COMPUTER NETWORKS

M.Sc. Computer Science

First Year, Semester-II, Paper-IV

Lesson Writers

Dr. Kampa Lavanya
Assistant Professor
Department of CS&E
Acharya Nagarjuna University

Dr. Neelima Guntupalli
Assistant Professor
Department of CS&E
Acharya Nagarjuna University

Dr. U. Surya Kameswari
Assistant Professor
Department of CS&E
Acharya Nagarjuna University

Dr. Vasantha Rudramalla
Assistant Professor
Faculty, Department of CS&E
Acharya Nagarjuna University

Mrs. Appikatla Pushpa Latha
Assistant Professor
Faculty, Deponent of CS&E
Acharya Nagarjuna University

Editor

Dr. U. Surya Kameswari
Assistant Professor
Department of CS&E
Acharya Nagarjuna University

Academic Advisor

Dr. Kampa Lavanya
Assistant Professor
Department of CS&E
Acharya Nagarjuna University

DIRECTOR, I/c.
Prof. V. Venkateswarlu
M.A., M.P.S., M.S.W., M.Phil., Ph.D.
CENTRE FOR DISTANCE EDUCATION
ACHARYA NAGARJUNA UNIVERSITY
NAGARJUNA NAGAR 522 510
Ph: 0863-2346222, 2346208
0863- 2346259 (Study Material)
Website www.anucde.info
E-mail: anucdedirector@gmail.com

M.Sc., (Computer Science) : COMPUTER NETWORKS

First Edition : 2025

No. of Copies :

This book is exclusively prepared for the use of students of M.Sc. (Computer Science), Centre for Distance Education, Acharya Nagarjuna University and this book is meant for limited circulation only.

Published by:

Prof. V. VENKATESWARLU
Director, I/c
Centre for Distance Education,
Acharya Nagarjuna University

Printed at:

FOREWORD

Since its establishment in 1976, Acharya Nagarjuna University has been forging ahead in the path of progress and dynamism, offering a variety of courses and research contributions. I am extremely happy that by gaining 'A+' grade from the NAAC in the year 2024, Acharya Nagarjuna University is offering educational opportunities at the UG, PG levels apart from research degrees to students from over 221 affiliated colleges spread over the two districts of Guntur and Prakasam.

The University has also started the Centre for Distance Education in 2003-04 with the aim of taking higher education to the door step of all the sectors of the society. The centre will be a great help to those who cannot join in colleges, those who cannot afford the exorbitant fees as regular students, and even to housewives desirous of pursuing higher studies. Acharya Nagarjuna University has started offering B.Sc., B.A., B.B.A., and B.Com courses at the Degree level and M.A., M.Com., M.Sc., M.B.A., and L.L.M., courses at the PG level from the academic year 2003-2004 onwards.

To facilitate easier understanding by students studying through the distance mode, these self-instruction materials have been prepared by eminent and experienced teachers. The lessons have been drafted with great care and expertise in the stipulated time by these teachers. Constructive ideas and scholarly suggestions are welcome from students and teachers involved respectively. Such ideas will be incorporated for the greater efficacy of this distance mode of education. For clarification of doubts and feedback, weekly classes and contact classes will be arranged at the UG and PG levels respectively.

It is my aim that students getting higher education through the Centre for Distance Education should improve their qualification, have better employment opportunities and in turn be part of country's progress. It is my fond desire that in the years to come, the Centre for Distance Education will go from strength to strength in the form of new courses and by catering to larger number of people. My congratulations to all the Directors, Academic Coordinators, Editors and Lesson-writers of the Centre who have helped in these endeavors.

Prof. K. Gangadhara Rao
M.Tech., Ph.D.,
Vice-Chancellor I/c
Acharya Nagarjuna University.

204CP24 COMPUTER NETWORKS

SYLLABUS

UNIT _ I

Introduction: Uses of Computer Networks - Business Applications, Home Applications, Mobile Users, Social Issues. Network Hardware - Local Area Networks - Metropolitan Area Networks Wide Area Networks - Wireless Networks - Home Networks - Internetworks. Network Software - Protocol Hierarchies - Design Issues for the Layers - Connection Oriented and Connectionless Services - Service Primitives - The relationship of Services to protocols. Reference Models - The OSI Reference Model - The TCP/IP Referlnce Model - A Comparison of OSI and TCP/iP reference Model - A Critique of the OSI Model and protocols - A Critique of the TCP/IP reference model. Example Networks - The Internet - Connection Oriented Networksx.25, Frame Relay, and ATM - Ethernet - Wireless LANs Network Standardization Who's who in the Telecommunication World - Who's who in the International Standards World- Who's who in the Internet Standards World.

Physical Layer: Guided Transmission Media - Magnetic Media - Twisted Pair - Coaxial Cable- Fiber Optics

Data Link Layer: Data Link Layer Design Issues - Services Provided to the Network Layer Framing - Error Control - Flow Control. Error Detection and Correction - Error correcting Codes - Error Detecting Codes. Elementary Data Link Protocols - An unrestricted Simplex Protocol- A simplex Stop-and-wait Protocol - A simplex Protocol for a Noisy channel. Sliding Window Protocols - A one-bit sliding Window Protocol - A Protocol using Go Back N - A Protocol using selective Repeat. Example Data Link Protocols - HDLC - The bata Link Laver in the Internet

UNIT -I I

The Medium Access Control Sublayer: Ethernet -Ethernet Cabling - Manchester Encoding The Ethernet MAC sublayer Protocol - The Binary Exponential backoff Algorithm - Ethernet Performance - Switched Ethernet - Fast Ethernet - Gigabit Ethernet - IEEE 802.2 Logical Link Control - Retrospective on Ethernet. WIRELESS LANS- The 802.11 protocol Stack - The 802'll Physical Layer The 802.11 MAC sublayer Protocol - The 802.11 Frame Structure. BLUETOOTH - Bluetooth Architecture - Bluetooth Applications - The Bluetooth Protocol Stack - The Bluetooth Radio Layer - The Bluetooth Baseband Layer - The Bluetooth L2CAP layer - The Bluetooth Frame Structure. Data Link Layer Switching - Bridges from 802.x to 802.y - Local Internetworking - Spanning Tree Bridges - Remote bridges J Repeaters, Hubs, Bridges, Switches, Routers and Gateways - Virtual LANs.

UNIT - III

The Network Layer: Network Layer Design Issues - Store-and- Forward packet Switching Services Provided to the Transport Layer - Implementation of Connectionless Services Implementation of Connection Oriented Services - -Comparison Of Virtual Circuit and Datagram subnets. Routing Algorithms - The Optimality Principle - Shortest path Routing - Flooding Distance Vector Routing - Link State Routing - Hierarchical Routing - Broad-cast Routing Multicast Routing - Routing for Mobile Hosts. Internet Working - How Networks Differ - How Networks can be connected - Concatenated Virtual Circuits - Connectionless Internetworking Tunneling - lnternet work Routing - Fragmentation. The Network Layer in the Internet - The IP Protocol - lP address - Internet Control Protocols - OSPF - The Internet Gateway Routing Protocol - BGP - The Exterior Gateway Routing Protocol.

UNIT - IV

The Transport Layer: The Transport Service - Services provided to the Upper Layers Transport Services Primitives - Berkeley Sockets. Elements of Transport Protocols - Addressing- Connection Establishment - Connection Release - Flow Control and Buffering - Multiplexing- Crash Recovery. The Intemet Transport Protocols : UDP
Introduction to UDP -
Remote Procedure Call- The Real Time Transport Protocol. The Internet Transport Protocols : TCP - Introduction to TCP - The TCP Service Model- the TCP Protocol The TCP segment header - TCP connection establishment - TCP connection release - Modeling TCP connection management- TCP Transmission Policy - TCP congestion Control - TCP Timer Management - Wireless TCP and UDP - Transactional TCP.

UNIT _ V

The Application Layer: The Domain Name System - The DNS Name Space - Resource Records - Name Servers. Electronic Mail - Architecture and Services - The User Agent Message Formats - Message Transfer - Final Delivery. The World Wide Web - Architecture Overview - Static Web Documents - Dynamic Web Documents - HTTP - The Hyper Text Transfer Protocol - Performance Enhancements - The Wireless Web. Multimedia - Introduction to Digital Audio - Audio Compression - Streaming Audio - Internet Radio - Voice Over IP Introduction to Video - Video Compression - Video on Demand.

Prescribed Book

Andrew S. Tanenbaum, "Computer Networks", Fourth Edition, PHI.
chapters l.l to 1.6,2.2,3.1 to 3.4,3.6,4.3,4.4,4.6.4.7,5.1,5.2.1to5.2.9,5.5,5.6.1to 5.6.5,6.1.1 to 6.1.3, 6.2, 6.4, 6.5, 7.1 to 7.4

Reference Books

l. James F.Kurose, Keith W.Ross, "Computer Networking", Third Edition. Pearson Education
2. Behrouz A Forouzan, "Data Communications and Networking", Fourth Edition, TMH (2007)
3. MichaelA. Gallo, William M. Hancock, "Computer Communications and Networking Technologies", Cengage Learning (2008).

M.Sc., (Computer Science)
MODEL QUESTION PAPER

204CP24 COMPUTER NETWORKS

Time: 3 Hours **Max. Marks: 70**

Answer ONE Question from Each Unit **5 × 14 = 70 Marks**

Answer ONE Question from each unit **5 x 14 = 70 M**

UNIT – I

1. a). Explain about the OSI reference model.
   b). Explain about the TCP/IP reference model.

(OR)

2. a). Write about the data link layer design issues.
   b). Explain about sliding window protocols.

UNIT – II

3. a). Explain Switched Ethernet.
   b). Give and explain 802.11 frame structure, services.

(OR)

4. a). Describe architecture, applications, protocol stack of Bluetooth.
   b). Explain Spanning Tree Bridges.

UNIT – III

5. a). Discuss about Flooding routing algorithm.
   b). Explain Distance Vector Routing Algorithm.

(OR)

6. a). Discuss about Tunneling and Fragmentation.
   b). Explain about IP Header Format and IP addresses.

UNIT – IV

7. a).Discuss about elements of transport protocols.
   b). Discuss about remote procedure call.

(OR)

8. a).Write about the TCP Protocol.
   b). Explain TCP Congestion Control.

UNIT – V

9. a) Explain about DNS.
   b) Write about URL's.

(OR)

10. a) Explain about Electronic Mail.
    b) Discuss JPEG Compression mechanism.

CONTENTS

LESSON- 1
INTRODUCTION

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the basic concept of computer networks.
- Identify major business applications of networks.
- Describe common home uses of computer networks.
- Recognize the role of mobile users in networking.
- Discuss key social issues related to network use.

STRUCTURE OF THE LESSON:

1.1 INTRODUCTION

The 18th, 19th, and 20th centuries each witnessed the emergence of transformative technologies that defined their eras. The 18th century was characterized by mechanical systems and marked the beginning of the Industrial Revolution. The 19th century saw the rise of the steam engine, a driving force for transportation and manufacturing. In the 20th century, the spotlight shifted to information—its collection, processing, and dissemination. This era gave birth to global telephone networks, radio, television, computers, communication satellites, and the Internet. These innovations reshaped how societies functioned and interacted. Moving into the 21st century, the previously distinct fields of data collection, transmission, storage, and processing are converging rapidly. Technological integration allows organizations with vast geographical footprints to access real-time data from any location with ease. This rapid advancement has fueled a growing demand for increasingly sophisticated information systems.

Computers, despite being a relatively new technology compared to industries like automobile manufacturing or aviation, have evolved at an astonishing pace. Initially, computers were massive, centralized machines housed in specialized rooms, often behind glass walls for public admiration. In the early days, even large institutions had only a few of these machines.

Yet, in less than half a century, computers have shrunk dramatically in size and exploded in power, leading to compact, powerful devices manufactured in the billions. This revolution has been further amplified by the fusion of computing and communication technologies, fundamentally altering how computing systems are structured. The outdated model of a single, centralized computer has given way to networks of interconnected systems, known as computer networks. These networks consist of multiple autonomous computers linked through various technologies such as copper wire, fiber optics, microwaves, infrared, or satellites. Whether small or large, these networks often interconnect to form even larger systems, with the Internet being the most prominent example of a global network of networks.

To clarify, a "computer network" is a group of independent computers that can exchange data through a common communication medium. It is important not to confuse this with a "distributed system." While both involve multiple computers, a distributed system operates in a way that appears as a unified whole to users. This illusion is created by middleware—a software layer that harmonizes the individual computers into a single user experience. A good example of a distributed system is the World Wide Web, which operates on the Internet but presents content uniformly as web documents or pages. In contrast, a computer network does not attempt to create a unified system image. The underlying differences in hardware, operating systems, and system behavior remain visible to users. For instance, if someone needs to use a program on a remote computer in a network, they must directly access that specific machine and run the program there. Essentially, while a distributed system overlays a cohesive software environment on a network, a basic computer network simply provides the infrastructure for communication and data exchange between machines.

1.2 USES OF COMPUTER NETWORKS

1.2.1 Business Applications

Most modern companies rely heavily on a significant number of computers to perform daily operations such as designing products, preparing documents, and managing payroll. Initially, these computers might have operated independently, but over time, the need to distribute and share information efficiently across the organization leads to connecting them into a network. This setup supports the broader objective of resource sharing, allowing employees to access software, data, and devices regardless of their physical location. For instance, instead of providing each employee with a separate printer, a single high-performance networked printer can serve an entire office, saving both costs and maintenance efforts.

Beyond sharing physical resources, the true strength of computer networks lies in their ability to facilitate information sharing. In today’s information-driven environment, data such as customer records, product specifications, inventory databases, and financial documents are vital for day-to-day operations. If a company loses access to its computerized information systems, it can suffer significant operational disruptions — banks may fail to function within minutes, and automated factories could halt instantly. Even small businesses like travel agencies or law firms heavily depend on networked systems to allow employees instant access to essential information.

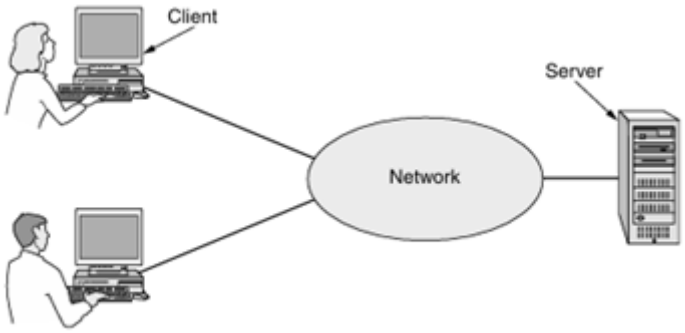


Figure 1.1 A network with two clients and one server

While small companies might host all their computers in one office or building, large enterprises are usually spread across various locations — sometimes even in different countries. Employees in distant locations, like a salesperson in New York needing access to a database in Singapore, depend on Virtual Private Networks (VPNs) to connect various geographically separated branches into one cohesive network. This arrangement allows remote users to access data as if it were stored locally, thus overcoming the limitations imposed by physical distance — often referred to as ending the "tyranny of geography."

In a simplified representation, a company's information infrastructure typically involves a centralized database hosted on powerful computers called servers. Employees, using their own desktop or laptop computers — known as clients — interact with these servers to access or update information, such as data used in spreadsheets. These client and server machines communicate via a network. While the client is the system requesting data or services, the server responds by providing the necessary resources. This interaction model is known as the client-server model and is fundamental to how modern networks operate.

This model is extensively applied in various domains, the most popular being web applications. In such applications, the client (user's web browser) sends a request to a server, which then processes the request — possibly using its internal database — and sends back a web page in response. The model works equally well whether the server and client are located in the same building or across the globe. For example, when someone browses a website from home, their computer functions as a client while the distant website's host acts as the server. Importantly, a single server is typically capable of managing communication with hundreds or even thousands of clients simultaneously, making this a highly scalable and efficient architecture.

When we examine the client-server model more closely, it becomes evident that it involves two distinct processes operating on separate machines — one on the client side and one on the server side. These processes are essentially running programs that communicate across the network. The communication begins when the client process initiates a message, known as a request, which is transmitted over the network to the server process. Once the server process receives this request, it processes the information or carries out the necessary action, and then returns a corresponding reply back to the client. This interaction between client and server — request and response — is central to the functioning of many network-based applications and services.

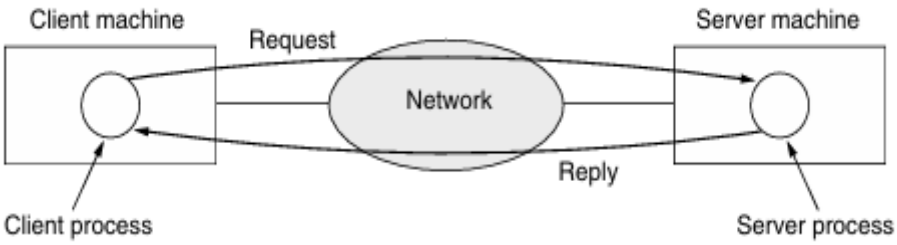


Figure 1.2. the client-server model involves requests and replies.

Beyond the exchange of data and the execution of programs, networks play a crucial role in facilitating communication among people. Most organizations today, regardless of their size, have adopted email as a standard communication tool, allowing employees to send and receive messages with ease. Email has become so pervasive in office culture that managing a high volume of messages — some of which may be trivial or unnecessary — is a common workplace complaint. Nevertheless, email significantly enhances internal communication, reducing the need for in-person meetings or memos.

Networks also support voice communication through technologies like IP telephony or Voice over IP (VoIP), which enable phone calls to be conducted over the Internet rather than traditional telephone lines. This allows companies to make cost-effective voice calls using VoIP-enabled phones or even computer microphones and speakers. Furthermore, richer communication experiences are made possible by the integration of audio and video. Video conferencing enables real-time face-to-face meetings between geographically distant colleagues, eliminating the need for travel and saving both time and resources. Similarly, desktop sharing allows multiple users to view and interact with the same computer screen, facilitating collaborative tasks such as editing documents or presentations. These features enable teams in different parts of the world to collaborate as if they were in the same room, with immediate updates and feedback. In advanced cases, such capabilities are being extended to fields like telemedicine, where healthcare providers can remotely monitor patients or consult on treatments from afar. These developments suggest a future where digital communication might surpass the need for physical travel altogether.

Another significant objective for companies in deploying computer networks is to engage in electronic business, particularly with external parties like customers and suppliers. This practice, known as e-commerce, has expanded rapidly due to its convenience and efficiency. Many industries — from airlines to retail — have embraced online platforms where consumers can browse products, compare options, and place orders without leaving their homes. Similarly, manufacturers who rely on parts from multiple suppliers use computer networks to coordinate their procurement processes. Orders can be placed electronically on demand, minimizing the need to maintain large inventories and improving overall productivity. In essence, networks are revolutionizing not only how companies operate internally but also how they conduct business with the outside world.

1.2.2 Home Application

The section discusses the various uses of computer networks in modern life. Initially, personal computers were used for tasks like word processing and games, but now their primary value lies in Internet connectivity. Computer networks enable access to vast remote information (e.g., websites, digital libraries), support person-to-person communication (e.g., email, messaging, video calls), and drive social networking and collaborative platforms.

They also power electronic commerce, including online shopping, banking, and auctions. Entertainment has shifted online through music, video streaming, and multiplayer gaming. Lastly, with ubiquitous computing, everyday devices like smart appliances and sensors are now networked, leading to innovations such as smart homes and the Internet of Things (IoT), where even simple objects like cameras and showers can communicate over networks.

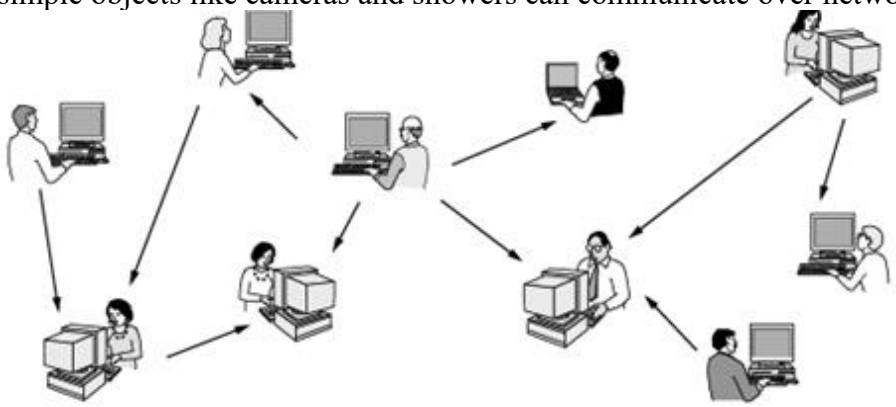


Figure 1.3 in a peer -to-peer system there are no fixed clients and servers

Computer networks have transformed the role of personal computers from standalone tools used for basic tasks like word processing or gaming into powerful gateways to the global digital world. Today, the real value of computers lies in their ability to connect to the Internet and tap into an immense reservoir of information and services. Networks provide access to countless remote resources such as websites, online encyclopedias, digital libraries, and educational content that can be reached instantly from anywhere in the world. In addition to accessing information, networks have become the backbone of human communication in the digital age. They support various forms of interpersonal interaction — including email, instant messaging, audio calls, and video conferencing — that allow people to communicate seamlessly across the globe, whether for personal or professional purposes.

Furthermore, the advent of social media platforms has enabled users not just to communicate but also to share their lives, collaborate on projects, and form digital communities. Beyond communication and information, networks have revolutionized how we conduct business through e-commerce. Consumers can now shop online, transfer money via digital banking, and participate in online auctions, all from the comfort of their homes. The entertainment industry has also undergone a radical shift, with music, movies, television shows, and multiplayer online games all being streamed or played over the Internet.

Perhaps most revolutionary is the integration of networks into everyday physical devices. With the rise of ubiquitous computing and the Internet of Things (IoT), an increasing number of appliances — from refrigerators and washing machines to security cameras and smart showers — are now equipped with sensors and network connectivity. These smart devices

can collect data, communicate with one another, and be remotely controlled or automated, giving rise to intelligent environments such as smart homes. In such settings, users can manage lighting, temperature, security, and even daily routines using their smartphones or voice commands, marking a significant leap in convenience and efficiency driven entirely by computer networks.

1.2.3 Mobile Users

Mobile computers, such as laptops, tablets, and smartphones, have become indispensable in modern life due to the flexibility they provide in accessing digital content and online services from nearly any location. People rely on these devices to perform everyday activities such as checking emails, streaming videos, browsing the web, using social media, and even completing professional work on the go. In many situations—such as inside vehicles, airplanes, public parks, or other areas where wired infrastructure is impractical or unavailable—wireless communication becomes essential. Technologies like Wi-Fi, cellular networks (3G, 4G, 5G), and satellite communication allow mobile devices to maintain Internet connectivity and perform real-time data transactions efficiently and reliably.

The influence of mobile computing extends across various industries and public services. In the transportation sector, wireless systems help taxi fleets with real-time dispatching and routing, while rental car agencies use handheld devices to track vehicle returns and availability. In retail, mobile point-of-sale systems and inventory scanners streamline customer service and stock management. Mobile devices also play a significant role in military operations and healthcare, where quick access to data and communication can be critical. Smartphones in particular have evolved from simple communication tools into sophisticated computing platforms. Equipped with high-resolution cameras, GPS receivers, accelerometers, and secure payment options through technologies like NFC (Near Field Communication) and RFID (Radio Frequency Identification), they support a wide range of services—from turn-by-turn navigation and mobile banking to barcode scanning for price comparisons and contactless purchases.

The development of mobile and wireless computing has also led to the rise of advanced technologies like sensor networks and wearable or implantable computing devices. These innovations allow for remote monitoring and automation in applications such as wildlife tracking, smart parking systems, and telemedicine. For example, wearable health devices can transmit patient data to doctors in real time, while sensors embedded in the environment or in equipment can optimize energy use or detect anomalies. As mobile and wireless technologies continue to evolve, their integration into everyday life is expected to deepen, fostering new levels of convenience, efficiency, and interconnectivity that will fuel further innovations in areas we are only beginning to imagine.

1.2.4 Social Issues

Computer networks have revolutionized modern communication, enabling ordinary individuals to distribute and access content in unprecedented ways—much like the printing press did centuries ago. This transformation has empowered users across the globe to freely express their views, share information, and engage in collaborative efforts online. While this democratization of content creation and sharing has opened up countless positive avenues—such as educational discussions, hobby forums, and knowledge exchange—it also introduces complex and unresolved social, political, and ethical challenges.

One significant issue arises from the type of content shared online. While sharing neutral or niche interests, like gardening or technology, rarely causes disputes, the landscape becomes far more contentious when discussions involve sensitive topics such as politics, religion, or sex. People can easily publish text, images, and videos that others may find offensive or inflammatory. Some users advocate for freedom of expression under any circumstances, whereas others call for strict content regulation or censorship, especially when the material is seen as hateful, obscene, or politically incorrect. However, opinions vary widely across cultures and countries, making global regulation of online content nearly impossible. This ongoing tension has sparked heated debates about censorship, freedom of speech, and the responsibility of network operators.

Legal complications further complicate the matter. Individuals and organizations have attempted to hold network operators accountable for the content transmitted over their infrastructure, likening them to publishers who are responsible for what they distribute. In contrast, operators argue that they are more like telephone companies or postal services—merely conduits that should not be held liable for the messages their users send. Nevertheless, some network providers have taken steps to restrict or shape traffic for business reasons. For example, users of peer-to-peer (P2P) file-sharing applications have found their connections throttled or cut off entirely because these applications consume large amounts of bandwidth. This selective treatment of traffic has led to a fierce debate over "network neutrality," a principle asserting that all Internet traffic should be treated equally, regardless of its source, destination, or content. Advocates for neutrality argue that preferential treatment could undermine the open and fair nature of the Internet.

Adding to this complex mix is the widespread issue of copyright infringement, especially through file-sharing services. P2P networks became notorious for enabling the illegal distribution of music, movies, and other copyrighted content. This angered rights holders, prompting legal battles and the development of automated systems to monitor and respond to violations. In the U.S., these efforts include DMCA takedown notices, which inform users and service providers of alleged copyright breaches. Despite such mechanisms, identifying infringers reliably remains difficult, leading to an ongoing cat-and-mouse game between enforcers and violators.

Privacy concerns are another major consequence of widespread networking. The same technologies that make communication seamless also make surveillance easier. Employers often monitor their employees' emails and online activity—sometimes even messages sent from personal devices outside of work hours—raising concerns about the balance between corporate rights and individual privacy. On a broader scale, governments have developed sophisticated surveillance tools to monitor email and Internet activity. For instance, the FBI once used a system called "Carnivore" (later renamed DCS1000) to scan incoming and outgoing emails for signs of criminal activity. Critics argue such systems infringe upon constitutional rights, such as the Fourth Amendment in the United States, which prohibits unreasonable searches and seizures without a warrant. Nevertheless, these programs often operate under a veil of secrecy and questionable legality.

Private companies also play a significant role in compromising user privacy. Web services and browsers track users using small files called cookies, which collect data about browsing habits and personal information. This data is often used for targeted advertising or sold to third parties, raising ethical concerns about consent and data security. Services like Gmail scan user emails to deliver personalized ads, and companies accumulate vast amounts of user

data, giving them the power to create detailed behavioral profiles. The rise of mobile devices has added another layer to this issue: location tracking. To function, mobile networks must know a user's location, enabling operators to build detailed logs of where individuals go, when, and how often—information that can be sensitive or intrusive.

On the flip side, computer networks can enhance privacy in certain cases. Tools for sending anonymous messages allow whistleblowers—such as employees, students, or citizens—to report illegal or unethical conduct without fear of retaliation. This form of anonymity can be crucial in authoritarian regimes or hostile environments. However, it also poses challenges for the legal system. In many democracies, an accused individual has the right to confront their accuser, making anonymous accusations inadmissible in court. Balancing anonymity for protection with accountability for fairness remains an unresolved issue.

Despite the Internet's convenience, the quality and accuracy of information online can vary wildly. While valuable resources from reputable experts are available, misinformation and false content are equally rampant. Users often encounter medical advice or legal information that is incorrect or dangerous. Compounding the issue is spam—unwanted electronic junk mail that clogs inboxes and consumes bandwidth. Spammers exploit email databases to send unsolicited messages, though spam filters now attempt to mitigate this problem with varying levels of success.

More dangerous still are malicious uses of the Internet, including cybercrime. Emails and websites with embedded active content—scripts, macros, or programs—can infect a user's computer with viruses. These may be used to steal personal data, access bank accounts, or hijack machines into botnets used for further attacks. Phishing is another serious threat, in which attackers impersonate trusted entities, like banks, to trick users into revealing sensitive information. This form of fraud contributes to identity theft, where criminals gather enough personal data to pose as someone else and commit financial crimes.

A unique challenge on the Internet is verifying whether someone is human. To combat bots pretending to be people—for instance, during online registrations or ticket purchases—developers use CAPTCHAs, which require users to solve simple puzzles or recognize distorted text. This approach is inspired by the Turing Test, which evaluates whether a machine can mimic human behavior convincingly. While effective, such tests also highlight the difficulty of distinguishing between real users and automated systems in a digital environment.

Many of the issues outlined above stem from a lack of serious commitment to computer security. Although technologies like encryption and authentication are available and mature, they are not widely implemented. Hardware and software vendors often prioritize adding new features over improving security, resulting in bloated, bug-ridden software vulnerable to attacks. Introducing penalties for buggy or insecure products might help—but such measures are unlikely to be accepted widely and could devastate the software industry.

Finally, legal complications arise when digital actions intersect with outdated laws. A notable example is online gambling. While gambling may be banned in some countries, it is legal in others, and online casinos hosted abroad create complex jurisdictional dilemmas. When a user, server, and casino operator are located in different legal territories, enforcing any particular nation's laws becomes extraordinarily difficult. As a result, the legal system struggles to keep pace with the realities of global computer networks.

1.3 SUMMARY

Computer networks connect devices to share data and resources efficiently. They are widely used in business for communication, online services, and data management; in homes for internet access, entertainment, and smart devices; and by mobile users for wireless connectivity on the go. However, increased network use also brings social issues such as privacy risks, security threats, and digital inequality.

1.4 TECHNICAL TERMS

Computer Networks, Smart devices, Wireless, Internet access

1.5 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the business applications of computer networks.
2. Describe various home applications of computer networks.
3. Discuss the importance of mobile users in networking.
4. Explain the major social issues caused by network usage.
5. Compare business, home, and mobile network applications

Short Questions:

1. What is a computer network?
2. Mention any two uses of networks in business.
3. Write two examples of home applications of networks.
4. Who are mobile users?
5. List two social issues arising from computer networks.

1.6 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008)

Dr. Kampa Lavanya

LESSON- 2

NETWORK HARDWARE AND SOFTWARE

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the types and functions of network hardware.
- Differentiate LAN, MAN, WAN, and wireless networks.
- Describe the concept of internetworks and their importance.
- Explain network software and protocol hierarchies.
- Understand the relationship between services and protocols.

STRUCTURE OF THE LESSON:

2.1 NETWORK HARDWARE

This section shifts focus from the social aspects of networking to the technical foundations, particularly transmission technology and network scale. There are two main types of transmission: point-to-point, where individual machines are connected directly and data may travel through intermediate nodes (unicast), and broadcast, where all machines share the communication channel and packets are received by all, though only the intended recipient processes it. Broadcast networks can also support broadcasting (to all machines) and multicasting (to selected groups). Networks are also classified by scale, ranging from Personal Area Networks (PANs) for a single user, to Local Area Networks (LANs),

Metropolitan Area Networks (MANs), and Wide Area Networks (WANs). Connecting multiple networks forms an internetwork, with the Internet being the most prominent example. Future developments may even extend to the Interplanetary Internet, linking networks across space.

Interprocessor distance	Processors located in same	Example
1 m	Square meter	Personal area network
10 m	Room	Local area network
100 m	Building	
1 km	Campus	
10 km	City	Metropolitan area network
100 km	Country	Wide area network
1000 km	Continent	
10,000 km	Planet	The Internet

Figure 2.1 classification of interconnected processors by scale

2.2. PERSONAL AREA NETWORK

Personal Area Networks (PANs) are small-scale networks designed to enable communication between devices located within the direct proximity of an individual, usually within a range of a few meters. These networks allow personal devices to share data and resources without requiring complex infrastructure. The primary purpose of a PAN is to support the connectivity needs of a single user, enabling seamless interaction among personal devices such as smartphones, laptops, tablets, wearables, and peripherals.

One of the most common wireless technologies used in PANs is Bluetooth. Bluetooth removes the need for physical cables by allowing devices to communicate wirelessly. For instance, a computer can connect to a mouse, keyboard, speakers, or printer via Bluetooth, creating a convenient and clutter-free working environment. The Bluetooth communication model typically operates using a master-slave configuration, where one device acts as the master (often the computer or smartphone), controlling and coordinating communication with multiple slave devices. This structure ensures efficient and orderly data exchange among connected devices.

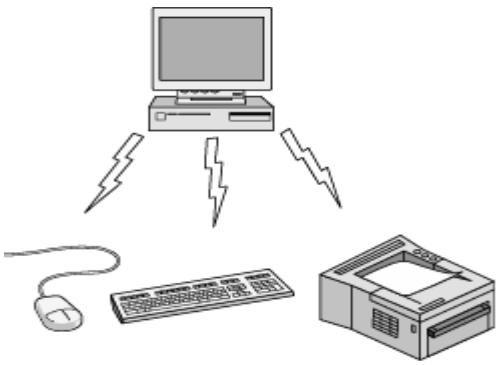


Figure 2.2 Bluetooth PAN configuration.

Bluetooth technology extends beyond computers and office peripherals. It is widely used in mobile applications, such as connecting smartphones to wireless headsets, earbuds, smartwatches, car infotainment systems, and fitness trackers. In the healthcare domain, Bluetooth plays a crucial role in medical monitoring devices, such as pacemakers, glucose monitors, and fitness sensors, enabling real-time data transmission to user interfaces or healthcare providers. These applications highlight Bluetooth's importance in supporting mobility, accessibility, and user convenience.

Other short-range wireless technologies also contribute to PANs. Radio-Frequency Identification (RFID) is another popular technology used in scenarios where automatic identification and tracking are required. RFID is commonly used in smartcards, contactless payment systems, library book tracking, tollgate systems, and access control environments. Unlike Bluetooth, RFID does not necessarily require active user participation and can operate with minimal power, making it suitable for embedded or tag-based systems. Together, Bluetooth and RFID demonstrate the versatility of PANs in everyday life, enabling seamless and efficient communication across a variety of personal and professional applications.

2.3 LOCAL AREA NETWORK

A Local Area Network (LAN) is a privately managed communication network that connects multiple devices within a relatively small geographical area, such as a home, office, school, or a single building. The primary purpose of a LAN is to enable connected devices to share resources—including files, printers, software applications, and internet connectivity—and to facilitate fast and efficient communication. Because of its limited coverage area, a LAN typically offers high data transfer speeds and reliable performance compared to networks spanning larger regions.

LANs can be established using either wired or wireless technologies. Wired LANs usually rely on Ethernet cables, switches, and routers. They are known for their high speed, stable connections, low latency, and low susceptibility to interference. Due to these features, wired LANs are often preferred in environments where performance and reliability are critical, such as offices, laboratories, and data centers. On the other hand, wireless LANs (WLANs), typically implemented using Wi-Fi and access points, provide greater flexibility and convenience. Wireless networks eliminate the need for physical cables, making them ideal for homes, public spaces, and workplaces where mobility and ease of installation are important.

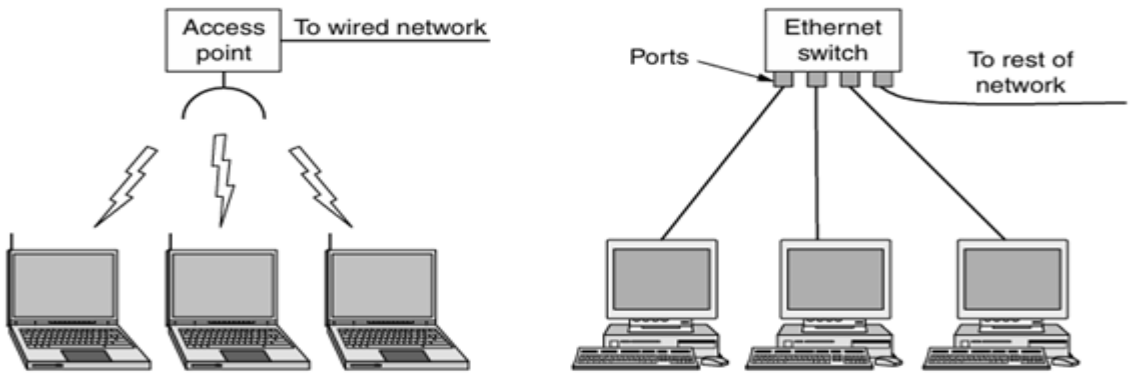


Figure 2.3 Wireless and wired LANs. (a) 802.11 (b) Switched Ethernet

LANs can be scaled and managed using various networking devices and techniques. Switches allow more devices to be added to the network without compromising performance, while Virtual LANs (VLANs) enable logical segmentation of the network for better organization, security, and traffic management. VLANs are particularly useful in large organizations where different departments or groups need isolated network environments while still sharing the same physical infrastructure.

In modern homes, the importance of LANs has increased significantly due to the rise of smart devices, including smart TVs, gaming consoles, IoT appliances, and home automation systems. However, setting up and maintaining a LAN still presents challenges, especially for non-technical users. Factors such as affordability, ease of installation, network security, and user-friendly configuration play a crucial role in widespread adoption. To address these challenges, alternative networking solutions like power-line communication networks are emerging. These networks utilize existing electrical wiring to transmit data, offering a cost-effective and convenient solution for extending connectivity without installing new cables. Such advancements continue to enhance the usability and accessibility of LANs in everyday environments.

2.4 METROPOLITAN AREA NETWORK

A Metropolitan Area Network (MAN) is a type of network that covers a larger geographic area than a Local Area Network (LAN), typically spanning an entire city or a large campus. MANs are designed to provide high-speed connectivity across urban environments, connecting multiple LANs together to form a unified network infrastructure. They are commonly used by organizations, government agencies, and service providers to support data communication, Internet access, and distributed services across various locations within a metropolitan region. MANs play an important role in enabling efficient communications, especially in densely populated areas where reliable and widespread network coverage is essential.

One of the earliest and most recognizable examples of a MAN is the cable television network. Cable TV systems originated as community antenna networks, where a shared antenna was installed to improve television reception for areas with poor broadcast signal quality. Over time, these systems expanded and developed into city-wide distribution networks operated by cable service providers under government regulation. As technology advanced, cable networks began to utilize unused frequency bands in their coaxial cables to support two-way data transmission. This innovation allowed cable companies not only to deliver television programming but also to offer broadband Internet services, effectively transforming cable TV networks into large-scale MANs serving millions of users.

Another important modern example of a MAN is WiMAX, based on the IEEE 802.16 standard. WiMAX provides wireless broadband connectivity across large urban and suburban areas, making it suitable for delivering Internet access to locations where wired infrastructure is difficult or expensive to install. Unlike Wi-Fi, which is typically limited to smaller local areas, WiMAX can cover distances of several kilometers, offering high-speed data services to homes, businesses, and public hotspots. This capability makes WiMAX a powerful solution for expanding digital access, improving communication infrastructure, and supporting smart city applications.

2.5 WIDE AREA NETWORK

A Wide Area Network (WAN) is designed to connect computers and networks over large geographical distances, often spanning cities, states, countries, or even continents. WANs allow organizations with multiple branches or remote locations to share data and communicate as though they were part of a single unified network. For example, a business with offices in Perth, Melbourne, and Brisbane may use a WAN to link those offices together so employees can collaborate and access shared systems seamlessly. Within each branch, individual computers (hosts) carry out user tasks, while the underlying communication infrastructure—known as the subnet—transmits data between locations. This subnet consists of transmission lines, such as fiber-optic cables, copper wires, or radio links, and routers that direct data packets across the network.

A key characteristic of WANs is that their components often have different ownership. The hosts (like company laptops and desktops) are owned and controlled by the organization, while the transmission lines and sometimes even the routing infrastructure are typically operated by telecommunications companies. Businesses frequently lease bandwidth or dedicated lines from network carriers to enable long-distance communication. Furthermore, WANs commonly interconnect multiple local area networks (LANs), creating larger internetworks that support diverse communication technologies. Routers play a crucial role here by linking different network types, such as office Ethernet networks with long-distance fiber-based carrier systems.

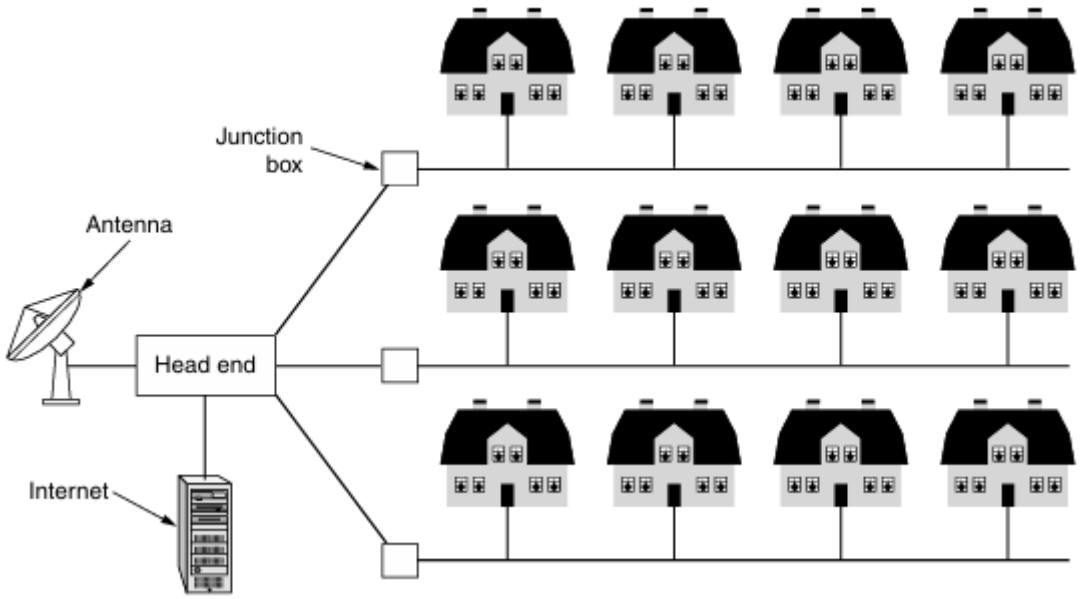


Figure 2.4 A metropolitan area network based on cable TV

WANs may be implemented in different ways depending on organizational needs. One common approach is the use of a Virtual Private Network (VPN), where secure, encrypted communication "tunnels" are created over the public Internet. VPNs offer flexibility and cost savings because they do not require renting dedicated physical lines, but they depend on the performance of the public Internet, which may sometimes limit reliability or bandwidth control. Another approach is to use a WAN provided and managed directly by an Internet Service Provider (ISP), which owns and operates the subnet infrastructure. In this model, the ISP connects customers to each other and to the broader global network.

Routing and forwarding are critical processes in WAN operations. The routing algorithm determines the best path that data should follow from source to destination, while the forwarding process is carried out by routers, which move packets step-by-step through the network. WANs may also include wireless technologies. Satellite-based networks provide broad geographic coverage and are useful in remote or rural regions, while cellular networks use base stations with ranges of several kilometers. These cellular networks have evolved through multiple generations, beginning with analog voice (1G), advancing to digital voice (2G), adding mobile data (3G), and later offering high-speed broadband data services (4G and beyond). Although cellular data rates are typically lower than those of wireless LANs, their coverage areas are much larger.

Together, these technologies and network structures form the communication backbone that supports global connectivity. Each WAN implementation involves trade-offs among performance, cost, reliability, and scalability, and organizations choose their WAN setups based on their operational requirements and service needs.

2.6 INTERNETWORKS

An internetwork, commonly referred to as an internet (with a small "i"), is created when multiple individual networks are connected to enable communication across them. These networks may vary in size, structure, and underlying technology. The most prominent and globally recognized example of such an internetwork is the Internet (with a capital "I"), which interconnects millions of networks worldwide. The concept of internetworking allows users and devices located in different regions and operating in different environments to exchange information as if they were part of the same network.

In understanding internetworks, it is helpful to distinguish between the terms network and subnet. A network typically includes both the hosts—such as computers, smartphones, servers, and other devices used by end users—and the communication system connecting them. The communication system itself, which consists of routers, switches, communication links, and transmission technologies, is called the subnet. The subnet's role is to carry data from one host to another, whether across a room or across continents.

Internetworks are particularly important when networks are owned, managed, or maintained by different organizations or when they use different communication technologies. For example, a wired office LAN, a wireless campus network, and a cellular network operated by a telecom provider may all be interconnected to enable seamless communication. Because these networks are heterogeneous, specialized devices are needed to link them together. The devices used to connect and translate between different networks are called gateways.

Among gateway devices, routers are the most commonly used. Routers operate at the network layer of the OSI model, which allows them to make forwarding decisions based on IP addresses rather than hardware characteristics. This independence from physical technology makes routers well-suited for integrating different types of networks, whether wired, wireless, or satellite-based. By examining the destination address of each data packet and determining the best available route, routers ensure that data is transmitted efficiently between networks, enabling global communication regardless of underlying differences in hardware, network structure, or application software.

2.7 NETWORK SOFTWARE

In the early days of computer networking, the primary focus was on developing the hardware components required to transmit digital data across communication channels. Software played only a minor role and was often considered after the physical devices were in place. However, as networks grew in size, complexity, and application diversity, it became apparent that hardware alone could not efficiently manage the wide range of communication tasks. This led to the development of network software, which provides structure, control, reliability, and flexibility to communication systems.

Modern computer networks are now designed with a highly organized approach to software, using layered architectures. Each layer is responsible for a specific set of functions and interacts with the layers above and below it. This modular approach allows network systems to be designed, implemented, updated, and managed more efficiently. It also encourages interoperability across different hardware platforms, operating systems, and vendors.

This chapter explores the foundational concepts that make up network software. It begins with protocol hierarchies, which describe how communication tasks are divided into layers. Then, it addresses design issues that must be considered when creating layered network systems. It also explains the difference between connectionoriented and connectionless communication services, which define how communication occurs between devices. The chapter further discusses service primitives, which describe how network services are invoked by programs, and concludes with the relationship between services and protocols, clarifying how they work together to achieve seamless data communication.

2.8 PROTOCOL HIERARCHIES

To simplify the design and management of complex networks, most network systems are built using a layered architecture. In this structure, the complete networking functionality is divided into multiple layers, where each layer focuses on a specific set of tasks. The main idea behind this approach is to divide large, complex problems into smaller, manageable parts. Each layer provides certain services to the layer above it and relies on the layer below it to perform more basic operations. What is important is that each layer performs its role independently, without needing to know the internal details of how other layers work. This separation ensures clarity, maintainability, and ease of troubleshooting.

Communication in such layered systems happens in two ways. On one hand, layers communicate horizontally with their corresponding layers on another machine using protocols. A protocol is a strict set of rules that govern how data is formatted, transmitted, received, and interpreted. For example, two transportlayer protocols on different devices must agree on how to establish connections, detect errors, and manage data flow. On the other hand, actual data movement occurs vertically within a single system, from one layer to the next. As data moves down the layers for transmission, each layer may add control information in the form of headers (and sometimes trailers) which help the receiving system process the data correctly.

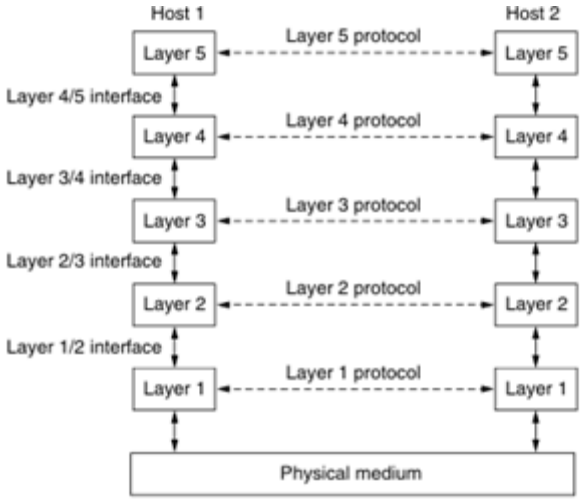


Figure 2.5 Layers ,protocols and interface.

The point where two layers meet is known as an interface. This interface clearly defines what services the lower layer provides to the upper one and how those services can be accessed. Importantly, it does not reveal how the services are implemented internally—this allows technology to evolve without affecting the whole network. A network architecture is the complete set of layers along with their functions and interactions. The collection of all protocols used across these layers is known as the protocol stack. Wellknown examples include the TCP/IP protocol stack used on the Internet and the OSI reference model commonly used for understanding network concepts.

For example, when two users on different machines communicate, the sender's data starts at the topmost layer and travels downward. Each layer adds its own header containing control information, such as addressing details or errorchecking codes. The data is then transmitted through physical media to the destination machine. On the receiving end, this process occurs in reverse: as data travels upward through the layers, each layer removes its corresponding header and processes the information. This structured approach not only simplifies network design but also makes it easy to modify or upgrade individual layers without affecting the entire system—as long as the interfaces between layers remain unchanged.

2.9 DESIGN ISSUES FOR LAYERS

When designing layered network systems, several important issues must be considered to ensure smooth and efficient communication. One of the primary concerns is reliability. Networks often operate over channels that may introduce noise, interference, or packet loss. To ensure that data reaches its destination correctly, mechanisms such as error detection, error correction, and automatic retransmission are used. Additionally, networks may have multiple possible paths between devices, so alternate routing techniques can be used to reroute data if one path fails, thereby enhancing fault tolerance.

Another crucial issue is scalability, which refers to the network's ability to expand without suffering performance degradation. As more devices, users, and data flows are added, the network must continue to operate efficiently. Proper addressing mechanisms help support large numbers of devices by assigning unique identifiers to senders and receivers. Addressing also allows routers and switches to correctly forward data to the intended destination.

Modern networks consist of different types such as LANs, WANs, and wireless networks. Internetworking ensures that these diverse networks can communicate seamlessly. Protocols and gateways are often required to translate formats and manage differences in transmission rates and addressing schemes.

Resource allocation is another key design issue. Since network bandwidth is limited, it must be shared efficiently among multiple users and applications. Techniques such as flow control prevent a fast sender from overwhelming a slow receiver, while congestion control helps avoid overload in the network itself. Methods like statistical multiplexing allow multiple data streams to share capacity dynamically, improving utilization of network resources.

Networks must also consider Quality of Service (QoS), especially for applications like video streaming, voice calls, and online gaming. QoS mechanisms help ensure timely delivery, minimize delays, reduce packet loss, and maintain consistent performance according to the needs of each application.

Finally, security is an essential design consideration. Data transmitted across networks may be exposed to threats such as eavesdropping, tampering, identity spoofing, or malicious attacks. To defend against these risks, techniques such as encryption protect confidentiality, while authentication verifies identity, and integrity checks ensure that transmitted data has not been altered.

2.10 CONNECTION-ORIENTED VS CONNECTIONLESS SERVERS

Network communication services are generally classified into two categories: connection-oriented and connectionless services. In a connection-oriented service, a communication path is first established between the sender and the receiver before any data transfer takes place. This is similar to making a telephone call, where the caller and receiver are connected first, and only then does the conversation begin. Once the connection is set up, data is transmitted in a reliable and sequential manner, ensuring that packets arrive in the correct order and without errors. If a packet is lost or corrupted, the system detects it and retransmits the packet. This type of service is suitable for applications that require high reliability and data integrity, such as file transfer, remote login, or audio/video streaming where correct and ordered delivery of data is essential.

In contrast, a connectionless service does not establish any dedicated path before sending data. Each message, called a datagram, is sent independently, much like sending postal mail. There is no guarantee that the data will arrive at the destination, arrive in the correct order, or arrive only once. Because of this, connectionless services are often referred to as unreliable services. However, connectionless communication has the advantage of speed and low overhead, as it does not require connection setup or maintenance. This makes it highly efficient for applications where quick delivery is more important than perfect accuracy, such as broadcasts, live audio/video calls, or sensor networks. In some cases, reliability can still be added to connectionless communication through the use of acknowledgements and retransmission mechanisms, but this is optional and depends on application requirements.

	Service	Example
Connection-oriented	Reliable message stream	Sequence of pages
	Reliable byte stream	Movie download
	Unreliable connection	Voice over IP
Connectionless	Unreliable datagram	Electronic junk mail
	Acknowledged datagram	Text messaging
	Request-reply	Database query

Figure 2.6 Six different types of service

Figure 2.6 illustrates that network services can vary widely in terms of reliability, ordering, and delivery guarantees. Some services provide reliable, ordered delivery, while others offer best-effort delivery with no guarantees. Both connection-oriented and connectionless services coexist because different applications have different needs. For example, voice and video conferencing require fast delivery even if some data is lost, whereas activities like bank transactions and file downloads require perfect accuracy and sequencing. Therefore, choosing the appropriate type of service depends on the nature and requirements of the application.

2.11 SERVICE PRIMITIVES

To enable communication between applications over a network, the operating system provides a set of basic operations known as service primitives. These primitives act as an interface between the network software and user-level programs, allowing them to request network services. In a connection-oriented service, these primitives are used to establish a connection, exchange data, and then properly terminate the connection. The most commonly used primitives are LISTEN, CONNECT, ACCEPT, SEND, RECEIVE, and DISCONNECT.

Primitive	Meaning
LISTEN	Block waiting for an incoming connection
CONNECT	Establish a connection with a waiting peer
ACCEPT	Accept an incoming connection from a peer
RECEIVE	Block waiting for an incoming message
SEND	Send a message to the peer
DISCONNECT	Terminate a connection

Figure 2.7 Six service primitives that provide a simple connection

In a typical client–server model, the server begins by executing the LISTEN primitive, which instructs the server to wait for incoming connection requests. Meanwhile, the client uses the CONNECT primitive to request a connection with the server. Once the request arrives, the server responds by executing the ACCEPT primitive, which formally establishes the

connection. At this point, a communication channel is ready, and both client and server can exchange data.

The data transmission takes place using the SEND and RECEIVE primitives. The SEND primitive allows a program to transmit data to the other side, while the RECEIVE primitive retrieves incoming data. These operations ensure that both parties can communicate back and forth in an orderly manner. After the communication is complete, either side can invoke the DISCONNECT primitive to close the connection and free network resources.

This process is very similar to how a telephone conversation works: a call is first established, conversation takes place, and finally the call is ended properly. Although connectionless communication requires fewer steps—since data is sent without establishing a prior connection—it lacks the reliability and ordering that many applications require. For tasks such as remote login, file transfer, and database transactions, connection-oriented communication using service primitives ensures accurate, sequential, and error-free data delivery.

2.12 THE RELATIONSHIP OF SERVICES TO PROTOCOL

In layered network architectures, it is essential to clearly distinguish between services and protocols. A service refers to the functionality that a lower layer provides to the layer above it. It defines what operations can be performed, usually described in terms of service primitives such as connect, send, receive, or disconnect. Importantly, a service specifies what is available, but not how the operations are implemented. This creates a stable and consistent interface, allowing the upper layers to use the service without needing to know the internal workings of the lower layer.

A protocol, by contrast, defines the rules and conventions used by corresponding layers (called peer entities) on different machines to communicate. Protocols describe the format, meaning, and order of messages exchanged between peers to ensure that data is transmitted correctly. These rules include how errors are detected, how data is acknowledged, how flow is controlled, and how messages are sequenced. Protocols operate horizontally across the network, enabling machines to coordinate and interpret each other's data.

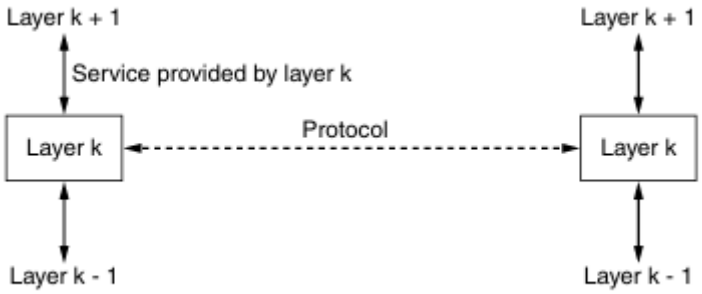


Figure 2.8 The relationship between a service and a protocol

The key idea is that services and protocols are conceptually separate. Services define the interface between layers within the same machine, while protocols define the communication mechanism between equivalent layers on different machines. Because of this separation, protocols can be modified or optimized (for example, to improve efficiency, reliability, or

performance) without changing the service interface seen by the higher layers. Likewise, different protocols may provide the same service, as long as the expected service behavior remains consistent.

An analogy helps clarify this distinction: a service is like an abstract data type or API that specifies available operations, while a protocol is the behind-the-scenes implementation logic that makes those operations work. Earlier network designs often mixed these two concepts, making networks harder to modify or troubleshoot. Modern layered network architectures maintain a clear separation, which enhances flexibility, modularity, and maintainability.

2.13 SUMMARY

This chapter explains the hardware and software components of computer networks. It introduces different types of network hardware such as LANs, MANs, WANs, wireless networks, home networks, and internetworks, each serving different geographical areas and communication needs. On the software side, the chapter covers protocol hierarchies, layer design issues, and the distinction between connection-oriented and connectionless services. It also describes service primitives and explains how services and protocols work together to ensure reliable communication between networked systems.

2.14 TECHNICAL TERMS

LAN, WAN, MAN, Connection-oriented, Connectionless, Protocol

2.15 SELF ASSESSMENT QUESTIONS

Essay Questions:

1. Explain different types of network hardware: LAN, MAN, WAN, and wireless networks.
2. Describe the architecture and components of home and internetworks.
3. Explain protocol hierarchies and the design issues for network layers.
4. Differentiate between connection-oriented and connectionless services with examples.
5. Discuss service primitives and how services relate to protocols in network communication.

Short Questions:

1. What is a Local Area Network (LAN)?
2. Define a Wide Area Network (WAN).
3. What is meant by a protocol hierarchy?
4. Distinguish between connection-oriented and connectionless services.
5. What is the relationship between services and protocols?

2.16 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008).

Dr. Kampa Lavanya

LESSON- 3

REFERENCE MODELS

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the concept of network reference models.
- Describe the OSI reference model and its seven layers.
- Explain the structure and layers of the TCP/IP model.
- Compare the OSI and TCP/IP reference models.
- Discuss the critiques and real-world relevance of both models.

STRUCTURE OF THE LESSON:

3.1 REFERENCE MODELS

This section introduces two key network architectures: the OSI model and the TCP/IP model. The OSI model is valuable for understanding concepts and design, even though its protocols are outdated. In contrast, the TCP/IP model is widely used in real-world networks, though its structure is less formal. Studying both provides a complete understanding—highlighting practical success (TCP/IP) and conceptual clarity (OSI), as well as lessons from each model's strengths and weaknesses.

3.2 OSI REFERENCE MODEL

The OSI model is a seven-layer reference model developed by the ISO to standardize how open systems communicate over a network. It outlines what each layer should do, not how to do it, making it a conceptual framework rather than a concrete architecture. The layers, from bottom to top, are: Physical, Data Link, Network, Transport, Session, Presentation, and Application. Each layer performs a specific function, and the model was designed to support international protocol standards, reduce complexity, and clearly separate concerns. Though the OSI protocols are rarely used today, the model remains influential in understanding network design.

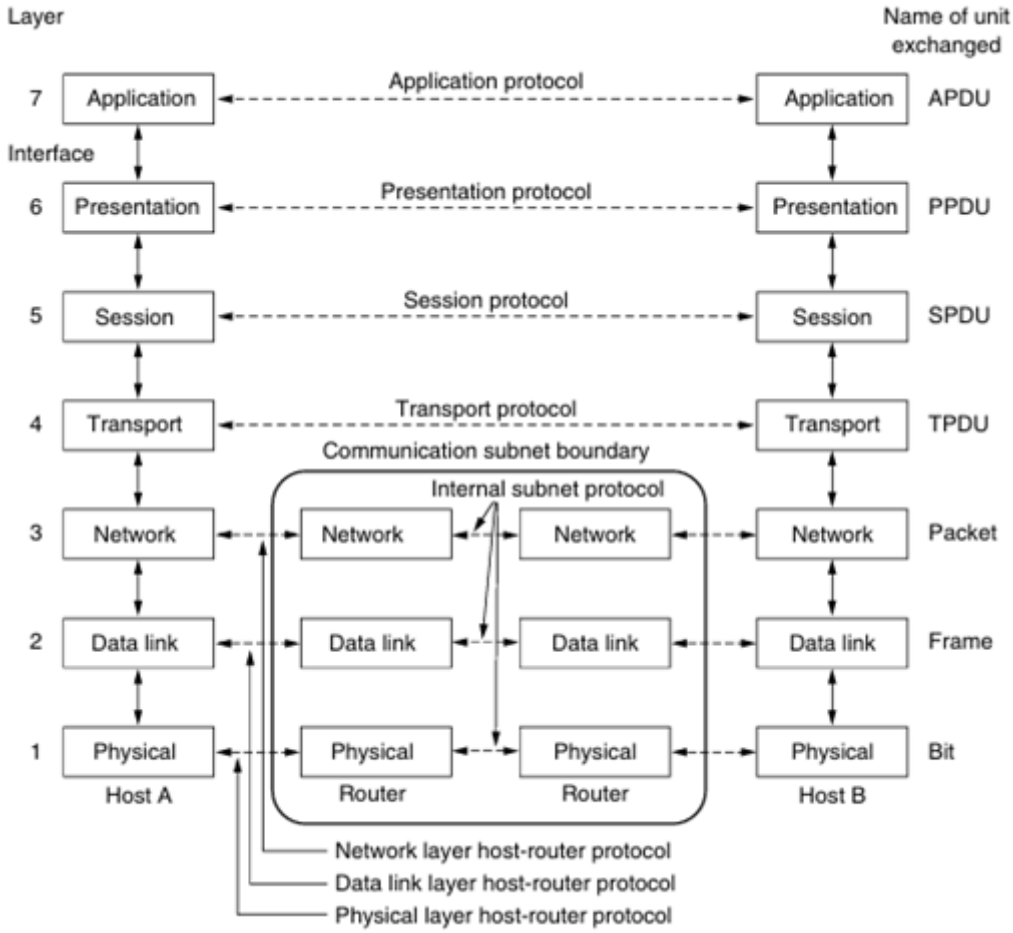


Figure 3.1 The OSI reference model

3.2.1 The Physical Layer

The physical layer is responsible for the actual transmission of raw bits over a communication medium. It ensures that a bit sent (like a 1) is received correctly as the same bit on the other end. Key concerns at this layer include the electrical or optical signal representation of bits, bit timing, direction of transmission (simplex, half-duplex, or full-duplex), and the setup and termination of physical connections. It also addresses the physical aspects such as connectors, pin configurations, and the characteristics of the transmission medium itself.

Protocols/Standards:
Ethernet (Physical specifications), RS-232, DSL, USB, Bluetooth, IEEE 802.11 (physical aspects).

Unit of Data:
Bit – the smallest unit of data representing binary 0s and 1s.

3.2.2 The Data Link Layer

The data link layer ensures reliable communication over a physical link by detecting and correcting transmission errors. It breaks data into frames, sends them sequentially, and often expects acknowledgments to confirm receipt. It also handles flow control to prevent a fast sender from overwhelming a slow receiver. In broadcast networks, it includes a medium access control (MAC) sublayer to regulate access to the shared communication medium, ensuring that devices transmit without collisions.

Protocols/Standards:

Ethernet, PPP (Point-to-Point Protocol), Frame Relay, HDLC, ARP, MAC.

Unit of Data:
Frame – contains source and destination MAC addresses and error-checking information

3.2.3 The Network Layer

The network layer manages the movement of packets across the communication subnet, determining how data is routed from source to destination. It handles routing—either statically, dynamically, or per conversation—and addresses issues like congestion control, ensuring efficient data flow and quality of service. When packets move between different networks, the network layer deals with incompatibilities in addressing, packet size, and protocols to ensure interoperability. In broadcast networks, where routing is simpler, the network layer may be minimal or omitted altogether.

Protocols/Standards:
IP (Internet Protocol), ICMP, IGMP, OSPF, BGP, RIP.
Unit of Data:
Packet – includes logical address information and routing details.

3.2.4 The Transport Layer

The transport layer is responsible for reliable, end-to-end delivery of data between source and destination processes. It breaks data from the upper layers into smaller units, passes them to the network layer, and ensures all pieces arrive correctly and in order. It provides different types of services, such as reliable, ordered delivery or fast, unordered delivery, based on application needs. Unlike the lower layers that manage hop-by-hop communication, the transport layer operates across the entire network path, maintaining a connection directly between the communicating applications on different machines.

Protocols/Standards:
TCP (Transmission Control Protocol), UDP (User Datagram Protocol), SCTP.

Unit of Data:
Segment / TPDU (Transport Protocol Data Unit) – contains port numbers and sequencing information for end-to-end communication.

3.2.5 The Session Layer

The session layer enables users on different machines to establish and manage sessions, which are prolonged interactions between applications. It provides services like dialog control (to coordinate communication direction), token management (to avoid simultaneous access to critical resources), and synchronization (to insert checkpoints in long data transfers so that progress can be saved and resumed after a failure).

Protocols/Standards:
NetBIOS, RPC (Remote Procedure Call), SQL session protocols.
Unit of Data:
SPDU (Session Protocol Data Unit) – used to maintain session control information.

3.2.6 The Presentation Layer

The presentation layer focuses on the syntax and semantics of the data exchanged between systems, rather than just its movement. It ensures that data sent by one system is readable and properly interpreted by another, even if they use different internal data formats. This is done by defining abstract data structures and standard encodings, allowing consistent communication—for example, in sharing complex structures like banking records.

Protocols/Standards:
SSL/TLS, MIME, JPEG, MPEG, ASCII, EBCDIC.

Unit of Data:
PPDU (Presentation Protocol Data Unit) – represents formatted or transformed data for application use.

3.2.7 The Application Layer

The application layer is the topmost layer and provides protocols directly used by end users for network services. It includes widely used protocols like HTTP for web browsing, where a

browser requests web pages from a server. Other important application layer protocols support services like file transfer, email, and network news, enabling user-level communication over the network.

Protocols/Standards:
HTTP, HTTPS, FTP, SMTP, DNS, SNMP, Telnet.

Unit of Data:
APDU (Application Protocol Data Unit) – represents the actual user data exchanged between applications.

3.3 THE TCP/IP REFERENCE MODEL

This passage introduces the TCP/IP Reference Model, developed from the ARPANET, the first major wide-area computer network funded by the U.S. Department of Defense. The model was created to solve interconnection problems among different kinds of networks (like satellite, radio, and wired).

Key design goals of the TCP/IP model were:

- Robustness: Keep communications alive even if parts of the network are damaged (e.g., during war).
- Inter-networking: Seamlessly connect multiple, diverse networks.
- Flexibility: Support different applications—from file transfers to real-time communication (like voice).

The model is named after its two main protocols: TCP (Transmission Control Protocol) and IP (Internet Protocol).

3.3.1The Link Layer

The TCP/IP model uses a packet-switching approach with a connectionless internet layer that can work across different types of networks. The lowest part, called the link layer, defines how physical connections like Ethernet or serial lines interact with the internet layer. Unlike traditional layers, the link layer acts more like an interface between the host and the physical transmission medium. Early descriptions of TCP/IP didn't focus much on this layer.

Protocols/Standards:
Ethernet, ARP (Address Resolution Protocol), PPP, Frame Relay, Token Ring, Wi-Fi (IEEE 802.11).

Unit of Data:
Frame / Bit – data transmitted over the physical medium.

3.3.2 The Internet Layer

The internet layer in the TCP/IP model is crucial because it connects different networks and makes sure packets can travel independently from the source to the destination, even if they arrive out of order. It roughly corresponds to the OSI network layer.

Think of it like sending letters through the mail: you drop a batch of letters in one country's mailbox, and they get delivered internationally, passing through different postal systems without you worrying about their details.

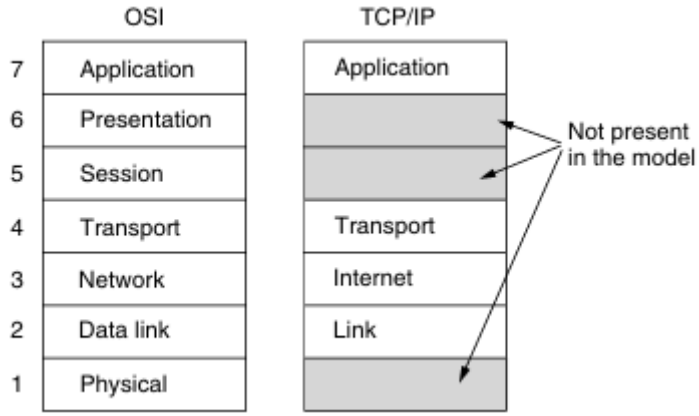


Figure 3.2 The TCP/IP Reference Model

This layer defines the Internet Protocol (IP) for addressing and routing packets, and also uses ICMP to help manage and troubleshoot the network. Its main job is to get packets to their destinations, though managing congestion isn't its strong suit.

Protocols/Standards:
IP (Internet Protocol – IPv4/IPv6), ICMP (Internet Control Message Protocol), ARP, RARP, IGMP.

Unit of Data:
Packet – includes source and destination IP addresses.

3.3.3. The Transport Layer

The TCP/IP transport layer enables direct communication between programs on source and destination hosts. It uses two main protocols: TCP, which provides reliable, connection-oriented delivery by breaking data into segments, ensuring they arrive correctly and in order, and managing flow control; and UDP, a simpler, connectionless protocol that offers faster but less reliable delivery, suitable for applications like streaming or quick queries where speed is prioritized over accuracy. Together, these protocols work on top of the internet layer to support different types of data transmission needs.

Protocols/Standards:
TCP (Transmission Control Protocol), UDP (User Datagram Protocol).

Unit of Data:
Segment (TCP) or **Datagram (UDP)** – includes port numbers and sequence information.

3.3.4 The Application Layer

The TCP/IP model doesn't include separate session or presentation layers because these functions are handled within the applications themselves, which has proven effective in practice.

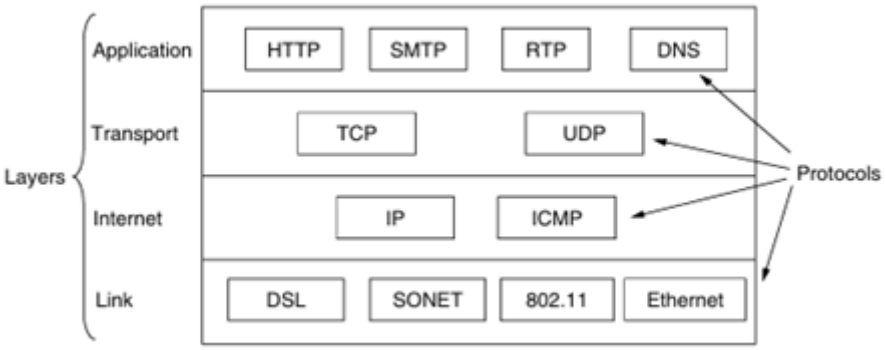


Figure 3.3 The TCP/IP model with some protocols we will study

Above the transport layer sits the application layer, which contains all the high-level protocols used by applications. Early protocols included TELNET for virtual terminals, FTP for file transfer, and SMTP for email. Over time, many more were added, such as DNS for translating domain names to IP addresses, HTTP for web pages, and RTP for real-time media like voice and video.

Protocols/Standards:
HTTP, HTTPS, FTP, SMTP, DNS, SNMP, Telnet, SSH.

Unit of Data:
Message / Data – actual application-level information exchanged between programs

3.4 THE COMPARISION OF OSI AND TCP/IP REFERENCE MODEL

The OSI and TCP/IP reference models share a common structure as layered protocol stacks, with similar functionality up to the transport layer, which provides an end-to-end communication service. Both models separate lower layers focused on network transport from upper layers that use this service for applications. However, they differ significantly in philosophy and design. The OSI model clearly distinguishes three key concepts—services (what a layer does), interfaces (how layers are accessed), and protocols (how layers internally operate)—making it very modular and adaptable. This aligns well with modern object-oriented ideas, allowing protocols within a layer to change without affecting others.

In contrast, TCP/IP did not originally separate these concepts clearly; its model was created to describe existing protocols rather than to define an abstract framework, making it tightly coupled to TCP/IP protocols and less flexible for other networks. The OSI model has seven layers, while TCP/IP has four, and they differ in support for connection-oriented and connectionless communication: OSI supports both modes in the network layer and only connection-oriented transport, whereas TCP/IP uses only connectionless networking but

offers both connectionless and connection-oriented options at the transport layer, catering to different application needs.

Table 3.1 Comparison between TCP/IP and OSI Reference Models

Feature / Basis of Comparison	**OSI Reference Model**	**TCP/IP Reference Model**
Developed By	ISO (International Organization for Standardization)	DoD (U.S. Department of Defense)
Year of Development	1978	1974
Number of Layers	7 Layers	4 Layers
Layer Names (Top to Bottom)	Application, Presentation, Session, Transport, Network, Data Link, Physical	Application, Transport, Internet, Network Interface (Link)
Concept	Theoretical model defining ideal communication framework	Practical model for real-world Internet communication
Primary Function	Standardizes communication functions between systems	Enables internetworking and Internet-based communication
Approach	Protocol-independent, general-purpose reference	Protocol-specific, focused on TCP/IP suite
Layered Structure	Distinct and detailed layers with clear separation	Layers are broader and combined (some OSI layers merged)
Reliability	Conceptual model; not directly implemented	Implemented model used in real networks
Model Type	Generic model – suitable for all network types	Practical model – specific to Internet communication
Error Handling and Flow Control	Done at Data Link and Transport Layers	Done at Transport Layer (TCP)
Examples of Protocols	HTTP, FTP, SMTP, TCP, IP, Ethernet	HTTP, FTP, TCP, IP, UDP, ARP
Data Unit Names (Top to Bottom)	APDU, PPDU, SPDU, TPDU, Packet, Frame, Bit	Data, Segment/Datagram, Packet, Frame/Bit
Dependency	Model is protocol-independent	Model is protocol-dependent (based on TCP/IP suite)
Usage	Used as a reference framework for understanding networking	Used in actual implementation of the Internet
Flexibility and Speed	More conceptual and layered (less efficient for practical use)	More streamlined and efficient for implementation
Example of Application	Used in education, standardization, and design	Used in all Internet-based systems today

3.5 A CRITIQUE OF OSI REFERENCE MODEL AND PROTOCOL

Here's a brief summary of the criticisms of the OSI model and protocols:
The OSI model and its protocols faced several major challenges that prevented them from becoming the dominant networking standard.

First, **bad timing** played a key role: the OSI standards appeared after TCP/IP had already gained significant traction in universities and early networks, so vendors were reluctant to support OSI, leading to a deadlock where no one adopted it first.

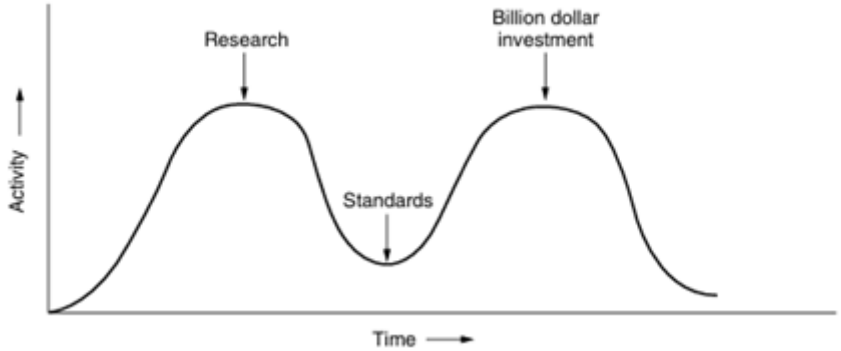


Figure 3.4 The apocalypse of the two elephants

Second, **bad technology** was a problem—OSI's seven-layer model was overly complex and included redundant features (like repeated error control across layers), making the protocols difficult to understand, implement, and inefficient to run.

Third, **bad implementations** hurt OSI's reputation early on; the initial OSI software was large, slow, and cumbersome, while TCP/IP implementations (like Berkeley UNIX's) were smaller, free, and effective, leading to rapid adoption and community improvement.

Finally, **bad politics** also played a part: OSI was associated with bureaucratic government bodies and viewed as a forced, top-down standard, whereas TCP/IP was linked to the open, academic UNIX culture, giving TCP/IP a more grassroots and trusted image. Together, these factors led to OSI's limited success despite its conceptual strengths.

3.6 A CRITIQUE OF TCP/IP REFERENCE MODEL

The TCP/IP model has several drawbacks. First, it does not clearly separate services, interfaces, and protocols, unlike the OSI model, which makes it less useful as a guide for designing new networks or technologies. Second, TCP/IP is not very general and struggles to describe protocols outside its own stack, like Bluetooth. Third, the so-called link layer in TCP/IP isn't really a true layer but more of an interface, which blurs important distinctions. Fourth, TCP/IP does not separate the physical and data link layers, though these have distinct roles in transmission and framing. Finally, while core protocols like IP and TCP were well-designed, many other protocols were developed in an ad hoc manner by students and became entrenched due to wide, free distribution—even when outdated. For example, TELNET was built for slow, text-only terminals but remains in use long after graphical interfaces became common.

3.7 SUMMARY

This chapter focuses on the reference models that define how computer networks operate. The OSI model divides network communication into seven layers—from physical transmission to application services—providing a standardized framework. The TCP/IP model, used by the Internet, has fewer layers and emphasizes practical implementation over

theoretical design. A comparison highlights that OSI is more conceptual, while TCP/IP is more implementation-oriented. The chapter also critiques both models, noting OSI's limited adoption and TCP/IP's lack of clear layer separation, concluding with the Internet as the best real-world example of the TCP/IP model in action

3.8 TECHNICAL TERMS

Transmission Control Protocol, Internet Protocol, TCP/IP, OSI, Layer

3.9 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the OSI reference model and describe the function of each layer.
2. Describe the TCP/IP reference model and its key layers.
3. Compare the OSI and TCP/IP models in terms of structure and usage.
4. Write a critique of the OSI and TCP/IP reference models.
5. Explain the role of the Internet as an example network based on the TCP/IP model.

Short Questions:

1. What is a reference model in networking?
2. Name the seven layers of the OSI model.
3. How many layers are there in the TCP/IP model?
4. Mention one major difference between OSI and TCP/IP models.
5. What is the main purpose of the Internet in networking?

3.10 FURTHER READINGS

1. Andrew S. Tanenbaum, "Computer Networks", Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, "Computer Networking", Third Edition, Pearson Education
3. Behrouz A Forouzan, "Data Communications and Networking", Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, "Computer Communications and Networking Technologies", Cengage Learning (2008)

Dr. Kampa Lavanya

LESSON- 4

EXAMPLE NETWORKS

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the concept of connection-oriented networks.
- Learn about X.25, Frame Relay, and ATM technologies.
- Study the working of Ethernet in local area networks.
- Recognize the features and importance of wireless LANs.
- Compare wired and wireless connection-oriented communication

STRUCTURE OF THE LESSON:

This section introduces the diverse world of computer networking, highlighting that networks vary widely in goals, size, and technology. It outlines the plan to explore different types of networks: starting with the well-known Internet, covering its history and technology; then the mobile phone network, which is quite different technically from the Internet; followed by IEEE 802.11, the main wireless LAN standard; and finally RFID and sensor networks, which extend networking to everyday physical objects.

4.1 THE INTERNET

The Internet is actually a huge collection of different networks that share common protocols and services, rather than a single network. What makes it unique is that it wasn't planned or controlled by any one person or organization. To understand it better, it helps to look at its history and how it evolved over time. For a detailed and enjoyable history, John Naughton's 2000 book is highly recommended. There are also many technical books about the Internet and its protocols, such as Maufer (1999), for those seeking deeper knowledge.

THE ARPANET

The Internet began as a project by the U.S. Department of Defense during the Cold War to create a communication network that could survive a nuclear attack. The existing telephone system was vulnerable because it relied on a centralized hierarchy of switching offices.

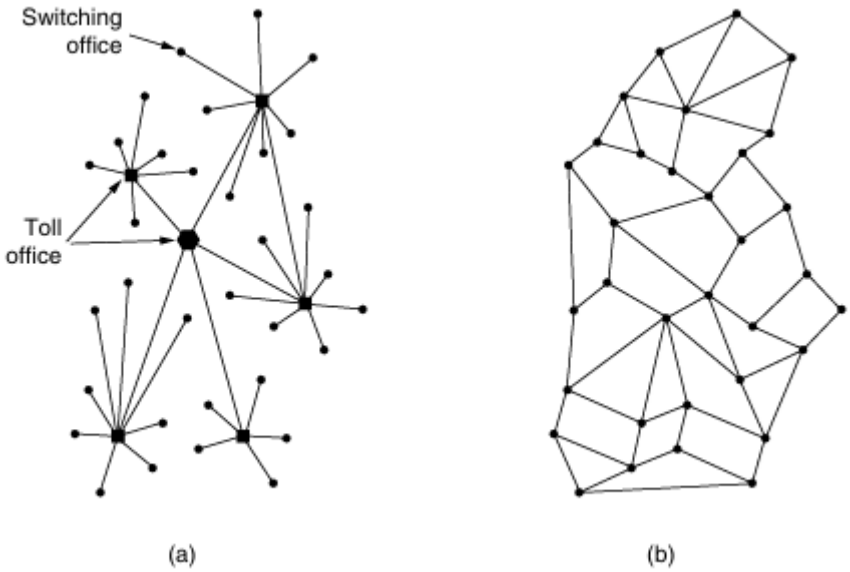


Figure 4.1 (a) Structure of the telephone system (b) Baran's proposed distributed switching system

Paul Baran proposed a distributed, fault-tolerant network using digital packet switching, but his ideas were initially rejected by AT&T.In response to Sputnik and inter-service conflicts, the government created ARPA, which funded research into networking.

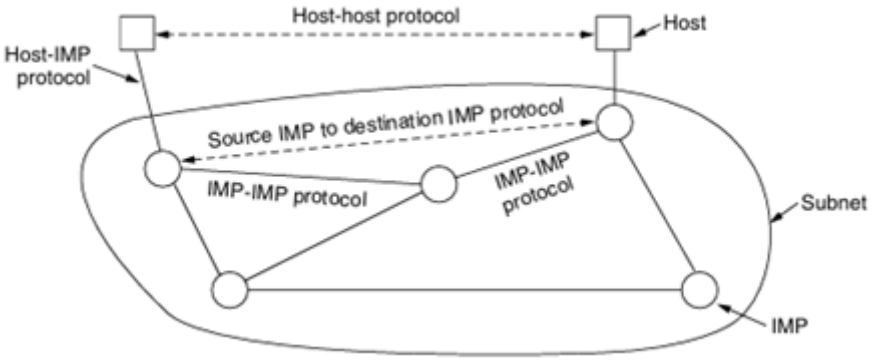


Figure 4.2 The original ARPANET design

Larry Roberts at ARPA developed the concept of the ARPANET, a packet-switched network connecting different computers through Interface Message Processors (IMPs). The first ARPANET nodes went live in 1969, quickly expanding across the U.S. Experiments with satellite and mobile networks highlighted the need for protocols that could connect different types of networks.

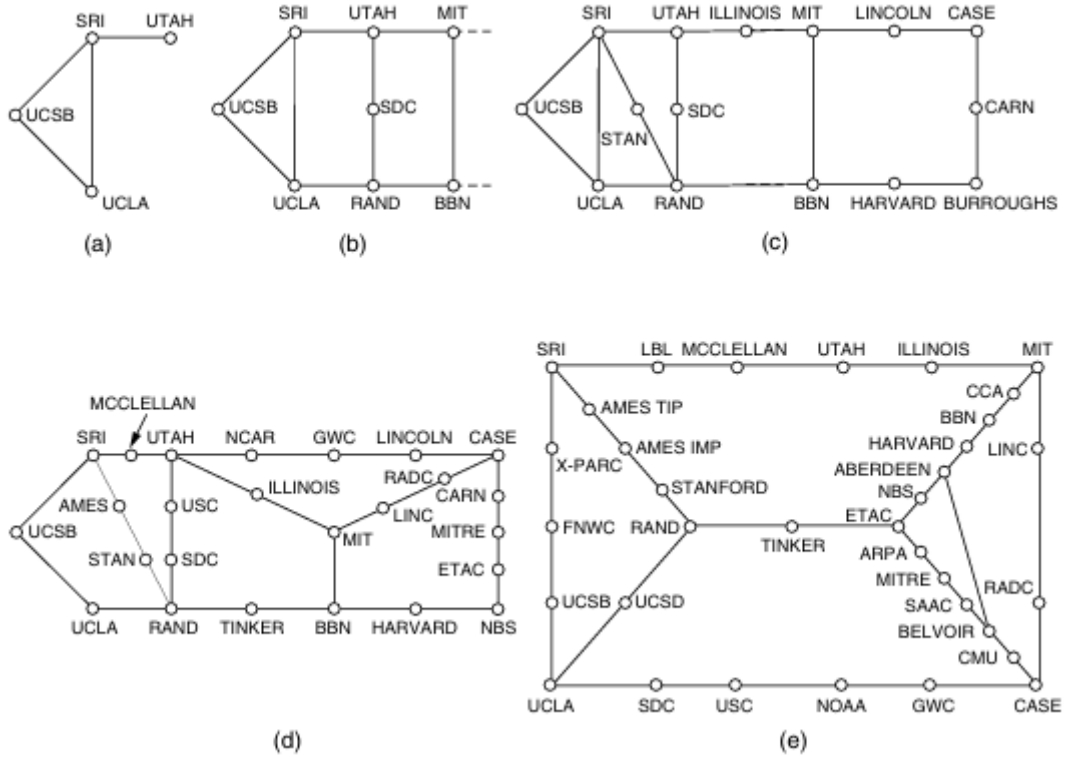


Figure 4.3 Growth of the ARPANET (a) December 1969 (b) July 1970 (c) March 1971 (d) April 1972 (e) September 1972

This led to the creation of TCP/IP in the 1970s, which was widely adopted with the help of Berkeley UNIX. As the network grew, the Domain Name System (DNS) was introduced in the 1980s to simplify finding hosts by translating domain names into IP addresses. The Internet evolved from this foundation into the vast, interconnected network we use today.

NSFNET

By the late 1970s, the National Science Foundation (NSF) recognized the ARPANET's significant impact on university research but noted that access was limited to institutions with Department of Defense contracts. To widen access, NSF funded the Computer Science Network (CSNET) in 1981, linking computer science departments and research labs to ARPANET. Later, NSF created NSFNET, a TCP/IP-based backbone network connecting six supercomputer centers via leased lines and regional networks, enabling thousands of institutions to communicate.

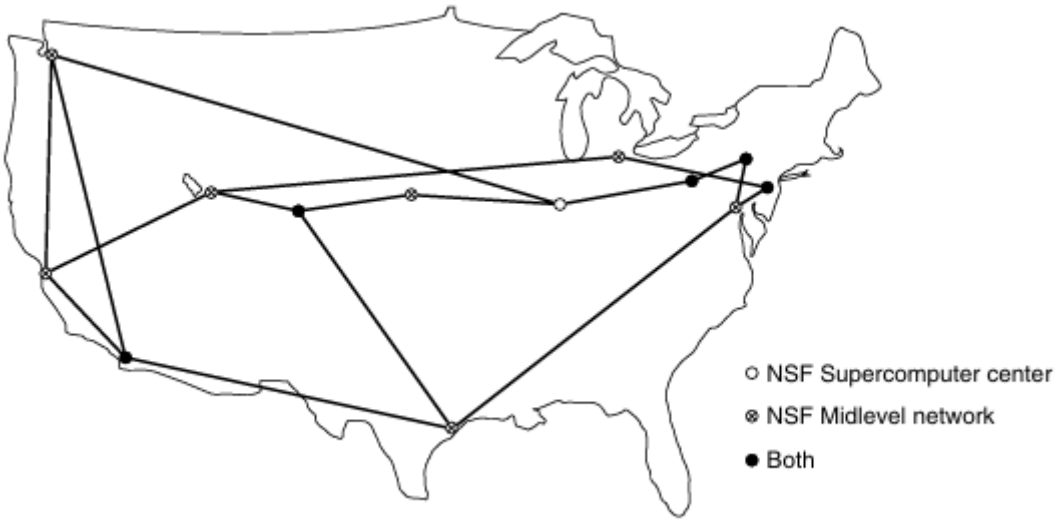


Figure 4.4 The NSFNET backbone in 1988

NSFNET quickly became overloaded, leading to upgrades and eventual commercialization. To transition from government to commercial operation, Network Access Points (NAPs) were established to foster competition among backbone providers. Meanwhile, similar national research networks emerged globally. The Internet then grew rapidly in the 1990s with the rise of the World Wide Web, expanding from academic email and file sharing to include rich media, social networks, and real-time communications, continually changing the dominant types of Intenet.

The Architecture of the Internet

The Internet connects users through ISPs using technologies like DSL, cable, fiber, or mobile networks. Data travels from the user's device to the ISP's network and, if needed, to other networks via Internet Exchange Points (IXPs), where ISPs share traffic.

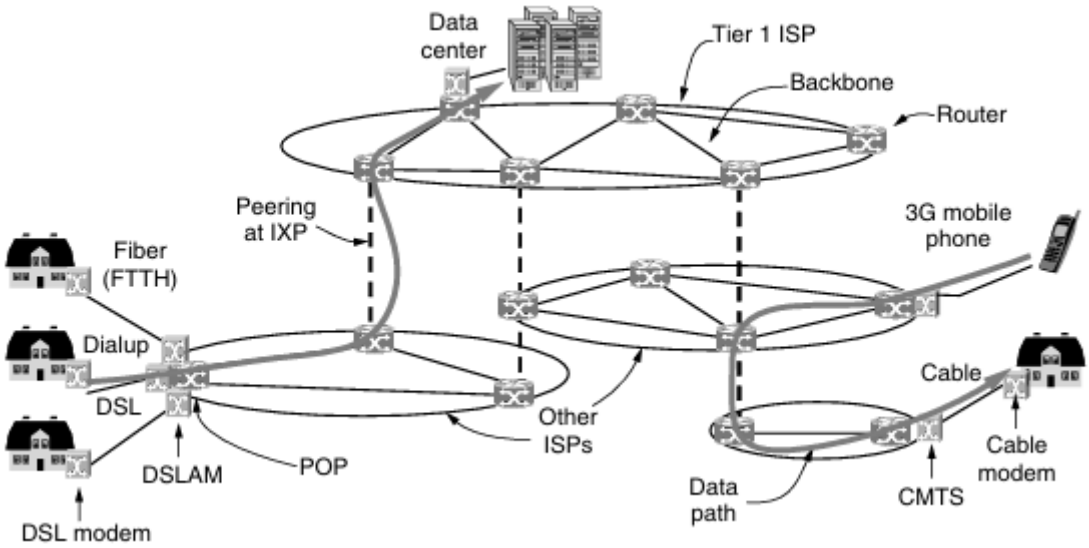


Figure 4.5 Overview of the Internet architecture

Large Tier 1 ISPs form the Internet's backbone. Companies like Google host services in powerful data centers for faster access. Over time, virtualization and shared IP addresses have

changed how we define being "on the Internet." Many businesses also use private internal networks called intranets that function like the Internet but are only for internal use.

4.2 THIRD-GENERATION MOBILE PHONE NETWORKS

Mobile phone networks have evolved significantly over time, starting from 1G analog systems to 2G digital systems like GSM,

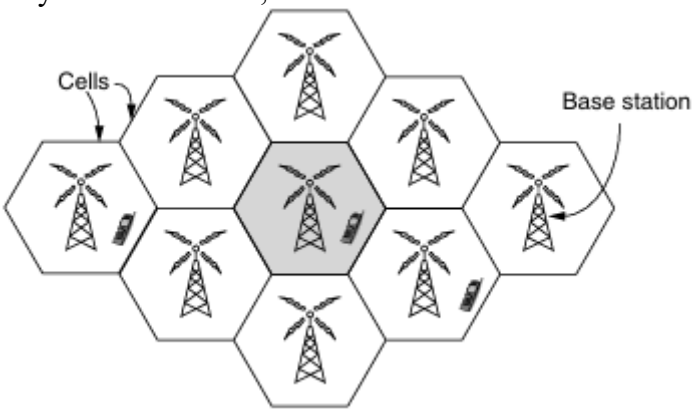


Figure 4.6 Cellular design of mobile phone networks

and then to 3G systems like UMTS that support both voice and high-speed data. These networks use radio spectrum, a limited resource, which is managed by dividing coverage into cells for efficient frequency reuse.

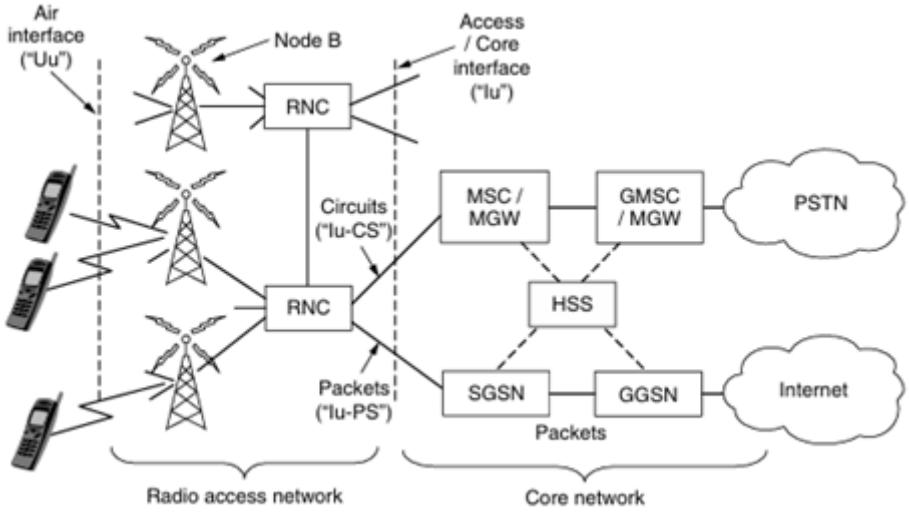


Figure 4.7 Architecture of the UMTS 3G mobile phone network

The architecture includes a radio access network (base stations and controllers) and a core network that handles call/data routing. Initially, voice used circuit-switched systems, while data used packet-switched systems, but newer networks increasingly use IP-based packet switching for everything

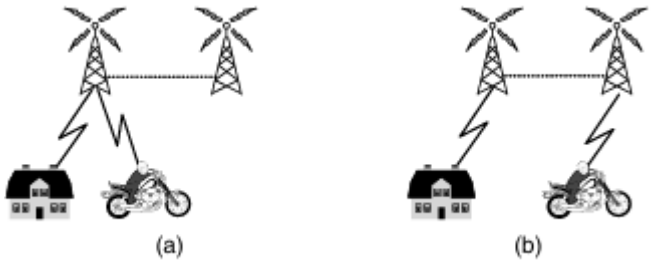


Figure 4.8 Mobile phone handover (a) before (b) after

Features like handover support mobility, while SIM cards ensure user authentication and security. Mobile networks are now more focused on data services, driving the development of faster 4G (LTE) and competing technologies like WiMAX.

4.3 WIRELESS LANs: 802.11

The development of wireless LANs began with the desire for laptops to connect to the Internet without wires, leading to the use of short-range radio communication.

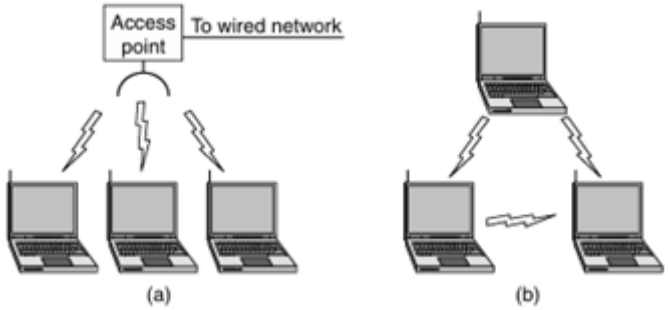


Figure 4.9 (a) Wireless network with an access point (b) Ad hoc network

Initially, different vendors used incompatible systems, prompting the IEEE to standardize wireless LANs as 802.11, commonly known as WiFi. 802.11 operates in unlicensed ISM bands and consists of clients (like laptops) and access points that connect to a wired network.

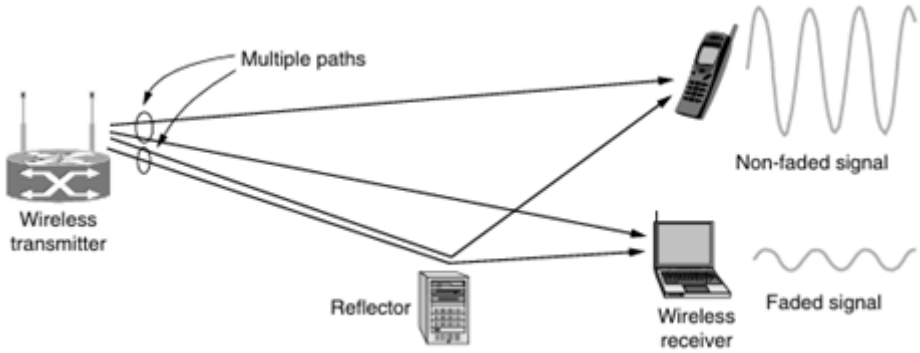


Figure 4.10 Multipath fading

It can also function in ad hoc mode, where clients communicate directly. Wireless signals face challenges like multipath fading, which are managed using techniques such as OFDM and multiple antennas. Over time, newer versions like 802.11b, a/g, and n improved data rates significantly.

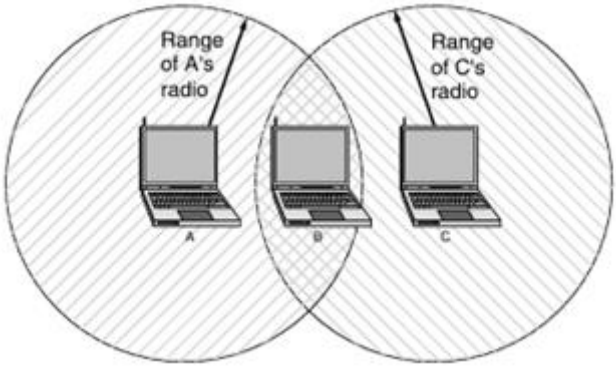


Figure 4.11 The range of a single radio may not cover the entire system

802.11 uses a CSMA protocol to avoid transmission collisions, but hidden terminal problems can still cause issues. For mobility, devices can switch between access points within a network. Security began with WEP but later moved to WPA and WPA2 due to vulnerabilities. Today, 802.11 is widely used in various settings, making wireless connectivity nearly ubiquitous.

4.4 RFID AND SENSOR NETWORKS

RFID (Radio Frequency Identification) extends computer networking to everyday objects using small tags with unique identifiers and antennas. These tags, often passive (without batteries), are powered by radio waves from RFID readers that query them when in range. RFID is used in supply chains, passports, credit cards, and more.

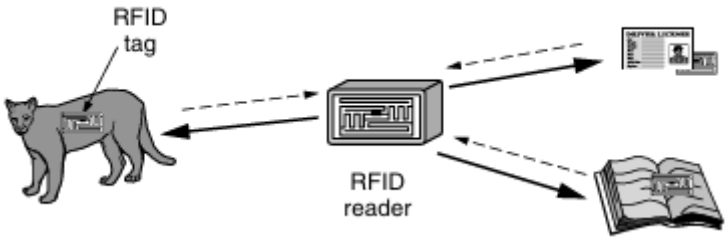


Figure 4.12 RFID used to network everyday objects

UHF RFID (902–928 MHz) uses backscatter for longer ranges, while HF RFID (13.56 MHz) uses induction for short-range applications. A challenge with RFID is managing multiple tags at once, which is handled using randomized response times, similar to 802.11 networks. However, RFID poses privacy concerns due to weak security and easy tracking.

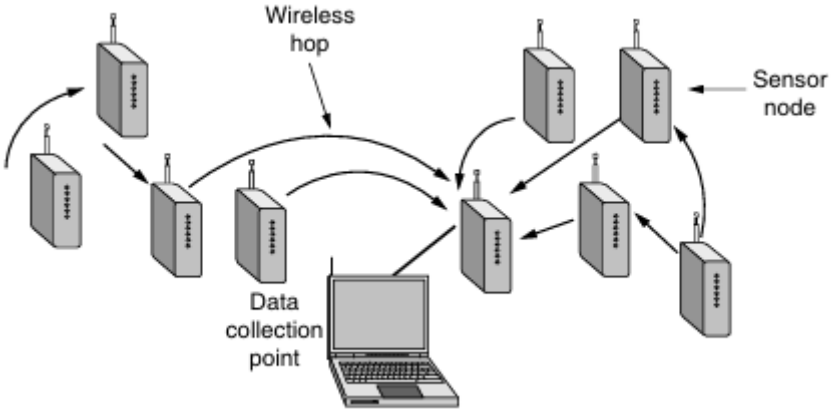


Figure 4.13 Multidrop topology of a sensor network

Meanwhile, sensor networks, composed of small, battery-powered devices with environmental sensors, self-organize into multihop networks to monitor physical conditions like temperature and vibration. These networks are used in science and industry, and the convergence of RFID and sensors suggests a future with even more integrated and intelligent monitoring systems.

4.5 SUMMARY

This chapter discusses connection-oriented networks, where a dedicated path is established before data transfer. It explains X.25, an early reliable packet-switched network; Frame Relay, a faster version optimized for digital connections; and ATM (Asynchronous Transfer Mode), which supports high-speed data, voice, and video transmission using fixed-size cells. The chapter also covers Ethernet, the most common LAN technology that provides efficient wired communication, and Wireless LANs (Wi-Fi), which offer mobility and flexibility without physical cables. Together, these technologies form the backbone of both wired and wireless communication networks

4.6 TECHNICAL TERMS

Connection oriented, X.25, ATM, Wireless LAN , Ethernet

SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the architecture and operation of the X.25 network.
2. Describe Frame Relay and how it improves over X.24.
3. Discuss the main features and advantages of ATM technology.
4. Explain the working of Ethernet and its importance in LAN communication.
5. Describe the structure, benefits, and applications of Wireless LANs.

Short Questions:

1. What is a connection-oriented network?
2. Name the three main connection-oriented technologies.
3. What is the main purpose of X.25?
4. Define Ethernet and its basic function.
5. What is a Wireless LAN?

FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008)

Dr. Kampa Lavanya

LESSON- 5

NETWORK STANDARIZATION

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the need for network standardization.
- Identify key organizations involved in telecommunication standards.
- Recognize international and Internet standardization bodies.
- recognize the organizations involved in defining Internet standards

STRUCTURE OF THE LESSON:

5.1 NETWORK STANDARDIZATION

In the vast and rapidly evolving world of computer networks, multiple vendors and developers often design systems and equipment based on their unique technologies and proprietary approaches. Without a unifying framework, this diversity can lead to chaos, incompatibility, and inefficiency, as devices from different manufacturers might fail to communicate effectively. To overcome this challenge, network standards play a critical role. These standards define the rules, formats, and protocols necessary for devices and systems to work together seamlessly, regardless of their origin.

By ensuring interoperability, standards enable users to mix and match components—routers, switches, cables, and software—from various vendors without worrying about compatibility issues. Moreover, standardized protocols foster healthy competition and market expansion, as manufacturers can produce devices at scale for a global market. This not only reduces production costs but also enhances performance, reliability, and user satisfaction. For instance, Ethernet and Wi-Fi became dominant in networking largely due to the widespread adoption of common standards, allowing users to connect devices effortlessly.

5.1.1. The Scope and Flexibility of Standards

Contrary to popular belief, standards do not specify every detail of how systems must be implemented. Instead, they focus only on what is necessary to ensure interoperability

between different systems. This design philosophy preserves flexibility and innovation, allowing companies to develop products with unique features or performance optimizations while still maintaining compatibility.

A good example of this is the IEEE 802.11 Wi-Fi standard, which defines multiple data transmission rates such as 11 Mbps, 54 Mbps, and beyond. The standard specifies that devices must support these rates but does not dictate the specific algorithm or conditions under which a device should switch between them. This freedom lets product designers optimize for factors like signal strength, power efficiency, or user experience. However, this flexibility sometimes leads to inconsistencies in how different products behave. To address such issues, organizations like the Wi-Fi Alliance certify devices to ensure that even with diverse implementations, they remain interoperable and meet quality expectations. Thus, while standards set the foundation, industry alliances ensure uniformity in practice.

5.1.2 Protocol Standards and Interface Design

Network protocol standards primarily define how data is transmitted and received between systems rather than how internal software or hardware components should be designed. For instance, the TCP/IP protocol suite focuses on reliable data communication over networks—specifying packet formats, addressing schemes, and error-handling mechanisms. However, it does not dictate how these functions should be implemented inside an operating system or networking device.

This approach allows different systems—Windows, Linux, macOS, or embedded routers—to communicate effortlessly, even though their internal architectures may differ significantly. Over time, some internal interfaces, although not officially standardized, become de facto industry standards due to their practicality and widespread use. An example is the Berkeley Sockets API, developed at the University of California, Berkeley. It provides a consistent programming interface for network communication, simplifying application development and becoming an essential tool for programmers across platforms. Thus, clear separation between protocol standards and implementation design encourages both interoperability and innovation.

5.1.3 Types of Standards: De Facto and De Jure

Standards can be broadly categorized into two types—*de facto* and *de jure.*

De facto standards emerge naturally through widespread use rather than formal approval. These standards often become popular because they solve real-world problems effectively or are backed by influential companies. A classic example is HTTP (Hypertext Transfer Protocol), the foundation of the World Wide Web. Initially developed by Tim Berners-Lee and his team at CERN, it gained rapid acceptance through web browsers like Mosaic and Netscape, eventually becoming the universal protocol for web communication. Similarly, Bluetooth began as a proprietary technology developed by Ericsson but gained global acceptance due to its convenience and support from major device manufacturers.

De jure standards, on the other hand, are officially approved and ratified by recognized standardization organizations. These may include governmental treaty-based bodies such as the International Telecommunication Union (ITU) or voluntary international organizations like the International Organization for Standardization (ISO), Internet Engineering Task

Force (IETF), and Institute of Electrical and Electronics Engineers (IEEE). These bodies establish and maintain formal standards through a rigorous review and consensus process, ensuring global reliability and consistency across technologies.

5.1.4 Collaboration and Evolution in Standardization

The relationship between companies, technologies, and standardization organizations is intricate and dynamic. Often, a successful de facto standard—developed and proven in practice—is later adopted and formalized by a standards body, ensuring its long-term stability and broader acceptance. For instance, many Internet protocols, originally developed by research groups and adopted widely by the community, were later standardized by the IETF.

In modern times, the process of developing standards has become highly collaborative and business-driven. Industry alliances and consortia are formed to address specific technological areas and to speed up the standardization process. A prime example is the 3rd Generation Partnership Project (3GPP), which unites several telecommunication standardization bodies across the world to create unified mobile communication standards, including 3G, 4G LTE, and 5G. This cooperative model ensures that emerging technologies evolve in harmony with both industry requirements and user expectations. As a result, network standards today not only ensure interoperability but also serve as catalysts for innovation and global connectivity.

5.2 WHO'S WHO IN THE TELECOMMUNICATION WORLD

5.2.1 Diversity in Global Telecommunication Structures

The legal and organizational structure of telephone services varies significantly across countries, shaped by historical, political, and economic influences. In some nations, telecommunications evolved as a public utility controlled by the government, while others embraced a marketdriven, privatized model.

In the United States, the telecommunication system developed as a highly decentralized network. Over 2,000 independent telephone companies operate across the country, most of which are privately owned and serve specific localities. This fragmentation can be traced back to America's early emphasis on private enterprise and competition. A defining moment in U.S. telecommunications history came in 1984, when the AT&T monopoly, which once controlled nearly 80% of all U.S. telephone services, was dismantled under antitrust law. The breakup resulted in the creation of several regional "Baby Bell" companies and fostered innovation by allowing new entrants into the market.

Further restructuring occurred with the Telecommunications Act of 1996, which was the first major overhaul of U.S. telecommunications law in over 60 years. This act promoted open competition, reduced regulatory restrictions, and encouraged convergence between telephone, cable, and Internet services. Together, these reforms transformed the American telecommunications landscape into one of the most dynamic and competitive in the world.

5.2.2. Government Controlled Systems and the PTT Model

In contrast, many other countries historically maintained stateowned monopolies over their communication systems. Governments often combined postal services, telegraphy, telephony,

and later, broadcasting under a single national authority known as the PTT (Post, Telegraph, and Telephone Administration).

Under the PTT model, the government was responsible for every aspect of telecommunications —from infrastructure construction and maintenance to service delivery and pricing. While this centralized system ensured nationwide connectivity and uniform service quality, it often limited competition and slowed technological innovation due to bureaucratic constraints.

In recent decades, however, global trends have shifted toward liberalization and privatization. Many countries in Europe and Asia have moved away from the traditional PTT model. For example, British Telecom (BT) in the United Kingdom and Deutsche Telekom in Germany were once governmentrun entities but have since been privatized. Even so, some nations, especially in the developing world, continue to rely on staterun systems as they gradually transition toward a mixed or fully privatized telecommunications framework.

5.2.3 The Need for International Compatibility

The growing number of telecommunication providers and the diversification of national policies have made international compatibility an essential concern. Individuals, businesses, and governments all require seamless communication across borders, whether through voice calls, data exchange, or Internetbased applications.

This necessity for interoperability is not a modern phenomenon. As early as 1865, European nations recognized the importance of standardizing communication systems and founded a cooperative organization to harmonize telegraph networks. This body later evolved into the International Telecommunication Union (ITU), the world's oldest international organization still in existence. The ITU's mission expanded over time from telegraphy to include telephone, radio, television, and modern digital communications, setting the foundation for global telecommunication cooperation.

5.2.4. The Role and Membership of the International Telecommunication Union

The International Telecommunication Union (ITU) became a specialized agency of the United Nations in 1947. Today, it is one of the most influential organizations in global telecommunications, comprising over 200 member states. The ITU provides a neutral platform where countries collaborate to establish global standards, allocate radio frequencies, and promote equitable access to communication technologies.

In the United States, where no centralized national telecommunication body like a PTT exists, representation in the ITU is managed by the U.S. Department of State. The ITU's membership extends beyond governments—it also includes more than 700 privatesector companies and academic institutions as sector or associate members.

Prominent members include telecommunication giants such as AT&T, Vodafone, and Verizon, equipment manufacturers like Cisco, Nokia, and Ericsson, chipmakers such as Intel and Texas Instruments, and major corporations involved in digital communication technologies like Microsoft, Toshiba, Boeing, and CBS. This inclusive model ensures that the ITU reflects both public and private interests in shaping the global telecommunication environment.

5.2.5 Organizational Structure of the ITU

The ITU is divided into three main sectors, each responsible for a distinct area of telecommunications:

1. ITUT (Telecommunication Standardization Sector) – This sector formulates international standards (known as Recommendations) for telephony and data communication systems. Until 1993, ITUT was known as CCITT (Comité Consultatif International Téléphonique et Télégraphique).
2. ITUR (Radiocommunication Sector) – Responsible for managing the global radiofrequency spectrum and satellite orbits, which are essential for services like broadcasting, mobile communications, and GPS.
3. ITUD (Development Sector) – Focused on promoting telecommunications development in developing countries, bridging the digital divide, and ensuring universal access to modern ICT infrastructure.

This sectorbased structure enables the ITU to effectively address both technical and developmental aspects of telecommunications.

5.2.6 ITUT Recommendations and Global Adoption

The core responsibility of ITUT is to develop technical recommendations that act as global standards for telecommunication interfaces, signaling systems, and data protocols. Although these recommendations are not legally binding, they are widely adopted worldwide to ensure interoperability. A nation or company that ignores ITU standards risks isolation from the global communication network.
For example, if a country uses incompatible frequency bands or signaling methods, it could not easily connect to international systems. While such isolation might be acceptable in highly controlled environments (e.g., North Korea), it would cause severe economic and technological disadvantages for most countries. Therefore, adherence to ITUT recommendations is a practical necessity for global integration.

5.2.7. Study Groups and Technical Work within ITUT

The ITUT's standardization work is carried out by Study Groups (SGs)—expert committees focusing on specific telecommunication domains. Currently, there are 10 active Study Groups, each composed of specialists from governments, academia, and industry. Their areas of work include:

- Broadband and nextgeneration networks
- Multimedia services and video communication
- Cybersecurity and data protection
- Network architecture and infrastructure
- Billing, numbering, and service management

Each Study Group is divided into Working Parties, which are further subdivided into Expert Teams and temporary Ad Hoc Groups to address specific issues. This hierarchical framework ensures that complex technical problems are studied and resolved efficiently through collaboration among global experts.

5.2.8 Major Achievements and Standard Contributions

Despite its complex structure, ITUT has been highly productive. Since its inception, it has issued over 3,000 recommendations, many of which underpin the modern telecommunication landscape. Notable examples include:

Recommendation H.264 (MPEG4 AVC) – A video compression standard widely used in digital television, online streaming platforms like YouTube, and Bluray discs.
Recommendation X.509 – The foundation for digital certificates used in secure web browsing (HTTPS) and online authentication systems.

These standards have become integral to daily life, enabling secure communication, multimedia sharing, and efficient use of network resources worldwide.

5.2.9. The Future of Global Telecommunications Standardization

As the world transitions toward 5G, 6G, satellite Internet, and the Internet of Things (IoT), international coordination has become even more critical. The ITU, along with other bodies like 3GPP, IETF, and IEEE, plays a vital role in ensuring that emerging technologies are compatible across countries and ecosystems.
Moreover, modern standardization is increasingly collaborative and inclusive, involving not only governments and corporations but also research institutions and regional alliances. This collective approach ensures that the benefits of global communication technologies extend to all nations—advancing connectivity, fostering economic development, and narrowing the digital divide.

5.3 WHO'S WHO IN THE INTERNATIONAL STANDARDS WORLD

5.3.1 Decentralized Telecommunications in the United States

The legal and organizational setup of telephone services varies significantly across countries, reflecting differences in historical development, political systems, and economic philosophies. In the United States, the telecommunications sector is characterized by decentralization and private ownership. More than 2,000 independent and mostly small-scale private telephone companies operate across the nation, reflecting America's strong commitment to free enterprise and competition.

Historically, the U.S. telecommunications industry was dominated by AT&T (American Telephone and Telegraph Company), which had grown into one of the world's largest corporations, controlling approximately 80% of the nation's telephone services. However, this monopoly was dismantled in 1984, when the U.S. government intervened under antitrust law to promote competition and innovation. The breakup resulted in the creation of several regional "Baby Bell" companies, each responsible for local telephone services.

Further reforms arrived with the Telecommunications Act of 1996, the first major overhaul of U.S. communications law since 1934. This act sought to open markets to competition, reduce regulatory barriers, and encourage convergence between telephony, broadcasting, and Internet services. The result was a more dynamic and technologically diverse

telecommunications environment, paving the way for advancements such as broadband Internet and mobile communication.

5.3.2 Centralized and Government-Controlled Systems

In contrast to the U.S. model, many other nations historically maintained centralized, government-controlled systems. These systems were typically managed by a national organization known as the PTT (Post, Telegraph, and Telephone Administration), which operated all communication services — including postal operations, telegraphy, telephony, and, in many cases, broadcasting.

Under the PTT framework, the government was responsible for every aspect of communication: constructing infrastructure, providing services, setting tariffs, and managing customer relations. This model ensured uniform service coverage and strategic national control, especially important during the early and mid-20th century when communications were viewed as essential public utilities.

However, as technology advanced and globalization increased, many nations realized that government monopolies could not adapt quickly to changing technological and market conditions. Consequently, the last few decades have witnessed widespread liberalization and privatization. For instance, the United Kingdom transformed its British Post Office Telecommunications into British Telecom (BT), a privatized entity, while France Télécom and Deutsche Telekom underwent similar reforms. Despite these trends, the pace of transformation has not been uniform—some countries continue to retain partial or full governmental control while cautiously moving toward privatization.

5.3.3 The Need for Global Compatibility and Standardization

With so many diverse national systems and regulatory frameworks, international compatibility became a critical necessity. Seamless cross-border communication requires common technical standards for hardware, signaling, and protocols, ensuring that systems developed in one country can interconnect with those in another.

This need for harmonization was recognized as early as 1865, when representatives from several European nations met to standardize telegraph operations. Their efforts led to the formation of a cooperative body that would later evolve into the International Telecommunication Union (ITU). Over time, the ITU expanded its scope beyond telegraphy to encompass telephone, radio, television, and eventually digital communications.

In 1947, the ITU officially became a specialized agency of the United Nations, dedicated to coordinating international standards, allocating global radio frequencies, and promoting equitable access to communication technologies. Today, the ITU represents over 200 member states, virtually encompassing every nation, and collaborates with more than 700 associate and sector members from the private sector. These include global industry leaders such as AT&T, Vodafone, Cisco, Intel, Microsoft, Nokia, and Boeing, as well as media organizations like CBS. The diverse membership ensures that the ITU’s standards reflect both governmental priorities and private-sector innovation, fostering interoperability across the global communication ecosystem.

5.3.4 Structure and Functions of the ITU

The ITU's work is divided into three main sectors, each focusing on a distinct domain of telecommunications:

1. ITU-T (Telecommunication Standardization Sector):

Formerly known as CCITT (Comité Consultatif International Téléphonique et Télégraphique), this sector is responsible for developing international standards and technical recommendations for telephone and data communication systems. Its work ensures interoperability across equipment and networks produced by different vendors and used in different nations.

2. ITU-R (Radiocommunication Sector):

This sector manages the global radio-frequency spectrum and satellite orbits, both of which are essential for mobile communication, broadcasting, and satellite-based services. By coordinating frequency allocation, ITU-R prevents interference and ensures efficient global communication.

3. ITU-D (Development Sector):

Focused on bridging the digital divide, ITU-D helps developing nations expand their communication infrastructure and adopt modern technologies, ensuring equitable access to information and connectivity worldwide.

Among these, the ITU-T is particularly crucial because it develops the technical foundations for how communication networks operate. Although its recommendations are not legally binding, they are almost universally adopted since non-compliance would result in technical isolation from the global telecommunication network—a disadvantage that most nations cannot afford. Only a few highly restricted systems, such as North Korea's closed networks, operate outside of ITU frameworks.

Standardization through ITU-T Study Groups

To manage its extensive scope, the ITU-T organizes its work through specialized Study Groups, each focusing on a specific technical area such as:

- Broadband and next-generation networks
- Network security and cybersecurity
- Multimedia and video communication services
- Cloud computing and IoT infrastructure
- Billing, numbering, and service management systems

Each Study Group consists of hundreds of experts from around the world and is further divided into Working Parties, Expert Teams, and ad hoc groups to address complex issues efficiently. This structure allows for collaborative problem-solving and continuous innovation in global telecommunications.

Over the decades, the ITU-T has published more than 3,000 Recommendations, many of which have become cornerstones of modern communication. Prominent examples include:
Recommendation H.264 (MPEG-4 AVC): A widely used video compression standard essential for digital TV, video streaming, and Blu-ray discs.

Recommendation X.509: The foundation for digital certificates used in secure web communications (HTTPS), ensuring authentication and encryption across the Internet.

Global Collaboration and the Future of Telecommunications

As telecommunications rapidly evolve through innovations like 5G networks, satellite-based Internet systems, and the Internet of Things (IoT), the importance of international standardization continues to grow. The ITU, along with other bodies such as 3GPP (Third Generation Partnership Project), IETF (Internet Engineering Task Force), and IEEE (Institute of Electrical and Electronics Engineers), plays a vital role in ensuring that emerging technologies are interoperable, secure, and globally accessible.

Moreover, the standardization process has become increasingly collaborative, involving governments, private companies, research institutions, and business consortia. This shared responsibility ensures that new technologies contribute to a cohesive, inclusive, and sustainable global communication network, enabling innovation while preserving interoperability and equity among nations.

5.4 WHO'S WHO IN THE INTERNET STANDARDS WORLD

Internet standardization differs fundamentally from traditional formal organizations like ITUT and ISO, both in procedure and culture. Traditional bodies emphasize bureaucracy, consensus, and corporate or governmental representation, while Internet standardization—driven by engineers and researchers—values openness, practicality, and informal collaboration. This ethos is best captured by David Clark's phrase, "rough consensus and running code," emphasizing working solutions over theoretical ideals.

Early Internet standardization began with the Internet Activities Board (IAB) in 1983, formed under the U.S. Department of Defense to coordinate ARPANET and Internet research. Standards were documented through publicly available RFCs (Requests for Comments), which remain central to Internet governance today.

As the Internet expanded in the late 1980s, this informal system was reorganized into three main entities:

IAB (Internet Architecture Board) – oversight and coordination.
IRTF (Internet Research Task Force) – focused on longterm research.
IETF (Internet Engineering Task Force) – responsible for practical Internet standards.
The IETF, composed of numerous working groups, became the primary standardization body. Its standards process includes stages: Proposed Standard, Draft Standard, and finally Internet Standard, with implementation testing by independent groups before approval.
For the World Wide Web, a distinct body—the World Wide Web Consortium (W3C)—was founded in 1994 by Tim BernersLee. The W3C develops and maintains key web technologies such as HTML, CSS, XML, and web privacy standards. Together, IETF and W3C represent a

collaborative, decentralized model that mirrors the Internet's distributed structure and ensures interoperability and openness globally.

5.5 SUMMARY

This chapter highlights the importance of network standardization to ensure compatibility and interoperability among global communication systems. It introduces key organizations such as the ITU and IEEE in the telecommunication world, ISO in international standardization, and bodies like the IETF and W3C in Internet standard development. The chapter also covers the Physical Layer, which deals with the actual transmission of bits over physical media. It explains various guided transmission media—including magnetic media, twisted pair cables, coaxial cables, and fiber optics—describing how each medium carries data with different speed, cost, and reliability characteristics.

5.6 TECHNICAL TERMS

IUT, IEEE, W3C, ISO, IETF, IRTF

5.7 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the roles of major telecommunication standard organizations.
2. Describe the international bodies involved in developing network standards.
3. Discuss the Internet standardization process and the key organizations responsible.
4. Compare the characteristics of various standards.

Short Questions:

1. What is the main purpose of network standardization?
2. Name any two organizations involved in telecommunication standards.
3. What are Internet standards, and who maintains them?

5.8 FURTHER READINGS

1. Andrew S. Tanenbaum, "Computer Networks", Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, "Computer Networking", Third Edition, Pearson Education
3. Behrouz A Forouzan, "Data Communications and Networking", Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, "Computer Communications and NetworkingTechnologies", Cengage Learning (2008)

Dr. Neelima Guntupalli

LESSON- 6

GUIDED TRANSMISSION MEDIA

OBJECTIVES:

After going through this lesson, you will be able to

- Define guided transmission media and differentiate it from unguided media.
- Describe the types and characteristics of twisted pair, coaxial, and optical fiber cables.
- Compare the advantages, limitations, and applications of various guided media.
- Understand signal attenuation, noise, and methods to improve transmission quality.
- Explore modern applications and trends in high-speed guided communication networks

STRUCTURE OF THE LESSON:

The physical layer is responsible for transmitting raw bits from one machine to another using various transmission media. These media differ in bandwidth, delay, cost, and ease of installation and maintenance. They are broadly categorized into guided media, like copper wires and fiber optics, where signals follow a physical path, and unguided media, like wireless, satellite, and laser, where signals travel through open space.

Transmission media is the part of the physical layer that provides a path through which data is transmitted from one place to another in computer networks. It is also known as a transmission channel, Communication media, or communication channel.

The amount of data that can be transferred through a communication medium in a unit of time is called bandwidth.

The *bandwidth of the digital signal* is measured in *bits per second or bytes per second.*
The *bandwidth of Analog's signal* is measured in *cycles per second or Hertz*

Figure 6.1 Transmission media in Computer Networks

6.1 MAGNETIC MEDIA

One of the simplest yet surprisingly efficient methods for transferring large amounts of data between computers involves writing the data onto physical storage media—such as magnetic tapes or recordable DVDs—and then physically transporting these to the destination where they are read back into a machine. While this may seem rudimentary compared to advanced technologies like communication satellites, it is often far more cost-effective, particularly for applications where high bandwidth and low cost per bit are critical. This practicality becomes clearer when we consider a concrete example. An Ultrium magnetic tape, which is commonly used in industry, has a storage capacity of 800 gigabytes. A cubic box with dimensions 60 cm on each side can hold about 1000 such tapes, which translates to a total of 800 terabytes or 6.4 petabits of data. If this box is shipped overnight anywhere in the United States using courier services like FedEx, the effective data transmission rate is about 70 gigabits per second. If the destination is only an hour's drive away, the effective bandwidth soars to over 1700 gigabits per second. No current computer network, regardless of how advanced, can rival this level of data transfer speed, especially across such distances.

When examining cost, the picture is similarly striking. Each Ultrium tape costs around $40 when bought in bulk, and each tape can be reused at least ten times. This means the cost of the tapes for a single use is around $4000 per shipment. Adding in an estimated shipping cost of $1000, the total expense to move 800 terabytes comes to about $5000. This works out to a cost of just over half a cent per gigabyte transferred—an extremely economical rate that no digital network can match. This example highlights the often-overlooked but crucial truth that, in certain contexts, physical data transport using conventional logistics can outpace and outprice even the most high-tech digital networks. As humorous as it may sound, the lesson to take away is that one should never underestimate the effective bandwidth of a station wagon full of data tapes speeding down the highway.

6.2 TWISTED PAIRS

While magnetic tape offers impressive bandwidth for data transfer, it suffers significantly in terms of delay. Data transmission using tapes takes minutes or even hours, making it unsuitable for applications that require real-time or near-instantaneous communication. In such scenarios, online connections become essential. One of the oldest and most widely used transmission media for online communication is the twisted pair. This medium consists of two insulated copper wires, typically around one millimeter in diameter, twisted together in a helical shape resembling the structure of DNA. The purpose of twisting the wires is to reduce electromagnetic interference. When wires are simply laid out in parallel, they can act as antennas, picking up unwanted signals. Twisting them causes electromagnetic waves from different twists to cancel each other out, which reduces radiation and improves noise

immunity. The signal is transmitted as a voltage difference between the two wires, and since external noise tends to affect both wires similarly, the signal integrity is preserved.

Twisted pair cables are most commonly used in the telephone system. They are the primary medium connecting individual telephones to the telephone company's central office, and they also support ADSL-based Internet connections. These cables can carry signals for several kilometers without amplification, but beyond that, the signal weakens and repeaters are required. When multiple twisted pairs are used together, as in the case of all the lines from an apartment complex heading to a telephone exchange, they are bundled and enclosed in a protective outer sheath. Without the twisting, the wires in such a bundle would interfere with each other. In areas where telephone lines are suspended on poles, it is common to see thick bundles containing many twisted pairs. Twisted pair cables can carry both analog and digital signals, and their bandwidth depends on factors such as the cable thickness and the transmission distance. Generally, these cables can support several megabits per second over a few kilometers, making them a popular and cost-effective choice for many communication needs. Because of their practicality and performance, twisted pair cables are expected to remain in use for the foreseeable future.

There are various types of twisted pair cables. A widely used variant in office environments is the Category 5 (Cat 5) cable, which contains four pairs of insulated wires gently twisted together and encased in a plastic sheath for protection and organization. Different LAN standards utilize these wires in various ways. For instance, 100-Mbps Ethernet typically uses two of the four pairs, with each pair handling data in one direction. In contrast, 1-Gbps Ethernet employs all four pairs for bidirectional communication, requiring sophisticated receivers that can separate outgoing and incoming signals. Communication links also differ in terms of directionality. Full-duplex links can transmit data in both directions simultaneously, much like a two-lane road. Half-duplex links allow data transmission in either direction but not at the same time, resembling a single-track railway. Simplex links, on the other hand, only allow data to travel in one direction, like a one-way street.

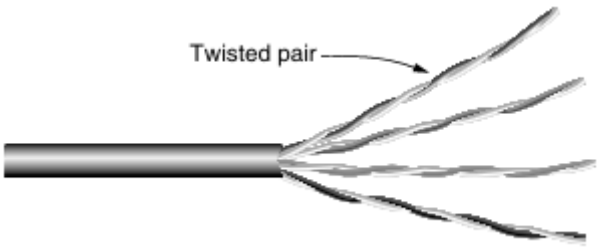


Figure 6.2 Category 5 UTP cable with four twisted pairs

As technology evolved, Cat 5 cables replaced the earlier Category 3 standard by incorporating more twists per meter, which reduced signal crosstalk and allowed for higher-quality transmissions over longer distances—especially for 100-Mbps and 1-Gbps Ethernet networks. Modern installations are more likely to use Category 6 or even Category 7 cables. These newer categories offer enhanced specifications to support higher bandwidths. Category 6 cables can handle signals up to 500 MHz and are capable of supporting 10-Gbps transmission speeds. Up to Category 6, these cables are classified as UTP, or Unshielded Twisted Pair, meaning they consist only of insulated wires without any additional shielding. Category 7 cables, however, introduce shielding for each twisted pair and for the entire cable

assembly within the outer sheath. This extra protection reduces external interference and crosstalk from adjacent cables, making them suitable for more demanding communication environments. Interestingly, this approach mirrors IBM's early use of shielded twisted pair cables in the 1980s, which did not gain much popularity outside IBM at the time. However, with today's growing demands for higher data rates and cleaner signal transmission, shielded cables are making a comeback.

6.3 COAXIAL CABLE

Coaxial cable, commonly referred to as "coax," is another widely used transmission medium that offers superior shielding and bandwidth capabilities compared to unshielded twisted pair cables. Its construction allows it to support higher speeds over longer distances, making it suitable for various digital and analog communication applications. Coaxial cables are broadly categorized into two main types based on their impedance. The 50-ohm coaxial cable is typically used for digital transmission, having been designed with data communication in mind from the outset. In contrast, the 75-ohm coaxial cable is traditionally used for analog signals and is prevalent in cable television systems. The historical preference for 75-ohm cables in television applications can be traced back to the impedance characteristics of early antennas and the convenience of using available impedance-matching transformers. However, with the evolution of technology and services, 75-ohm coaxial cables have taken on a more prominent role in data communications, particularly since the 1990s when cable television companies began offering broadband Internet services over the same infrastructure.

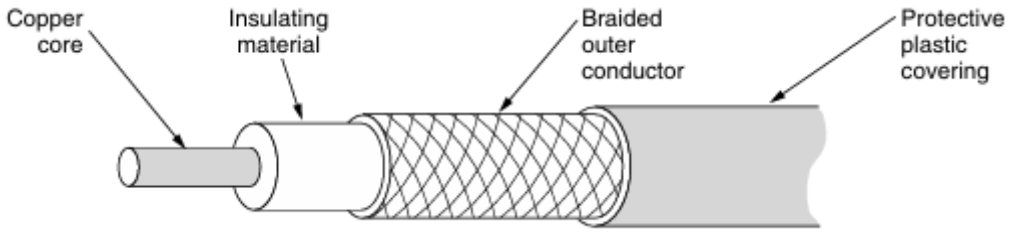


Figure 6.3 A Coaxial cable

The physical structure of a coaxial cable consists of a solid copper core that acts as the central conductor. This core is surrounded by an insulating layer, which in turn is encased in a cylindrical outer conductor made of a woven braided mesh that provides shielding from electromagnetic interference. The entire assembly is then protected by an external plastic sheath that offers durability and environmental resistance. This layered construction enables the coaxial cable to achieve a robust balance between high bandwidth capacity and strong noise immunity. Depending on the quality and length of the cable, coaxial systems can support frequencies up to several gigahertz, making them suitable for high-speed transmissions.

Although coaxial cables were once a staple in the telephone industry for long-distance communication, their role in such applications has diminished significantly with the advent and widespread adoption of fiber optic technology, which offers even higher bandwidth and lower signal degradation over long distances. Nonetheless, coaxial cables remain important in specific domains. They continue to be widely used for distributing cable television signals to households and buildings and also serve as a backbone for some metropolitan area networks (MANs). Their reliability, ease of installation, and compatibility with existing infrastructure

ensure that coaxial cables will continue to play a valuable role in communication systems for years to come.

6.4 POWER LINES

Electrical power lines, commonly used to deliver electricity to homes and distribute it through internal wiring, are also being explored for data communication. This idea is not entirely new—power companies have long used power lines for low-speed communication tasks like remote metering and home automation through standards such as X10. However, recent developments have renewed interest in using these lines for high-speed data communication, particularly within homes to create local area networks (LANs) and even for broadband Internet access.

One of the major advantages of using power lines for networking is their ubiquity and convenience. Every home is already equipped with power outlets in virtually every room, eliminating the need for additional wiring. Devices such as televisions and media players already need to be plugged into power outlets, so enabling them to communicate over the same wires simplifies installation and setup. The system works by superimposing a high-frequency data signal onto the low-frequency 50–60 Hz power signal carried by the active wire, allowing both signals to coexist on the same electrical path. This setup is straightforward and requires no additional networking infrastructure like Ethernet cables or wireless routers.

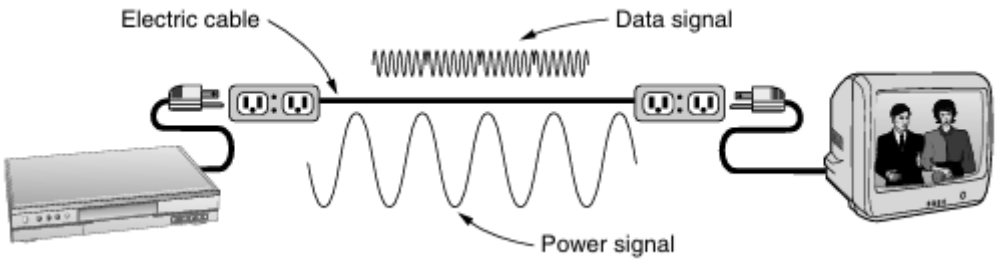


Figure 6.4 A network that uses household electrical wiring

However, using electrical wiring for data transmission comes with significant technical challenges. Household wiring was designed exclusively for carrying low-frequency power, not high-frequency data. As a result, it performs poorly when tasked with transmitting signals in the MHz range, which are essential for high-speed communication. Signal quality is highly inconsistent because the electrical characteristics of household wiring differ from one home to another and fluctuate depending on which appliances are in use. When devices are turned on or off, they introduce electrical noise and sudden surges that disrupt the data signal. Moreover, since household wiring lacks the shielding and twisting found in specialized communication cables, it acts like an antenna—absorbing unwanted external signals and emitting interference that can disrupt other devices.

Another complication is that data signals must avoid using frequency bands that are already licensed, such as those used by amateur radio operators. This requirement limits the range of usable frequencies and adds to the technical complexity. Despite these challenges, it is still feasible to achieve data rates of 100 Mbps or more over standard home electrical wiring. This

is made possible by using advanced communication techniques that can withstand noise, frequency disturbances, and bursts of errors. While many current power-line networking products rely on proprietary standards, international organizations are working to develop universal standards to ensure compatibility and reliability across devices and regions.

6.5 FIBER OPTICS

The rate of advancement in computer technology has been a source of immense pride within the industry, often highlighted by Moore's Law, which suggests that the number of transistors on a chip doubles approximately every two years. This prediction has translated into remarkable improvements in computing performance. For instance, the original IBM PC, introduced in 1981, operated at a clock speed of 4.77 MHz. Over the course of nearly three decades, this evolved into computers running multiple-core processors at speeds of around 3 GHz. This increase represents a performance gain of roughly 2500 times, or about sixteenfold per decade, which is indeed an impressive achievement.

During the same time frame, the progress in wide-area communication has been equally extraordinary. Communication speeds rose from 45 Mbps, as seen in traditional T3 telephone lines, to 100 Gbps in modern long-distance connections. This progress, matching the scale of advancement in computing, also reflects a similar rate of sixteen times per decade. Additionally, there has been a drastic reduction in transmission errors, with bit error rates decreasing from one in 100,000 to nearly zero. While computing is approaching physical limitations that constrain performance enhancements, such as those related to chip size and heat dissipation, communication through fiber optic technology still offers vast untapped potential. The theoretical bandwidth of fiber optics exceeds 50,000 Gbps (or 50 Tbps), which remains far above current practical transmission capabilities.

The practical limit in communication speeds today is not due to the fiber itself but the inability to perform rapid electrical-to-optical signal conversions. To bypass this limitation, multiple channels are transmitted in parallel through a single fiber. As a result, communication technology, particularly through fiber optics, may eventually surpass computing in terms of scalability and potential. This shift could change the prevailing mindset among engineers and computer scientists, who have traditionally focused on conserving bandwidth due to the limitations of copper wires and the low Shannon limits. In contrast, fiber optics could enable an era of virtually unlimited bandwidth.

However, the scenario is more complex when costs are considered. Installing fiber optic cables to reach every household, especially in the last-mile segment, involves substantial financial investment. Furthermore, the energy cost of moving data over networks is often higher than that of performing computations locally. This economic and energy cost imbalance suggests that while some areas of the network may have access to abundant bandwidth, others will continue to rely more heavily on computation and storage to optimize data usage. For example, on the user side of the Internet, heavy use of caching and compression helps manage limited bandwidth effectively. Meanwhile, in core networks, large-scale data transfers are used strategically by companies like Google to take advantage of cheaper storage and computational resources in different geographic locations.

Fiber optics are widely used in long-distance backbone transmission, high-speed local area networks, and Internet access technologies like Fiber to the Home (FttH). A fiber optic transmission system typically consists of a light source, the fiber medium, and a light

detector. Data is transmitted by converting electrical signals into pulses of light, where the presence of light represents a binary 1 and its absence a binary 0. These light pulses travel through a thin strand of glass, and at the receiving end, the light detector converts the signal back into electrical form. This process allows for high-speed, long-distance data transmission with minimal signal degradation.

The efficiency of fiber optic communication relies on a physical phenomenon known as total internal reflection. When light passes from one medium to another—for example, from glass to air—it bends, or refracts. If the angle of incidence exceeds a certain critical threshold, the light reflects entirely within the glass medium, rather than escaping. This reflection enables light to travel through the fiber with very little loss over long distances. While many rays may reflect internally at various angles, a fiber that supports multiple paths or modes is called a multimode fiber.

To further improve performance, the diameter of the fiber can be reduced to a few wavelengths of light, allowing the light to travel in a straight path without bouncing. This configuration is known as a single-mode fiber. Though more expensive, single-mode fibers are ideal for long-distance communication and are capable of transmitting data at speeds of up to 100 Gbps over distances of 100 kilometers without amplification. Experimental setups have even achieved higher speeds over shorter distances, demonstrating the immense potential of this technology for future communication systems.

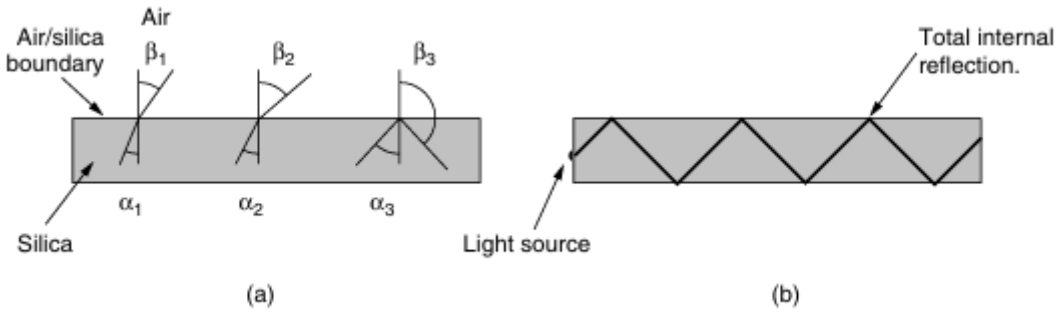


Figure 6.5 (a) Three examples of a light ray form inside a silica fiber impinging on the air/silica boundary at different angles. (b) Light trapped by total internal reflection

Transmission of light through fiber

Optical fibers are made from glass, a material derived from sand, which is both inexpensive and abundantly available in nature. Although glassmaking is an ancient skill, known even to the Egyptians, early glass was limited in thickness and transparency. It wasn't until the Renaissance that techniques were developed to create glass clear enough for use in windows. Modern optical fiber glass, however, has achieved an extraordinary level of transparency—so much so that if oceans were filled with it instead of water, the seabed would appear as clearly from the surface as the earth does from an airplane on a clear day. This exceptional clarity is crucial for transmitting light signals across long distances with minimal loss.

The degree to which light attenuates, or weakens, as it passes through glass depends on both the physical properties of the glass and the wavelength of the light. Attenuation is usually expressed in decibels per kilometer, representing the logarithmic ratio of the input to the

output power of the signal. A signal power loss by a factor of two corresponds to 3 dB of attenuation. In practical optical fiber communication, light in the near-infrared part of the electromagnetic spectrum is used rather than visible light, which spans from approximately 0.4 to 0.7 microns in wavelength. For fiber optics, the preferred wavelengths are centered around 0.85, 1.30, and 1.55 microns, which fall in the near-infrared range. Each of these wavelength bands supports a very wide frequency spectrum—about 25,000 to 30,000 GHz—making them highly suitable for high-speed data transmission.

Among the three bands, the 0.85-micron band was the first to be used. Although it has higher attenuation and is therefore less efficient over long distances, it offered the advantage that both the lasers and electronic components could be made from the same material, gallium arsenide, simplifying early development. The other two bands, particularly the 1.55-micron band, offer much lower attenuation—less than 5% loss per kilometer—and are now more widely used. In fact, the 1.55-micron band is commonly employed in modern fiber optic systems, often in conjunction with erbium-doped fiber amplifiers that can boost the signal directly in the optical domain without the need for conversion to electrical signals.

As light pulses travel through a fiber, they tend to spread out—a phenomenon known as chromatic dispersion. This spreading is wavelength dependent and can lead to overlapping pulses, which degrades signal clarity and limits transmission speed. One straightforward way to counteract this is by increasing the distance between the pulses, but this approach also reduces the overall data rate. A more sophisticated solution lies in shaping the light pulses in a special mathematical form related to the reciprocal of the hyperbolic cosine function. These specially shaped pulses, known as solitons, have the remarkable property of resisting dispersion as they travel. As a result, they maintain their shape over very long distances, allowing for high-speed communication without significant distortion. Although still largely experimental, solitons hold great promise for the future of ultra-long-distance optical communication, and significant research is ongoing to bring them from laboratory environments into practical, real-world applications.

Fiber cables

Fiber optic cables are constructed similarly to coaxial cables but without the braided outer conductor. At the center of a fiber optic cable lies a glass core through which light signals travel. In multimode fibers, this core is about 50 microns in diameter—roughly the thickness of a human hair—while in single-mode fibers, the core diameter is only around 8 to 10 microns. Surrounding the core is a layer of glass called the cladding, which has a lower refractive index than the core. This difference in refractive indices ensures that light remains confined within the core by reflecting internally along the fiber's length. Outside the cladding is a protective plastic jacket, and typically, several of these fiber strands are bundled together and covered by an outer sheath for additional protection.

These fiber optic cables are usually installed underground at shallow depths, where they can be vulnerable to damage from construction equipment or burrowing animals. Underwater cables that connect continents are buried near coastlines using specialized machines called seaplows, but in deeper oceanic regions, the cables are left to rest directly on the seabed. In such environments, they face threats from fishing operations and even large marine creatures.
Fiber optic cables can be connected in three main ways. The first is through the use of connectors, which allow fibers to be plugged into sockets. While this method results in some light loss—typically between 10 to 20 percent—it allows for easy reconfiguration of systems.

The second method involves mechanical splicing, where two fibers are precisely aligned and clamped together in a sleeve. Light is passed through the junction to fine-tune the alignment, and though the process takes a few minutes and results in about 10% light loss, it provides a reliable connection. The third and most seamless method is fusion splicing, where the fiber ends are melted and fused into a single, solid piece. Fusion splices have minimal attenuation and are nearly as efficient as a continuous fiber strand. However, all splicing techniques may introduce reflections at the joint, which can interfere with signal quality.

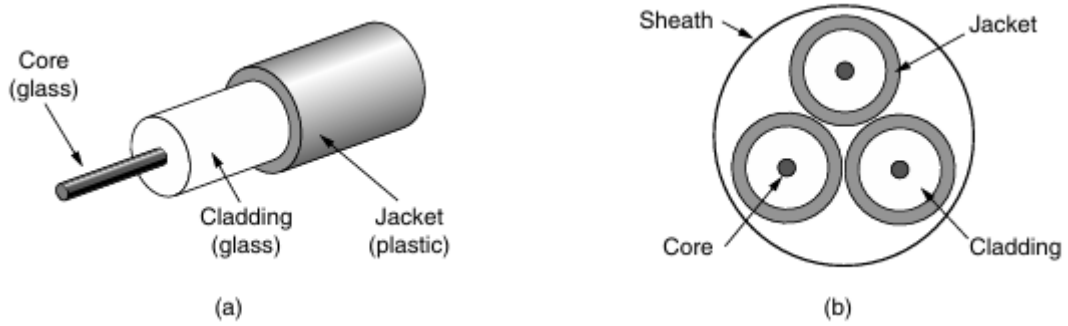


Figure 6.6 (a) side view of a single fiber (b) End view of a sheath with three fibers

The light sources used in fiber optic systems are typically either LEDs (Light Emitting Diodes) or semiconductor lasers. LEDs are more suitable for lower data rates and shorter distances and are primarily used with multimode fibers. They have long lifespans, are inexpensive, and exhibit minimal sensitivity to temperature changes. On the other hand, semiconductor lasers can support higher data rates and longer distances and can be used with both multimode and single-mode fibers. However, they are more expensive, have shorter lifespans, and are more sensitive to temperature variations. The wavelength of the light they emit can be precisely tuned using devices such as Fabry-Perot and Mach-Zehnder interferometers. Fabry-Perot interferometers use two parallel mirrors to form a resonant cavity that filters specific wavelengths. Mach-Zehnder interferometers, meanwhile, split light into two beams, allow them to travel different distances, and then recombine them to select specific wavelengths based on phase interference.

Item	LED	Semiconductor laser
Data rate	Low	High
Fiber type	Multi-mode	Multi-mode or single-mode
Distance	Short	Long
Lifetime	Long life	Short life
Temperature sensitivity	Minor	Substantial
Cost	Low cost	Expensive

Figure 6.7 A comparison of semiconductor diodes and LEDs as light sources

At the receiving end of a fiber optic link, photodiodes are used to detect the incoming light pulses and convert them back into electrical signals. These photodiodes are limited by their response time, capping practical data rates at around 100 Gbps. Additionally, thermal noise can interfere with detection, so the light pulses must carry enough energy to be reliably sensed. By ensuring that each pulse has sufficient power, the error rate can be minimized to

extremely low levels, enabling highly reliable and efficient communication over long distances.

Comparison of fiber optics and copper wire

Fiber optic cables offer a range of advantages over traditional copper cables, making them highly suitable for modern high-speed communication networks. One of the most significant benefits of fiber is its ability to support much higher bandwidths than copper. This makes it an ideal choice for high-end networks that require rapid data transmission. Another major advantage is its low signal attenuation, which means that repeaters—devices used to regenerate signals—are needed far less frequently. While copper cables require repeaters approximately every 5 kilometers, fiber optics only need them about every 50 kilometers. This difference results in considerable cost savings for long-distance communication infrastructure.

Fiber optics are also immune to a range of environmental and electrical disturbances. Unlike copper, they are not affected by electromagnetic interference, power surges, or electrical failures. They are also resistant to corrosive chemicals in the atmosphere, which is particularly beneficial in industrial or factory settings where such conditions are common. Interestingly, telephone companies have embraced fiber not just for its technical superiority, but also for its physical characteristics. Fiber cables are much thinner and lighter than copper cables, which is crucial in urban settings where existing underground ducts are already crowded. Replacing bulky copper cables with slim fiber optic lines frees up space in these ducts and offers the additional benefit of reclaiming the copper, which has high resale value due to its purity.

The weight difference between the two is also striking. A bundle of 1,000 twisted copper pairs extending one kilometer weighs about 8,000 kilograms. In contrast, just two fiber strands with significantly more data-carrying capacity weigh only around 100 kilograms. This substantial reduction in weight minimizes the need for heavy-duty support systems, further lowering maintenance and infrastructure costs. Fiber also excels in terms of security. Because it does not emit electromagnetic signals and is extremely difficult to tap without detection, it offers strong resistance to eavesdropping.

However, despite these advantages, fiber optics do come with a few drawbacks. The technology is less familiar to many engineers, which means specialized skills and training are required for installation and maintenance. The physical nature of fiber makes it more fragile than copper, as it can be damaged if bent too sharply. Additionally, because optical transmission is inherently unidirectional, two-way communication demands either two separate fibers or the use of different frequency bands on a single fiber. Furthermore, the interfaces used with fiber optics are typically more expensive than those used with electrical systems. Nonetheless, despite these challenges, fiber optics clearly represent the future of fixed long-distance data communication due to their superior performance and overall cost-effectiveness.

6.6 SUMMARY

This chapter highlights the importance of network standardization to ensure compatibility and interoperability among global communication systems. It introduces key organizations such as the ITU and IEEE in the telecommunication world, ISO in international standardization,

and bodies like the IETF and W3C in Internet standard development. The chapter also covers the Physical Layer, which deals with the actual transmission of bits over physical media. It explains various guided transmission media—including magnetic media, twisted pair cables, coaxial cables, and fiber optics—describing how each medium carries data with different speed, cost, and reliability characteristics.

6.7 TECHNICAL TERMS

IUT, IEEE, W3C, ISO, IETF, IRTF

6.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain different guided transmission media: magnetic media, twisted pair, coaxial cable, and fiber optics.
2. Compare the characteristics, advantages, and limitations of various guided transmission media.
3. Describe twisted pair cables applications in communication networks.
4. Discuss the structure, working principle, and performance of fiber optic cables.

Short Questions:

1. What is magnetic media, and where is it commonly used in communication systems?
2. List any two types of guided transmission media.
3. Write any two differences between coaxial cable and twisted pair cable.
4. What is fiber attenuation, and why is it a concern in optical fiber transmission?

6.9 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008).

Dr. Neelima Guntupalli

LESSON- 7

DATA LINK LAYER DESIGN ISSUES

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the functions of the data link layer.
- Learn the services provided to the network layer.
- Explain the concept and methods of framing.
- Describe error detection and correction mechanisms.
- Understand flow control and its importance in data transmission.

STRUCTURE OF THE LESSON:

7.1 INTRODUCTION

The data link layer is the second layer in the OSI (Open Systems Interconnection) model, and it plays a critical role in ensuring that data is transferred reliably and efficiently across a physical communication channel. It builds upon the raw transmission capability provided by the physical layer, which is responsible only for transmitting individual bits over a medium such as a wire, fiber optic cable, or wireless link. While the physical layer focuses on sending these bits as electrical signals, light pulses, or radio waves, it does not understand the structure or meaning of the data. That responsibility falls to the data link layer.

One of the primary functions of the data link layer is to provide a well-defined interface to the network layer, which is the layer above it. This means that the network layer should not have to worry about how data is transmitted across the physical medium. The data link layer takes care of that complexity and presents the network layer with a reliable service, allowing it to send and receive complete data units, or packets, without dealing with the noise and errors of the physical medium.

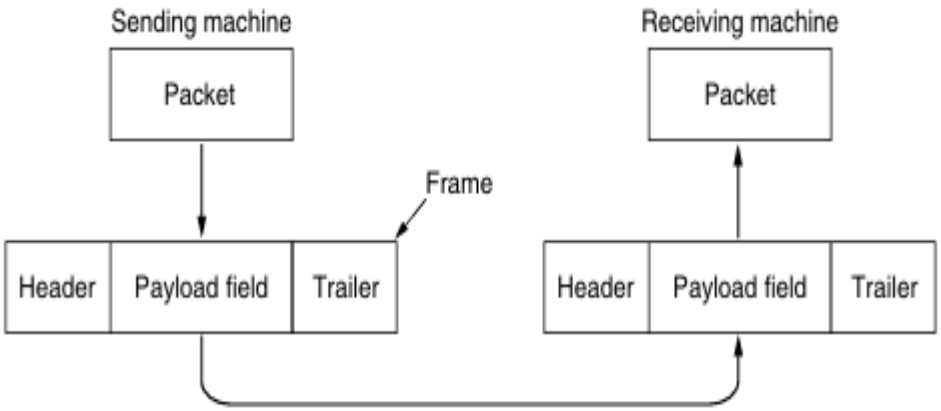


Figure 7.1 Relationship between packets and frames

Another crucial role of the data link layer is error handling. Physical transmission is often prone to noise, interference, and other factors that can result in bit errors (i.e., a 0 being received as a 1 or vice versa). To detect and sometimes even correct these errors, the data link layer uses techniques like checksums, cyclic redundancy checks (CRC), or parity bits. These error detection mechanisms are implemented in the frame trailer, which is part of the structured format used by the data link layer.

In addition to error control, the data link layer also manages flow control. This refers to mechanisms that regulate the rate of data transmission between a sender and a receiver. If a fast sender transmits data at a speed that a slow receiver cannot keep up with, it may lead to buffer overflow and data loss. The data link layer prevents this by ensuring that the sender only sends data when the receiver is ready to accept it, often using techniques such as stop-and-wait or sliding window protocols.

To perform these functions effectively, the data link layer groups bits into structured data units called frames. When a packet is passed down from the network layer, the data link layer encapsulates it into a frame. This frame includes a header, a payload field, and a trailer. The header contains information like source and destination MAC addresses, frame type, and control information. The payload field carries the actual data or packet from the network layer. The trailer, typically, includes the error-detection bits such as the checksum or CRC. On the receiving side, the frame is decoded, the payload is extracted and passed to the network layer, and the header/trailer are processed or discarded after performing their functions.

The diagram referred to in Figure 7.1 (though not shown here) typically illustrates the encapsulation process: the network layer sends a packet to the data link layer, which wraps it with a header and a trailer to form a frame. This frame is what is actually transmitted over the physical medium. When the frame reaches the destination, the data link layer on the receiving machine checks it for errors, removes the header and trailer, and forwards the packet up to the network layer.

Although the discussion is centered on the data link layer, it's important to note that the concepts of reliability, error control, and flow control appear in other layers too—especially in the transport layer. That’s because achieving end-to-end reliability in data communication is a shared responsibility across all layers of the network stack. Some networks rely heavily on the data link layer for these functions, while others delegate most of them to the upper

layers. Regardless of where these mechanisms are implemented, their fundamental principles remain consistent across layers.

In fact, many of these mechanisms are often found in their simplest and most fundamental forms at the data link layer, making it an excellent place to study and understand them. The straightforward and well-defined environment at this layer allows learners to grasp the core concepts of reliable communication, which are later built upon and extended in the upper layers of the network architecture

7.2 SERVICES PROVIDED TO THE NETWORK LAYER

The primary function of the data link layer is to provide communication services to the network layer, which resides immediately above it in the OSI model. Its core responsibility is to transfer data from the network layer of the source machine to the network layer of the destination machine in a reliable and efficient manner. At the source, a process in the network layer generates a packet, which is handed down to the data link layer. This layer then manages the transmission of the packet across the physical medium. Upon reaching the destination, the data is passed up to the network layer of the receiving machine. Conceptually, we often model this interaction as if two peer processes in the data link layer—one at the sender and one at the receiver—are directly communicating with each other using a data link protocol, even though in reality, the bits travel across the physical layer as shown in the actual communication path.

To meet the needs of different types of communication and environments, the data link layer can be designed to offer multiple types of services. These services are not fixed and can vary depending on the specific data link protocol in use. The three main types of services that a data link protocol may provide are:

1. Unacknowledged Connectionless Service,
2. Acknowledged Connectionless Service, and
3. Acknowledged Connection-Oriented Service.

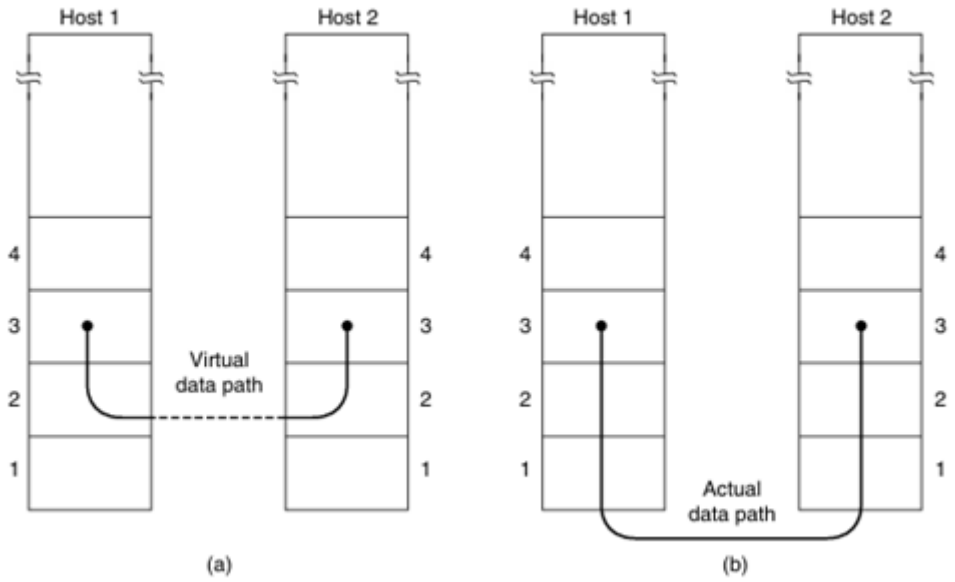


Figure 7.2 (a) Virtual communication (b) Actual communication

7.2.1. Unacknowledged Connectionless Service

This is the simplest and least reliable service provided by the data link layer. In this mode, the sender transmits independent frames to the destination without expecting or receiving any form of acknowledgement. There is no logical connection established between the sender and receiver before data transfer begins, nor is there any procedure to release a connection afterward. If a frame is lost during transmission—say, due to noise or interference—the data link layer does not attempt any recovery. This type of service is ideal for environments where the likelihood of error is very low, such as high-quality wired networks. It is also well-suited for real-time applications, such as audio or voice transmission, where receiving delayed data is worse than receiving slightly corrupted data. An example of a system using this service is Ethernet, which prioritizes speed and simplicity over guaranteed delivery.

7.2.2 Acknowledged Connectionless Service

The next level of reliability is provided by acknowledged connectionless service. Like the unacknowledged version, this service also does not involve setting up a dedicated logical connection. However, in this case, each frame is individually acknowledged by the receiver. This feedback allows the sender to detect whether a frame has been lost or corrupted. If an acknowledgment is not received within a specified time period, the sender can retransmit the frame, ensuring better reliability. This service is particularly useful in environments with higher error rates, such as wireless communication systems. A common example of this service is found in 802.11 Wi-Fi protocols, where wireless signals are more prone to disruption, and acknowledgements help ensure the successful delivery of data.

It is important to understand that adding acknowledgements at the data link layer is optional and often seen as an optimization rather than a strict necessity. The network layer, which sits above, can also handle acknowledgements by managing timers and retransmissions itself. However, relying solely on the network layer can be inefficient, especially because it is unaware of the physical characteristics of the communication channel, such as frame size limits or propagation delays. For example, if the network layer sends a large packet that is broken into several frames and some of those frames are lost, it would have to retransmit the entire packet, not just the missing frames. In contrast, the data link layer, with its understanding of frame sizes and local errors, can retransmit only the lost frames, saving time and bandwidth. This optimization becomes especially valuable on unreliable links, such as wireless channels, where frame loss is more common.

7.2.3 Acknowledged Connection-Oriented Service

The most sophisticated and reliable service offered by the data link layer is the acknowledged connection-oriented service. In this mode, a logical connection is established between the source and destination data link layer processes before any data is transmitted. During this connection establishment phase, both ends initialize necessary variables, buffers, and sequence counters. Once the connection is set up, data can be exchanged in the form of numbered frames, which ensures that the receiver can detect duplicates, reorder frames if needed, and confirm successful delivery. This service guarantees that every frame sent is received exactly once and in the correct order. After the data transfer is complete, the connection is properly terminated, and resources are freed.

This type of service is particularly well-suited for long-distance or unreliable links, such as satellite communications or telephone circuits, where errors and delays are more likely to occur. Unlike the acknowledged connectionless mode, which might result in repeated transmissions due to lost acknowledgements, connection-oriented service maintains more robust state information, allowing it to manage retransmissions, acknowledgements, and sequencing in a coordinated way. While this adds complexity and overhead, the benefits are significant in terms of reliability, especially over noisy or high-latency channels.

7.2 FRAMING

To deliver services to the network layer, the data link layer depends on the services offered by the physical layer. The physical layer is responsible for accepting a raw stream of bits and attempting to transmit them over a communication channel to the destination. However, the physical layer operates in a noisy and error-prone environment—especially in wireless channels and even in some wired links—which means that despite any redundancy mechanisms built into the physical signal encoding, errors are still possible. The physical layer may attempt to reduce the bit error rate by adding redundancy (such as error-correcting codes), but it does not guarantee error-free delivery. As a result, the bit stream received by the data link layer may be corrupted, with altered bits or a different number of bits than what was sent.

This is where the data link layer's responsibilities begin. One of its key roles is to detect and, when possible, correct errors in the incoming data. To do this, the data link layer breaks the continuous incoming bit stream into discrete units called frames. For each frame, it computes a short value known as a checksum, using a predefined algorithm. The checksum is then appended to the frame before transmission. When the frame arrives at the receiver’s data link layer, it recalculates the checksum based on the received data. If the newly calculated checksum matches the one sent in the frame, the data is considered valid; otherwise, it is assumed to be corrupted. In such cases, the data link layer may discard the faulty frame and possibly send back an error message or request a retransmission, depending on the protocol in use.

However, framing—the process of segmenting the bit stream into frames—is more challenging than it may appear. A robust framing strategy must allow the receiver to accurately identify where each frame begins and ends, and it should do this efficiently, without consuming too much of the available bandwidth. Over time, engineers have developed four main framing techniques to achieve this:

7.2.1. Byte Count Method

The byte count method is the simplest way to implement framing. In this approach, a field in the header of each frame explicitly states the number of bytes that follow. Using this information, the receiver can determine where the frame ends. For example, a frame header may contain the number 5, indicating that the next 5 bytes belong to the payload. While this technique works in error-free environments, it has a major drawback: if the byte count is corrupted, the receiver loses synchronization and cannot determine where the next frame begins. Even if the checksum indicates that the current frame is invalid, the receiver has no reliable way to realign itself with the incoming stream, as the corrupted length makes skipping to the next frame impossible. This issue makes the byte count method unreliable for noisy channels, and as a result, it is rarely used alone in modern data link protocols.

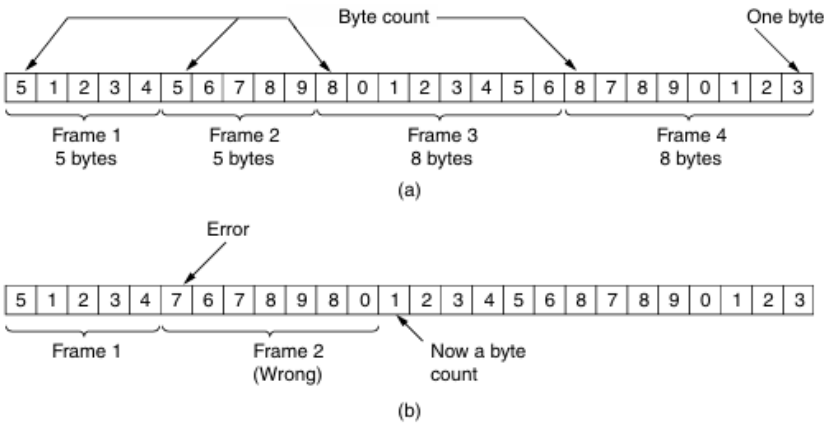


Figure 3-3. A byte stream. (a) Without errors. (b) With one error.

Figure 7.3 A byte stream (a) without errors (b) With one error

7.2.2 Flag Bytes with Byte Stuffing

To overcome the synchronization problem of byte counts, the flag byte method is used. In this approach, each frame begins and ends with a special byte pattern, typically known as the flag byte (for example, 0x7E). The receiver scans the incoming byte stream for these flags. When it detects two consecutive flags, it interprets the first as the end of the previous frame and the second as the start of the next. This ensures that even if part of a frame is lost or corrupted, the receiver can resynchronize by locating the next pair of flags.

However, a new problem arises: what if the flag byte appears in the actual data? For instance, when transmitting images, music files, or other binary content, the data may unintentionally contain the flag byte. To distinguish between real frame delimiters and flag bytes in the data, the sender uses a method called byte stuffing. Whenever a flag byte appears in the payload, the sender inserts an escape byte (ESC) before it. If an actual ESC byte appears in the data, it is also escaped by inserting another ESC byte before it. At the receiver end, the ESC byte is used as a signal to unescape the following byte—either restoring a flag byte or an ESC byte—and thus recover the original data. This approach allows the receiver to maintain accurate framing boundaries while ensuring that the original data remains intact.

7.2.3 Flag Bits with Bit Stuffing

Although byte stuffing works well, it assumes that the transmission is organized into 8-bit bytes. To support devices or media that operate at the bit level, protocols like HDLC (High-Level Data Link Control) use bit stuffing instead. In this approach, frames start and end with a special bit pattern, commonly 01111110 (0x7E in hexadecimal). To prevent this bit pattern from appearing inside the actual data, the sender inserts a 0 bit after any sequence of five consecutive 1 bits in the payload. This extra 0 ensures that the special pattern never occurs within the frame data.

At the receiver side, the process is reversed. Upon detecting five 1s followed by a 0, the receiver removes (destuffs) the 0, restoring the original bit stream. If it sees five 1s followed by another 1, and then a 0, it recognizes this sequence as the frame delimiter. Like byte stuffing, bit stuffing is transparent to the network layer—it does not alter the original data

content but guarantees clear boundaries between frames. This method is not only effective for maintaining synchronization but also ensures that there are enough signal transitions to help the physical layer maintain clock synchronization during transmission.

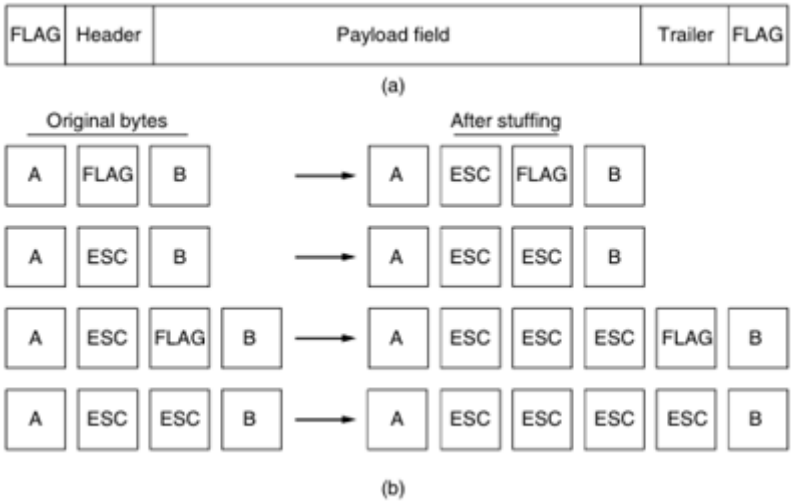


Figure 7.4 (a) A Frame delimited by flag bytes (b) Four examples of byte sequences before and after byte stuffing

7.2.4. Physical Layer Coding Violations

Another clever framing method leverages redundancy in the physical layer's signal encoding scheme. Many line coding schemes, such as 4B/5B encoding, translate blocks of 4 data bits into 5 signal bits. This creates unused or invalid signal combinations that don't map to valid data. These "coding violations" can be used to mark frame boundaries. When the receiver detects a signal that violates the standard encoding rules, it interprets this as the start or end of a frame.

The beauty of this method is that no additional bytes or bits are added to the data stream. The special codes used to delimit frames are reserved and never appear in the user data, which eliminates the need for byte or bit stuffing. This technique is highly efficient and works well when supported by the physical layer. Some protocols combine this approach with preambles or headers for extra robustness.

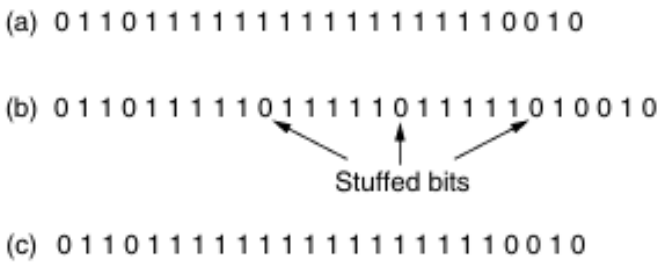


Figure 7.5 Bit Stuffing (a) The original data (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing

Framing in Real-World Protocols

In practice, real-world protocols often combine multiple framing techniques for improved reliability and synchronization. For instance, Ethernet and IEEE 802.11 (Wi-Fi) frames begin with a well-defined preamble—a special pattern (often 72 bits long in 802.11) that helps the receiver synchronize its clock with the incoming bit stream. This is followed by a length field that helps determine the end of the frame. These combined approaches ensure that frames are both easily detected and accurately interpreted, even in the presence of noise or data corruption.

7.3 ERROR CONTROL

Once the data link layer has successfully solved the problem of framing—that is, correctly identifying where each frame starts and ends—it faces another critical challenge: ensuring that all frames are reliably delivered to the destination's network layer and in the correct order. This is especially important in connection-oriented services, where reliability and sequencing are crucial. For unacknowledged connectionless service, reliability isn't guaranteed, and the sender simply transmits frames without expecting confirmation. However, such an approach is unacceptable when applications require dependable communication.

To provide reliable transmission, the data link layer uses feedback mechanisms that allow the receiver to communicate back to the sender. These feedback signals are typically in the form of control frames known as acknowledgements (ACKs) or negative acknowledgements (NAKs). When a frame is received correctly, the receiver sends a positive acknowledgement (ACK), letting the sender know the frame arrived safely. If an error is detected in the frame (e.g., through a failed checksum), a negative acknowledgement (NAK) may be sent, indicating the need for retransmission.

However, data transmission over networks is not always perfect. It is possible for a frame to completely vanish due to issues like noise bursts, signal loss, or hardware malfunctions. In such cases, the receiver won't send an acknowledgement because it never received the frame in the first place. Similarly, the acknowledgement frame itself could be lost while traveling back to the sender. If the protocol only relied on waiting for ACKs or NAKs, the sender would be stuck indefinitely, unsure of whether the frame was successfully received or not. This situation would cause the protocol to hang and data delivery to fail.

To handle such uncertainties, the data link layer employs a mechanism called a timer. Every time the sender transmits a frame, it simultaneously starts a timer. This timer is set for a duration that allows enough time for the frame to reach the receiver, be processed, and for the acknowledgement to return. If the timer expires before an acknowledgement is received, the sender assumes something went wrong—either the original frame or the acknowledgement was lost—and it proceeds to retransmit the frame.

While retransmissions solve the problem of lost frames, they introduce another issue: the receiver might mistakenly accept a duplicate frame. For instance, suppose the receiver gets a frame and sends an acknowledgement, but that acknowledgement is lost. The sender, unaware that the frame was actually received, resends the same frame. If the receiver accepts the duplicate as new, it would deliver the same data twice to the network layer, violating the principle of exactly-once delivery.

To address this problem, the data link layer uses sequence numbers. Each frame sent is assigned a unique sequence number, which the receiver can use to identify duplicates. If a frame with the same sequence number arrives more than once, the receiver knows it's a retransmission and can discard it rather than delivering it again. This technique helps maintain the integrity and correctness of the data flow.

In summary, the combined use of acknowledgements, timers, and sequence numbers enables the data link layer to ensure that each frame is delivered exactly once to the receiving network layer. This careful coordination helps prevent data loss, duplication, and reordering, which are essential requirements for reliable communication. Later sections of the chapter will explore how these mechanisms are implemented through various protocol designs, progressively building more robust and efficient systems for error control and flow regulation.

7.4 FLOW CONTROL

Another significant design issue that arises in the data link layer, and in higher layers as well, is the problem of flow control—specifically, how to prevent a fast sender from overwhelming a slower receiver. This mismatch can occur when, for example, a powerful computer or server is sending data to a relatively slower device such as a smartphone. Even if the connection is reliable and error-free, the receiver might not be able to process incoming frames quickly enough. As a result, it might lose frames simply because its processing speed or buffer capacity is insufficient to keep up with the transmission rate of the sender.

To handle this, flow control mechanisms are implemented to regulate the rate at which data is transmitted. There are two fundamental approaches to flow control. The first is called feedback-based flow control, where the receiver sends control information back to the sender, indicating either how much data it can currently handle or whether it is ready to receive more frames. This method allows for dynamic adjustment based on the real-time state of the receiver. The second approach is known as rate-based flow control, which does not rely on receiver feedback. Instead, the protocol imposes an intrinsic limit on the rate at which the sender can transmit data, regardless of the receiver’s state. This form of control is more common in the transport layer and less so in the data link layer.

In the context of the data link layer, the focus is primarily on feedback-based flow control. These systems are widely used because they are more adaptable to different transmission conditions and hardware capabilities. In many modern networks, especially those with high-speed hardware such as Network Interface Cards (NICs), the data link layer is designed to operate at “wire speed,” meaning that it can handle frames as quickly as they arrive. In such cases, frame loss due to overflow is unlikely to occur at the data link layer, and any residual flow control responsibilities are typically shifted to higher layers, such as the transport layer.

Despite this, flow control at the data link layer remains essential in many practical scenarios. Feedback-based schemes usually rely on protocol rules that determine when the sender is permitted to send the next frame. These rules can vary in complexity but generally involve the receiver explicitly or implicitly granting permission to the sender. For instance, during the setup of a communication session, the receiver might specify, “You are allowed to send me n frames at once, but after that, you must wait for my signal before sending more.” This approach helps avoid overwhelming the receiver and ensures smooth and efficient communication.

Overall, flow control is a critical feature in data link protocols, ensuring that data is delivered at a rate the receiver can handle without loss or congestion. By using feedback-based control, the data link layer adds a layer of robustness to the communication process, adapting dynamically to varying network conditions and maintaining reliable transmission even in the presence of performance disparities between sender and receiver.

7.5 SUMMARY

This chapter explains the data link layer, which ensures reliable communication between directly connected devices. It provides essential services to the network layer, such as framing, error control, and flow control. Framing divides the data stream into manageable units for transmission. Error control detects and corrects transmission errors, ensuring data integrity. Flow control manages the data transmission rate between sender and receiver to prevent buffer overflow. Together, these mechanisms enable smooth, reliable, and efficient data transfer across network links.

7.6 TECHNICAL TERMS

Data link layer, Frame, connected devices, flow control, error control

7.7 SELF ASSESSMENT QUESTIONS

Essay questions:

1. What is the main function of the data link layer?
2. List any two services provided by the data link layer to the network layer.
3. What is framing in the data link layer?
4. Define error control.
5. What is the purpose of flow control?

Short Questions:

1. Explain the services that the data link layer provides to the network layer.
2. Describe the process of framing and different framing methods.
3. Discuss error control techniques used at the data link layer.
4. Explain flow control and its role in reliable data transmission.
5. Discuss how the data link layer ensures reliable and efficient communication.

7.8 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008).

Dr. Neelima Guntupalli

LESSON- 8

ERROR DETECTION AND CORRECTION

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the need for error detection and correction in data transmission.
- Learn the concept of error-correcting codes.
- Understand error-detecting codes and their working principles.
- Differentiate between error detection and error correction methods.
- Recognize common techniques used for reliable communication

STRUCTURE OF THE LESSON:

8.1 INTRODUCTION

Communication channels, as discussed earlier, vary widely in their reliability. Some channels, such as optical fibers used in telecommunications, experience extremely low error rates, making transmission errors a rare occurrence. On the other hand, channels like wireless links and aging telephone lines are far more prone to errors, often suffering from significantly higher error rates. For these error-prone channels, errors cannot be completely avoided without incurring substantial costs or compromising performance. As a result, the reality of communication systems is that transmission errors are inevitable, and network designers must find ways to manage and mitigate them effectively.

To address this challenge, two primary strategies have been developed. Both involve adding redundant bits to the original data being transmitted. One strategy adds enough redundancy to

enable the receiver to infer the original message even if some bits are received in error. This approach is called error correction, and the codes used are known as error-correcting codes, or more formally, Forward Error Correction (FEC). The second strategy adds only enough redundancy to detect the presence of an error. If an error is detected, the receiver requests that the sender retransmit the data. These are called error-detecting codes. Each of these approaches is suitable in different scenarios depending on the nature of the channel and the expected error characteristics.

On highly reliable communication links, such as those using optical fiber, it is more efficient to use error-detecting codes and retransmit data only when necessary. However, in environments like wireless communication, where errors are frequent and retransmissions may also be affected by errors, FEC becomes a better choice. By enabling the receiver to correct errors without needing a retransmission, FEC improves performance in noisy conditions. Nevertheless, neither technique is foolproof. The additional redundant bits are subject to the same channel conditions as the original data, meaning they can also be corrupted. Therefore, the strength of the code must be carefully chosen based on the expected type and frequency of errors.

Two general models are used to describe how errors occur. One assumes that errors are random and isolated, caused by brief spikes in thermal noise that disrupt individual bits. The other assumes errors occur in bursts, often due to physical disturbances like deep fades in wireless signals or transient electrical interference in wired systems. Both error models have practical relevance and different implications. Interestingly, burst errors, despite being more difficult to correct, can sometimes be advantageous. For example, if data is transmitted in blocks of 1000 bits and the bit error rate is 0.001, random errors might affect nearly every block. But if errors occur in bursts of 100 bits, only one out of every 100 blocks would be affected, on average.

Other error types include cases where the system knows the location of an error but not its value. This scenario is called an erasure channel and occurs when a received signal deviates so far from what is expected that the system cannot determine whether the bit was a 0 or a 1. Erasures are easier to correct than general bit errors because the error's location is known, even if the actual value is not. However, most communication systems cannot rely on erasures being clearly identified and must treat all errors as unknown and unpredictable.

Error codes are primarily addressed in the data link layer because this is the first layer responsible for ensuring the reliable transmission of data packets. However, their usage is not limited to this layer. Error-correcting codes are also found in the physical layer, especially in channels with high noise levels, and in the higher layers, where they support real-time media and the distribution of digital content. Error-detecting codes, on the other hand, are widely used across the link, network, and transport layers to support error checking and reliable transmission.

Finally, it is important to recognize that error codes are rooted in applied mathematics, involving concepts such as Galois fields and matrix operations. Creating effective error codes requires a deep understanding of these mathematical principles. For this reason, most systems rely on well-established and thoroughly tested codes provided by standards bodies or developed by experts. Protocol standards tend to reuse proven codes across different applications, and it is recommended to adopt these established methods rather than attempting to design new codes from scratch. In practice, this approach leads to robust

systems capable of handling the wide variety of errors encountered in real-world communication networks.

8.2 ERROR CORRECTING CODES

Error detection and correction is a crucial aspect of reliable data transmission, especially over noisy or unreliable channels. To achieve this, various coding techniques are employed, all of which introduce redundancy into the data. This redundancy helps in detecting and correcting errors at the receiver's end. There are multiple types of codes used for this purpose, such as

1. Hamming codes.
2. Binary convolutional codes.
3. Reed-Solomon codes.
4. Low-Density Parity Check codes.

All of these codes follow a fundamental idea: a message or data block of m bits is extended with r check or redundant bits, resulting in a total block length of n = m + r bits. This block is referred to as a codeword. In *block codes*, these check bits are derived purely as a function of the data bits. A systematic code sends the original data bits along with the check bits, making it easy to extract the message without decoding. In *linear codes*, which are the focus of most practical systems, the check bits are computed using linear functions like XOR, which simplifies both encoding and decoding processes. The effectiveness of these codes is measured by the *Hamming distance*, which is the number of differing bits between two codewords. A code with a higher Hamming distance can detect and correct more errors. For example, to correct d errors, a code must have a Hamming distance of at least 2d + 1.

8.2.1 Hamming Codes

Hamming codes, one of the earliest and simplest forms of error-correcting codes, were designed to correct single-bit errors. They are systematic linear block codes that place check bits at positions that are powers of two in the codeword. These check bits are calculated such that certain groups of bits, including the check bit itself, have even parity. When a codeword is received, the parity of these groups is recalculated, and the results form a binary number known as the *syndrome*. If the syndrome is zero, no error has occurred; if not, it pinpoints the location of the erroneous bit, which can then be flipped to correct the error. For instance, in an (11,7) Hamming code, 7 data bits are sent along with 4 check bits. If a single-bit error occurs, the receiver can detect and correct it using the computed syndrome. However, Hamming codes are limited in their error correction capabilities as they can only correct single-bit errors and detect double-bit errors. This makes them suitable for applications where errors are relatively rare, such as error-correcting memory systems.

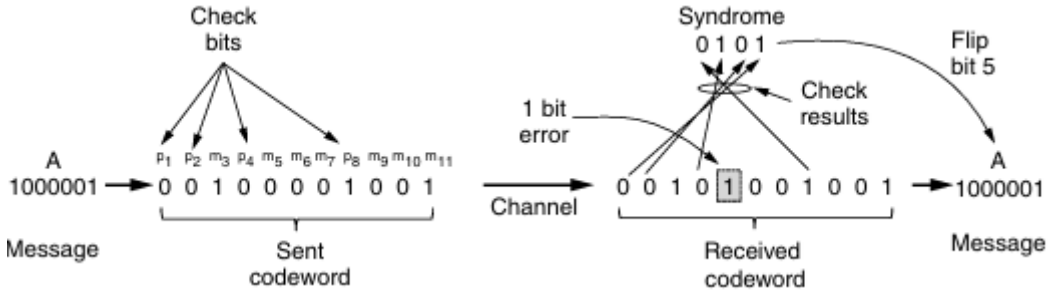


Figure 8.1 Example of an (11,7) Hamming code correcting a single-bit error

8.2.2 Binary Convolutional Codes

Convolutional codes differ significantly from block codes in that they do not work with fixed-size message blocks. Instead, they encode data as a continuous stream, with the output depending not only on the current input bit but also on several previous input bits, defined by the *constraint length*. This gives convolutional codes a memory-like behavior. A well-known example is the NASA convolutional code used in satellite communication and IEEE 802.11 wireless standards. In this system, for every input bit, two output bits are generated based on XOR operations involving the input bit and the internal memory state, leading to a code rate of 1/2. Because of their structure, convolutional codes are typically decoded using the Viterbi algorithm, which traces all possible paths the input could have taken and selects the most likely one by minimizing the total number of errors. This approach is efficient and effective for small constraint lengths and can even handle signal uncertainty through *soft-decision decoding*, where received signal levels are considered to estimate the most probable transmitted bits. Convolutional codes are especially effective at correcting isolated bit errors and are widely used in mobile and satellite communication systems.

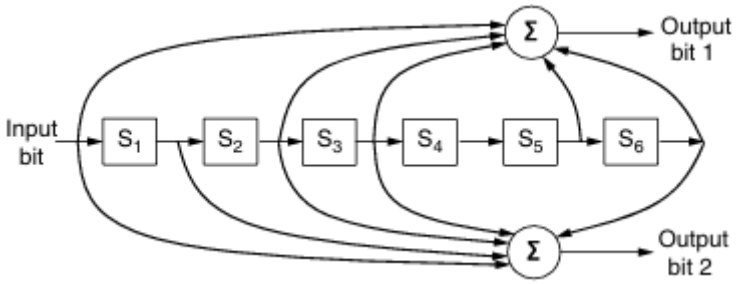


Figure 8.2 The NASA binary convolutional code used in 802.11

8.2.3 Reed-Solomon Codes

Reed-Solomon (RS) codes are another powerful family of error-correcting codes that are widely used in practical systems, particularly for correcting burst errors. Unlike Hamming codes, RS codes operate on symbols made up of multiple bits, typically bytes (8-bit symbols), and use polynomial mathematics over finite fields. They are systematic linear block codes and work by fitting polynomials to a number of data points. Additional check symbols are generated by evaluating the same polynomial at extra points. When errors occur during transmission, the original data can be recovered by identifying and discarding erroneous symbols and refitting the polynomial to the remaining valid points. A common configuration is the (255, 233) RS code, where 233 data symbols are extended with 22 check symbols, allowing the correction of up to 11 symbol errors. These codes are particularly effective against burst errors because a burst that affects several bits within a symbol still counts as a single symbol error. RS codes are extensively used in storage media like CDs and DVDs, DSL, and satellite communication. They are often combined with other codes, such as convolutional codes, to enhance error correction capabilities in different types of channels—handling both isolated and burst errors effectively.

8.2.4 Low-Density Parity Check (LDPC) Codes

Low-Density Parity Check (LDPC) codes are among the most advanced error-correcting codes in use today. Although proposed in the 1960s, they only became practical in the 1990s due to improvements in computing power. LDPC codes are linear block codes represented by sparse matrices, meaning that most of the entries in the parity-check matrix are zeros. This sparsity makes both encoding and decoding more efficient. LDPC decoding uses an iterative approximation algorithm, often called belief propagation, which incrementally refines its estimate of the transmitted message by comparing received data with possible valid codewords and updating beliefs about bit values based on inconsistencies. LDPC codes can achieve performance very close to the theoretical limits of error correction defined by Shannon's theorem, especially with large block sizes. They are now used in high-performance systems such as 10 Gbps Ethernet, digital television broadcasting (DVB), power-line communication systems, and newer versions of Wi-Fi (IEEE 802.11). Their excellent error correction capability and scalability make them a promising standard for the future of data communication systems.

8.3 ERROR DETECTING CODES

Error-correcting codes play a critical role in ensuring the reliability of data transmission, especially over unreliable communication channels like wireless links, where noise and interference can introduce frequent errors. These codes help detect and sometimes correct errors without the need for retransmission. However, in high-quality transmission media such as fiber optics or quality copper wires, where error rates are significantly lower, using error-detecting codes combined with retransmission mechanisms is often more efficient. In this context, three main types of linear, systematic block error-detecting codes are commonly used: parity bits, checksums, and Cyclic Redundancy Checks (CRCs).

Parity bits are the simplest form of error detection. A single bit is added to a block of data to make the total number of 1s either even (even parity) or odd (odd parity). This method can detect any single-bit error by verifying if the parity still holds after transmission. However, it cannot detect errors that affect two bits in such a way that the parity remains unchanged. Parity checking is very efficient in terms of overhead, especially when the bit error rate is low. For instance, adding just one bit per 1000-bit block results in minimal redundancy, and any detected error can be corrected by retransmitting the entire block. Nevertheless, its limitation lies in its inability to reliably detect multi-bit or burst errors, which are more complex and frequent in noisy environments.

To improve upon simple parity, more advanced strategies such as using a rectangular matrix of bits are employed. Here, parity bits are calculated for both rows and columns of the data block, enhancing the ability to detect multiple errors spread across different areas of the block. Even better results are achieved through a technique called *interleaving*, where the order of bits is rearranged during transmission, so a burst error affecting a continuous sequence of bits in the transmission gets spread across various rows and columns in the matrix. This method ensures that the likelihood of undetected burst errors is significantly reduced, as each column gets its own parity bit, and the burst error is unlikely to affect all columns simultaneously.

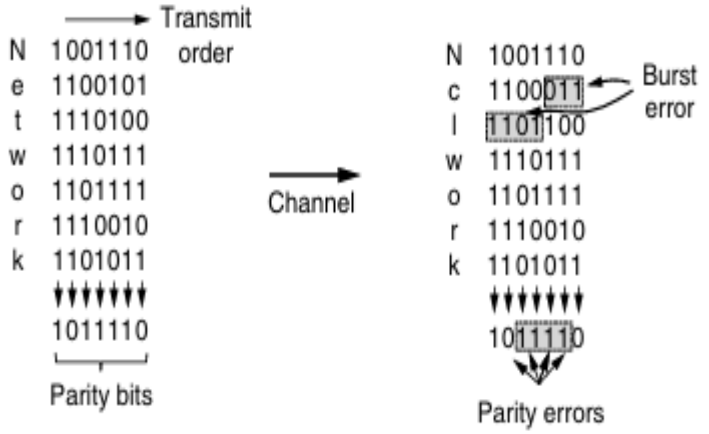


Figure 8.3 Interleaving of parity bits to detect a burst error

8.3.1 Checksums

Checksums represent a broader category of error-detecting techniques and are commonly found in higher-level protocols such as those used on the Internet. A checksum is typically computed as a running sum of the data words (not bits), and then its complement is appended to the message. When the receiver computes the sum of the received data plus the checksum, the result should be zero if no error occurred. Unlike parity checks, which work on individual bits, checksums are effective against errors that parity might miss, like two-bit errors occurring in different locations that cancel each other out. The Internet checksum is an example that operates on 16-bit words and uses one's complement arithmetic, which enhances error detection coverage and allows convenient representation of a "null checksum."

Despite their simplicity, checksums have limitations, especially in detecting certain structured errors, such as swapped sections of data or the insertion of zeroes. To overcome these weaknesses, more sophisticated checksum variants like Fletcher's checksum include a positional component, where each data word's position in the message is factored into the sum, significantly improving the error detection ability for certain types of errors.

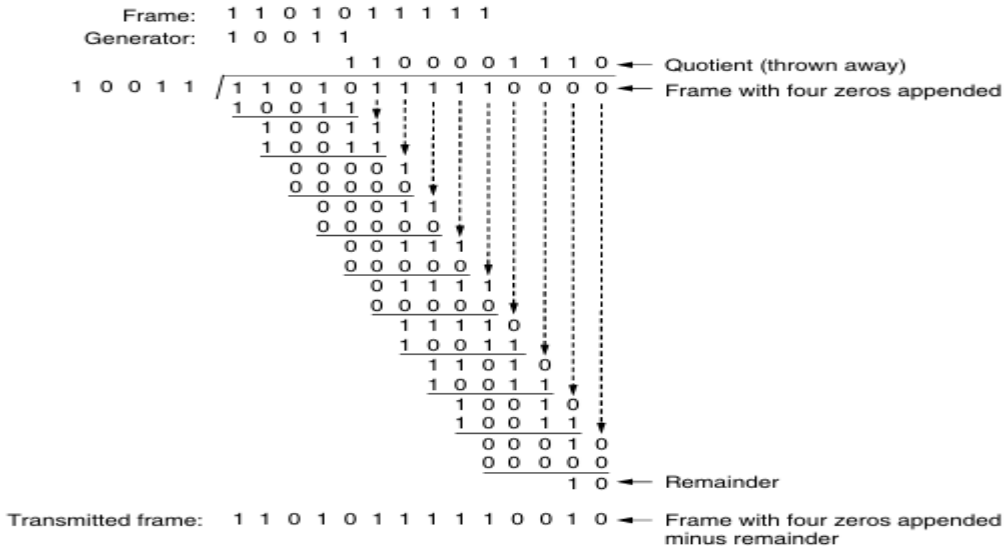


Figure 8.4 Example calculation of the CRC

8.3.2 CRCs

Cyclic Redundancy Checks, are among the most robust and widely used error-detecting codes at the data link layer. They treat the binary message as a polynomial, where each bit represents a coefficient (either 0 or 1). Using polynomial division in modulo-2 arithmetic, a remainder is computed by dividing the message polynomial, extended with zero bits, by a fixed generator polynomial agreed upon by both sender and receiver. This remainder is then appended to the original data. At the receiving end, the entire message (data plus remainder) is divided by the same generator polynomial. If the result is not divisible (i.e., the remainder is nonzero), an error has occurred.

CRCs can detect a wide variety of error types, including all single-bit errors, all double-bit errors (assuming appropriate generator polynomial selection), all odd numbers of bit errors, and all burst errors up to a certain length. For example, a generator polynomial with a degree of r will detect all burst errors of length $\leq r$, and will detect longer bursts with a high probability. Carefully chosen generator polynomials, such as the standard CRC-32 used in IEEE 802 networks, are designed to offer strong error-detecting properties for typical message sizes and transmission environments.

8.4 SUMMARY

This chapter explains how error detection and correction maintain data accuracy during transmission. Error-correcting codes not only detect but also fix errors at the receiver end, ensuring reliability without retransmission. Error-detecting codes, such as parity bits, checksums, and CRC, identify transmission errors so data can be resent. Together, these methods protect information integrity and enhance communication reliability across noisy or unreliable network channels.

8.5 TECHNICAL TERMS

Error detection, error correction, parity bit, check sum, CRC

8.6 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the concept of error detection and correction in data communication.
2. Describe different types of error-correcting codes with examples.
3. Discuss the working of error-detecting codes such as parity and checksum.
4. Compare error detection and error correction methods.
5. Explain how error control ensures reliable data transfer across networks.

Short Questions:

1. What is the purpose of error detection and correction?
2. Define an error-correcting code.
3. What is an error-detecting code?
4. Mention one technique used for error detection.
5. Mention one method used for error correction.

8.7 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008)

Dr. Neelima Guntupalli

LESSON- 9

ELMENTARY DATA LINK PROTOCOLS

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the concept of data link layer protocols.
- Learn about the unrestricted simplex protocol.
- Study the simplex stop-and-wait protocol.
- Understand the simplex protocol for a noisy channel.
- Compare different elementary data link protocols.

STRUCTURE OF THE LESSON:

9.1 INTRODUCTION

This section introduces the foundational model for data link layer protocols by assuming that the physical, data link, and network layers operate as independent processes that communicate through message passing. Typically, part of this functionality is handled by hardware (like the NIC), while the rest runs in software on the main CPU. The sender (machine A) is assumed to always have data ready to send to the receiver (machine B) over a reliable, connection-oriented service, and machines do not crash. A packet is passed from the network layer to the data link layer, which encapsulates it into a frame with a header and trailer before sending it across the physical layer. Key C structures and procedures are defined to handle events (e.g., frame arrival, timeout), manage timers, and ensure modular interaction between layers.

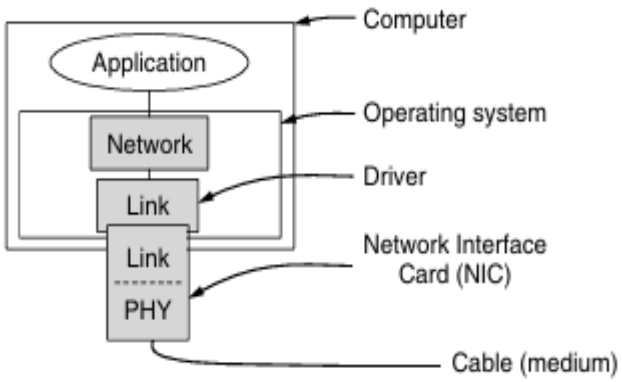


Figure 9.1 Implementation of the physical, data link and network layers

Importantly, the network layer is unaware of frame headers, allowing for clean separation and easier system updates or changes. This setup supports building and analyzing simple to complex data link protocols in a modular and event-driven way.

9.2 THE UTOPIAN SIMPLEX PROTOCOL

The "Utopia" protocol is a simple, one-way data link layer protocol designed to illustrate the basic structure of communication between sender and receiver, assuming perfect conditions. It assumes that the channel never loses or corrupts frames, both the sender and receiver are always ready, and there is infinite buffer space and processing speed. The sender continuously fetches packets from the network layer, encapsulates them into frames, and sends them out without any flow control or error checking. On the other side, the receiver waits for frame arrival, retrieves the frame from the physical layer, and passes its data to the network layer. Since no sequence numbers or acknowledgments are used, this protocol reflects an idealized model that lacks the robustness needed for real-world communication.

```
/* Protocol 1 (Utopia) provides for data transmission in one direction only, from
   sender to receiver. The communication channel is assumed to be error free
   and the receiver is assumed to be able to process all the input infinitely quickly.
   Consequently, the sender just sits in a loop pumping data out onto the line as
   fast as it can. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
  frame s;                              /* buffer for an outbound frame */
  packet buffer;                        /* buffer for an outbound packet */

  while (true) {
      from_network_layer(&buffer);      /* go get something to send */
      s.info = buffer;                  /* copy it into s for transmission */
      to_physical_layer(&s);            /* send it on its way */
  }                                     /* Tomorrow, and tomorrow, and tomorrow,
                                           Creeps in this petty pace from day to day
                                           To the last syllable of recorded time.
                                               – Macbeth, V, v */
}

void receiver1(void)
{
  frame r;
  event_type event;                     /* filled in by wait, but not used here */

  while (true) {
      wait_for_event(&event);           /* only possibility is frame_arrival */
      from_physical_layer(&r);          /* go get the inbound frame */
      to_network_layer(&r.info);        /* pass the data to the network layer */
  }
}
```

Figure 9.2 A utopian simplex protocol

The Utopia protocol is a basic and highly idealized example of a data link layer protocol used to illustrate core concepts without involving any complexity. In this protocol, data transmission is strictly one-way, from sender to receiver, with several unrealistic assumptions: both ends are always ready, there is no processing delay, infinite buffer space is available, and the channel is perfect, meaning no data loss or corruption ever occurs. As such, the protocol requires no acknowledgements or sequence numbers, and the only event the receiver handles is the arrival of a correct frame. The sender simply sends frames continuously, while the receiver immediately delivers each received frame to the network layer. Although completely impractical, Utopia serves as a foundation to understand how more realistic protocols build upon this simple structure.

The Utopia protocol represents the simplest form of a data link layer protocol, operating under ideal conditions with no errors, no lost frames, infinite buffer space, and immediate readiness of both the sender and receiver. In this protocol, the sender runs in an infinite loop where it continuously fetches packets from the network layer, constructs frames using only the data field (info), and sends them to the physical layer without any concern for acknowledgements, sequence numbers, or retransmissions. On the receiving end, the receiver also operates in an infinite loop, waiting for frames to arrive. Once a frame is received, it extracts the data from the frame and passes it to the network layer, then returns to waiting for the next frame. This protocol illustrates the basic structure of a data link layer protocol in a perfect environment, serving as a foundation for understanding more realistic protocols that handle errors and flow control.

9.3 A SIMPLEX STOP-AND-WAIT PROTOCOL FOR AN ERROR- FREE CHANNEL

To address the problem of a fast sender overwhelming a slower receiver, a flow control mechanism is introduced using a protocol called stop-and-wait. Unlike the earlier "Utopia" protocol, where the sender sent frames without restriction, this protocol ensures that the sender sends one frame and then waits for an acknowledgment from the receiver before sending the next. Although the data itself flows only from sender to receiver (simplex), acknowledgments travel back, requiring the communication channel to support bidirectional transfer. After processing a received frame and passing its data to the network layer, the receiver sends an acknowledgment frame to the sender. This handshake-like process ensures the receiver isn’t flooded and represents a simple yet effective form of flow control.

To prevent the sender from overwhelming the receiver with frames faster than it can handle—a common and important practical issue—the text considers a few strategies. Although the communication channel is assumed to be error-free and the data flow is simplex (one-way), problems still arise if the receiver cannot process incoming frames quickly enough. One naive solution is to build the receiver to be extremely fast and powerful, capable of handling a continuous stream of back-to-back frames without delay. This would require it to have ample buffering and high processing speeds, enabling it to pass received frames to the network layer as fast as they arrive. However, this approach is inefficient and unrealistic in many cases. It demands expensive, dedicated hardware and wastes resources when traffic load is low. Worse, it doesn’t truly solve the problem—it simply shifts the burden from the data link layer to the network layer. Thus, a more balanced and scalable solution is needed to manage flow control effectively.

To prevent the sender from overwhelming the receiver, a stop-and-wait protocol is used. In this method, after sending one frame, the sender waits for an acknowledgement from the receiver before sending the next frame. The receiver, after receiving and processing a frame, sends back a simple acknowledgement frame. This ensures the receiver can keep up with the sender, preventing buffer overflow. Even though data flows in one direction, acknowledgement frames flow back, so the channel must support two-way communication. This method provides basic flow control in an error-free communication environment.

9.4 A SIMPLEX STOP AND WAIT PROTOCOL FOR A NOISY CHANNEL

This section introduces the stop-and-wait ARQ (Automatic Repeat reQuest) protocol, which improves upon previous protocols by handling errors in transmission such as lost or damaged frames. In this protocol, the sender attaches a sequence number (0 or 1) to each frame. When a frame is received correctly, the receiver acknowledges it and updates its expectation to the next sequence number. If the frame is damaged or a duplicate (same sequence number as the previous one), it is discarded, and the last valid acknowledgement is resent. The sender starts a timer after sending a frame; if no acknowledgement is received before the timer expires, it retransmits the frame. This method ensures that frames are delivered reliably and exactly once, even over unreliable channels. Using a 1-bit sequence number suffices because the only ambiguity that needs to be resolved is between a frame and its immediate predecessor. This simple yet powerful mechanism ensures correctness while enabling basic error and flow control in one-way data transmission.

In real-world communication channels, errors such as damaged or lost frames can occur. Typically, damaged frames are detected using a checksum mechanism. If a frame is corrupted but still passes the checksum (rare but possible), it may lead to incorrect delivery of data. A seemingly simple solution is to enhance the stop-and-wait protocol by adding a timer: the sender waits for an acknowledgment (ACK), and if it doesn't arrive within a certain time, it resends the frame.

```
/* Protocol 2 (Stop-and-wait) also provides for a one-directional flow of data from
   sender to receiver. The communication channel is once again assumed to be error
   free, as in protocol 1. However, this time the receiver has only a finite buffer
   capacity and a finite processing speed, so the protocol must explicitly prevent
   the sender from flooding the receiver with data faster than it can be handled. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
  frame s;                              /* buffer for an outbound frame */
  packet buffer;                        /* buffer for an outbound packet */
  event_type event;                     /* frame_arrival is the only possibility */

  while (true) {
      from_network_layer(&buffer);      /* go get something to send */
      s.info = buffer;                  /* copy it into s for transmission */
      to_physical_layer(&s);            /* bye-bye little frame */
      wait_for_event(&event);           /* do not proceed until given the go ahead */
  }
}

void receiver2(void)
{
  frame r, s;                           /* buffers for frames */
  event_type event;                     /* frame_arrival is the only possibility */
  while (true) {
      wait_for_event(&event);           /* only possibility is frame_arrival */
      from_physical_layer(&r);          /* go get the inbound frame */
      to_network_layer(&r.info);        /* pass the data to the network layer */
      to_physical_layer(&s);            /* send a dummy frame to awaken sender */
  }
}
```

Figure 9.3 A simple stop-and-wait protocol

However, this approach has a critical flaw. Suppose the receiver gets a frame and processes it correctly, but the ACK is lost on the way back. The sender then retransmits the frame, thinking it was lost. The receiver, unaware it's a duplicate, processes and delivers it again to the network layer, resulting in duplicate data delivery.
This violates the core responsibility of the data link layer: to ensure error-free, in-order, non-duplicated delivery to the network layer. To fix this issue, frames must include sequence numbers so the receiver can detect duplicates and discard them, ensuring reliable and accurate communication.

Consider the following scenario:

1. Packet 1 is sent by machine A and correctly received by machine B. B then passes it to its network layer and sends an acknowledgement (ACK) back to A.
2. The ACK is lost in the communication channel. Since the channel is unreliable, it can drop control frames (ACKs) just like data frames.
3. After waiting for a while, A does not receive the ACK, so it assumes the packet was lost and resends packet 1.
4. Machine B, seeing a valid packet, assumes it's new and delivers it again to its network layer, causing duplicate delivery.

This is a protocol failure: B's network layer ends up with two copies of the same packet. If A was sending a file, B would get some part of the file twice, corrupting the output.

Solution:

To fix this, we need to identify duplicates, which is done by adding sequence numbers to the frames. This way, the receiver can detect that a frame it has already processed is being repeated and ignore it, thus preventing duplication and ensuring reliable, ordered delivery.
To ensure reliable communication over a channel that may lose or damage frames, the protocol needs a way for the receiver to tell whether an incoming frame is new or a retransmission of a previously received frame. Without this distinction, the receiver might mistakenly deliver duplicate packets to the network layer, causing errors such as repeated data. The common solution is for the sender to include a sequence number in each frame's header. This sequence number allows the receiver to identify each frame uniquely and decide whether to accept or discard it.

A key design question is how many bits the sequence number needs. Since the header size affects the efficiency of the protocol, the sequence number field should be as small as possible while still preventing errors. Interestingly, in a simple stop-and-wait protocol, only a 1-bit sequence number is necessary. This bit alternates between 0 and 1 for consecutive frames. At any time, the receiver expects a frame with a specific sequence number. When a frame with that number arrives, it accepts the frame, passes the data to the network layer, and sends an acknowledgement. The expected sequence number then flips to the other value (0 becomes 1, or 1 becomes 0).

The reason one bit suffices is that the only possible confusion occurs between a frame and its immediate predecessor or successor. The sender only sends the next frame after receiving an acknowledgement for the current one, ensuring no overlap beyond adjacent frames. If an acknowledgement is lost, the sender will retransmit the same frame, and the receiver will

recognize it as a duplicate by its sequence number and discard it. This simple use of sequence numbers helps the protocol provide reliable, error-free communication over an imperfect channel.

This protocol (called ARQ) ensures reliable data transfer by having the sender wait for an acknowledgement before sending the next frame. Both sender and receiver use sequence numbers to track frames and detect duplicates.

The sender starts a timer after sending a frame and waits for an acknowledgement. If the timer expires or a damaged acknowledgement arrives, the sender retransmits the frame. If a valid acknowledgement arrives, the sender moves on to the next frame.

The receiver accepts frames with the expected sequence number, delivers them to the network layer, and sends acknowledgements. Duplicate or damaged frames are ignored, but the receiver still sends acknowledgements to keep the sender informed. This process helps handle lost or damaged frames and acknowledgements, ensuring correct data delivery.

9.5 SUMMARY

This chapter introduces elementary data link protocols used for reliable communication between two directly connected devices. The unrestricted simplex protocol assumes an ideal channel with no errors or flow control needs. The simplex stop-and-wait protocol adds synchronization by allowing the sender to wait for an acknowledgment before sending the next frame. The simplex protocol for a noisy channel further enhances reliability by including error detection and retransmission mechanisms. These protocols form the foundation for understanding more advanced communication techniques.

9.6 TECHNICAL TERMS

Simplex protocol, Unrestricted simplex protocol, stop-and-wait protocol, frame,

9.7 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the working of an unrestricted simplex protocol.
2. Describe the operation of the simplex stop-and-wait protocol with an example.
3. Discuss the design and working of a simplex protocol for a noisy channel.
4. Compare unrestricted simplex, stop-and-wait, and noisy channel protocols.
5. Explain how these elementary protocols contribute to reliable data communication.

Short Questions:

1. What is a simplex communication channel?
2. Define an unrestricted simplex protocol.
3. What is the main idea of the stop-and-wait protocol?
4. How does a noisy channel affect data transmission?
5. What is the purpose of acknowledgments in data link protocols?

9.8 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008)

Dr. U. Surya Kameswari

LESSON- 10

SLIDING WINDOW PROTOCOLS AND STANDARD DATA LINK LAYER PROTOCOLS

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the concept of sliding window protocols for reliable data transfer.
- Learn the one-bit sliding window protocol.
- Study the Go-Back-N protocol and its working.
- Understand the selective repeat protocol.
- Examine example data link protocols such as HDLC and the Internet data link layer.

STRUCTURE OF THE LESSION:

10.1 INTRODUCTION

Earlier protocols only sent data in one direction, which is often impractical because most communications require two-way (full-duplex) data transfer. One simple solution is to run two separate one-way protocols—one for each direction—but this wastes the capacity of the reverse channel, which mainly carries acknowledgements.

A more efficient approach is to use the same communication link for data going both ways. Since acknowledgements and data frames both travel over the same link, the receiver distinguishes between them by checking the frame's header. This way, data and acknowledgements are interleaved on a single channel, making better use of the available capacity.

Interleaving data and control frames on the same link is an improvement, but it can be made even more efficient through piggybacking. Instead of sending a separate acknowledgement frame immediately when a data frame arrives, the receiver waits until it has its own data to

send back. Then it attaches the acknowledgement to that outgoing data frame in the header's ack field.

Piggybacking improves channel bandwidth use because the acknowledgement only adds a few bits to the data frame header, whereas a separate acknowledgement frame requires a full header, checksum, and processing. This reduces the number of frames sent and eases the receiver's processing load. Typically, the piggyback field is just 1 bit in size, making it a very lightweight and efficient method.

Piggybacking introduces a timing challenge: the data link layer must decide how long to wait for a new outgoing data packet to attach the acknowledgement to. If it waits too long—longer than the sender's timeout period—the sender will retransmit the frame unnecessarily, defeating the purpose of piggybacking.

Since the data link layer cannot predict when the next packet will arrive, it uses a practical approach like waiting a fixed short time. If a new packet comes in quickly, the acknowledgement is piggybacked onto it; otherwise, a separate acknowledgement frame is sent after the timeout.

Following this, the text introduces sliding window protocols, which are bidirectional and more efficient. These protocols use sequence numbers for frames, typically fitting into an n-bit field, allowing for multiple outstanding frames. The simplest sliding window protocol is the stop-and-wait with just one-bit sequence numbers (0 and 1), while more advanced ones use larger sequence numbers for greater efficiency.

Sliding window protocols work by having the sender keep track of a "sending window," which is a set of sequence numbers representing frames it is allowed to send. At the same time, the receiver keeps a "receiving window," representing the frames it is ready to accept. These windows control which frames can be sent and received at any given moment. The size and position of these windows can be fixed or dynamic, depending on the protocol.

Even with this flexibility, the protocols still ensure that packets reach the network layer in the correct order, maintaining a "wire-like" behaviour where frames arrive in the same sequence they were sent.

The sender's window includes frames that have either been sent but not yet acknowledged or are allowed to be sent. When a new packet arrives from the network layer, it is assigned the next sequence number, and the upper edge of the sending window moves forward. When the sender receives an acknowledgement, the lower edge of the window advances, effectively sliding the window forward. This way, the sender keeps track of unacknowledged frames.

Because frames within the sending window might get lost or damaged, the sender must keep copies of all these unacknowledged frames in memory so they can be retransmitted if needed. If the window reaches its maximum size (meaning all buffers are full), the sender must pause accepting new packets from the network layer until some acknowledgements free up buffer space.

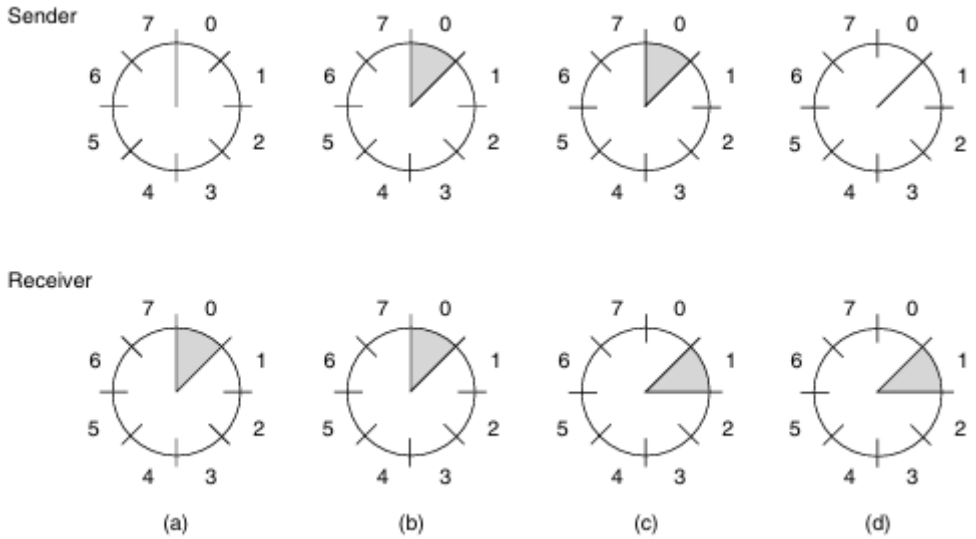


Figure 10.1 A sliding window of size 1, with a 3-bit sequence number (a) Initially, (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgment has been received

The receiver's data link layer maintains a "receiving window" that defines which frames it is allowed to accept. Any frame whose sequence number falls inside this window is stored in the receiver's buffer. When the receiver gets the frame that matches the lower edge (the next expected sequence number) of the window, it passes that frame up to the network layer and then slides the window forward by one position. Frames that arrive outside this window are discarded because they are either duplicates or out of order.

The receiver sends acknowledgements to inform the sender about which frames it has successfully received, helping the sender decide what to do next.

If the window size is 1, the receiver only accepts frames strictly in order. However, for larger window sizes, the receiver can accept out-of-order frames and buffer them until the missing frames arrive, ensuring the network layer always receives data in the correct order.

Unlike the sender's window, which can grow and shrink, the receiver's window size remains fixed but "rotates" as frames are accepted and delivered to the network layer.

This code implements Protocol 3, a simple stop-and-wait Automatic Repeat reQuest (ARQ) protocol for reliable unidirectional data transmission over an unreliable channel. The sender maintains a sequence number (either 0 or 1) for each frame it sends and waits for a positive acknowledgement before sending the next frame. When the sender transmits a frame, it starts a timer and waits for an acknowledgement. If the acknowledgement matches the sequence number of the sent frame, the sender stops the timer, increments the sequence number, fetches the next packet from the network layer, and sends the next frame. If no valid acknowledgement arrives before the timer expires, the sender retransmits the same frame. On the receiving side, the receiver keeps track of the expected sequence number. When it receives a frame with the expected sequence number, it delivers the data to the network layer and sends back an acknowledgement indicating the last correctly received frame. Frames with unexpected sequence numbers are discarded as duplicates. This protocol ensures reliable and ordered delivery of data frames despite possible loss or corruption in the transmission

channel, using only a single-bit sequence number to differentiate between new frames and retransmissions.

```
/* Protocol 3 (PAR) allows unidirectional data flow over an unreliable channel. */

#define MAX_SEQ 1                                   /* must be 1 for protocol 3 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
  seq_nr next_frame_to_send;                        /* seq number of next outgoing frame */
  frame s;                                          /* scratch variable */
  packet buffer;                                    /* buffer for an outbound packet */
  event_type event;

  next_frame_to_send = 0;                           /* initialize outbound sequence numbers */
  from_network_layer(&buffer);                      /* fetch first packet */
  while (true) {
      s.info = buffer;                              /* construct a frame for transmission */
      s.seq = next_frame_to_send;                   /* insert sequence number in frame */
      to_physical_layer(&s);                        /* send it on its way */
      start_timer(s.seq);                           /* if answer takes too long, time out */
      wait_for_event(&event);                       /* frame_arrival, cksum_err, timeout */
      if (event == frame_arrival) {
          from_physical_layer(&s);                  /* get the acknowledgement */
          if (s.ack == next_frame_to_send) {
              stop_timer(s.ack);                    /* turn the timer off */
              from_network_layer(&buffer);          /* get the next one to send */
              inc(next_frame_to_send);              /* invert next_frame_to_send */
          }
      }
  }
}

void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
      wait_for_event(&event);                       /* possibilities: frame_arrival, cksum_err */
      if (event == frame_arrival) {                 /* a valid frame has arrived */
          from_physical_layer(&r);                  /* go get the newly arrived frame */
          if (r.seq == frame_expected) {            /* this is what we have been waiting for */
              to_network_layer(&r.info);            /* pass the data to the network layer */
              inc(frame_expected);                  /* next time expect the other sequence nr */
          }
          s.ack = 1 - frame_expected;               /* tell which frame is being acked */
          to_physical_layer(&s);                    /* send acknowledgement */
      }
  }
}
```

Figure 3-14. A positive acknowledgement with retransmission protocol.

Figure 10.2 A positive acknowledgement with retransmission protocol

10.2 A ONE BIT SLIDING WINDOWS PROTOCOL

This passage explains a sliding window protocol with a window size of 1, which essentially operates as a stop-and-wait protocol. In this setup, the sender transmits one frame and then waits for its acknowledgement before sending the next frame. At the start, only one side initiates transmission by sending the first frame outside the main loop. The receiver checks incoming frames to detect duplicates; if the frame is the expected one, it passes the data to the network layer and slides its receiving window forward. The acknowledgement sent back

contains the sequence number of the last correctly received frame. The sender uses this acknowledgement to decide whether to move on to the next packet or to retransmit the same frame if the acknowledgement does not match. Whenever a frame is received, an acknowledgement frame is also sent in return.

The passage then discusses potential issues with this protocol in a two-way communication scenario (Protocol 4). For example, if both computers A and B simultaneously send frame 0 to each other, and A's timeout is too short, A might repeatedly resend the same frame before receiving a valid acknowledgement. The receiver (B) will accept the first frame, update its expected sequence number, and reject all subsequent duplicate frames. B will keep sending acknowledgements for the last correctly received frame until A eventually gets the correct acknowledgement and moves on. This ensures the protocol is reliable—it prevents duplicates, dropped packets, or deadlocks.

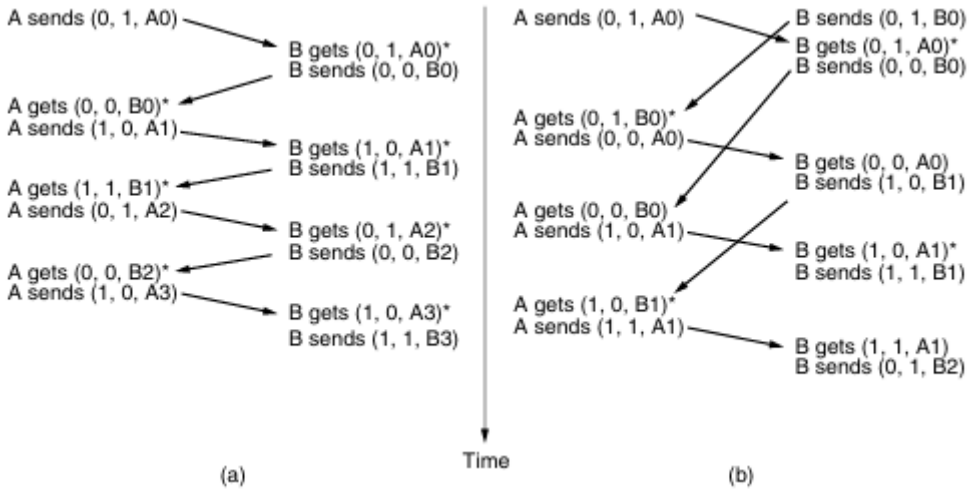


Figure 10.3 Two scenarios for protocol 4. (a) Normal case. (b) Abnormal case. The notation is (seq.ack, packget number). An asterisk indicates where a network layer accepts a packet.

However, the text also highlights a subtle synchronization problem when both sides start transmitting at exactly the same time. Their frames may cross paths, causing duplicate frames to appear even though there are no transmission errors. This can waste bandwidth due to multiple retransmissions triggered by premature timeouts, showing how timing and protocol interactions can complicate even seemingly straightforward communication processes.

This code snippet illustrates Protocol 4, a simple 1-bit sliding window protocol used for bidirectional communication. It works with sequence numbers limited to 0 and 1, which keeps track of the frames being sent and received.

- Variables:
  - next frame to send indicates the sequence number of the next frame the sender wants to transmit.
  - frame expected indicates the sequence number the receiver is waiting for. Both are either 0 or 1.

- Sending:

Initially, the sender fetches a packet from the network layer, creates a frame with the current next frame to send sequence number, and includes an acknowledgement (ack) for the last correctly received frame from the other side. It then sends the frame and starts a timer.

```
/* Protocol 4 (Sliding window) is bidirectional. */

#define MAX_SEQ 1                                   /* must be 1 for protocol 4 */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"
void protocol4 (void)
{
  seq_nr next_frame_to_send;                        /* 0 or 1 only */
  seq_nr frame_expected;                            /* 0 or 1 only */
  frame r, s;                                       /* scratch variables */
  packet buffer;                                    /* current packet being sent */
  event_type event;
  next_frame_to_send = 0;                           /* next frame on the outbound stream */
  frame_expected = 0;                               /* frame expected next */
  from_network_layer(&buffer);                      /* fetch a packet from the network layer */
  s.info = buffer;                                  /* prepare to send the initial frame */
  s.seq = next_frame_to_send;                       /* insert sequence number into frame */
  s.ack = 1 - frame_expected;                       /* piggybacked ack */
  to_physical_layer(&s);                            /* transmit the frame */
  start_timer(s.seq);                               /* start the timer running */
  while (true) {
      wait_for_event(&event);                       /* frame_arrival, cksum_err, or timeout */
      if (event == frame_arrival) {                 /* a frame has arrived undamaged */
          from_physical_layer(&r);                  /* go get it */
          if (r.seq == frame_expected) {            /* handle inbound frame stream */
              to_network_layer(&r.info);            /* pass packet to network layer */
              inc(frame_expected);                  /* invert seq number expected next */
          }
          if (r.ack == next_frame_to_send) {        /* handle outbound frame stream */
              stop_timer(r.ack);                    /* turn the timer off */
              from_network_layer(&buffer);          /* fetch new pkt from network layer */
              inc(next_frame_to_send);              /* invert sender's sequence number */
          }
      }
      s.info = buffer;                              /* construct outbound frame */
      s.seq = next_frame_to_send;                   /* insert sequence number into it */
      s.ack = 1 - frame_expected;                   /* seq number of last received frame */
      to_physical_layer(&s);                        /* transmit a frame */
      start_timer(s.seq);                           /* start the timer running */
  }
}
```

Figure 10.4 A 1-bit sliding window protocol

- Receiving:
  When a frame arrives without errors, the receiver checks if its sequence number matches frame expected. If yes, it delivers the data to the network layer and
- increments frame expected. It also checks the acknowledgement field in the received frame; if the ack matches next frame to send, it stops the timer and fetches
- a new packet to send, incrementing the sequence number.
- Bidirectional flow:
  Each frame carries both data and an acknowledgement (piggybacking), so communication happens simultaneously in both directions on the same link.

- Timer:

The timer ensures retransmission if an acknowledgement is not received in time, handling potential frame loss or errors.

In summary, this protocol maintains simple flow control and reliable delivery by sending frames one at a time, acknowledging received frames, and resending if necessary — all while supporting simultaneous two-way communication.

10.3 A PROTOCOL USING GO-BACK-N

This passage discusses a major limitation of stop-and-wait protocols—their inefficiency over high-delay links, such as satellite channels. The key problem arises when the round-trip time (RTT) is significant compared to the transmission time of a single frame. For example, consider sending 1000-bit frames over a 50-kbps satellite link with a 500-millisecond round-trip delay. When the sender transmits a frame, it finishes sending in 20 milliseconds, but the receiver doesn't get it until 270 ms, and the acknowledgement returns only after 520 ms. This means that the sender is blocked (i.e., waiting idle) for 500 out of every 520 milliseconds—wasting 96% of the available bandwidth.

The root cause is the stop-and-wait approach, where the sender waits for an acknowledgement before sending the next frame. To fix this, the protocol must allow the sender to transmit multiple frames before needing to stop. This is achieved using sliding window protocols, where the sender can transmit up to w frames before blocking.

To choose an efficient value for w, we calculate the bandwidth-delay product (BD)—the number of bits that can "fit" in the channel at one time. BD is computed as
BD = bandwidth (in bits/sec) × one-way propagation delay (in sec)

Dividing BD by the number of bits per frame gives the number of frames that can be in transit. The optimal window size is then:
$w = 2 \times BD + 1$
The $2 \times BD$ accounts for round-trip propagation, and the +1 ensures the sender always has something to do while waiting for acknowledgements. This approach dramatically improves bandwidth utilization, especially on long-delay or high-speed links.

The technique of pipelining, where multiple frames are sent without waiting for individual acknowledgements, greatly improves efficiency but also introduces complex reliability challenges over unreliable communication channels. A key issue arises when a frame in the middle of the transmission is lost or damaged. Because the sender continues to transmit additional frames before receiving feedback, the receiver might get many correct frames after the lost one.

However, the receiver cannot deliver these subsequent frames to the network layer immediately, even if they are error-free. This is because frames must be delivered in order, and the one that was lost or damaged creates a gap in the sequence. Therefore, the receiver is forced to discard or buffer the correct frames that follow until the missing one is properly retransmitted and received.

This situation emphasizes the complexity of pipelined communication—it requires sophisticated mechanisms for error detection, acknowledgement, retransmission, and buffer

management, all while ensuring that packet order is preserved for delivery to higher layers. These challenges are addressed by advanced sliding window protocols such as Go-Back-N and Selective Repeat, which differ in how they handle errors and out-of-order frames.

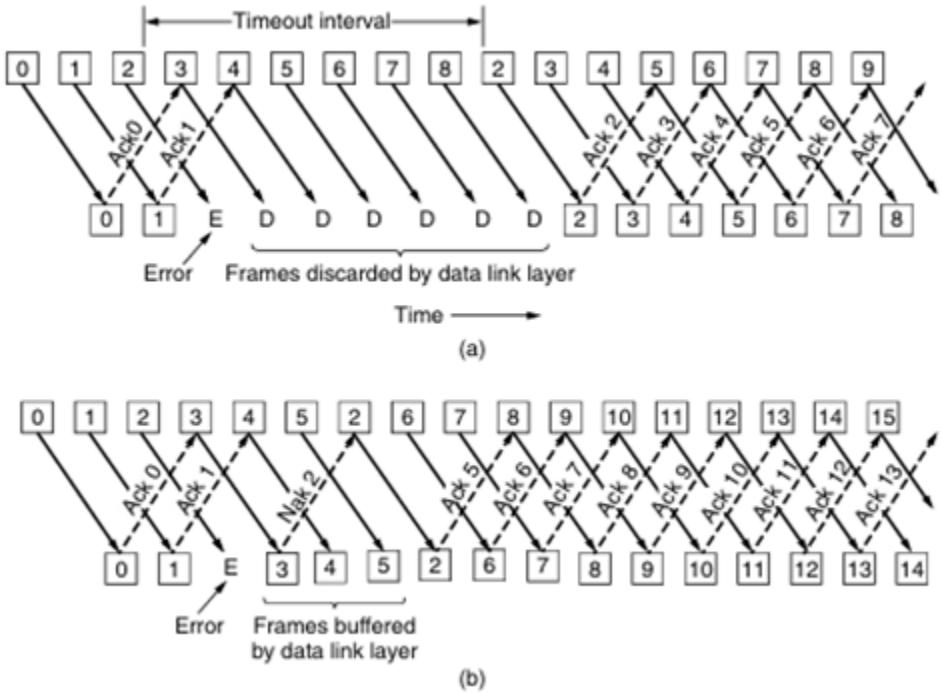


Figure 10.5 Pipelining and error recovery. Effect of an error when (a) receiver's window size is 1 and (b) receiver's window size is large

The diagram discusses two major strategies for handling errors in sliding window protocols when multiple frames are in transit, known as Go-Back-N and Selective Repeat.

In the Go-Back-N strategy, the receiver only accepts frames in order, effectively maintaining a receive window of size 1. If a frame is lost or damaged, the receiver discards that frame and all subsequent ones until the missing frame is correctly received. It doesn't send acknowledgements for the discarded frames, causing the sender to eventually time out and retransmit all unacknowledged frames starting from the lost one. This approach is simple but can be inefficient, especially when the error rate is high, as it wastes bandwidth by resending frames that were already received correctly but not acknowledged due to one earlier error.

In contrast, the Selective Repeat protocol uses a larger receive window and is more efficient. If a frame is lost or damaged, the receiver discards just that frame but buffers all subsequent correct frames. It sends a Negative Acknowledgement (NAK) for the missing frame, prompting the sender to retransmit only the missing frame. Once it is received, the buffered frames can be delivered to the network layer in order. This approach makes better use of bandwidth but requires more memory at the receiver to store out-of-order frames.

The choice between Go-Back-N and Selective Repeat involves a trade-off between bandwidth efficiency and memory usage. Go-Back-N is simpler and uses less memory but can lead to more retransmissions, while Selective Repeat is more complex and memory-intensive but reduces unnecessary retransmissions. Additionally, mechanisms like NAK and flow control (e.g., using enable_network_layer () and disable_network_layer () functions)

help manage when the sender can send new packets, ensuring the protocol doesn't overwhelm the receiver or violate window limits.

Protocol 5 introduces a more efficient approach to sliding window communication by allowing multiple frames to be in transit at once, but it still faces the challenge of buffer management. Although it doesn't require the receiver to buffer out-of-order frames (as in Selective Repeat), the sender must retain copies of all unacknowledged frames because they may need to be retransmitted if an error occurs. A key feature of this protocol is its use of cumulative acknowledgements, where receiving an ACK for frame n implies that all frames up to n have also been successfully received. This is particularly useful when some ACKs have been lost or corrupted — the receipt of any later ACK can still trigger the release of earlier frames from the sender's buffer. This helps free up space in the sender's window, allowing the network layer to send more packets.

```
/* Protocol 5 (Go-back-n) allows multiple outstanding frames. The sender may transmit up
   to MAX_SEQ frames without waiting for an ack. In addition, unlike in the previous
   protocols, the network layer is not assumed to have a new packet all the time. Instead,
   the network layer causes a network_layer_ready event when there is a packet to send. */

#define MAX_SEQ 7
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Return true if a <= b < c circularly; false otherwise. */
  if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
       return(true);
    else
       return(false);
}

static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[ ])
{
/* Construct and send a data frame. */
  frame s;                                    /* scratch variable */

  s.info = buffer[frame_nr];                  /* insert packet into frame */
  s.seq = frame_nr;                           /* insert sequence number into frame */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);/* piggyback ack */
  to_physical_layer(&s);                      /* transmit the frame */
  start_timer(frame_nr);                      /* start the timer running */
}

void protocol5(void)
{
  seq_nr next_frame_to_send;                  /* MAX_SEQ > 1; used for outbound stream */
  seq_nr ack_expected;                        /* oldest frame as yet unacknowledged */
  seq_nr frame_expected;                      /* next frame expected on inbound stream */
  frame r;                                    /* scratch variable */
  packet buffer[MAX_SEQ + 1];                 /* buffers for the outbound stream */
  seq_nr nbuffered;                           /* number of output buffers currently in use */
  seq_nr i;                                   /* used to index into the buffer array */
  event_type event;

  enable_network_layer();                     /* allow network_layer_ready events */
  ack_expected = 0;                           /* next ack expected inbound */
  next_frame_to_send = 0;                     /* next frame going out */
  frame_expected = 0;                         /* number of frame expected inbound */
  nbuffered = 0;                              /* initially no packets are buffered */

  while (true) {
    wait_for_event(&event);                   /* four possibilities: see event_type above */
```

```
switch(event) {
  case network_layer_ready:                 /* the network layer has a packet to send */
       /* Accept, save, and transmit a new frame. */
       from_network_layer(&buffer[next_frame_to_send]); /* fetch new packet */
       nbuffered = nbuffered + 1;           /* expand the sender's window */
       send_data(next_frame_to_send, frame_expected, buffer);/* transmit the frame */
       inc(next_frame_to_send);             /* advance sender's upper window edge */
       break;

  case frame_arrival:                       /* a data or control frame has arrived */
       from_physical_layer(&r);             /* get incoming frame from physical layer */

       if (r.seq == frame_expected) {
            /* Frames are accepted only in order. */
            to_network_layer(&r.info);      /* pass packet to network layer */
            inc(frame_expected);            /* advance lower edge of receiver's window */
       }

       /* Ack n implies n - 1, n - 2, etc.  Check for this. */
       while (between(ack_expected, r.ack, next_frame_to_send)) {
            /* Handle piggybacked ack. */
            nbuffered = nbuffered - 1;      /* one frame fewer buffered */
            stop_timer(ack_expected);       /* frame arrived intact; stop timer */
            inc(ack_expected);              /* contract sender's window */
       }
       break;

  case cksum_err: break;                    /* just ignore bad frames */

  case timeout:                             /* trouble; retransmit all outstanding frames */
       next_frame_to_send = ack_expected;   /* start retransmitting here */
       for (i = 1; i <= nbuffered; i++) {
            send_data(next_frame_to_send, frame_expected, buffer);/* resend frame */
            inc(next_frame_to_send);        /* prepare to send the next one */
       }

  }

  if (nbuffered < MAX_SEQ)
       enable_network_layer();
  else
       disable_network_layer();
}
```

Figure 10.6 A sliding window protocol using go-back-n

Unlike Protocol 4, which relies on a frame being sent in response to each frame received, Protocol 5 assumes there is always reverse traffic (traffic in the opposite direction) to piggyback acknowledgements. This assumption improves efficiency but might not hold in one-way communication scenarios, which will be addressed in the next protocol.

Since multiple frames can be outstanding, Protocol 5 requires a way to track timeouts individually for each one. Instead of needing a separate hardware timer for each frame, this can be efficiently simulated in software using a single hardware clock and a linked list of pending timeouts. Each node in the list holds the time remaining for the frame, its sequence number, and a pointer to the next node, allowing the system to manage retransmissions effectively with minimal hardware resources.

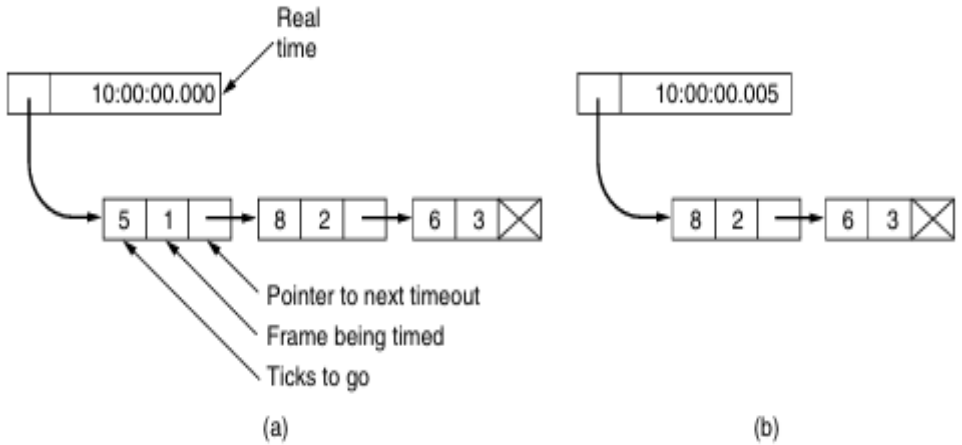


Figure 10.7 Simulation of multiple timers in software (a) The queued timeouts. (b) The situation after the first timeout has expired.

In Protocol 5, each frame needs its own timer since multiple frames can be sent before getting acknowledgements. Instead of using separate hardware timers for each frame, the system uses one hardware clock and manages all timers in a software list. Each timer is stored as a node in a list with the time left, the frame number, and a pointer to the next timer.

Every time the clock ticks (e.g., every 1 millisecond), the timer at the head of the list is decreased. When it reaches zero, that frame times out, and its timer is removed. This method saves resources and is efficient because only the first timer is updated on every tick, and new timers can be added or removed by scanning the list when needed.

10.4 A PROTOCOL USING SELECTIVE REPEAT

The Go-Back-N protocol is efficient when transmission errors are rare but wastes bandwidth on retransmissions over unreliable links. The Selective Repeat protocol overcomes this limitation by allowing the receiver to accept and buffer correctly received frames that follow a lost or damaged one. Both sender and receiver maintain windows: the sender's window grows dynamically up to a predefined maximum, while the receiver's window remains fixed. Each receiver buffer has an arrived flag indicating whether the slot is full. When a frame with a sequence number within the window arrives and has not been received before, it is stored but not delivered to the network layer until all preceding frames are received in sequence.

Allowing out-of-order reception introduces tighter constraints on sequence numbering. For example, with a 3-bit sequence number, the sender can transmit seven frames (0–6) before awaiting acknowledgements. If all arrive correctly, the receiver shifts its window to accept frames 7–5. If all acknowledgements are lost, the sender retransmits frame 0, which now falls within the receiver's new window and is incorrectly accepted as new, causing the network layer to receive a duplicate packet.

This ambiguity arises because the receiver's new window overlaps the previous one, making it impossible to distinguish old retransmissions from new data. To avoid overlap, the maximum window size must be at most half the sequence number range. With 3-bit numbering (0–7), the sender may have only four outstanding frames. Thus, when frames 0–3 are acknowledged, frames 4–7 are the next valid set. In general, the window size is limited to (MAX SEQ + 1)/2.

The receiver requires a number of buffers equal to the window size. Frames outside the window are discarded. For a 3-bit system, four buffers (0–3) are sufficient, with frame *i* stored in buffer *i mod 4*. Although *i* and *(i + 4) mod 4* share the same buffer, they never coexist within the window. The number of timers equals the number of buffers, as each buffer is associated with a retransmission timer.

Protocol 6 removes the dependence on heavy reverse traffic assumed in Go-Back-N. When reverse traffic is sparse, acknowledgements may be delayed, blocking transmission. An auxiliary acknowledgement timer ensures progress by sending standalone ACKs if no reverse data frames appear before timeout. This timer's duration must be significantly shorter than the data-frame timeout to prevent unnecessary retransmissions.

Selective Repeat also employs Negative Acknowledgements (NAKs) for faster error recovery. A NAK is sent when a damaged or unexpected frame is received, requesting retransmission of the missing frame. The variable *no_nak* prevents duplicate NAKs for the same sequence number. Lost NAKs are harmless since the sender's timeout eventually triggers retransmission. If a NAK is lost and an out-of-order frame arrives, the auxiliary ACK timer resynchronizes sender and receiver.

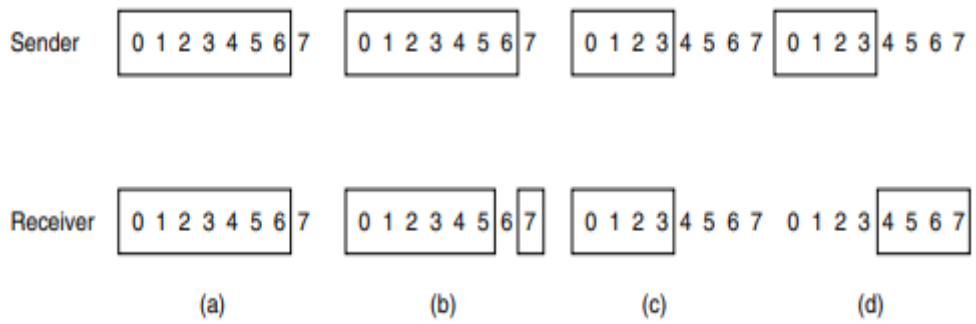


Figure 10.8 (a) Initial situation with a window of size . (b) After 7 frames have been sent and received but not acknowledged (c) Initial situation with a window size of 4. (d) After 4 frames have been sent and received but not acknowledged.

Timer configuration depends on delay variability. When round-trip times are nearly constant, the sender's timer can be tightly set. In channels with high delay variance, the timer must be loose to avoid false retransmissions, though this can increase idle time. In such environments, NAKs accelerate recovery, improving channel utilization.

Identifying which frame caused a timeout is complex since multiple frames may be outstanding. Unlike Go-Back-N, where the oldest frame always times out first, Selective Repeat maintains independent timers. If frames 0–4 are sent and 0, 1, 2 time out while new frames 5 and 6 are transmitted, the outstanding list may be 3405126. Each timer ensures that

retransmissions occur only for the specific frame whose timeout expires, maintaining correctness and efficiency.

10.5 HDLC (High-Level Data Link Control)

The High-Level Data Link Control (HDLC) protocol and its derivatives—SDLC, ADCCP, LAP, and LAPB—form a family of bit-oriented protocols derived from IBM's Synchronous Data Link Control (SDLC). IBM submitted SDLC to ANSI and ISO, resulting in ADCCP and HDLC, respectively. Later, CCITT adapted HDLC for LAP and LAPB within the X.25 standard. Though differing in details, these protocols share common principles: they are bit-oriented, employ bit stuffing for transparency, and use similar frame formats (Fig. 10.9).

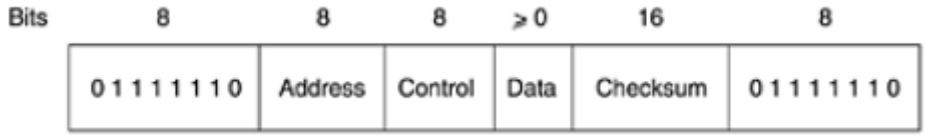


Figure 10.9 Frame format for a bit oriented protocols

All bit-oriented protocols use a standardized frame structure consisting of an Address, Control, Data, and Checksum field, delimited by flag sequences (01111110). The Address field identifies the terminal on multi-drop lines, while in point-to-point links it may distinguish commands from responses. The Control field carries sequence numbers, acknowledgements, and control information. The Data field can be of variable length, and the Checksum field employs Cyclic Redundancy Check (CRC). On idle lines, flag sequences are transmitted continuously. The minimum frame consists of 32 bits excluding flags.

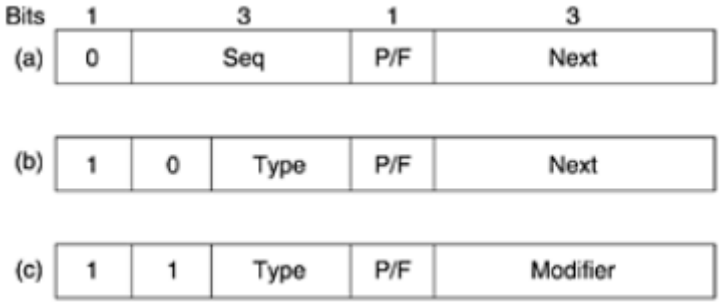


Figure 10.10 Control field of (a) an information frame, (b) a supervisory frame, (c) an unnumbered frame

There are three types of frames: Information, Supervisory, and Unnumbered (Fig. 10.10). The protocol uses a sliding window with 3-bit sequence numbers, allowing up to seven unacknowledged frames. In Information frames, Seq specifies the frame number and Next indicates the next expected frame (i.e., the first not yet received). The P/F bit (Poll/Final) is used for polling terminals or forcing the peer to send a Supervisory frame immediately.

Supervisory frames are of four types:

Type 0 (Receive Ready) – acknowledges all frames up to Next and indicates readiness to receive.

Type 1 (Reject) – requests retransmission starting from Next, similar to Go-Back-N.

Type 2 (Receive Not Ready) – acknowledges all frames up to Next but instructs the sender to pause, used during temporary receiver congestion.

Type 3 (Selective Reject) – requests retransmission of a specific frame, similar to Selective Repeat. This is supported in HDLC and ADCCP but not in SDLC and LAPB.

The Unnumbered frame type serves for control functions and sometimes for connectionless data transmission. Within this class, several control commands are defined. DISC (Disconnect) allows a device to terminate a session, while SNRM (Set Normal Response Mode) initializes a master-slave link. To support peer-to-peer operation, HDLC and LAPB include SABM (Set Asynchronous Balanced Mode), which establishes a symmetric relationship. SNRME and SABME are extended versions using 7-bit sequence numbers.

The FRMR (Frame Reject) command reports syntactically valid but semantically invalid frames, such as illegal control codes or out-of-window acknowledgements. The FRMR frame includes 24 bits of diagnostic information, including the faulty control field and status flags. Since control frames can also be lost or corrupted, they require acknowledgement through UA (Unnumbered Acknowledgement) frames. Only one control frame can be outstanding at a time, avoiding ambiguity. Other unnumbered control frames support initialization, polling, and link status reporting. The UI (Unnumbered Information) frame carries control information for the data link layer itself, not for higher layers.

Despite its maturity and widespread use, HDLC has known limitations and complexities in implementation and error handling (Fiorini et al., 1994). Nevertheless, it remains the foundation for many modern data link control procedures and network interface standards.

10.6 SUMMARY

This chapter explains sliding window protocols, which improve efficiency and reliability in data transfer by allowing multiple frames to be sent before waiting for acknowledgments. The one-bit sliding window protocol is the simplest form, suitable for reliable channels. Go-Back-N allows multiple outstanding frames but retransmits all frames after an error. Selective Repeat retransmits only erroneous frames, improving efficiency.

10.7 TECHNICAL TERMS

Sliding window protocol, go-back-N protocol, HDLC

10.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the operation of the one-bit sliding window protocol.
2. Describe the Go-Back-N protocol with an example.
3. Explain the selective repeat protocol and its advantages over Go-Back-N.
4. Discuss HDLC as an example of a sliding window protocol.
5. Describe how the data link layer in the Internet implements reliable communication using these protocols.

Short Questions:

1. What is a sliding window protocol?
2. Define the one-bit sliding window protocol.
3. What is the key idea behind the Go-Back-N protocol?
4. How does the selective repeat protocol differ from Go-Back-N?
5. Name one example of a data link protocol using sliding windows.

10.9 FURTHER READINGS

1. Andrew S. Tanenbaum, "Computer Networks", Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, "Computer Networking", Third Edition, Pearson Education
3. Behrouz A Forouzan, "Data Communications and Networking", Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, "Computer Communications and Networking Technologies", Cengage Learning (2008)

Dr. U. Surya Kameswari

LESSON- 11
ETHERNET

OBJECTIVES:

After going through this lesson, you will be able to

- Understand Ethernet cabling and physical layer encoding.
- Learn about Manchester encoding and its role in Ethernet.
- Describe the Ethernet MAC sublayer protocol.
- Study Ethernet performance, including backoff algorithms.
- Explore advanced Ethernet types: Switched, Fast, and Gigabit Ethernet.

STRUCTURE OF THE LESSON:

11.1 INTRODUCTION

The IEEE 802 standards define how different types of networks operate, including Ethernet (802.3) and Wi-Fi (802.11), which are the most widely used today. While many IEEE 802 protocols were created, only a few remain popular. Ethernet exists in two main forms: classic Ethernet, which used shared communication and operated at 3–10 Mbps, and switched Ethernet, which uses switches for faster and more efficient connections, now running at speeds up to 10 Gbps. Although both are called Ethernet, switched Ethernet is the modern standard, and the terms "Ethernet" and "802.3" are often used interchangeably.

11.2 CLASSIC ETHERNET PHYSICAL LAYER

Ethernet was developed in the 1970s by Bob Metcalfe, inspired by his exposure to the ALOHA system during his studies at Harvard and a summer spent in Hawaii with its creator, Norman Abramson. At Xerox PARC, Metcalfe and David Boggs designed the first Ethernet system to connect personal computers, using a thick coaxial cable and running at 3 Mbps. They named it "Ethernet" after the old scientific concept of the "ether" through which electromagnetic waves were once thought to travel. The system's success led to the creation of a 10-Mbps standard by DEC, Intel, and Xerox in 1978, known as the DIX standard, which became the IEEE 802.3 standard in 1983. Despite inventing Ethernet, Xerox failed to capitalize on it, leading Metcalfe to start 3Com, a company that sold millions of Ethernet adapters and helped popularize the technology.

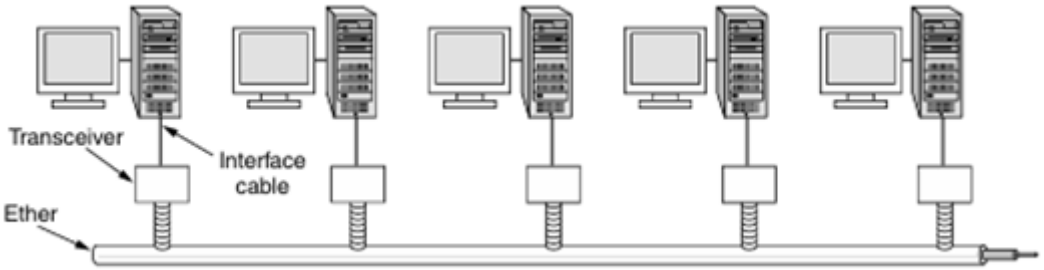


Figure 11.1 Architecture of classic Ethernet

Classic Ethernet used a single long coaxial cable that ran through the building, connecting all computers along it. The first version, called thick Ethernet, was a stiff, yellow cable marked every 2.5 meters for connections. It was later replaced by thin Ethernet, which was more flexible and cheaper, but supported shorter distances (185 meters vs. 500 meters) and fewer machines per segment (30 vs. 100). To build larger networks, multiple segments were connected using repeaters—devices that regenerate and retransmit signals to maintain strength. Ethernet used Manchester encoding for data transmission, and the network design limited the maximum distance between transceivers to 2.5 km and allowed only up to four repeaters per path to ensure the MAC protocol functioned correctly.

11.3 CLASSIC ETHERNET MAC SUBLAYER PROTOCOL

In classic Ethernet (IEEE 802.3), each frame starts with an 8-byte preamble. The first 7 bytes contain the repeating bit pattern 10101010, which helps the receiver synchronize its clock with the sender's signal using Manchester encoding. The 8th byte, known as the Start of Frame Delimiter (SFD), ends with 11, signaling the start of the actual frame data. This setup ensures reliable detection and timing alignment before the transmission of meaningful data begins.

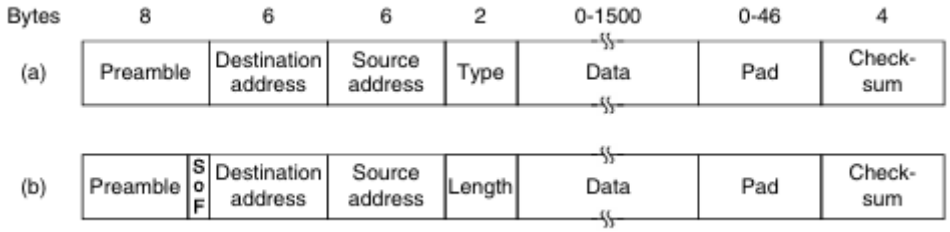


Figure 11.2 Frame formats (a) Ehternet (DIX). (b) IEEE 802.3

In Ethernet frames, after the preamble, there are two 6-byte address fields: the destination and source addresses. The destination address begins with a 0 for unicast (single receiver) or 1 for group addresses (multicast). A special address of all 1s is used for broadcasting, meaning every station on the network receives the frame. Source addresses are globally unique 48-bit identifiers assigned by manufacturers, who receive address blocks from the IEEE.

After the addresses comes a Type or Length field: Ethernet uses it to indicate the protocol type (e.g., IPv4), while IEEE 802.3 originally used it to indicate the frame length, with protocol type handled by an extra LLC header. To resolve this confusion, a rule was adopted—values above 1536 (0x600) are treated as Type, while values below are treated as Length—allowing both standards to coexist.

Ethernet frames contain a data field of up to 1500 bytes, a limit set in 1978 due to RAM constraints in transceivers. There's also a minimum frame size of 64 bytes (including addresses and checksum) to distinguish valid frames from noise caused by collisions and to ensure collision detection works correctly.

If the actual data is less than 46 bytes, padding is added. A minimum frame length is crucial to allow a collision signal (jam) from a distant station to reach the sender before it finishes transmitting, thus detecting the collision. At 10 Mbps, this means frames must last at least 512 bits (64 bytes).

Finally, a 32-bit CRC checksum at the end checks for transmission errors; if any are found, the frame is discarded.

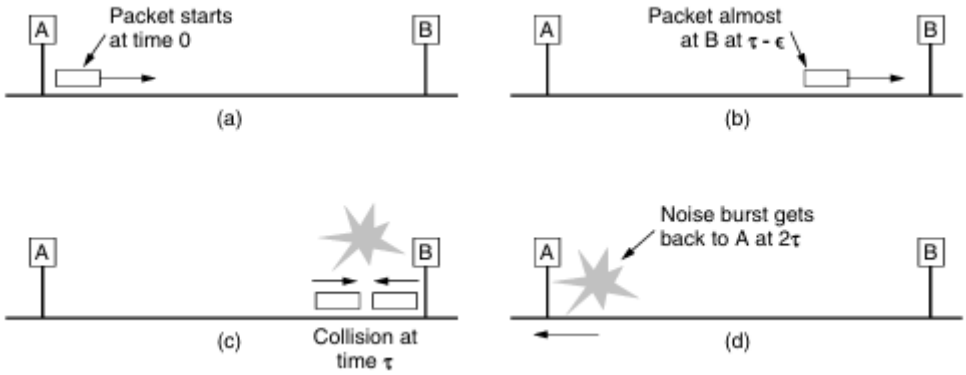


Figure 11.3 Collision detection can take as long as 2τ

CSMA/CD with Binary Exponential Backoff

Classic Ethernet uses the 1-persistent CSMA/CD protocol, where a station with data to send listens to the channel and transmits immediately when the channel is idle. While transmitting, it monitors for collisions. If a collision is detected, it sends a jam signal, stops transmission, and waits a random backoff time before retrying. This time is calculated using exponential backoff: after the first collision, it waits 0 or 1 time slots (each slot is 512 bit times or 51.2 μs). After each subsequent collision, the range of slots doubles (0 to 2^i-1, for the *i-th* collision), up to a cap at 1023 slots. If 16 collisions occur for the same frame, the transmission is aborted and reported as a failure.

The binary exponential backoff algorithm in Ethernet helps efficiently manage collisions. If only a few stations collide, the backoff time is short, keeping delay low. But if many stations collide, the random wait time increases exponentially, reducing the chance of repeated collisions. This dynamic adjustment balances speed and fairness. The backoff caps at 1023 slots to avoid excessive delays. Ethernet does not use acknowledgements, assuming reliable delivery due to low error rates on wired media. Any errors are detected by CRC, and recovery is handled by higher network layers.

11.4 ETHERNET PERFORMANCE

Under heavy and constant load, classic Ethernet performance is affected by how many stations are trying to send data and how long the cable is. To understand this, Metcalfe and Boggs proposed a simplified model using a constant probability of retransmission in each time slot. They found that the optimal condition for successful transmission occurs when each of the k stations transmits with probability 1/k, which leads to a maximum efficiency of about 1/e (~37%). The average time spent in contention (waiting due to collisions) increases with cable length, as it depends on the round-trip signal time (2τ). The channel efficiency, or how well the network uses its bandwidth, decreases as either the bandwidth (B) or cable length (L) increases relative to the frame size (F). In formula terms, the efficiency becomes worse when 2BLe/cF grows, where c is the speed of signal propagation. This means classic Ethernet is not ideal for high-speed or long-distance networks, which is why newer implementations were developed.

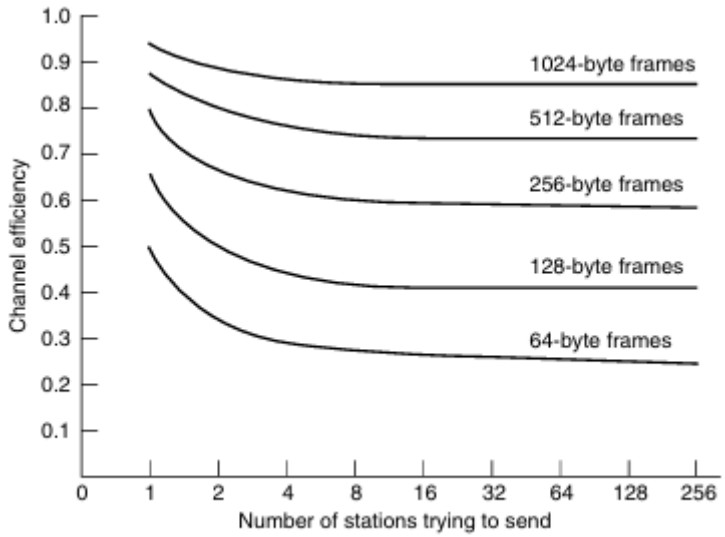


Figure 11.4 Efficiency of Ethernet at 10 Mbps with 512-bit slot times

Classic Ethernet performance under heavy load is evaluated using probability models. When multiple stations try to send data at once, they wait random time intervals (based on binary exponential backoff) to reduce collisions. Efficiency depends on frame size and network length — small frames cause more collisions and lower efficiency, while larger frames (like 1024 bytes) can achieve up to 85% efficiency.

The channel efficiency decreases if the product of bandwidth and cable length increases without increasing frame size. Although theoretical models predict poor performance under high load, real-world experiments show Ethernet works well even at moderately high traffic

levels. Also, traffic patterns are not always random (Poisson); they're often bursty, which affects performance analysis.

11.5 SWITCHED ETHERNET

Ethernet evolved from the original single long cable setup due to practical challenges like detecting cable breaks and connection issues. A more manageable approach was introduced where each station connects directly to a central hub via its own cable. This configuration makes it easier to maintain and troubleshoot the network. These hubs use twisted pair wiring, often reusing existing telephone lines, but this limits cable lengths to about 100–200 meters.
However, while hubs simplify physical setup, they do not increase network capacity—all connected devices still share the same bandwidth, just like in classic Ethernet. As more devices are added, network performance degrades due to increased competition for bandwidth.

To solve this, switched Ethernet was introduced. A switch looks like a hub externally but works differently internally. It uses a high-speed internal backplane to directly connect ports, allowing multiple simultaneous conversations between devices. This drastically improves network performance because devices no longer share a single communication channel. Switches also maintain the maintenance simplicity of hubs—faulty connections typically only affect one device, and switches can be replaced easily if they fail.

Classic Ethernet evolved from using a single long cable to using hubs and then switches for better performance and reliability. A hub simply connects all devices, acting like one big shared cable—so collisions happen often.

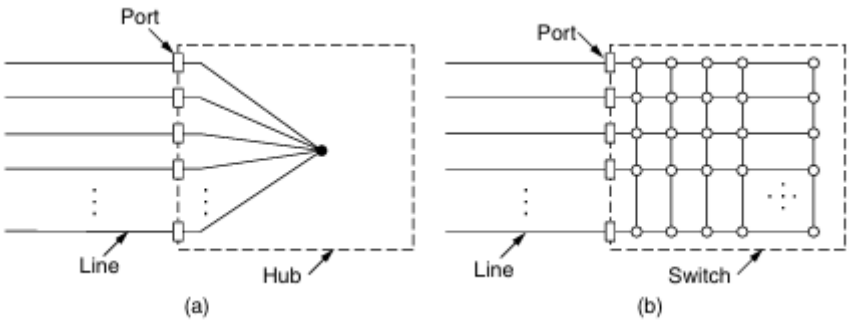


Figure 11. 5 (a) Hub (b) Switch

A switch, however, is smarter. It sends each Ethernet frame only to the correct destination port, not to all devices. This means:

No collisions (especially with full-duplex cables),
Multiple devices can send/receive at the same time, increasing speed,
Better security, since data isn't shared with all devices.
Inside a switch, Ethernet frames are handled much more intelligently than in a hub. When a switch receives a frame on one of its ports, it checks the destination Ethernet (MAC) address and sends the frame only to the specific port that corresponds to that address. This selective forwarding is possible because the switch maintains an internal table that maps MAC addresses to switch ports.

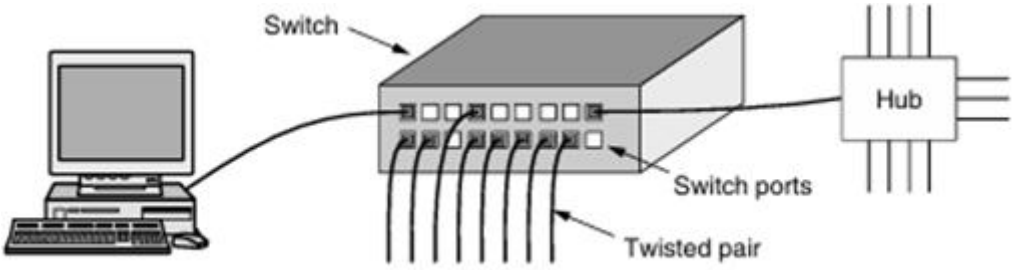


Figure 11.6 An Ethernet switch

The switch uses a high-speed internal backplane—often several Gbps—to move the frame from the input port to the destination port. This backplane uses proprietary methods and doesn't need to follow Ethernet standards because it's entirely internal. Once the frame reaches the destination port, it's transmitted over the wire to the target device. Importantly, other ports are unaware of the frame, which boosts both performance and security.

Switches also use buffers to handle multiple frames and support hubs on some ports, though hubs are now mostly outdated due to cheaper switches. Modern networks mostly use switched Ethernet for high efficiency and reliability.

11.6 FAST ETHERNET

As Ethernet usage grew, the 10 Mbps speed became insufficient due to increasing data demand. To handle this, many networks used combinations of repeaters, hubs, and switches—but each computer was still limited by its cable speed.

To address this, the IEEE formed the 802.3u committee in 1992 to develop a faster Ethernet. After some debate, the committee decided to keep the existing Ethernet design but increase the speed—leading to the creation of Fast Ethernet (802.3u), approved in 1995. It was fully backward compatible, using the same frame formats and procedures, but running at 100 Mbps instead of 10 Mbps.

Fast Ethernet used twisted-pair wiring (not the old coaxial cables), especially Category 5 cables, which could reliably support 100 Mbps over 100 meters. To remain flexible, support was also added for Category 3 twisted pair (with enhancements) and fiber optics, allowing fast Ethernet deployment without needing to rewire office buildings.

Name	Cable	Max. segment	Advantages
100Base-T4	Twisted pair	100 m	Uses category 3 UTP
100Base-TX	Twisted pair	100 m	Full duplex at 100 Mbps (Cat 5 UTP)
100Base-FX	Fiber optics	2000 m	Full duplex at 100 Mbps; long runs

Figure 11.7 The original fast Ethernet cabling

The 100Base-T4 standard used Category 3 twisted pair wiring and achieved 100 Mbps by using four twisted pairs, with a complex signalling scheme involving three voltage levels. This allowed offices with existing telephone wiring (which usually has four pairs) to upgrade without rewiring, though at the cost of dedicating those wires solely for networking (no phone line).

However, 100Base-T4 was soon replaced by 100Base-TX, which uses Category 5 cables with just two twisted pairs, simpler signalling (4B/5B encoding at 125 MHz), and supports full duplex (simultaneous send and receive at 100 Mbps).

Another option, 100Base-FX, uses fiber optics for longer distances (up to 2 km) and full duplex communication.

To maintain Ethernet's collision detection rules (CSMA/CD) at higher speeds, cable lengths had to be shortened proportionally. For 100 Mbps, maximum cable lengths were reduced, and longer fiber links must operate in full duplex mode to avoid collisions.

Fast Ethernet switches support both 10 Mbps and 100 Mbps devices and use auto-negotiation to automatically choose the best speed and duplex settings, although this sometimes causes duplex mismatch problems if not all devices negotiate properly.

11.7 GIGABIT ETHERNET

Gigabit Ethernet (standardized as IEEE 802.3ab in 1999) was designed to increase Ethernet speed to 1 Gbps (1000 Mbps) while staying fully compatible with previous Ethernet standards (same addressing, frame formats, and unacknowledged datagram service).

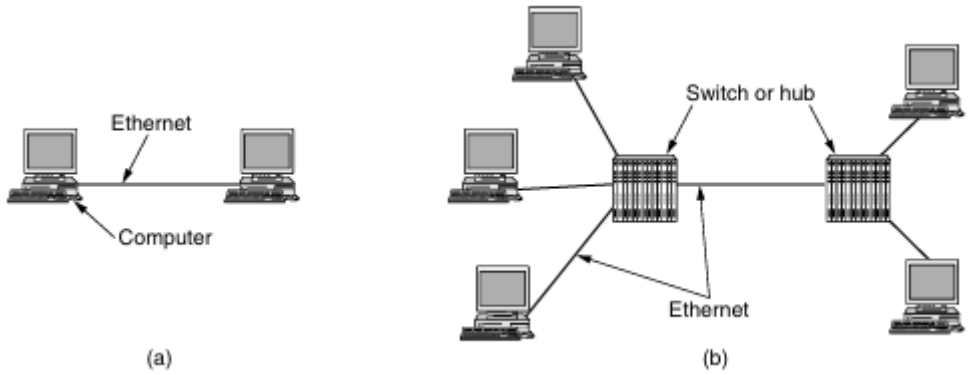


Figure 11.8 (a) A two-station Ehternet (b) A multistation Ethernet

It uses point-to-point links, meaning each cable connects exactly two devices — either directly between two computers or between a computer and a switch/hub.

Like fast Ethernet, gigabit Ethernet supports full-duplex (send and receive at the same time) and half-duplex modes, but full duplex is the common setup with switches.

In full-duplex mode, devices can send frames whenever they want without worrying about collisions or checking if the line is free, so CSMA/CD is not needed.

Because collisions don't happen in full duplex, cable length limits are based on signal quality, not collision detection timing.

Gigabit Ethernet switches support auto negotiation to automatically select the best speed among 10 Mbps, 100 Mbps, and 1 Gbps.

Half-duplex mode is used when devices connect through a hub instead of a switch. Unlike switches, hubs don't buffer data — they just electrically connect all devices on one shared line, so collisions can happen.

Because gigabit Ethernet transmits data 100 times faster than classic 10 Mbps Ethernet, the maximum cable length in half-duplex mode has to shrink proportionally — from 2500 meters down to 25 meters — to make sure collision detection (CSMA/CD) still works properly.

This 25-meter limit is often too short for practical use, so the standard added two features to extend the cable length to 200 meters, suitable for most offices:

Carrier extension: The hardware adds padding to extend short frames (64 bytes) to 512 bytes, allowing collision detection timing to work over longer cables. But this wastes bandwidth because many bytes sent are just padding.

Frame bursting: Allows sending multiple frames back-to-back as one burst, making transmission more efficient than carrier extension by reducing overhead.

In reality, hubs are rarely used with gigabit Ethernet now because switches are cheaper and better. Most modern computers support all speeds (10, 100, 1000 Mbps) and automatically negotiate the best option, including full duplex.

Gigabit Ethernet Cabling Types (Fig. 11.9):

Name	Cable	Max. segment	Advantages
1000Base-SX	Fiber optics	550 m	Multimode fiber (50, 62.5 microns)
1000Base-LX	Fiber optics	5000 m	Single (10 μ) or multimode (50, 62.5 μ)
1000Base-CX	2 Pairs of STP	25 m	Shielded twisted pair
1000Base-T	4 Pairs of UTP	100 m	Standard category 5 UTP

Figure 11.9 Gigabit Ethernet cabling

Signaling Details:

Speed & Encoding:

Gigabit Ethernet signals at around 1 Gbps, meaning sending a bit every nanosecond. This requires very precise and efficient encoding.

Optical Fiber Versions:

1000Base-SX: Uses short wavelength (0.85 microns) LEDs on multimode fiber; good for up to 500 m inside buildings.

1000Base-LX: Uses long wavelength (1.3 microns) lasers on single-mode or multimode fiber; can go up to 5 km, suitable for campus backbones.

Encoding Scheme (8B/10B):

To maintain signal integrity and clock recovery, data is encoded using 8B/10B encoding, which converts every 8 bits into a 10-bit codeword. This coding balances zeros and ones and ensures enough transitions for synchronization.

This adds a 25% overhead in bandwidth (sending 10 bits to represent 8 bits), which is much better than Manchester coding's 100% overhead.

Copper Cabling Challenges:
Initially, the faster signaling required new cables (shielded or fiber). But since Category 5 UTP wiring was widely deployed, a new version, 1000Base-T, was created to run Gigabit Ethernet over standard Cat 5 UTP cables.

1000Base-T Signaling:
Uses all 4 twisted pairs simultaneously, each running in both directions at once (full duplex on each pair) using sophisticated digital signal processing to separate send and receive signals.

Uses 5 voltage levels per pair, encoding 2 bits per symbol at 125 million symbols per second. The encoding involves scrambling and error correction to improve reliability.

Flow Control & Jumbo Frames:

At 1 Gbps, a receiver overwhelmed or busy for even 1 millisecond can accumulate nearly 2000 frames waiting to be processed, causing buffer overruns.

To manage this, flow control was introduced using PAUSE frames:
Special Ethernet frames with type 0x8808 tell the sender to pause transmission for a specified duration.

The pause time units are multiples of 512 nanoseconds, allowing pauses up to about 33.6 milliseconds.

Jumbo Frames:
Allow frames larger than the standard 1500 bytes (up to 9 KB).
Not part of the official standard (proprietary), but widely supported.
Bigger frames reduce CPU overhead by decreasing the number of frames processed per second, which improves efficiency at gigabit speeds.

11.8 10-GIGABIT ETHERNET

After gigabit Ethernet was standardized, the IEEE 802 committee moved quickly to develop 10-gigabit Ethernet (10GbE), which offers speeds 1000 times faster than the original Ethernet. This extremely high speed is primarily needed in data centres to connect high-performance routers, switches, and servers, as well as in long-distance metropolitan area networks using fiber optic cables. While long-distance connections rely on fiber, shorter connections can use either fiber or copper cables.

All versions of 10GbE operate in full-duplex mode only, meaning data can be sent and received simultaneously, eliminating collisions and the need for the CSMA/CD protocol. Devices support auto negotiation to fall back to the highest speed both ends support, maintaining compatibility. The standard focuses heavily on the physical layer to achieve these high speeds.

Name	Cable	Max. segment	Advantages
10GBase-SR	Fiber optics	Up to 300 m	Multimode fiber (0.85μ)
10GBase-LR	Fiber optics	10 km	Single-mode fiber (1.3μ)
10GBase-ER	Fiber optics	40 km	Single-mode fiber (1.5μ)
10GBase-CX4	4 Pairs of twinax	15 m	Twinaxial copper
10GBase-T	4 Pairs of UTP	100 m	Category 6a UTP

Figure 11.10 10-Gigabit Ethernet cabling

Several cabling options exist for 10GbE. Fiber optic versions include 10GBase-SR for short distances over multimode fiber, 10GBase-LR for up to 10 km over single-mode fiber, and 10GBase-ER for distances up to 40 km. Copper options include 10GBase-CX4, which uses twinax cables for short connections, and 10GBase-T, which runs 10 Gbps over standard twisted pair cables (Category 6a or better) for up to 100 meters.

Technically, 10GbE uses advanced encoding and signalling techniques. Data is scrambled and encoded with a 64B/66B code to reduce overhead compared to earlier methods. The copper version, 10GBase-T, uses complex modulation with 16 voltage levels and sophisticated error correction to transmit high-speed data reliably over twisted pair cables.

Finally, the IEEE committee is already working on even faster Ethernet standards operating at 40 and 100 Gbps to meet future networking demands in backbone and data centre environments. These faster standards are still under development, but some proprietary solutions are already available.

11.9 RETROSPECTIVE ON ETHERNET

Ethernet has remained dominant for over 30 years because of its simplicity, flexibility, and reliability. Its straightforward design makes it cheap, easy to maintain, and highly dependable—qualities that encourage users to keep existing Ethernet infrastructure rather than replace it. Ethernet's use of inexpensive twisted-pair wiring and minimal configuration keeps costs low and setup easy, while adding new devices is as simple as plugging them in.
Another key factor is Ethernet's seamless compatibility with the TCP/IP protocol suite, which is connectionless and pairs naturally with Ethernet's connectionless design. This compatibility helped Ethernet outlast competing technologies like ATM, FDDI, and Fibre Channel, which were more complex, incompatible, and costly. Interestingly, Ethernet incorporated some of their innovations, such as advanced encoding methods, to boost its own performance.

Looking ahead, Ethernet continues to evolve and expand its applications. The introduction of 10-gigabit Ethernet overcame previous distance limits, and carrier-grade Ethernet is being developed for reliable, high-quality services over metropolitan and wide area networks. Additionally, very high-speed Ethernet finds use inside large routers and servers, demonstrating Ethernet's adaptability beyond traditional office networking.

11.10 SUMMARY

This chapter focuses on Ethernet, a widely used LAN technology. It covers Ethernet cabling and Manchester encoding, which ensures proper signal timing. The MAC sublayer protocol manages access to the shared medium using CSMA/CD and the Binary Exponential Backoff

Algorithm to handle collisions. Ethernet performance is enhanced with Switched Ethernet, Fast Ethernet, and Gigabit Ethernet, providing higher speeds and reduced congestion. The chapter also introduces IEEE 802.2 Logical Link Control, which separates addressing and control from physical transmission, offering a structured and efficient LAN communication system

11.11 TECHNICAL TERMS

LAN , CSMA/CD, Ethernet, Fast Ethernet, Gigabit Ethernet, IEEE 802.2

11.12 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain Ethernet cabling and Manchester encoding in detail.
2. Describe the Ethernet MAC sublayer protocol and how it handles collisions.
3. Discuss the Binary Exponential Backoff Algorithm and its role in network performance.
4. Compare standard Ethernet, Fast Ethernet, and Gigabit Ethernet.
5. Explain the IEEE 802.2 Logical Link Control and its relation to Ethernet.

Short Questions:

1. What type of cabling is commonly used in Ethernet?
2. Define Manchester encoding.
3. What is the purpose of the Ethernet MAC sublayer?
4. What is the Binary Exponential Backoff Algorithm?
5. Name two advanced versions of Ethernet

11.13 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008)

Dr. Uduga Surya Kameswari

LESSON- 12

WIRELESS LANs AND BLUETOOTH

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the 802.11 wireless LAN protocol stack and architecture.
- Learn about the 802.11 physical layer and MAC sublayer.
- Describe the 802.11 frame structure and communication process.
- Understand Bluetooth architecture, layers, and applications.
- Study the Bluetooth protocol stack and frame structure.

STRUCTURE OF THE LESSON:

12.1 INTRODUCTION

Wireless LANs (Local Area Networks) have become increasingly popular in various places like homes, offices, cafes, airports, and even public spaces such as zoos, allowing devices like computers, smartphones, and PDAs to connect to the Internet without cables. They also enable nearby devices to communicate directly without needing an Internet connection. The main technology behind wireless LANs is the IEEE 802.11 standard, commonly known as Wi-Fi. While an overview of this was provided earlier in the text, this section takes a deeper

look into its key components, including the protocol stack, radio transmission methods at the physical layer, the MAC (Medium Access Control) sublayer protocol, the structure of data frames, and the different services 802.11 provides. For more detailed information, readers are referred to Gast (2005), a widely cited source on the subject.

12.2 THE 802.11 ARCHITECTURE AND PROTOCOL STACK

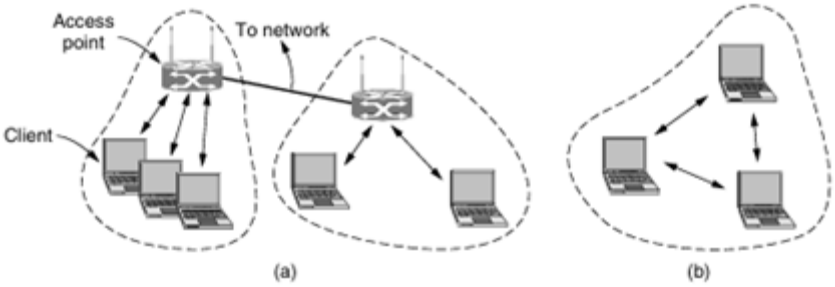


Figure 12.1 802.11 architecture (a) Infrastructure mode (b) Ad-hoc mode

802.11 networks can operate in two modes: infrastructure mode and ad hoc mode. In the more common infrastructure mode, devices like laptops and smartphones connect to an Access Point (AP), which is further linked to a larger network such as the Internet or a company intranet. Multiple APs can be connected through a distribution system, allowing communication between clients via their respective APs. In contrast, ad hoc mode involves devices connecting directly to each other without any AP, forming a peer-to-peer network, though this mode is less popular due to limited Internet access. The 802.11 protocol stack, used by both clients and APs, follows the general structure of other IEEE 802 protocols. It includes a physical layer, which aligns with the OSI model's physical layer, and a data link layer that is divided into two sublayers: the MAC (Medium Access Control) sublayer, which manages channel access, and the LLC (Logical Link Control) sublayer, which provides protocol identification and ensures consistency across different 802 variants.

In 802.11 networks, the physical layer closely aligns with the OSI physical layer, handling the actual transmission of bits over the air. However, the data link layer is divided into two sublayers: MAC (Medium Access Control) and LLC (Logical Link Control).

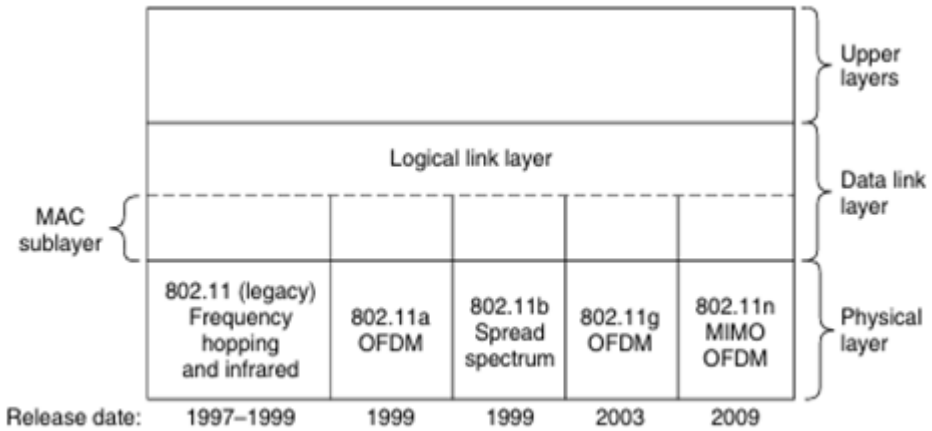


Figure 12.2 Part of the 802.11 protocol stack

The MAC sublayer is responsible for managing access to the shared wireless channel, deciding which device gets to transmit next. Above it, the LLC sublayer works to hide the technical differences among various 802 protocols, making them appear uniform to the network layer. While this could have been a complex role, the LLC is now mainly a simple "glue" layer that identifies which protocol, such as IP, is being carried in the 802.11 frame.

12.3 THE 802.11 PHYSICAL LAYER

The wireless transmission techniques used in 802.11 (Wi-Fi) networks. All these methods enable the transmission of MAC frames between devices using short-range radio signals, typically in the 2.4 GHz or 5 GHz ISM frequency bands. These bands are popular because they are unlicensed, meaning anyone can use them within certain power limits, but they are also crowded due to interference from devices like microwave ovens and cordless phones. The 5 GHz band, though less congested, has a shorter range than 2.4 GHz. Each 802.11 transmission technique supports multiple data rates, allowing devices to adapt the rate based on signal quality—low rates in poor conditions and high rates when the signal is strong. This is known as rate adaptation, but the specific method for it is not standardized. The passage focuses particularly on 802.11b, a spread-spectrum technique supporting speeds of 1, 2, 5.5, and 11 Mbps. It uses a Barker sequence for spreading, which helps receivers identify the start of a transmission. Lower speeds use BPSK or QPSK modulation, while higher speeds rely on CCK (Complementary Code Keying) for efficient encoding of bits into chip sequences.

The evolution of 802.11 wireless standards and the technologies they use to improve speed and reliability. 802.11a operates in the 5-GHz ISM band and supports data rates up to 54 Mbps using OFDM (Orthogonal Frequency Division Multiplexing), which is efficient and resists signal issues like multipath interference. Although 802.11a was developed before 802.11b, it was released later due to technical challenges. In contrast, 802.11b, which uses spread-spectrum techniques in the 2.4-GHz band, became popular earlier due to its longer range. Later, 802.11g combined the high speed of 802.11a with the wide compatibility and longer range of 802.11b by using OFDM in the 2.4-GHz band. To reduce customer confusion, many network interface cards (NICs) support 802.11a/b/g standards together. The next major upgrade was 802.11n, ratified in 2009, which aimed for throughput over 100 Mbps by doubling the channel width (from 20 MHz to 40 MHz), reducing overhead, and most importantly, using MIMO (Multiple Input Multiple Output) technology. MIMO allows multiple data streams to be sent simultaneously using multiple antennas, boosting speed, range, and reliability. This innovation, alongside OFDM, represents a significant leap in wireless communication technology.

12.4 THE 802.11 MAC SUBLAYER PROTOCOL

The 802.11 MAC sublayer protocol handles wireless communication differently from Ethernet, mainly due to the limitations of radio transmission. Unlike Ethernet, wireless radios are typically half-duplex, meaning they can't transmit and receive at the same time. This makes collision detection—used in Ethernet—impractical, because the transmitted signal overwhelms any incoming signals.

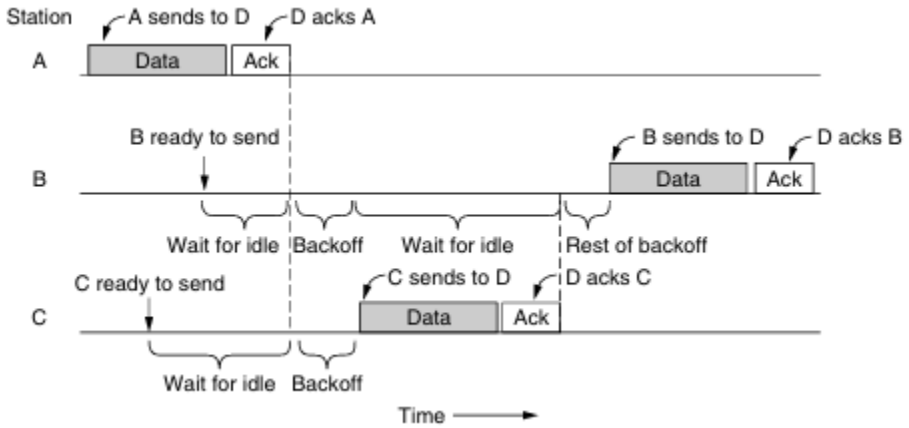


Figure 12.3 Sending a frame with CSMA/CA

Instead, 802.11 uses CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance). In this protocol, before sending, a station first senses if the channel is idle. If it is, the station waits for a short time (called DIFS) and then starts a random backoff countdown to avoid collisions. If another frame is transmitted during the countdown, the station pauses and resumes after the channel is idle again. When the countdown reaches zero, the station sends its frame. If the transmission is successful, the destination sends an acknowledgment (ACK). If no ACK is received, the sender assumes a collision or error occurred and retries, doubling the backoff window (exponential backoff), similar to Ethernet. The passage also describes a timeline where multiple stations coordinate their transmissions using this method to avoid collisions and ensure smooth communication.

Compared to Ethernet, the 802.11 wireless protocol has two major differences. First, it uses collision avoidance rather than collision detection. This is necessary because in wireless communication, collisions are expensive—the entire frame is lost—and cannot be detected during transmission due to the half-duplex nature of radios. So, 802.11 uses early backoff and acknowledgements (ACKs) to avoid and infer collisions. This operation mode is called DCF (Distributed Coordination Function), where each station operates independently. An optional mode, PCF (Point Coordination Function), allows the access point to control all traffic, but it is rarely used due to interference from nearby networks.

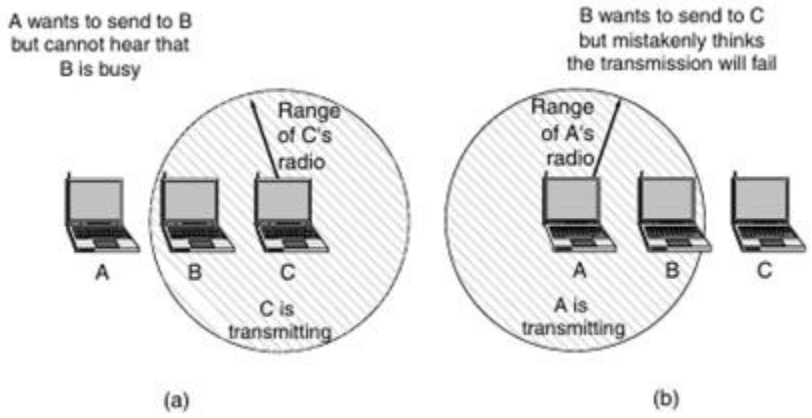


Figure 12.4 (a) The hidden terminal problem (b) The exposed terminal problem

The second major issue in wireless networks is the uneven transmission range of devices, leading to problems like the hidden terminal problem—where one station cannot detect another's transmission, causing unintended collisions—and the exposed terminal problem, where a station refrains from sending due to sensing another transmission that actually wouldn't interfere. These range-related issues don't exist in Ethernet, where all nodes are connected via cable and can detect each other's signals reliably.

When a station like C hears the RTS frame from A, it knows the channel will be busy for a certain time and updates its Network Allocation Vector (NAV) to stay silent during that period. Similarly, station D, which may only hear the CTS from B, also updates its NAV and defers transmission. These NAV updates are internal timers that help prevent collisions. However, in practice, the RTS/CTS mechanism is rarely used because it adds overhead, especially for short frames or when the access point controls the channel, which all devices can hear anyway. RTS/CTS mainly helps with hidden terminals but does not solve exposed terminal problems.

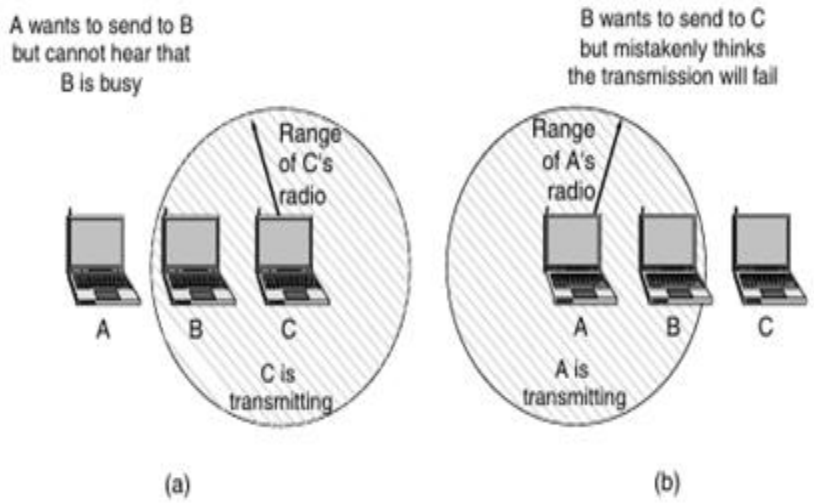


Figure 12.5 Virtual channel sensing using CSMA/CA

The core of 802.11 is CSMA/CA combined with physical and virtual sensing, but wireless networks face challenges due to interference and noise, making transmissions unreliable. To improve reliability, stations often reduce their transmission rates, using more robust modulations that work better in noisy environments. They may also send shorter frames, since shorter frames have a higher chance of being received without errors, reducing the need for retransmissions and improving overall network performance.

802.11 improves reliability by breaking large frames into smaller fragments, each acknowledged separately, reducing errors. For power saving, devices can sleep and wake up periodically to check buffered data from the access point, using beacon signals and mechanisms like APSD for efficient two-way traffic. To maintain quality of service, 802.11 prioritizes delay-sensitive traffic like voice calls over less urgent data to avoid delays and ensure smooth performance.

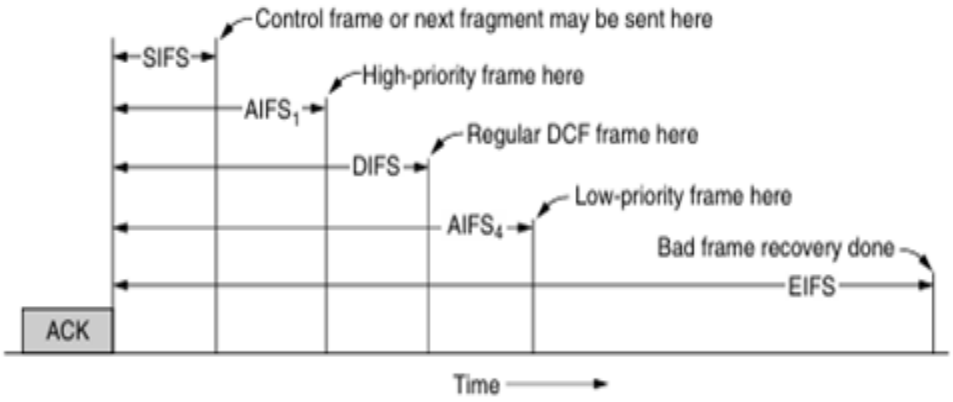


Figure 12.6 Interframe spacing in 802.11

The 802.11e extension improves quality of service by using different wait times between frames to prioritize traffic. Shorter wait times (like SIFS) let important frames like acknowledgments or voice data go first, while longer wait times delay less important background traffic. This way, high-priority data, such as voice or video, gets transmitted faster.

Also, 802.11e introduces Transmission Opportunities (TXOPs), allowing stations to send multiple frames during their turn. This helps fix the "rate anomaly" problem where slow senders reduce the overall network speed. By giving each station equal airtime instead of equal frame counts, faster stations achieve higher throughput, improving overall performance and fairness.

12.5 THE 802.11 FRAME STRUCTURE

The 802.11 standard defines three types of frames: data, control, and management, each with its own header fields used by the MAC sublayer. Focusing on the data frame, its header starts with a Frame Control field, which includes several subfields. These specify the protocol version, frame type (data, control, or management), and subtype (such as RTS or CTS). Other important bits indicate whether the frame is going to or coming from the distribution system (network connected to the access point), if more fragments will follow, if it's a retransmission, and whether the sender is entering power-save mode. Additional bits signal if more data is pending, if the frame is encrypted, and whether frames need to be processed in order. This header helps manage and control the flow and security of data over the wireless network.

The second field in the data frame is the Duration field, which indicates how long the current frame and its acknowledgement will occupy the wireless channel, measured in microseconds. This helps stations manage the Network Allocation Vector (NAV) to avoid collisions.

Next are three addresses: the receiver's address, the transmitter's address, and a third address that represents the final destination or source beyond the access point (AP). Since the AP acts as a relay between clients and the wider network, this third address identifies the distant endpoint.

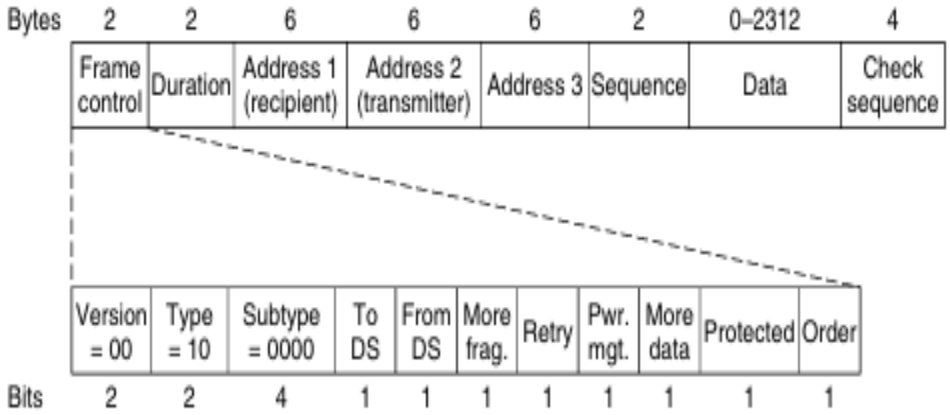


Figure 12.7 Format of the 802.11 data frame

The Sequence field numbers the frames and fragments to detect duplicates and manage reassembly. The main Data field carries the payload (up to 2312 bytes), starting with an LLC header that helps identify the higher-layer protocol, such as IP.

Finally, the Frame Check Sequence (FCS) provides error detection using a 32-bit cyclic redundancy check (CRC).

Management frames have a similar format but contain different types of data depending on their subtype (e.g., beacon frames). Control frames are shorter, containing only essential fields like Frame Control, Duration, and FCS, usually with one address and no data payload; their subtype (e.g., ACK, RTS, CTS) carries most of the control information.

12.6 SERVICES

Association: When a mobile device comes in range, it connects to an Access Point (AP) by learning its capabilities (data rates, security, power-saving, QoS). The device sends an association request, which the AP can accept or reject.

Reassociation: Allows a device to switch from one AP to another within the same network (like a handover) without losing data.

Disassociation: Either the device or AP can break their connection when leaving the network or for maintenance.

Authentication: Devices must prove they are allowed to connect. WPA2 is the recommended, secure method using a password or credentials checked by an authentication server. Older WEP method is insecure and discouraged.

Distribution service: Once the AP receives frames, it routes them locally or forwards them to the wired network.

Integration service: Handles communication between the wireless LAN and other networks, like the Internet.

Data delivery service: Manages sending and receiving data over the wireless LAN but does not guarantee perfect reliability; upper layers handle errors.

Privacy service: Encrypts data for confidentiality using AES encryption with keys established during authentication.

Quality of Service (QoS) service: Prioritizes important traffic like voice and video over regular or background traffic.

Transmit Power Control: Helps devices comply with regional power limits.

Dynamic Frequency Selection (DFS): Helps devices avoid interfering with radar signals in certain frequency bands.

Together, these services enable robust, secure, and efficient wireless networking for mobile clients.

12.7 BLUETOOTH

In 1994, Ericsson sought to connect its mobile phones wirelessly to other devices like laptops, leading to the formation of a Special Interest Group (SIG) in 1998 with IBM, Intel, Nokia, and Toshiba to develop a short-range, low-power wireless standard called Bluetooth, named after the Viking king Harald Bluetooth. Bluetooth 1.0 was released in 1999, enabling devices to pair and securely transfer data without cables. Since then, Bluetooth has become widely adopted in consumer electronics such as phones, headsets, keyboards, and more. The technology has evolved through several versions: Bluetooth 2.0 introduced higher data rates in 2004; Bluetooth 3.0 in 2009 combined Bluetooth pairing with faster data transfer using 802.11 Wi-Fi; and Bluetooth 12.0, also released in 2009, added low-power operation to extend battery life in devices. These advancements have made Bluetooth a standard for convenient, wireless communication across a wide range of devices.

12.8 BLUETOOTH ARCHITECTURE

Bluetooth technology is designed to enable short-range wireless communication between devices using small networks called piconets. Each piconet has one master device and up to seven active slave devices that are connected within a range of about 10 meters. The master device acts as the coordinator and controls how communication happens within the piconet.
In addition to these active slaves, a piconet can also have up to 255 parked devices. These parked devices are essentially in a low-power or standby mode to conserve battery life. While parked, these devices do not actively communicate but remain synchronized with the master so they can be quickly reactivated when needed. There are also two other intermediate power-saving modes called hold and sniff, which reduce power consumption while maintaining some level of responsiveness.

When multiple piconets operate in the same area, they can overlap and interconnect through a special device called a bridge that participates in multiple piconets. This interconnection forms a larger network known as a scatternet. The scatternet allows more devices to communicate across a broader network beyond the limits of a single piconet.

The master-slave architecture of Bluetooth is intentional to keep the system simple and cost-effective. Because the slaves are “dumb” devices, meaning they follow strict instructions from the master, Bluetooth chips can be manufactured cheaply (often under $5). The master

controls the clock and manages which device can transmit data during specific time slots, using a

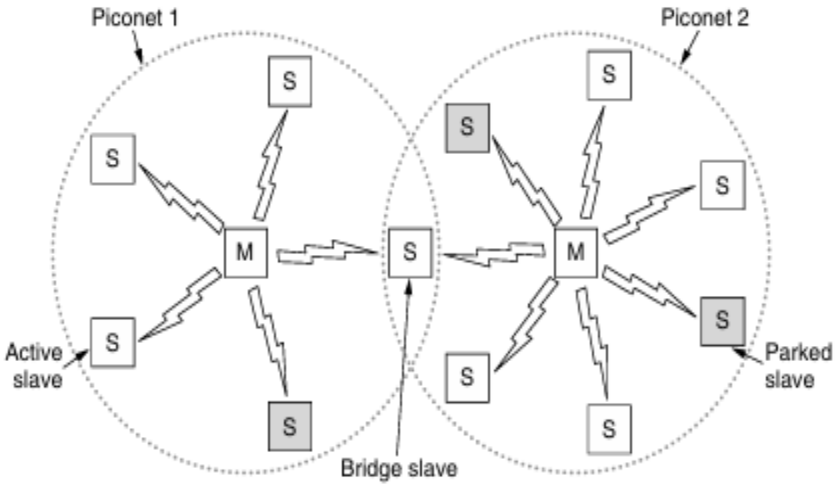


Figure 12.8 Two piconets can be connected to form a scatternet

Centralized time-division multiplexing (TDM) system. This design ensures organized communication without collisions.

Importantly, Bluetooth communication only occurs between the master and individual slaves. Slaves cannot directly talk to each other; instead, any data sent from one slave to another must first go through the master device. This central control simplifies the protocol but limits peer-to-peer interactions within a piconet.

Overall, this design balances simplicity, low cost, and efficient use of the limited wireless spectrum for short-range device connectivity. It enables Bluetooth to be widely used in consumer electronics such as phones, headsets, keyboards, and more.

12.9 BLUETOOTH APPLICATIONS

Unlike many network protocols that simply provide a communication channel and leave applications up to developers, Bluetooth defines a wide range of specific applications called profiles—currently about 25 of them. These profiles cover various uses like audio and video streaming (e.g., walkie-talkie style phone calls, hands-free headsets, music streaming), connecting keyboards and mice, sending images between devices, and even using phones as remote controls for TVs. Other profiles support networking functions, such as forming personal area networks or connecting laptops to phones for internet access.

Each profile has its own protocol stack tailored to its specific use, which adds complexity. This complexity arose because different working groups independently developed profiles for their specific needs, leading to many specialized stacks rather than a simpler unified approach. This situation reflects Conway's Law, which states that software structure often mirrors the organizational structure of the teams that create it. In hindsight, Bluetooth might have functioned well with far fewer profiles and protocol stacks.

12.10 THE BLUETOOTH PROTOCOL STACK

The Bluetooth standard organizes its many protocols into layers, but unlike typical models like OSI or TCP/IP, its structure is unique. At the bottom is the physical radio layer, which handles the actual radio transmission and modulation, focusing on keeping costs low for mass-market devices.

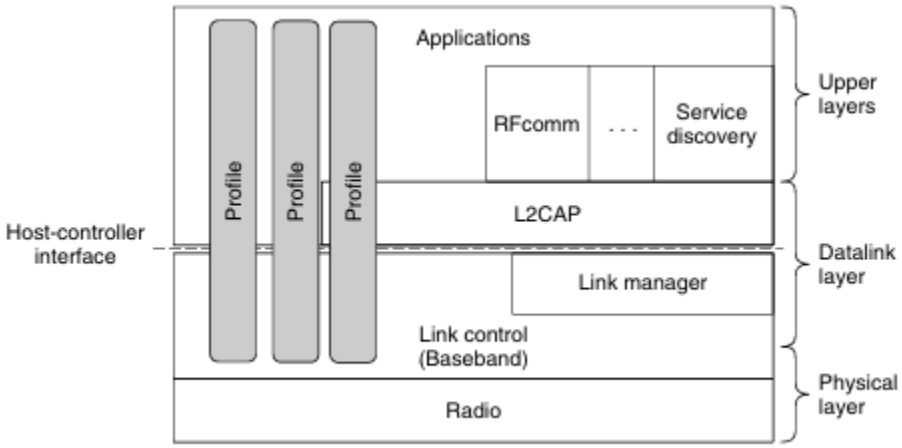


Figure 12.9 The Bluetooth protocol architecture

Above that is the link control (or baseband) layer, similar to a MAC sublayer but also handling some physical layer tasks. It manages how the master device controls timing, organizing communication into time slots and frames.

Next, two key protocols run on top of the baseband:

The link manager manages logical connections, handling pairing, encryption, power management, and quality of service. It operates below the Host Controller Interface (HCI), which separates the Bluetooth chip's low-level functions from the higher-level device functions.

Above the HCI is the L2CAP (Logical Link Control and Adaptation Protocol), which packages variable-length data messages and can provide reliability. Many other protocols use L2CAP, including utility protocols like:

Service Discovery Protocol (SDP) to find services on devices RFCOMM, which emulates a serial port to connect peripherals like keyboards and modems.

At the top layer are the profiles, each representing a specific application and defining which protocols they use. For example, a headset profile might use only what's necessary for streaming audio, skipping other protocols.

12.11 THE BLUETOOTH RADIO LAYER

The Bluetooth radio layer handles the actual transmission of bits between the master and slave devices. It's designed to be low power and has a typical range of about 10 meters. It operates in the 2.4 GHz ISM band, the same band used by Wi-Fi (802.11), which is divided into 79 channels, each 1 MHz wide.

To avoid interference and coexist with other devices in this crowded band, Bluetooth uses frequency hopping spread spectrum — rapidly switching (up to 1600 times per second) across different channels in a pseudorandom sequence controlled by the master device. All devices in the piconet hop together, synchronized by the master's timing.

Initially, Bluetooth and Wi-Fi signals interfered with each other, causing problems. The solution, called adaptive frequency hopping, lets Bluetooth detect which channels are busy with other signals and avoid them, reducing interference.

Bluetooth uses different modulation schemes to send data:

The basic method is frequency shift keying (FSK), sending 1 bit per microsecond, resulting in 1 Mbps data rate.

Later versions (Bluetooth 2.0 and onward) added phase shift keying (PSK), allowing 2 or 3 bits per symbol, increasing data rates to 2 or 3 Mbps. These faster rates are used only for the data parts of transmissions.

12.12 THE BLUETOOTH LINK LAYERS

The Bluetooth link control, or baseband layer, manages how devices share the communication channel by dividing time into 625-microsecond slots. The master transmits in even slots, and slaves transmit in odd slots, using time-division multiplexing. Frames can last 1, 3, or 5 slots, with longer frames being more efficient due to fixed overhead costs. Frequency hopping between channels happens only between frames to reduce interference, and data can be encrypted for security.

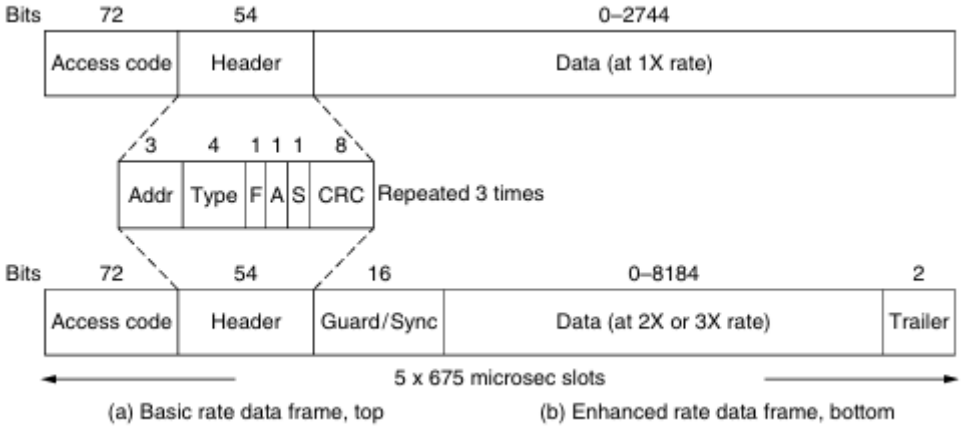
Above this, the link manager protocol establishes logical connections called links between paired devices. Pairing ensures devices are authorized to communicate, initially using simple PINs but now using a more secure passkey confirmation method. Two main link types carry data: SCO links for real-time voice with fixed time slots, and ACL links for irregular, packet-based data delivered on a best-effort basis.

The L2CAP layer handles large packets by breaking them into smaller frames and reassembling them on reception. It multiplexes multiple protocols over one link, manages error control by retransmitting lost packets, and enforces quality of service. Together, these layers allow Bluetooth devices to communicate efficiently and securely over short distances.

12.13 THE BLUETOOTH FRAME STRUCTURE

Bluetooth defines several frame formats, with the most important including an access code that identifies the master device so slaves can distinguish their traffic. The frame header is 54 bits long and repeated three times to ensure reliability in noisy environments using simple hardware. This header contains fields like address, frame type, flow control, acknowledgment, and sequencing bits.

At the basic data rate, frames carry up to 2744 bits of data in a five-slot transmission, or 240 bits in a single slot. Enhanced data rates increase the data portion by sending 2 or 3 bits per symbol instead of 1, but the access code and header remain at the basic rate. Enhanced frames also include guard fields and synchronization patterns for the faster data transmission.



12.10 Typical Bluetooth data frame at (a) basic and (b) enhanced, data rates.

The data payload differs for ACL and SCO frames. SCO frames carry fixed-size payloads designed for voice data, with error correction built in. Despite Bluetooth's raw 1 Mbps bandwidth, overhead from headers, error correction, and radio settling time reduce the effective data capacity to about 64,000 bps per direction — enough for a single full-duplex uncompressed voice channel. This efficiency limitation motivates the use of enhanced rates and longer frames in modern Bluetooth versions.

12.14 SUMMARY

This chapter covers wireless communication technologies such as 802.11 WLANs and Bluetooth. The 802.11 standard defines a protocol stack, including a physical layer and MAC sublayer, managing wireless access and frame structure for efficient communication. Bluetooth enables short-range wireless connectivity for devices, with a layered protocol stack including the radio layer, baseband, and L2CAP, supporting various applications like audio streaming, file transfer, and device networking. Both technologies provide flexible, wireless data transmission with structured protocols and frame formats to ensure reliable communication.

12.15 TECHNICAL TERMS

802.11 Wireless LAN, L2CAP

12.16 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the 802.11 protocol stack and the role of each layer.
2. Describe the 802.11 physical layer and MAC sublayer protocol in detail.
3. Discuss the 802.11 frame structure and its components.
4. Explain the Bluetooth architecture and its major applications.
5. Describe the Bluetooth protocol stack, including the radio, baseband, and L2CAP layers, and explain the frame structure

Short Questions:

1. What is the 802.11 protocol stack?
2. Name two functions of the 802.11 MAC sublayer.
3. What does the 802.11 frame structure include?
4. List one application of Bluetooth.
5. Name two layers of the Bluetooth protocol stack.

12.17 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008)

Dr. Uduga Surya Kameswari

LESSON- 13

DATA LINK LAYER SWITCHING

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the concept of data link layer switching.
- Learn about bridges and their role in connecting LANs.
- Describe the spanning tree algorithm and its application.
- Understand different network devices: repeaters, hubs, bridges, switches, routers, and gateways.
- Study Virtual LANs (VLANs) and their purpose.

STRUCTURE OF THE LESSON:

13.1 INTRODUCTION

Organizations often need to connect multiple LANs to create a larger, unified network, and this can be done using devices called bridges or switches (modern bridges). These devices operate at the data link layer, where they examine MAC addresses to forward frames between LANs without looking into the data itself, making them capable of handling various network protocols like IP or AppleTalk. Unlike routers, which work at the network layer and forward packets based on IP addresses, switches are protocol-independent and focus only on frame delivery. By using bridges or switches, multiple physical LANs can be joined into a single logical LAN. Additionally, VLANs (Virtual LANs) allow a single physical LAN to be divided into multiple logical networks, providing greater flexibility in managing and securing network traffic.

13.2 USES OF BRIDGES

Organizations often find themselves with multiple LANs for various reasons, and bridges (or switches) are essential for connecting these LANs into a cohesive network. One common reason is departmental independence—different departments within a university or corporation may set up their own LANs to support their specific needs and devices like PCs, printers, and servers. Since each department acts autonomously, there is no coordination in network setup. However, over time, the need for interdepartmental communication arises, making it necessary to use bridges to interconnect these independent LANs.

Another reason is geographical separation. When an organization spans several buildings, running a single LAN across all of them can be impractical or costly. Ethernet cables have physical length limitations (e.g., 200 meters for twisted-pair gigabit Ethernet), so connecting distant buildings requires separate LANs linked by long-distance cables and bridges. This setup extends the network's physical reach without violating cable length constraints and reduces signal loss and delay.

A third reason is scalability and performance. Large institutions like universities or corporations may have thousands of workstations, which exceeds the capacity of a single Ethernet LAN or hub. Even if physically possible, placing all devices on one LAN would lead to performance issues, as all devices share the same limited bandwidth. By splitting the network into multiple LANs and connecting them with bridges, the system can support more devices and double the bandwidth, as each LAN operates independently at full speed. Bridges also enhance reliability by isolating faults; a malfunctioning node on one LAN won't disrupt the entire network.

For maximum ease and flexibility, bridges are designed to be transparent. Ideally, a bridge should work instantly when plugged in—without requiring changes to existing hardware, software, or configurations. Stations on the network should not be able to tell whether they are part of a single LAN or a bridged LAN, and devices should be easily movable within the network. This plug-and-play functionality is made possible by two main algorithms: the backward learning algorithm, which ensures traffic is sent only where it is needed, and the spanning tree algorithm, which prevents network loops that could occur when multiple paths exist between bridges. Together, these algorithms allow bridges to manage traffic efficiently and maintain network stability.

13.3 LEARNING BRIDGES

Bridges (or Ethernet switches) are used to connect multiple LANs and can work with different network topologies, such as classic shared Ethernet or modern point-to-point links with hubs. In modern setups, switches usually replace hubs for better performance. Bridges work by listening to all incoming frames (promiscuous mode) and use the destination MAC address to decide whether to forward or discard a frame.

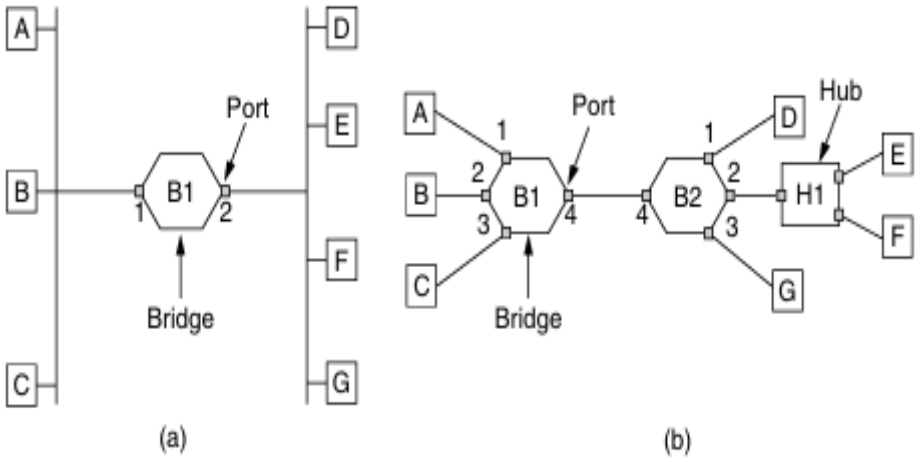


Figure 13.1 (a) Bridge connecting two multidrop LANs (b) Bridge (and a hub) connecting seven point-to-point stations

At first, bridges know nothing about device locations, so they flood unknown-destination frames to all ports. Over time, they use backward learning by observing the source addresses of frames to build a table that maps each device to a specific port. This helps send future frames only where needed, reducing traffic.

Bridges also adapt to changes in the network. If a device is moved or unplugged, its table entry is removed after some time. When the device becomes active again, the bridge learns its new location automatically. This self-learning and updating makes bridges efficient, flexible, and ideal for managing dynamic networks.

This section explains how bridges forward Ethernet frames using a simple 3-step algorithm:

1. If the destination port is the same as the source port, the bridge discards the frame since it is already on the correct segment.
2. If the destination port is different, the bridge forwards the frame to that port.
3. If the destination is unknown, the bridge uses flooding, sending the frame to all ports except the incoming one.

This logic even applies when hubs are used, as shown in the example where stations E and F are connected via hub H1 to bridge B2. If E sends a frame to F, the hub sends it both to F and to bridge B2. Since the frame already reached the right segment, B2 discards it.

The forwarding process is typically implemented in hardware using special-purpose VLSI chips to ensure very fast decisions. This allows cut-through switching, where forwarding can start as soon as the destination address is read—before the full frame is received—reducing delay and buffering needs.

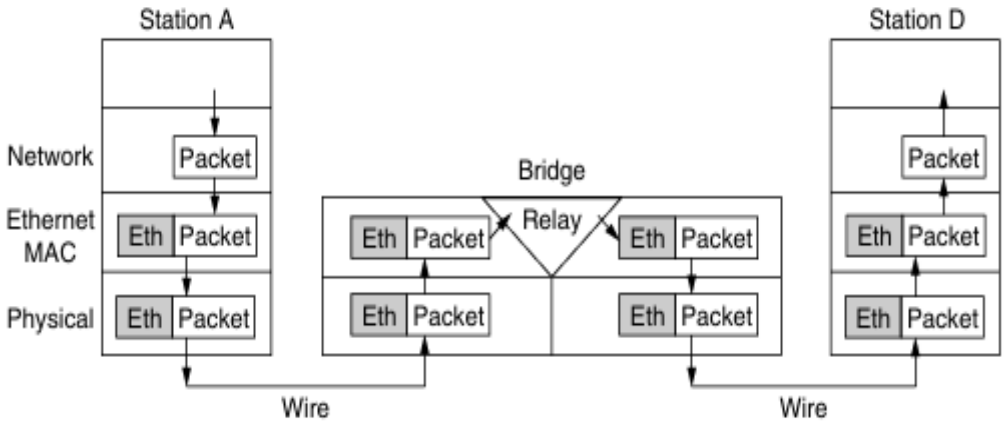


Figure 13.2 Protocol processing at a bridge

Bridges function strictly at the data link layer. They only examine Ethernet MAC headers, never the contents (like IP headers), preserving the integrity of the protocol stack. A bridge with k ports maintains k instances of MAC and physical layers. In the protocol stack view, a frame from station A to D is received by the bridge at its MAC layer, processed, and then sent out via the MAC layer of the appropriate output port, without altering the higher-layer content.

13.4 SPANNING TREE BRIDGES

Redundant links between bridges improve network reliability by providing backup paths if one link fails. However, they can also create loops in the network. When a frame is sent to an unknown destination, bridges flood it across all ports. If multiple paths exist, the same frame may circulate endlessly, causing a broadcast storm.

To solve this, the Spanning Tree Protocol (STP) is used. STP disables some redundant links to form a loop-free logical topology called a spanning tree, ensuring only one path exists between any two devices. Bridges communicate using special configuration messages to agree on this tree. Once established, only the active links in the tree are used for forwarding, preventing loops and ensuring stable communication.

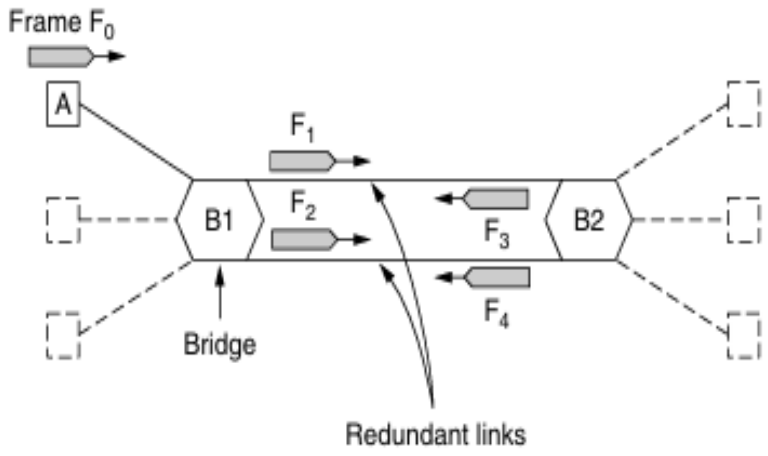


Figure 13.3 Bridges with two parallel links

To build a spanning tree, all bridges must first agree on one bridge to serve as the root. Each bridge sends configuration messages containing its own identifier (based on its MAC address) and the identifier of the bridge it currently believes to be the root. Since MAC addresses are globally unique, they serve as reliable identifiers. The bridge with the lowest MAC address becomes the root. In the example, bridge B1 has the lowest identifier and is chosen as the root.

Once the root is selected, each bridge calculates the shortest path to the root. For ties—where two paths are equal in length—the path through the bridge with the lower identifier is selected. Bridges then disable any ports not on the shortest path to the root, effectively removing redundant links to prevent loops. Although all bridges are part of the spanning tree, not all links are used.

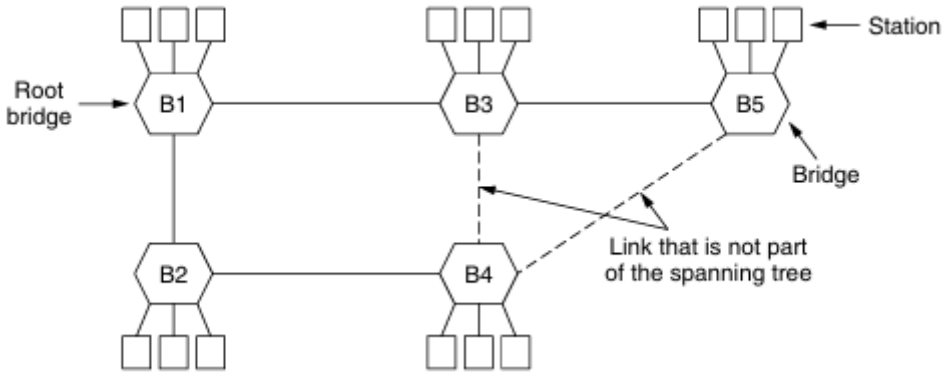


Figure 13.4 A spanning tree connecting five bridges. The dashed lines are links that are not part of the spanning tree.

The algorithm continues running even after the tree is built to adapt to network changes. This Spanning Tree Algorithm was invented by Radia Perlman, who famously wrote a poem about it. The algorithm became an industry standard as IEEE 802.1D, and later revisions improved its speed in reacting to topology changes.

13.5 REPEATERS, HUBS, BRIDGES, SWITCHES, ROUTERS AND GATEWAYS

This section compares different network devices—repeaters, hubs, bridges, switches, routers, and gateways—based on the layer of the OSI model in which they operate and the type of data they handle. Each device functions at a different layer, using specific information to make forwarding decisions.

At the physical layer, we find repeaters and hubs. A repeater is an analog device that amplifies and regenerates electrical signals, allowing data to travel longer distances. It does not understand frames or addresses—it simply boosts the signal. A hub is slightly more advanced, connecting multiple lines and broadcasting incoming frames to all other ports. Like repeaters, hubs don't examine or use link-layer addresses, and they also work at the physical layer.

At the data link layer, we find bridges and switches. These devices understand frame structures and use MAC addresses to forward frames to the correct port. Unlike hubs, bridges isolate traffic, reducing collisions and allowing different speeds on different ports. They can

buffer incoming frames and forward them as needed, but this buffering may run into limitations if traffic overloads the bridge. Switches are essentially high-speed bridges with many ports and are commonly used in modern LANs.

Bridges were initially intended to connect different types of LANs, like Ethernet and Token Ring. However, this proved difficult due to incompatible frame formats, different frame lengths, and reformatting requirements, which often caused data loss or errors. In practice, bridges work best when connecting similar LANs.

Network devices like repeaters, hubs, bridges, switches, routers, and gateways differ based on the OSI layer they operate in and how they handle data. Understanding these layers helps explain their functions and decisions in forwarding data.

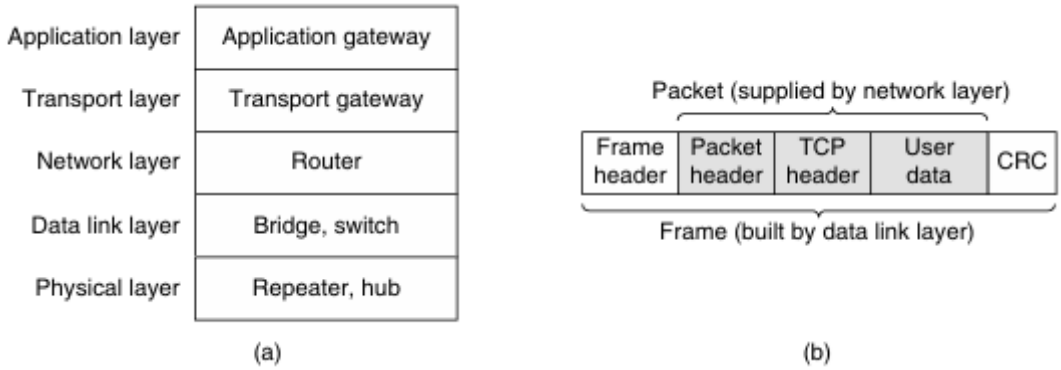


Figure 13.5 (a) Which device is in which layer (b) Frames, packets, and headers

Repeaters and hubs operate at the physical layer. A repeater simply amplifies and regenerates electrical signals to extend the distance a signal can travel. Hubs are similar but connect multiple lines and broadcast incoming signals to all ports. Both devices do not understand or use frame or address information—they only deal with raw signals.

Bridges and switches function at the data link layer. They can read frame headers and use MAC addresses to forward frames to the correct destination port, reducing unnecessary traffic. Switches are high-speed, multi-port bridges commonly used in modern networks. These devices isolate traffic between ports and can handle different network speeds. However, if data arrives faster than it can be sent out, they may run out of buffer space and start dropping frames.

Bridges were initially designed to connect different types of LANs, like Ethernet and Token Ring. However, due to issues like incompatible frame formats and maximum frame sizes, this approach was unreliable. In practice, bridges are best used to connect similar LANs.

13.6 VIRTUAL LANs

In the early days, LANs were set up based on physical layout—computers were connected as they were physically close, regardless of organizational structure. Later, with twisted pair cables and centralized hubs (later switches), it became possible to configure LANs based on logical groupings like departments.

This logical setup is useful for several reasons. Security is improved by isolating sensitive departments (like HR) from public servers. Load management becomes easier, as heavy users (like researchers) can be kept separate from others to prevent network slowdowns. Broadcast traffic is also reduced, which improves performance, since broadcasts go to all devices on a LAN and grow with network size.

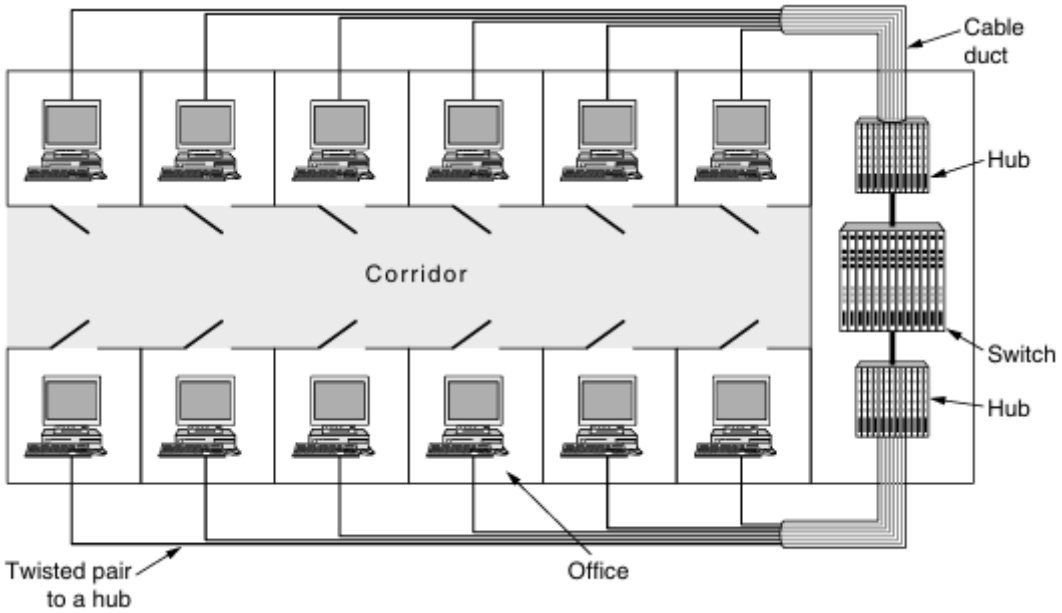


Figure 13.6 A building with centralized wiring using hubs and a switch

However, even with bridges and switches, some broadcasts still need to pass between LANs to maintain transparency. So, careful LAN design is important to keep networks secure, efficient, and stable.

This section introduces the concept of VLANs (Virtual LANs), which were developed to solve the limitations of organizing networks purely based on physical layout. In traditional LAN setups, computers were grouped by physical location, but as companies grew and changed, this became inefficient. For example, employees from the same department might be in different buildings, or departments might grow or shrink, creating mismatches with available switch ports. Constantly replugging cables to reassign LANs became a burden for administrators.

To address this, VLANs allow logical grouping of computers across physical locations using software, rather than physical cabling. Using VLAN-aware switches, administrators can assign each port to a specific VLAN. These VLANs can be configured based on department, function, or other logical groupings—regardless of where the machines are physically located. This simplifies reorganization, improves flexibility, and enhances management.

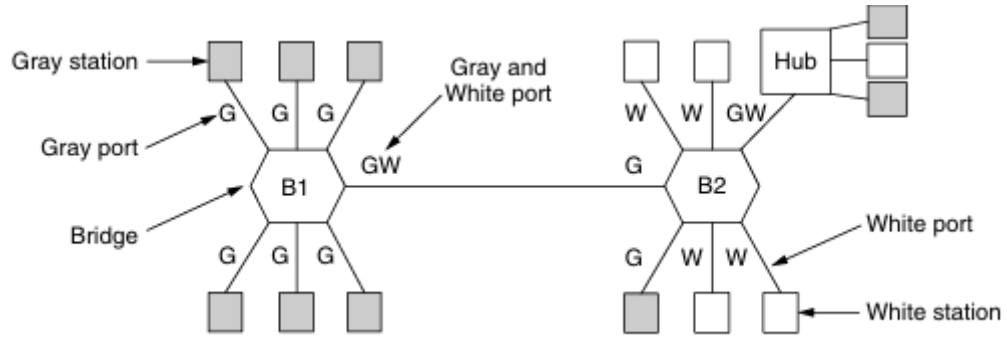


Figure 13.7 Two VLANs, gray and white, on a bridged LAN

In a VLAN setup, when a frame is received on a port, the switch forwards it only to ports that belong to the same VLAN. For example, in the illustrated network, machines in the gray VLAN (G) and white VLAN (W) are spread across different switches. When a gray machine sends a frame, it is only forwarded to ports marked "G," preventing it from reaching white VLAN machines. This separation ensures traffic isolation and efficient broadcast control, all while allowing centralized software-based network management.

The IEEE 802.1Q standards

To implement VLANs, network devices need a way to identify which VLAN each Ethernet frame belongs to. Since the original Ethernet frame format didn't include any space for VLAN information, the IEEE introduced the 802.1Q standard in 1998. This standard adds a special VLAN tag inside the Ethernet frame header. This tag tells VLAN-aware switches which VLAN the frame is part of, allowing them to properly separate and forward traffic based on VLAN membership.

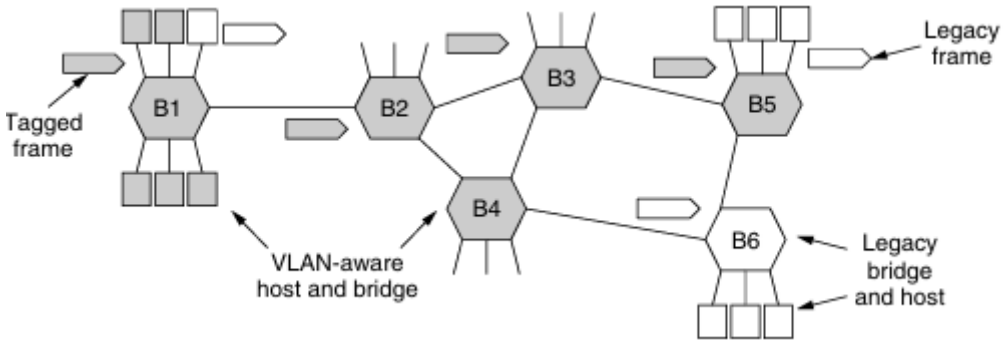


Figure 13.8 Bridged LAN that is only partly VLAN aware. The shaded symbols are VLAN aware. The empty ones are not.

Importantly, existing Ethernet hardware didn't need to be discarded because only VLAN-aware switches add and interpret these tags. Frames traveling between VLAN-aware switches carry these tags, but frames sent to regular (legacy) devices do not include them, so those devices continue to function normally.

The VLAN tag slightly increases the maximum Ethernet frame size from 1518 bytes to 1522 bytes, but only VLAN-aware devices must support this larger size. This approach allows organizations to introduce VLANs incrementally and maintain compatibility with older devices, making VLAN deployment practical and flexible.

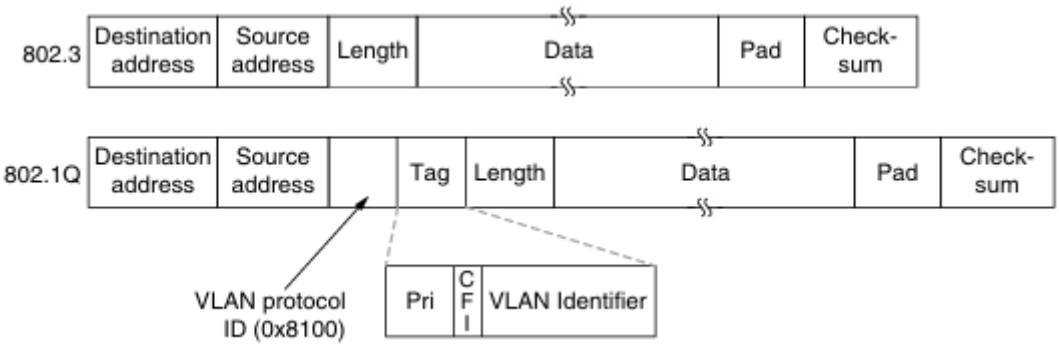


Figure 13.9 The 802.3 (legacy) and 802.1Q Ethernet frame formats

The 802.1Q VLAN tag adds a second 2-byte field with three parts. The most important is the 12-bit VLAN identifier, which marks the frame's VLAN "color" so switches know how to forward it properly.

There's also a 3-bit priority field for quality of service (QoS), helping prioritize time-sensitive traffic like voice or video over regular data.

The last part is the Canonical Format Indicator (CFI), originally meant to indicate bit order in MAC addresses but now mostly signals special token ring frames carried inside Ethernet. This part isn't related to VLANs but got included due to standards politics.

When a tagged frame reaches a VLAN-aware switch, the switch uses the VLAN ID to look up which ports to forward it to. These forwarding tables aren't manually set but are learned dynamically by observing incoming tagged frames (e.g., a VLAN 4 frame arriving on port 3 tells the switch VLAN 4 is reachable via port 3).

Interestingly, VLANs add a connection-like element to Ethernet switching: forwarding is based on VLAN IDs (like connection labels) rather than just destination addresses, marking a shift toward connection-oriented behavior in an otherwise connectionless network.

13.7 SUMMARY

This chapter focuses on data link layer switching and the devices that facilitate communication between networks. Bridges connect different LAN segments, using spanning tree protocols to prevent loops. Remote bridges extend connections over larger distances. The chapter also distinguishes common networking devices: repeaters regenerate signals, hubs broadcast frames, switches forward frames intelligently, routers direct traffic across networks, and gateways translate protocols. Virtual LANs (VLANs) allow logical segmentation of networks, improving security, efficiency, and management within LAN environments

13.8 TECHNICAL TERMS

Bridges, Virtual LANs, switches, routers, frames, network traffic

13.9 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain how bridges connect different 802.x LANs and support local internetworking.
2. Describe the spanning tree protocol and its importance in preventing loops.
3. Explain the functions of repeaters, hubs, bridges, switches, routers, and gateways.
4. Discuss remote bridges and their role in wide-area networking.
5. Explain the concept, benefits, and working of Virtual LANs (VLANs).

Short Questions:

1. What is a data link layer switch?
2. Define a network bridge.
3. What is the purpose of the spanning tree algorithm?
4. Name two network devices used to connect LANs.
5. What is a VLAN?

13.10 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008)

Dr. Vasantha Rudramalla

LESSON- 14

THE NETWORK LAYER DESIGN ISSUES

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the key design issues of the network layer.
- Learn the concept of store-and-forward packet switching.
- Describe the services provided to the transport layer.
- Understand implementation of connectionless and connection-oriented services.
- Compare virtual circuit and datagram subnet approaches

STRUCTURE OF THE LESSON:

14.1 STORE AND FORWARD PACKET SWITCHING

The network layer and its functions, it is important to understand the broader environment in which network layer protocols operate. This environment typically consists of two major components: the Internet Service Provider's (ISP's) infrastructure and the customer's equipment. These components are often depicted in diagrams such as Figure 14.1, where the ISP's network is represented inside a shaded oval, encompassing several routers interconnected via transmission lines. On the other hand, customer devices like personal computers or local networks are located outside this oval.

For instance, consider two hosts: H1 and H2. Host H1 could be a simple home computer directly connected to one of the ISP's routers (labeled A) through a DSL modem. This type of connection typically represents a residential broadband setup. In contrast, Host H2 might be part of a more complex local area network (LAN), such as an office Ethernet. This LAN is connected to a router (labeled F), which is owned and managed by the customer. Router F, although it is physically located on the customer's premises and managed by them, is considered part of the ISP's network for the purposes of studying network layer behavior. This is because router F runs the same routing algorithms and protocols as the routers owned by the ISP.

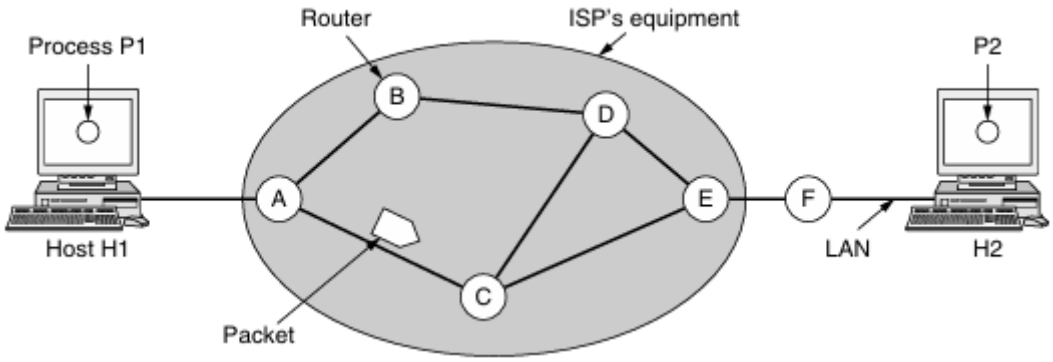


Figure 14.1 The environment of the network layer protocols

The communication process begins when a host, such as H1 or H2, wants to send a data packet. The packet is first sent to the nearest router. This can happen either through a point-to-point link—like the connection between H1 and router A—or via a shared LAN as is the case with H2. Once the packet arrives at the router, it undergoes a process called store-and-forward packet switching. In this mechanism, the router waits for the entire packet to arrive before it processes it. One key step in this process includes verifying the packet's checksum to ensure it hasn't been corrupted during transmission.

After the verification is complete, the router determines the next hop based on its routing table and forwards the packet to the next router along the path toward the destination. This process continues from router to router until the packet finally reaches the destination host, such as H2. The store-and-forward model, therefore, involves temporary storage at each hop and makes use of routing algorithms that determine the optimal path through the network.
By examining this setup, we gain a clearer understanding of how the network layer functions in real-world scenarios, handling responsibilities such as routing, addressing, and packet forwarding across a distributed infrastructure involving both ISP-controlled and customer-managed devices

14.2 SERVICES PROVIDED TO THE TRANSPORT LAYER

The network layer plays a crucial role in the overall architecture of computer networks by providing essential services to the transport layer. The interface between these two layers—the network layer/transport layer interface—is where the services of the network layer become visible to the transport layer. One of the key design questions in networking is determining what kind of services the network layer should offer to the transport layer. To answer this, designers must keep several critical goals in mind while creating the network layer's service specifications.

First and foremost, the services should be independent of the underlying router technology. This means the services must work uniformly regardless of the internal workings or designs of the routers in the network. Secondly, the transport layer should remain unaware of the number, types, or interconnections (topology) of routers present in the network. It should see only a simple, abstract service and not have to deal with the network's internal complexity. Thirdly, the network addressing scheme must be consistent and uniform, regardless of whether the communication occurs over a local area network (LAN) or a wide area network

(WAN). A uniform addressing approach simplifies the design of transport protocols and makes them more portable across different network environments.

Given these design principles, network architects have significant flexibility in defining the specific services that the network layer offers. However, this freedom has historically led to deep disagreement among experts, resulting in a long-standing debate between two major schools of thought. The central issue is whether the network layer should provide connection-oriented services or connectionless services.

One faction, often associated with the Internet community, advocates for a connectionless model. According to this group, the main responsibility of the network layer is simply to deliver packets from one point to another. Based on decades of practical experience with the Internet, they argue that networks are naturally unreliable and that attempts to make them perfectly reliable are often futile. Therefore, the hosts themselves should handle responsibilities like error detection, error correction, and flow control. From this viewpoint, the network layer should only offer basic primitives such as SEND PACKET and RECEIVE PACKET, without guarantees about delivery order or reliability. Since each packet is treated independently, it must carry the full destination address to ensure correct delivery, regardless of what came before or after it.

This approach exemplifies the well-known end-to-end argument, a fundamental design principle in network architecture. The end-to-end argument suggests that certain functions, such as reliability and security, are best implemented at the endpoints of a communication system (i.e., the hosts), rather than in the intermediate network. This principle has profoundly influenced the design of the modern Internet.

In contrast, the opposing faction—traditionally represented by telephone companies—advocates for a connection-oriented network service. Drawing on over a century of experience with the reliable global telephone system, they argue that providing a stable, predictable, and high-quality service requires explicit connections within the network. This is especially true for real-time applications like voice and video, where packet delay, jitter, and loss can severely impact performance. In their view, quality of service (QoS) is paramount, and QoS is difficult to guarantee without some form of connection establishment and management.

This debate has persisted for decades. In the early years of networking, many widely used protocols like X.25 (from the 1970s) and Frame Relay (from the 1980s) followed the connection-oriented model. However, over time, the connectionless model gained massive popularity, particularly due to the rise of the ARPANET and the Internet. The Internet Protocol (IP), which embodies the connectionless philosophy, became a universal standard and a symbol of success. Despite the introduction of advanced, connection-oriented technologies like ATM (Asynchronous Transfer Mode) in the 1980s—which was expected to replace IP—IP continued to thrive, while ATM faded into niche applications.

Interestingly, the story does not end there. As modern applications increasingly demand guaranteed performance, even the Internet, based on a connectionless model, has started adopting connection-oriented features. Examples of this include technologies like MPLS (MultiProtocol Label Switching) and VLANs (Virtual Local Area Networks). These solutions aim to combine the flexibility and scalability of connectionless networking with the performance guarantees typically associated with connection-oriented systems. This hybrid

evolution illustrates how the networking world continues to seek a balance between reliability, simplicity, and performance.

1. Connectionless service (Internet view):

- The network's job is simply to move packets, and the network itself is unreliable.
- Error control and flow control are handled by the hosts (end systems), not the network.
- The network layer offers simple primitives like SEND PACKET and RECEIVE PACKET, with no guarantees about ordering or delivery.
- Each packet carries a full destination address and is routed independently (no connections).
- This approach follows the end-to-end argument—tasks are best handled at the endpoints, not inside the network.

2. Connection-oriented service (Telephone companies' view):

- The network should provide reliable, connection-based services.
- Connections help ensure quality of service (QoS), which is important for real-time traffic like voice and video.
- This view is based on the success of the telephone system, which uses connections to manage calls reliably.

Historically, early networks like X.25 and Frame Relay used connection-oriented models. However, the Internet's connectionless IP protocol has become dominant and successful, even overtaking some connection-oriented technologies like ATM.

Still, modern networks are incorporating connection-oriented features (e.g., MPLS and VLANs) to improve QoS while keeping the Internet's basic connectionless nature.

In short, the network layer balances between simple, flexible connectionless service and more complex, reliable connection-oriented service depending on needs and technology trends.

14.3 IMPLEMENTATION OF CONNECTIONLESS SERVICES

After exploring the two major types of services that the network layer can offer—connectionless and connection-oriented—it becomes necessary to understand how the network layer operates internally. The internal organization of the network layer depends heavily on the kind of service it provides. In a connectionless network, packets are transmitted into the network individually, with each one being routed independently of the others. There is no need to establish a pre-arranged path before communication begins. In such a system, the packets are referred to as datagrams, drawing a parallel with telegrams, which are also delivered individually and possibly out of order. Networks that operate this way are called datagram networks.

On the other hand, if the network layer offers connection-oriented service, it requires the establishment of a specific path—called a Virtual Circuit (VC)—from the source to the destination before any actual data is transmitted. This setup resembles the way telephone systems operate by setting up a physical circuit before a call begins. Networks using this

approach are known as virtual-circuit networks. In this section, the focus is on how datagram networks function, and the discussion of virtual-circuit networks follows afterward.

To illustrate the working of a datagram network, consider a scenario shown in Figure 14.2. Suppose process P1 on host H1 wants to send a long message to process P2 on host H2. P1 passes the message to the transport layer, specifying that it should be delivered to P2 on H2. The transport layer, which typically operates within the operating system on H1, adds a transport header to the message and hands it over to the network layer.

Assume that the original message is four times longer than the maximum allowable packet size in the network. Consequently, the network layer must fragment the message into four separate packets: 1, 2, 3, and 4. These packets are then transmitted individually to router A using a point-to-point protocol such as PPP (Point-to-Point Protocol). At this point, the ISP's routers take control of the packet forwarding process.

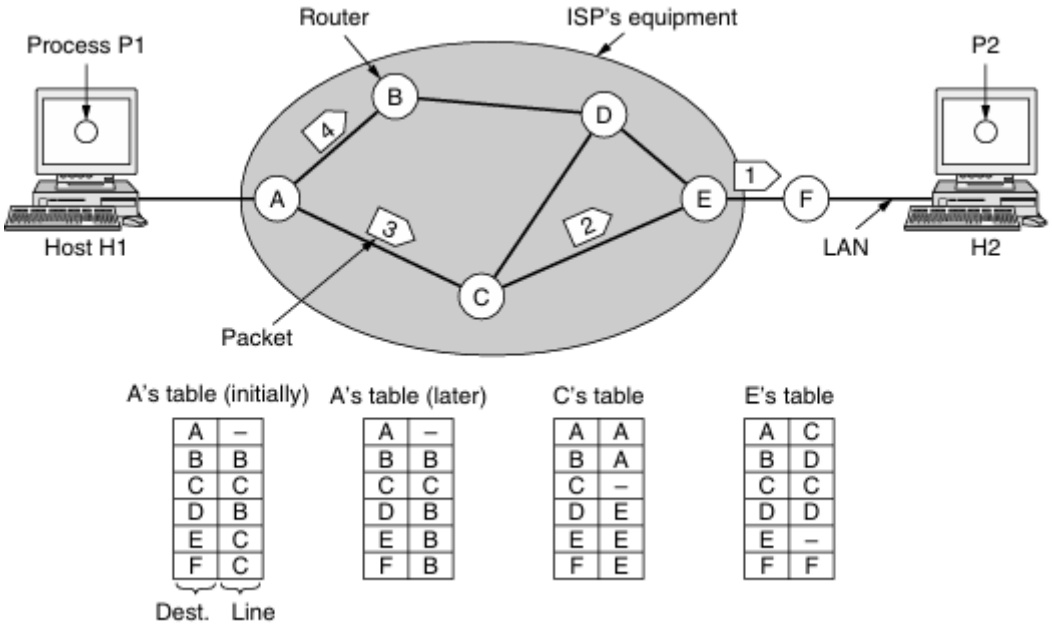


Figure 14.2 Routing within a datagram network

Each router in the network maintains an internal routing table that indicates where to send packets based on their destination address. Every entry in the routing table is a pair consisting of a destination and an outgoing line to reach that destination. Importantly, routers can only send packets over lines to which they are directly connected. For example, in the network depicted in Figure 14.2, router A has two outgoing links—one to router B and one to router C. Therefore, any packet arriving at A must be forwarded through one of these two routers, even if its final destination is much further away.

As packets 1, 2, and 3 arrive at router A, they are briefly stored so their checksums can be verified to ensure they were not corrupted during transmission. After verification, router A forwards each packet based on its routing table. Initially, A's table directs these packets to router C. From C, the packets proceed to router E, then to router F, and finally across the LAN to host H2, where process P2 resides. All three packets follow this same path.

However, when packet 4 arrives at router A, something different happens. Instead of forwarding it through the path used by the previous packets, A sends packet 4 to router B. This change in routing behavior may have occurred because router A updated its routing table after discovering congestion or failure on the previous path (ACE). This example highlights one of the key features of datagram networks: routing decisions are made independently for each packet, and routes can change dynamically. This flexibility helps networks adapt to real-time conditions such as traffic jams or failed links.

The logic that determines how routers update and use their routing tables is governed by the routing algorithm, which is a central topic in the study of the network layer. Different routing algorithms have different properties, goals, and behaviors, and they form the backbone of efficient data delivery in packet-switched networks.

A real-world example of a datagram-based connectionless service is the Internet Protocol (IP), which underpins the entire Internet. In IP networks, each packet carries its own destination address, allowing routers to process and forward it independently. IP comes in two major versions: IPv4, which uses 32-bit addresses, and IPv6, which uses 128-bit addresses to accommodate the growing number of devices on the Internet. The detailed mechanics of IP will be discussed later in the chapter.

14.4 IMPLEMENTATION OF CONNECTION-ORIENTED SERVICES

To provide connection-oriented service, the network layer must operate using a virtual-circuit network. Unlike the connectionless model where each packet finds its own route through the network, virtual circuits are designed to establish a fixed path between the source and destination before any data packets are transmitted. This means that during the connection setup phase, the routers involved determine and store a route in their routing tables, which will be used for all packets in that communication session. Once the communication is complete and the connection is released, the virtual circuit is terminated, and the related entries in the routers' tables are discarded. This model is very similar to how traditional telephone systems work.

In a virtual-circuit network, each packet does not carry a full destination address. Instead, it includes a connection identifier that refers to the established path. Each router uses this identifier to forward the packet along the predefined route. This approach reduces overhead and speeds up routing, since the path is fixed and known in advance.

To understand this mechanism, consider the example illustrated in Figure 14.3. Suppose host H1 wants to communicate with host H2. It begins by establishing a connection—referred to as connection 1. This connection is then recorded in the routing tables of all routers along the path from H1 to H2. For example, router A's table contains an entry stating that if it receives a packet with connection ID 1 from H1, it should forward that packet to router C, keeping the same connection ID. Similarly, router C forwards the packet to E, and from E it finally reaches F and then H2.

However, now suppose another host, H3, also wants to establish a connection to H2. Since H3 is just initiating communication, it selects connection ID 1 for its side of the communication. This introduces a potential conflict, because routers such as C may now receive multiple packets with the same connection identifier (1), one from H1 and another from H3. Since the connection ID must be unique on each incoming interface but can differ

across routers, this problem is resolved by allowing routers to assign new identifiers to outgoing traffic. In this case, when the packet from H3 passes through router A, A changes the identifier from 1 to a different value (say, 2) before forwarding it to C. This method enables routers to avoid conflicts by performing label translation at each hop.

This capability of assigning and switching labels (or identifiers) as packets move from one hop to the next is often referred to as label switching. A widely used example of a label switching, connection-oriented technology is MPLS (MultiProtocol Label Switching). MPLS is commonly used within ISP networks and works by attaching a 20-bit label to IP packets. These labels serve as connection identifiers, allowing for fast, efficient routing based on predetermined paths.

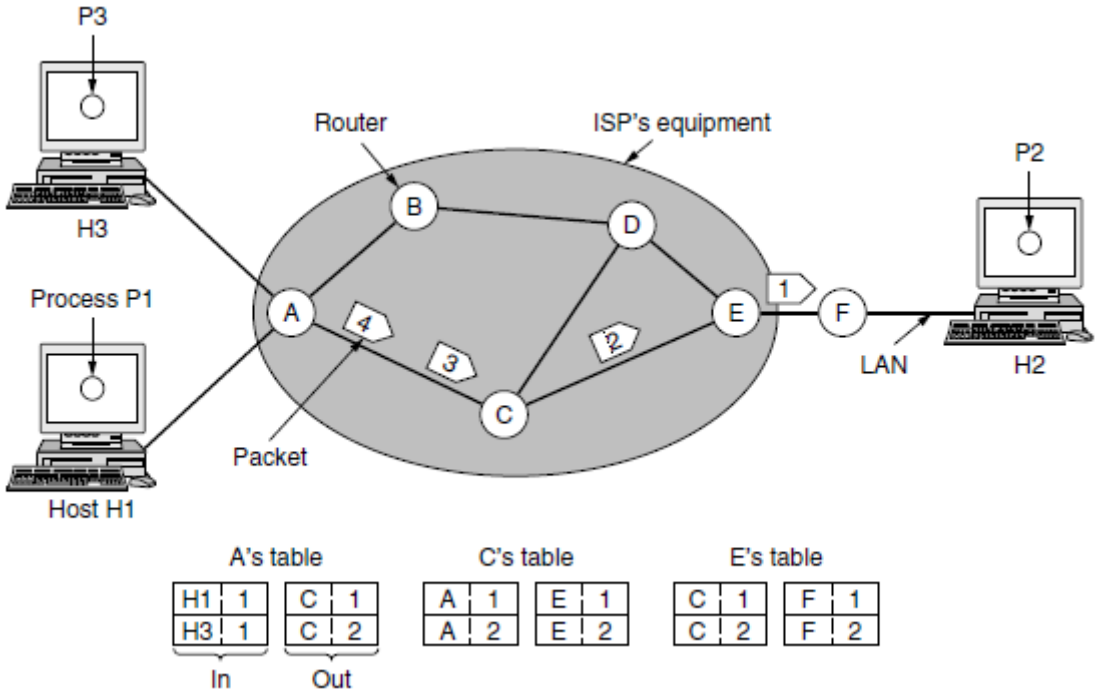


Figure 14.3 Routing within a virtual circuit network

Although MPLS operates beneath the traditional IP layer and is often invisible to end users, it is a powerful tool for traffic engineering and quality of service (QoS) management within service provider networks. ISPs can set up long-lived MPLS paths to handle high-volume or delay-sensitive traffic (such as voice and video), ensuring better performance and network efficiency. While we will explore the technical details of MPLS later in the chapter, it is important to note here that virtual-circuit techniques like MPLS are essential when guaranteed service quality and efficient traffic management are required.

14.5 COMPARISION OF VIRTUAL CIRCUITS AND DATAGRAM NETWORKS

Both datagram and virtual-circuit networks are widely used in computer networking, and each approach has its own set of advantages and drawbacks. The debate between their supporters centers on how each handles routing, addressing, setup, fault tolerance, and quality of service. Understanding the trade-offs between these two methods is crucial for network designers and architects who need to select the most appropriate model for their applications.

Issue	Datagram network	Virtual-circuit network
Circuit setup	Not needed	Required
Addressing	Each packet contains the full source and destination address	Each packet contains a short VC number
State information	Routers do not hold state information about connections	Each VC requires router table space per connection
Routing	Each packet is routed independently	Route chosen when VC is set up; all packets follow it
Effect of router failures	None, except for packets lost during the crash	All VCs that passed through the failed router are terminated
Quality of service	Difficult	Easy if enough resources can be allocated in advance for each VC
Congestion control	Difficult	Easy if enough resources can be allocated in advance for each VC

Figure 14.4 Comparison of datagram and virtual-circuit networks

One of the fundamental differences lies in the need for circuit setup. In datagram networks, no setup is required before sending packets. Each packet is treated independently and routed dynamically through the network based on its destination address. In contrast, virtual-circuit networks require an initial setup phase, during which a fixed path from the source to the destination is established. This path remains in place for the duration of the communication, and all packets follow it in sequence. Once the connection is complete, the virtual circuit is terminated, and the allocated resources are released.

The way packets carry addressing information also differs significantly. In a datagram network, each packet must carry the complete source and destination address since it may take a different path than other packets from the same message. This adds to packet overhead, particularly when packets are small. On the other hand, virtual-circuit networks require only a short virtual circuit identifier to be included in the packet header. This identifier is used by routers to forward the packet along the pre-established route, reducing header size and simplifying processing.

Another key difference is the handling of state information by routers. In datagram networks, routers remain stateless; they do not maintain any record of ongoing connections. Each routing decision is made afresh for every packet. However, in virtual-circuit networks, routers must store state information for every active connection. This state includes the mapping between incoming and outgoing virtual circuit numbers and interfaces, consuming memory and processing resources.

Routing behavior further distinguishes the two approaches. In a datagram network, routing decisions are made independently for each packet, allowing for dynamic adjustment to changing network conditions like congestion or link failure. In contrast, virtual-circuit networks use static routing for the duration of a connection, as the path is fixed during setup. While this improves consistency and ordering, it reduces flexibility.

One of the major strengths of datagram networks is their resilience to router failures. If a router crashes, only the packets currently being processed by that router may be lost. New packets can be rerouted dynamically through alternate paths. In virtual-circuit networks, however, a single router failure can terminate all virtual circuits passing through it, disrupting ongoing communications. Even if the router comes back online quickly, the connection information stored in its memory is lost.

When it comes to quality of service (QoS) and congestion control, virtual-circuit networks offer better capabilities. Since a path is established before communication begins, resources such as buffer space, bandwidth, and CPU cycles can be reserved in advance. This makes it easier to guarantee service levels for applications like video streaming or VoIP. Datagram networks, with their lack of prior setup, struggle to offer similar guarantees, making QoS management more difficult and often reactive rather than proactive.

There are additional practical considerations as well. In virtual-circuit networks, the setup phase introduces some delay and consumes processing time, but packet forwarding becomes faster and more efficient once the connection is established. In datagram networks, there is no initial delay, but each packet requires more complex address lookup, which increases per-packet processing time. Moreover, longer addresses in datagram packets contribute to higher bandwidth usage, particularly in networks where packet sizes are small.

Another aspect is router memory usage. Datagram routers must maintain routing entries for all possible destinations, which could be a large number. Virtual-circuit routers only need entries for active connections, which could be fewer, depending on the traffic pattern. However, this perceived advantage can be misleading because even virtual-circuit setup messages require full destination addresses to be routed initially—just like datagrams.

In terms of application suitability, datagram networks are well-suited for short, bursty communication where setting up and tearing down a connection would be inefficient. For example, in credit card verification systems where each transaction is brief, the overhead of a virtual circuit would outweigh the benefits. On the other hand, long-term communications, such as VPNs connecting branch offices, benefit from virtual circuits. These can be established manually and maintained over long periods, offering stability and performance for sustained data flows.

Lastly, fault tolerance and traffic balancing also favor datagram networks. If a link or router fails, new routes can be computed immediately for subsequent packets, ensuring continuity. Also, because each packet is routed independently, traffic can be dynamically balanced across the network to avoid congestion. Virtual circuits lack this flexibility; they are tied to specific paths, and rerouting requires establishing a new connection, which can be disruptive.

14.6 SUMMARY

This chapter covers the network layer design issues, focusing on how data is delivered across networks. Store-and-forward packet switching ensures each packet is temporarily stored at intermediate nodes before forwarding. The network layer provides essential services to the transport layer, supporting either connectionless or connection-oriented communication. The chapter compares virtual circuit subnets, which establish a dedicated path, and datagram subnets, which route packets independently, highlighting their trade-offs in efficiency, reliability, and complexity

14.7 TECHNICAL TERMS

Switching, packet switching, connectionless communication, connection-oriented communication, Virtual circuit.

14.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the design issues of the network layer in detail.
2. Describe the concept and working of store-and-forward packet switching.
3. Discuss the implementation of connectionless services in the network layer.
4. Explain how connection-oriented services are implemented in the network layer.
5. Compare virtual circuit and datagram subnets in terms of performance and reliability.

Short Questions:

1. What is the main function of the network layer?
2. Define store-and-forward packet switching.
3. Name two services provided by the network layer to the transport layer.
4. What is a connectionless service?
5. What is a virtual circuit subnet?

14.9 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008)

Dr. Vasantha Rudramalla

LESSON- 15

ROUTING ALGORITHMS

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the key principles of routing in networks.
- Learn about shortest path and optimality in routing.
- Study distance vector and link-state routing algorithms.
- Understand hierarchical, broadcast, and multicast routing.
- Learn about routing challenges for mobile hosts.

STRUCTURE OF THE LESSON:

15.1 INTRODUCTION

The network layer's job is to route packets from a source to a destination—often across multiple hops—and forward each packet by consulting routing tables to determine the correct outgoing interface. In connectionless (datagram) networks, routing decisions are made for every packet to adapt to topology changes, while in connection-oriented (virtual-circuit) networks, a route is established once per session and packets follow that fixed path. Routing algorithms (e.g., link-state and distance-vector) build and maintain these tables for correctness, efficiency, and robustness, while forwarding is the actual per-packet lookup and send operation performed at line speed.

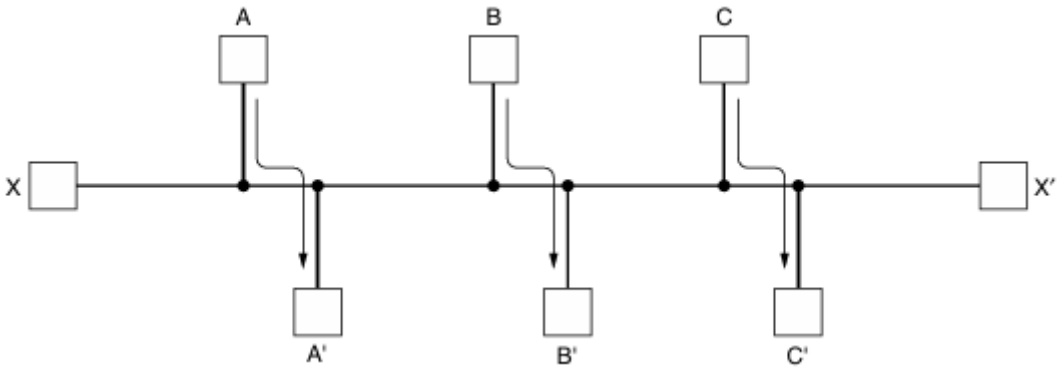


Figure 15.1 Network with a conflict between fairness and efficiency

A stable routing algorithm converges quickly to a consistent set of paths and remains there—avoiding endless fluctuations—because ongoing route instability can disrupt communication until equilibrium is reached. Fairness (treating all flows reasonably) often conflicts with efficiency (maximizing overall throughput); for example, shutting off a saturated flow may boost total traffic but be unfair to that flow. Simple solutions include optimizing hop counts or path length to strike a balance: fewer hops both reduce per-packet delay and free up bandwidth, improving throughput while maintaining relative fairness. Lastly, routing can be nonadaptive (static)—paths are preconfigured and never change—or adaptive (dynamic)—routes evolve in response to topology or traffic changes, using metrics like distance, delay, or load to adjust routing decisions.

15.2 THE OPTIMALITY PRINCIPLE

The optimality principle (Bellman, 1957) asserts that if router J lies on the optimal path from I to K, then the segment of that path from J to K must also be optimal; otherwise, a better overall path could be found—contradicting the assumption of optimality.

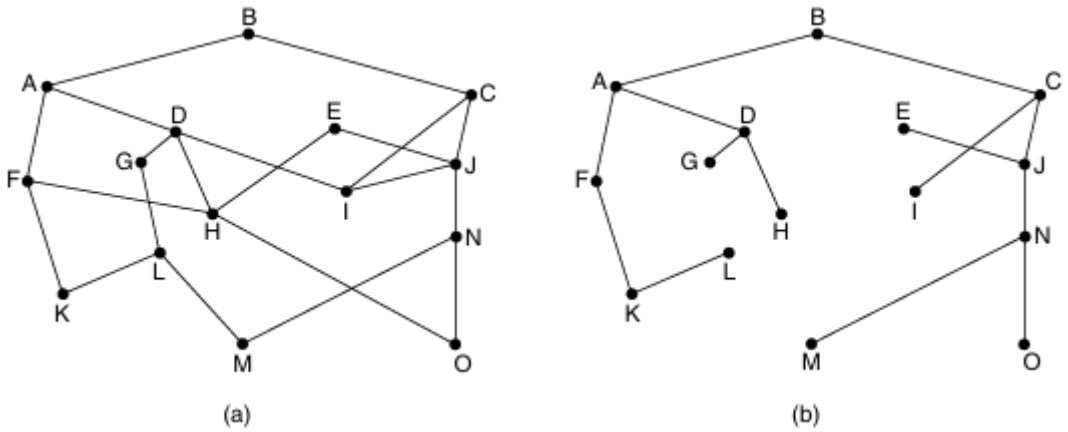


Figure 15.2 (a) Network (b) A sink tree for router B

Consequently, the collection of optimal paths from all sources to a particular destination forms a sink tree rooted at that destination—a loop-free structure ensuring finite delivery. While sink trees are not always unique (multiple equal-cost paths may exist, forming a DAG), they provide a standard benchmark for routing algorithms. In real networks, issues such as link failures, topology changes, and inconsistent knowledge across routers complicate discovering and maintaining these ideal structures.

15.3 SHORTEST PATH ALGORITHM

Dijkstra's algorithm is a classic method for computing shortest (optimal) paths in a fully known network: model routers as nodes and links as weighted edges (e.g., hops, delay, cost), initialize all distances from a source as infinity except the source itself (zero), then repeatedly select the unvisited node with the smallest tentative distance, mark it as permanent, and update its neighbours' distances if a shorter path is found via it. This greedy process continues until all nodes are settled, producing a shortest-path tree rooted at the source—perfectly aligning with the optimality principle and serving as the foundation for link-state routing protocols like OSPF and IS-IS.

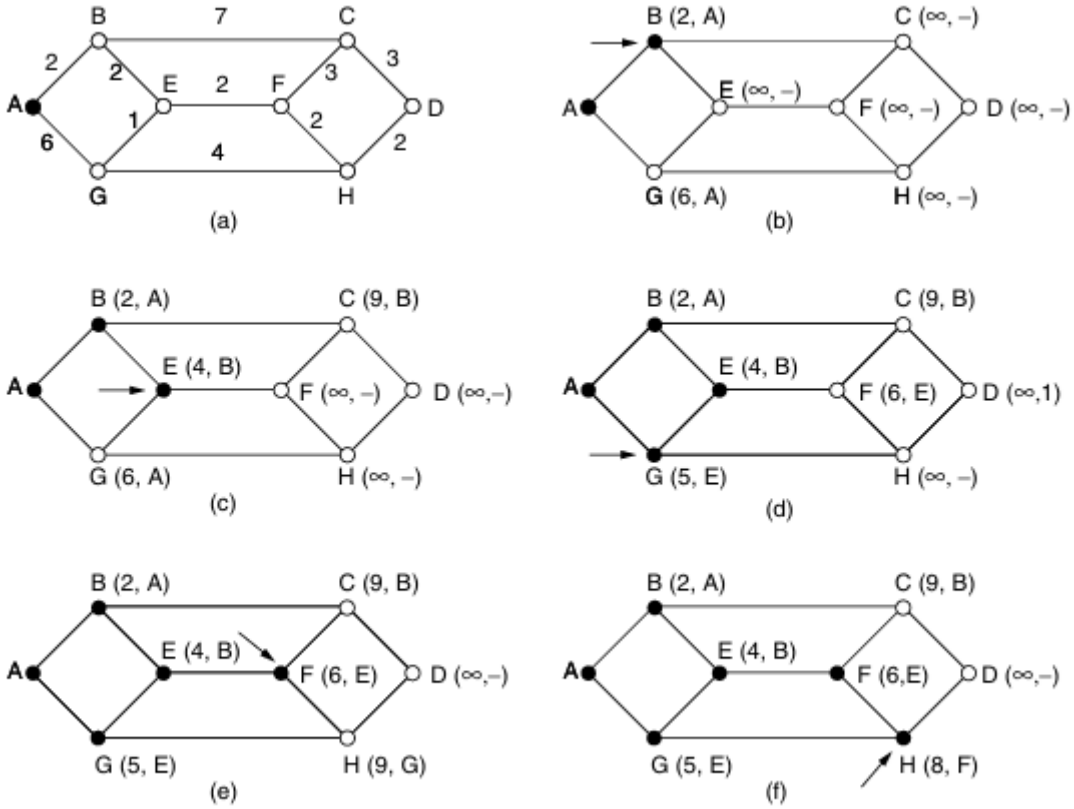


Figure 15.3 The first six steps used in computing the shortest path from A to D. The arrows indicate the working done.

Dijkstra's algorithm works like this: start at node A, mark it permanent, and tentatively label its neighbors (e.g. B, E) with their distances from A, noting A as their predecessor. Then, repeat: choose the tentatively labelled node with the smallest distance (say B), mark it permanent, and relax its neighbors by updating labels and predecessors if shorter paths are found through B. Continue picking the next smallest tentative node, making it permanent, and relaxing its neighbors until all nodes are permanent. By always extending the frontier via the closest tentative node, the algorithm ensures correctness—the key insight being that any shorter path to a permanent node would have already been found—ultimately producing a shortest-path tree (or sink tree) from A to every destination.

15.4 FLOODING

Flooding is a simple, local (nonadaptive) routing method where each router forwards every received packet on all outgoing links except the one it arrived on. To prevent infinite duplication, controlled flooding adds mechanisms such as a hop-count (TTL) that decrements at each hop until the packet is dropped, and sequence-number based suppression, where routers remember packets by (source, sequence) pairs and drop repeats. More advanced refinements include reverse-path forwarding (RPF)—forwarding only if the packet arrives on the router's shortest-path link back to the source—and selective flooding, which restricts forwarding to links roughly directed toward the destination. This method guarantees delivery along the shortest available path and is highly resilient, but it incurs substantial overhead in

bandwidth and packet processing, making it best suited for broadcasts, route discovery, or protocol updates rather than routine unicast forwarding.

Flooding is a simple, local, non-adaptive routing method where each router forwards every incoming packet on all outgoing links except the one it arrived on. Since uncontrolled flooding produces an infinite number of duplicates and broadcast storms, controlled flooding is used—techniques like hop-count (TTL) to limit packet lifespan, sequence-number suppression to discard duplicates, and reverse-path forwarding (RPF) to ensure packets only travel if they arrive via the router's shortest-path interface. A further refinement, selective flooding, restricts forwarding to links roughly pointing toward the destination. Flooding guarantees delivery over any existing path (often the shortest), requires minimal setup—routers need only know their neighbors—and is extremely robust, though it incurs high overhead in bandwidth and processing, making it best suited for broadcasts, route discovery, or protocol updates rather than routine unicast forwarding.

15.5 DISTANCE VECTOR ROUTING

Distance vector routing is a dynamic routing algorithm where each router maintains a table listing the best-known distance to every destination and the link to use to reach it. Routers exchange these tables with their immediate neighbors at regular intervals. Using the received information, each router updates its table by calculating the total distance to each destination via each neighbor and choosing the shortest one. This process continues until all routers have consistent and accurate routing information. The algorithm is based on the Bellman-Ford method and was used in early networks like ARPANET and in protocols like RIP.

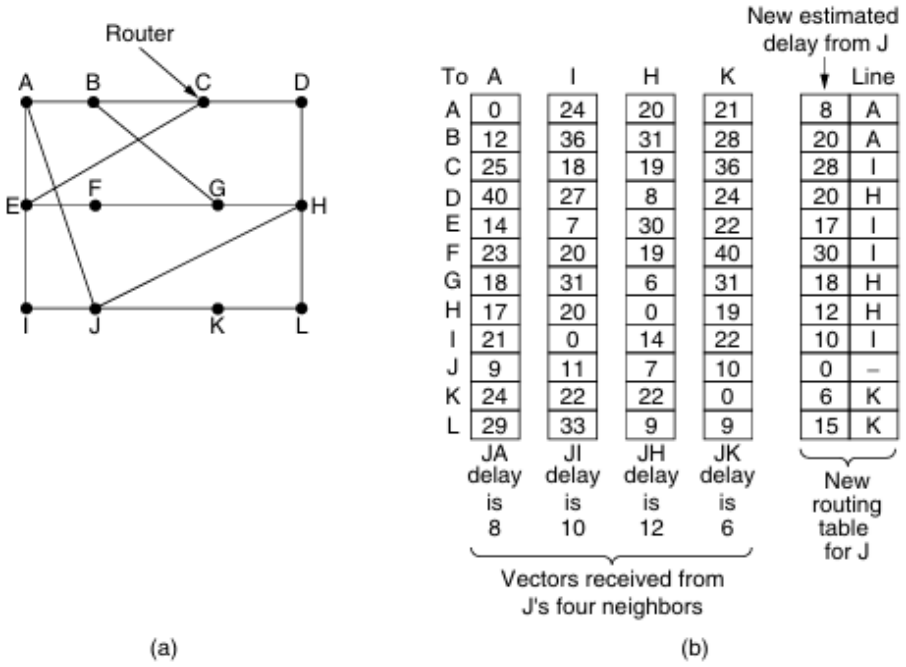


Figure 15.4 (a) A Network (b) Input from A, I, H, K and the new routing table for J

This picture describes how a router (J) updates its routing table using the distance vector routing algorithm. Router J receives delay information (distance vectors) from its neighbors A, I, H, and K. It also knows the delay to reach each neighbor: 8 ms to A, 10 ms to I, 12 ms to H, and 6 ms to K. To find the best route to a destination like router G, J calculates the total delay through each neighbor: via A it's 26 ms (8 + 18), via I it's 41 ms, via H it's 18 ms, and via K it's 37 ms. Since the lowest delay is through H (18 ms), J updates its table to show that the best route to G is via H with a delay of 18 ms. J repeats this process for all other destinations to build an updated routing table.

The count to infinity problem

The concept of convergence in distance vector routing, which is the process of routers gradually updating their tables until the best paths are known throughout the network. While this method is simple and effective, it has a major drawback: it responds quickly to improvements (good news), but slowly to failures or worse routes (bad news). For example, if a router suddenly hears from a neighbor that a destination is now closer, it quickly updates its table. However, if a destination becomes unreachable, the update spreads much more slowly. To illustrate this, consider a linear five-node network where node A is initially down and all routers have recorded infinite delay to A. When A comes back online, B (its immediate neighbor) detects this in the first vector exchange and updates its table to reflect that A is one hop away. However, the rest of the routers still think A is down until the updated information reaches them in subsequent exchanges. This shows how good news spreads quickly in one round, while bad news may take multiple rounds to fully propagate.

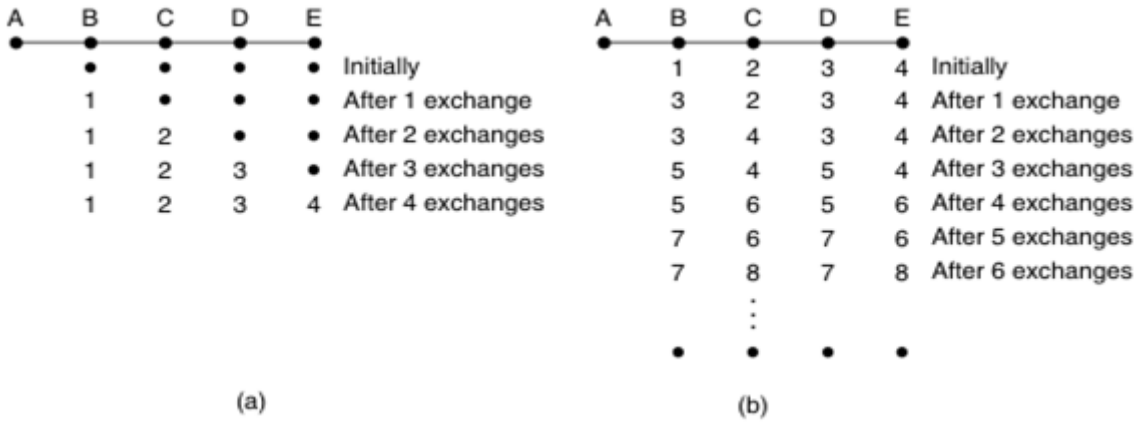


Figure 15.5 The count-to-infinity problem

The count-to-infinity problem occurs in distance vector routing when routers react slowly to bad news, such as a node or link failure. For example, if router A goes down, B may incorrectly believe A is still reachable via C—unaware that C's route actually loops back through B. This incorrect path gets passed along, with each router increasing the hop count by one in each exchange. Eventually, the count reaches "infinity," marking the destination as unreachable, but this can take many exchanges. The problem arises because routers lack full path awareness and update based only on neighbors' information. Techniques like split horizon with poisoned reverse try to help but don't fully solve the issue.

15.6 LINK STATE ROUTING

Distance vector routing was used in the ARPANET until 1979 but was eventually replaced by link state routing due to slow convergence caused by the count-to-infinity problem. Link state routing overcomes this issue by giving every router a complete view of the network.

It works in five steps:

(1) each router discovers its neighbour and their addresses
(2) measures the cost to each neighbor
(3) creates a packet with this information
(4) sends and receives such packets from all routers
(5) computes the shortest paths using Dijkstra's algorithm.

Modern protocols like OSPF and IS-IS are based on this method and are widely used in today's networks.

Learning about the neighbours

When a router starts up, its first job is to identify its neighbors. It does this by sending a HELLO packet on each link. The neighboring router responds with its unique name, which helps avoid confusion later when building the network map. In networks with broadcast LANs (like Ethernet), simply treating the LAN as many point-to-point links between routers would complicate the topology and increase message overhead. Instead, the LAN is modeled as a single virtual node (e.g., node N), with all connected routers linking to it. One router on the LAN is designated to represent this virtual node in the routing process. This way, connections like A to C via the LAN are represented as paths like A–N–C, simplifying link state routing on shared media.

Setting link costs

In link state routing, each link must have a cost metric to help routers find the shortest paths. This cost can be set manually or determined automatically. A common method is to make the cost inversely proportional to the link's bandwidth, so faster links (e.g., 1 Gbps) have lower costs than slower ones (e.g., 100 Mbps). In geographically large networks, link delay can also be included in the cost. Routers estimate delay by sending a special ECHO packet, which the neighbor immediately returns. By measuring the round-trip time and dividing it by two, the router estimates the link's delay and uses it in cost calculations.

Building link state packets

After a router gathers all the necessary data about its neighbors and link costs, it creates a link state packet. This packet includes the router's identity, a sequence number, an age, and a list of its neighbors along with the cost to reach each one. These packets are shared across the network to inform all routers of the current topology. Creating the packets is straightforward, but deciding when to create them is more challenging. They can be generated either periodically or when significant changes occur—like a link going down or coming back up, or if its cost changes notably.

Distributing the link state packets

In link state routing, once routers create their link state packets, they must be flooded across the network quickly and reliably so all routers maintain a consistent view of the topology. Each packet includes a sequence number and age to avoid loops and outdated data. Routers track received packets using the source and sequence number; new packets are forwarded, duplicates discarded, and older ones ignored.

			Send flags			ACK flags			
Source	Seq.	Age	A	C	F	A	C	F	Data
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

Figure 15.6 The packet buffer for router B in Figure 15.5 (a)

An Age field ensures stale packets eventually expire. To improve reliability, packets are acknowledged and may be delayed briefly before being forwarded, allowing updates to be merged. Routers maintain a packet buffer that includes flags for forwarding and acknowledging packets, ensuring efficient and accurate propagation throughout the network.

Computing the new routes

Once a router has received all link state packets, it can build the complete network graph, since each router shares its links and costs. Each link appears twice—once in each direction—which may have different costs. The router then runs Dijkstra's algorithm to find the shortest paths to all destinations and updates its routing table accordingly.

Compared to distance vector routing, link state routing uses more memory and processing power. In a network with n routers and k neighbors per router, storage needs are proportional to kn, and computation time increases even faster. Still, link state routing converges much faster and avoids the count-to-infinity problem, making it suitable for large networks.

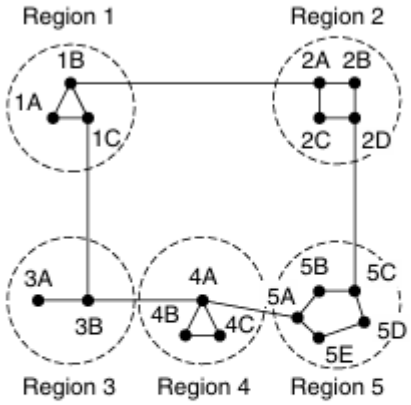
Two major link state protocols are:

- IS-IS: Originally for DEC net and later expanded to support multiple protocols like IP and AppleTalk.
- OSPF: Developed by the IETF with features like self-stabilizing updates and designated routers. It primarily supports IP but lacks IS-IS's multiprotocol flexibility.

Finally, all routing algorithms depend on correct behaviour of routers. A faulty router—due to hardware/software issues, incorrect link data, memory failures, or miscalculations—can disrupt the entire network. As networks scale, occasional router failures become more likely. Thus, robust design and error containment are critical, as discussed in depth by Perlman (1988).

15.7 THE HIERARCHICAL ROUTING

As networks expand, maintaining a flat routing structure—where each router stores routes to every other router—becomes impractical. The memory required to store routing tables, the CPU time needed to process them, and the bandwidth consumed for routing updates all increase rapidly. To manage this growth, hierarchical routing is used, where routers are grouped into regions. Each router maintains detailed routes only within its region and keeps summarized information for reaching other regions.



Full table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

(b)

Hierarchical table for 1A

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

(c)

Figure 15.7 Hierarchical routing

In hierarchical routing, inter-region routing is simplified. For example, in a multi-region setup, a router may know the exact path to local routers but treat all routers in another region as reachable through a single gateway router. This reduces the routing table size significantly. However, this simplification comes at the cost of potentially longer routes, since the most direct path might be ignored in favor of a more generalized one.

In large global networks, multilevel hierarchies can be applied. For instance, a packet from Berkeley (California) to Malindi (Kenya) may be routed first within California, then to a national gateway (e.g., Los Angeles), then to an international gateway (e.g., New York), and finally to Kenya. Each level of the hierarchy handles routing within its scope.

Quantitatively, using a two-level hierarchy in a network of 17 routers can reduce a routing table from 17 entries to just 7. In even larger networks, such as one with 720 routers, a three-level hierarchy can reduce the number of required entries per router from 720 to as few as 215. According to research by Kamoun and Kleinrock, the optimal number of hierarchical levels in a network is roughly the natural logarithm of the number of routers, with each router needing only e ln(N) entries. Although hierarchical routing may slightly increase the average path length, the benefits in scalability and manageability usually outweigh this drawback.

15.8 BROADCAST ROUTING

In some network applications, a host needs to send messages to many or all other hosts. This is called broadcasting and is useful for services like weather updates or live streams. One basic method is to send a separate packet to each destination. While simple, this approach is inefficient and requires the sender to know every destination. A better method is Multi destination routing, where a packet contains a list or bitmap of destinations. Routers examine the packet, split it as needed, and forward it along appropriate paths. Though more efficient in bandwidth, it still requires complex processing and complete destination knowledge at the source. Another method is flooding, where routers forward new packets on all outgoing links. It ensures delivery to all nodes but can create redundant traffic.

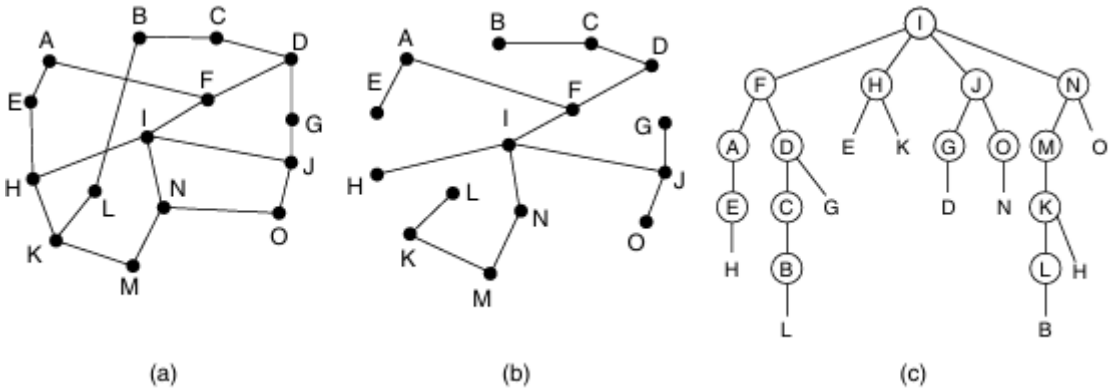


Figure 15.8 Reverse path forwarding (a) A Network (b) A sink tree (c) The tree built by reverse path forwarding

Reverse Path Forwarding (RPF) improves on flooding. A router forwards a broadcast packet only if it arrives on the link it would normally use to reach the source. This prevents duplicates and ensures efficient broadcasting using the existing shortest-path routing information. An example of reverse path forwarding (RPF) is illustrated in a network setup. Initially, router I sends packets to its direct neighbors—F, H, J, and N. These routers receive the packet through their preferred path back to I, which means the packet is forwarded further. In the second hop, these routers generate eight new packets. All of these reach new routers, and five of them arrive via the correct reverse path. By the third hop, six packets are

created, but only three arrive through the preferred paths—others are duplicates. Broadcasting finishes after five hops and 24 packets, although an exact sink tree would have only needed four hops and 14 packets. The key benefit of RPF is that it uses bandwidth efficiently and is easy to implement. Like flooding, each link is used just once per direction, but RPF doesn't need complex tracking like sequence numbers or destination lists—it only requires routing knowledge.

A further improvement uses spanning trees. A spanning tree includes all routers but no loops. If each router knows which links are part of the tree, it can forward the packet on all of them except the incoming one. This method minimizes the total packets sent. For instance, using the sink tree in the example, only 14 packets are needed. The downside is that routers must know the spanning tree, which is feasible with link state routing but harder with distance vector routing.

15.9 MULTICAST ROUTING

Some applications, such as multiplayer games or live sports streaming, need to send data to multiple recipients simultaneously. Sending a separate packet to each recipient becomes inefficient as the group size increases. On the other hand, broadcasting to the entire network wastes resources when most nodes aren't interested. This is where multicasting comes in—it allows data to be sent to a selected group of receivers that is large but still a small fraction of the whole network.

Multicast routing uses specialized routing algorithms to deliver messages only to the intended group members, without flooding the entire network. These routers must know which groups they belong to, and each group is identified by a multicast address. While group creation and membership management are separate concerns, multicast routing focuses on how to efficiently deliver packets using methods similar to broadcast routing.

For dense groups, where many routers belong to the group, broadcasting can initially be used, and then pruning techniques remove unnecessary paths—those that don’t lead to group members. This creates a more efficient multicast spanning tree. For example, a full spanning tree might have 10 links, but after pruning, a multicast tree for one group might only use 7, and another might use just 5, depending on the members' locations. If routers use link state routing, each has full knowledge of the network and group memberships. This allows them to compute pruned spanning trees effectively. One protocol that uses this method is MOSPF (Multicast OSPF), which is an extension of the OSPF link state protocol designed for multicast routing.

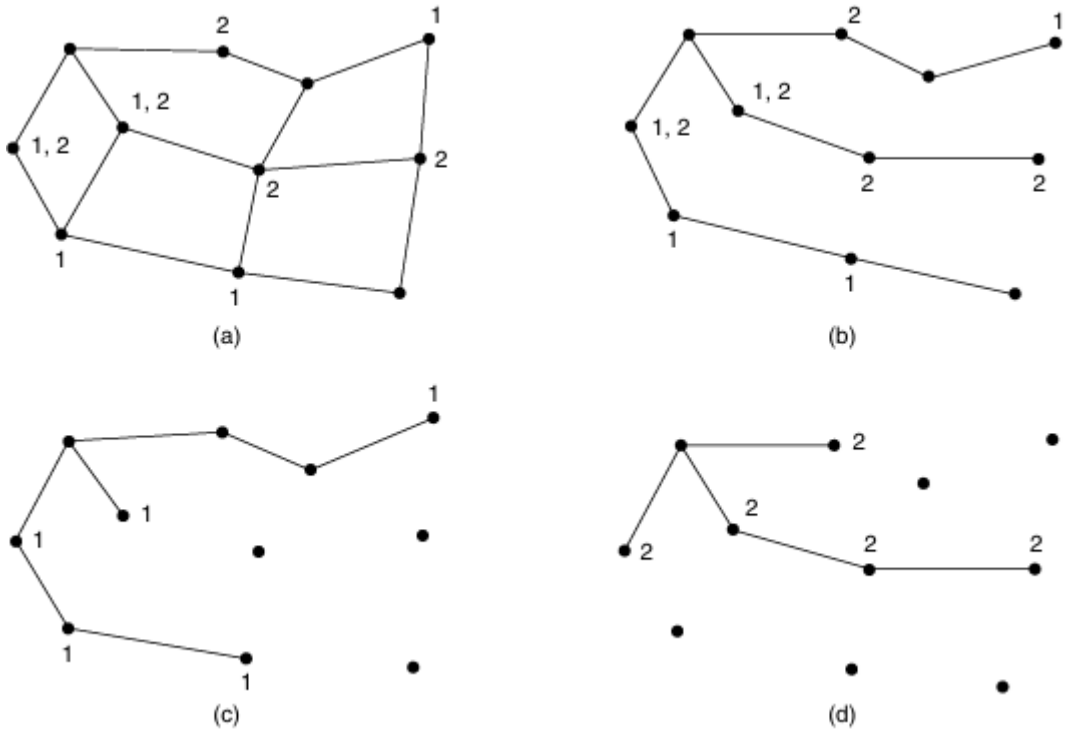


Figure 15.9 (a) A Network (b) A spanning tree for the leftmost router (c) A multicast tree for group 1. (d) A multicast tree for group 2.

In distance vector routing, multicast can be made more efficient using reverse path forwarding with pruning. When a router receives a multicast message for a group in which it has no interested hosts and no downstream routers needing it, it sends a PRUNE message to its upstream neighbor. This tells the sender not to forward more packets for that group. If a router receives PRUNE messages on all its multicast lines, it also sends a PRUNE, effectively recursively pruning the tree. This results in a minimized multicast tree. An example protocol using this method is DVMRP (Distance Vector Multicast Routing Protocol).

Pruning helps reduce bandwidth usage, but comes with a cost: each router may need to store many multicast trees. If there are n groups and each has m members, each router could store up to mn trees. This leads to high storage and processing overhead, especially in large-scale networks with many groups and senders. To address this, core-based trees are used. In this approach, routers agree on a core or rendezvous point for each multicast group. A single shared spanning tree is formed by sending messages from each group member to the core. When a sender wants to transmit, it sends the packet to the core, which then forwards it down the tree. This method is shown in Fig. 15.10.

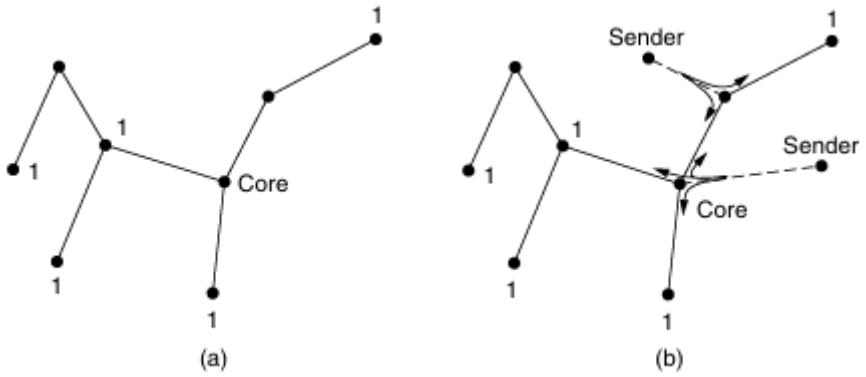


Figure 15.10 (a) Core-based tree for group 1 (b) Sending to group 1

A benefit of core-based trees is that each router needs to maintain only one tree per group, reducing memory and computation. However, shared trees may not always be optimal. For example, a packet from a sender might travel a longer path through the core than if it had gone directly to the destination. Despite this inefficiency, it is a practical trade-off for sparse groups, especially when storage and efficiency matter. Protocols like PIM (Protocol Independent Multicast) use this method for multicast in the Internet.

15.10 ANYCAST ROUTING

Anycast is a network communication method where data is sent from a source to the nearest node in a group of potential receivers that all share the same address. Unlike unicast (which targets one specific destination), broadcast (which targets all nodes), or multicast (which targets a group of specific nodes), anycast focuses on delivering data to the closest member of a group based on routing distance. This model is particularly useful for services where it doesn't matter which node replies, as long as the response is correct. Examples include services like DNS, CDNs (Content Delivery Networks), or time synchronization, where multiple servers can provide the same data or service. With anycast, a user automatically connects to the server that is topologically closest, improving response time and reducing overall network load.

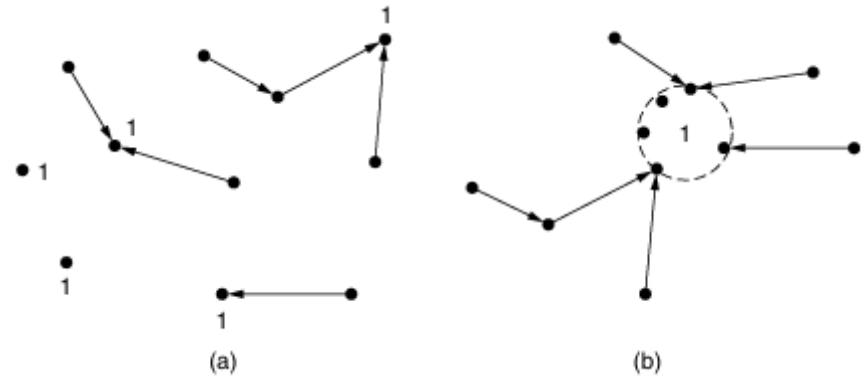


Figure 15.11 (a) Anycast routes to group 1. (b) Topology seen by the routing protocol

Interestingly, anycast does not require new routing protocols. Traditional routing algorithms like distance vector and link state can be used. All nodes providing the same service are assigned the same destination address. As routers calculate the shortest path to that address, the protocol naturally directs traffic to the nearest available server. The routing system treats all instances as one destination, making anycast both efficient and easy to implement using existing infrastructure.

15.11 ROUTING FOR MOBILE HOSTS

Millions of people use computers while on the move, in both mobile situations (like using wireless devices in cars) and nomadic ones (such as using laptops in different locations). These are collectively referred to as mobile hosts, in contrast to stationary hosts that do not move. With the increasing demand for constant connectivity regardless of location, a key challenge arises: how to route data packets to a host that keeps changing its network location. In the model under discussion, each host has a permanent home address and home location, much like a phone number reveals country and area. The goal is to ensure packets sent to a mobile host's fixed home address can still reach it wherever it is.

Recomputing routes every time a mobile host moves would overwhelm the network with constant updates. Instead, by using home addresses and assigning a home agent (a server at the host's home location), the network can manage mobility more efficiently. The mobile host, upon moving, obtains a temporary local address known as a care-of address. It then informs its home agent about this new address through a control message. When someone sends a data packet to the mobile host's permanent address, it reaches the home agent. The home agent then encapsulates this packet and forwards it to the care-of address—a technique known as tunneling. This approach allows seamless communication with mobile hosts while reducing the overhead of constantly updating routing tables.

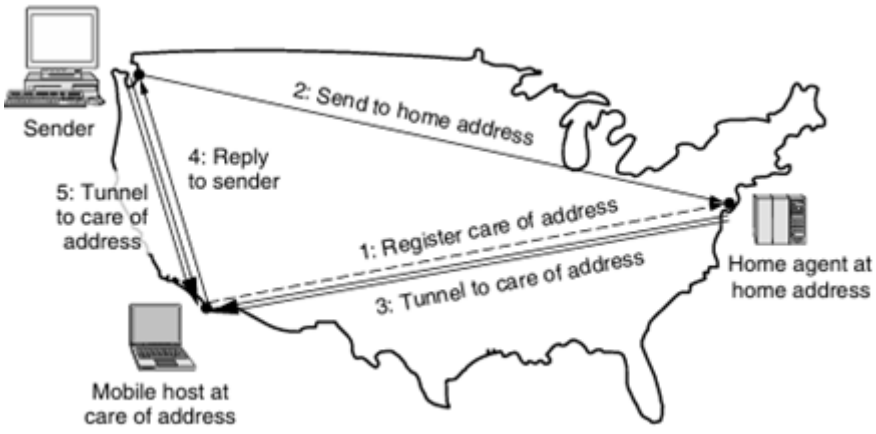


Figure 15.12 Packet routing for mobile hosts

When the encapsulated packet reaches the care-of address, the mobile host decapsulates (unwraps) it to retrieve the original message from the sender. It then sends a reply directly back to the sender. This results in what is called triangle routing—a potentially inefficient

path, especially if the mobile host is far from its home location. However, during this exchange, the sender may learn the mobile host's care-of address. In that case, future packets can be sent directly to the care-of address using tunnelling, skipping the home agent and making communication more efficient. Still, if direct communication fails (e.g., if the host moves again), the sender can always fall back to using the permanent home address.

Security is a key concern in this setup. Since messages like "Send all of Stephany's messages to me" could be misused, cryptographic mechanisms are included to authenticate and verify such messages. These security protocols ensure that only legitimate changes in location are acted upon.

Various mobile routing techniques exist. The method described here is based on IPv6 mobility, used in modern Internet and cellular networks such as UMTS. While the explanation assumes the sender is stationary, the protocol supports both sender and receiver being mobile. It can even handle entire mobile networks, such as on airplanes, without requiring individual mobile devices to take action.

Some versions of mobile routing use a foreign agent at the host's temporary location, much like the Visitor Location Register (VLR) in cellular systems. But modern systems often eliminate the need for a separate foreign agent, letting the mobile host handle this role itself. In all cases, only a small set of entities (typically the mobile host, its home agent, and any active senders) are aware of the mobile host's temporary location, keeping the rest of the network unaffected and avoiding the need to constantly recalculate routes.

15.12 ROUTING IN AD HOC NETWORKS

So far, we've looked at routing where mobile hosts move, but the routers themselves remain fixed. A more dynamic and complex scenario arises when the routers themselves are mobile. This situation occurs in environments like disaster recovery zones, military operations, fleets of ships, or groups of people with laptops in areas lacking wireless infrastructure. In such cases, each device communicates wirelessly and serves both as a host and as a router. These self-organizing, infrastructure-less networks are called ad hoc networks, or MANETs (Mobile Ad hoc NETworks).

What sets ad hoc networks apart is the highly unpredictable and ever-changing topology. Unlike wired networks, where routing paths remain stable unless a rare failure occurs, ad hoc network paths can break or become suboptimal suddenly as nodes move or go offline. This makes routing far more challenging than in traditional, fixed networks.

Numerous routing algorithms have been designed for ad hoc networks, though they haven't seen widespread deployment yet. One well-known example is AODV (Ad hoc On-demand Distance Vector), an algorithm that builds on the traditional distance vector approach but is specially adapted for mobile environments with limited bandwidth and battery power. Let's now look into how AODV discovers and maintains routes in such dynamic networks.

Route discovery

In AODV (Ad hoc On-demand Distance Vector) routing, routes are discovered only when they are needed—on demand. This approach conserves bandwidth and processing power by avoiding the maintenance of routes that may quickly become outdated due to frequent topology changes. At any given moment, an ad hoc network can be represented as a graph of nodes, where a direct connection (or an edge) exists between two nodes if they are within each other's wireless transmission range. A simple but effective model assumes that every node can communicate with all others within a fixed-radius coverage circle. While real-world conditions like buildings, terrain, and uneven transmitter power may cause asymmetric links (where node A can reach B, but not vice versa), for simplicity, we assume all links are symmetric.

To understand how AODV works, consider a newly formed ad hoc network. Suppose node A wants to send a packet to node I. Each node maintains a distance vector table that stores routes keyed by destination, including the next hop to use. Node A checks its table and finds no entry for I, which means it does not yet know how to reach the destination. It now needs to discover a route to node I. This need-driven behaviour—discovering routes only when necessary—is the core principle behind the on-demand nature of the AODV protocol.

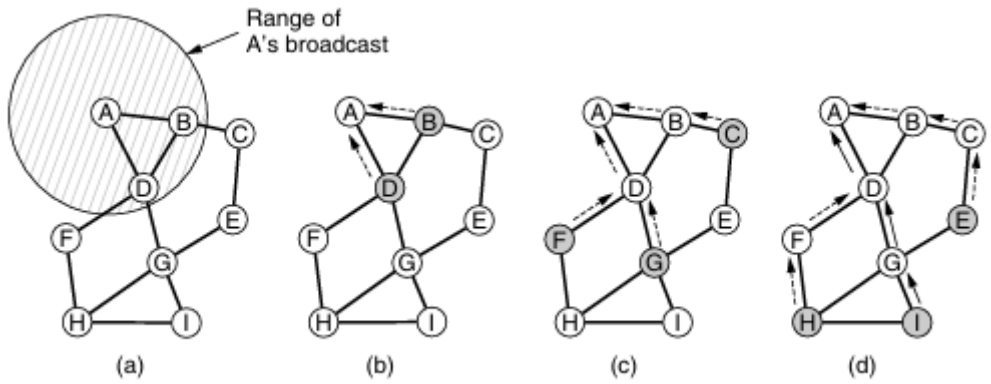


Figure 15.13 (a) Range of A's broadcast. (b) After B and D receive it. (c) After C, F, and G receive it. (d) After E, H, and I receive it. The shaded nodes are new recipients. The dashed lines show possible reverse routes. The solid lines show the discovered route.

To find a route to the destination node I, node A creates a ROUTE REQUEST (RREQ) packet and broadcasts it using flooding. This means the packet is sent to all neighboring nodes—in this case, nodes B and D. Each node that receives the RREQ rebroadcasts it, spreading it throughout the network. For example, the request reaches nodes F, G, and C, and then further reaches H, E, and finally I. To prevent the same request from being processed multiple times, a sequence number is included with the RREQ. Each node checks this number and ignores duplicates. For instance, node D discards a duplicate RREQ from node B if it has already forwarded the same request earlier.

When the destination node I receives the RREQ, it sends back a ROUTE REPLY (RREP) packet. This reply is unicast (sent directly) back to the source node A by retracing the reverse path taken by the request. For this to work, each intermediate node along the way must remember the node it received the request from, effectively building a reverse route. These nodes also count the number of hops from the destination as the reply travels back. Each node updates its routing table with the best route it receives—choosing the path with the fewest hops. When the RREP reaches node A, a complete route (e.g., A → D → G → I) is established.

While effective, this flooding-based route discovery can generate a lot of overhead in large networks, even for nearby destinations. To minimize unnecessary broadcasts, AODV uses the Time to Live (TTL) field in the IP header. TTL is initialized by the sender and decremented by one at each hop. When it reaches zero, the packet is discarded. AODV uses this feature to perform an expanding ring search: it starts with a ROUTE REQUEST with TTL = 1. If no reply is received, it tries again with TTL = 2, then 3, and so on. This approach limits flooding to progressively larger areas, reducing overall broadcast traffic.

Route maintenance

Since nodes in an ad hoc network can move or be switched off at any time, the network topology can change unpredictably. For example, in a previously discovered route like A → D → G → I, if node G is turned off, node A won't immediately know that the route is now broken. To handle such dynamic changes, AODV uses a mechanism where each node periodically sends out Hello messages. These are short broadcasts to nearby nodes, and neighbors are expected to respond. If a neighbor doesn't respond or fails to acknowledge a packet, the node concludes that the neighbor is no longer reachable.

When a node detects that a neighbor is no longer reachable, it updates its routing table to remove any routes that depended on that neighbor. Each node keeps track of its active neighbors for each destination—those that have recently sent packets through it. If a neighbor becomes unreachable, the node identifies which routes relied on it and informs all its active neighbors to purge those routes from their own tables. This notification process continues recursively until all affected nodes have removed the invalid routes.

Once invalid routes are cleared, new valid routes can be discovered using the regular on-demand route discovery process. However, this can be tricky, as traditional distance vector algorithms may suffer from slow convergence or count-to-infinity problems, mistaking stale routes for valid ones. To solve this, AODV uses sequence numbers managed by the destination node. Every time a destination sends a new ROUTE REPLY, it increments its sequence number. When a source requests a route, it includes the sequence number of the last known route. The request is broadcast until a route with a newer sequence number is found, ensuring up-to-date routing information. Intermediate nodes store only the routes with either the highest sequence number or the lowest hop count for a given sequence number.

To preserve bandwidth and battery life—important in mobile environments—AODV only stores active routes (those currently being used). Any other routing data learned through broadcast messages is discarded after a short timeout, unlike traditional distance vector protocols that maintain full routing tables.

Also, route discovery and maintenance are shared when routes overlap. For example, if node B wants to send data to I and node D already has a valid route to I, B can use D's knowledge and avoid initiating a new discovery. This reduces unnecessary network traffic.

Apart from AODV, other ad hoc routing protocols exist. DSR (Dynamic Source Routing) is another on-demand protocol, and GPSR (Greedy Perimeter Stateless Routing) uses geographic information instead of routing tables. In GPSR, packets are forwarded toward the geographic location of the destination, making routing decisions based on physical position rather than route discovery. Which protocol becomes dominant in real-world ad hoc networks will depend on practical use cases and network conditions.

15.13 SUMMARY

This chapter explains routing algorithms, which determine optimal paths for data delivery across networks. It covers shortest path routing guided by the optimality principle, and flooding, where packets are sent to all neighbors. Distance vector and link-state algorithms provide efficient routing through periodic updates or complete network views. The chapter also addresses hierarchical routing to scale large networks, broadcast and multicast routing for one-to-many communication, and special considerations for routing in mobile host networks, where dynamic topology requires adaptive techniques for reliability and efficiency.

15.14 TECHNICAL TERMS

Routing, distance vector algorithm, link-state algorithm, multicast routing.

15.15 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the optimality principle and shortest path routing with examples.
2. Describe flooding and its advantages and disadvantages.
3. Discuss distance vector and link-state routing algorithms in detail.
4. Explain hierarchical, broadcast, and multicast routing methods.
5. Describe the challenges and techniques for routing in mobile host networks

Short Questions:

1. What is the optimality principle in routing?
2. Define shortest path routing.
3. What is flooding in network routing?
4. Mention one key feature of distance vector routing.
5. What is multicast routing?

15.16 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008).

Dr. Vasantha Rudramalla

LESSON- 16
INTERNETWORKING AND INTERNET PROTOCOL SUITE

OBJECTIVES:

After going through this lesson, you will be able to

- Understand differences between networks and how they can be connected.
- Learn about concatenated virtual circuits and connectionless internetworking.
- Study tunneling, internetwork routing, and fragmentation.
- Understand the IP protocol, addressing, and Internet control protocols.
- Learn about OSPF, EGP, BGP, and Internet gateway routing

STRUCTURE OF THE LESSON:

16.1 INTRODUCTION

So far, we've been thinking as if all computers are connected through one big network using the same rules and protocols. But in the real world, that's not true at all. There are many different types of networks—like home networks (PANs), office networks (LANs), city-wide networks (MANs), and even global ones (WANs). Each of these uses different technologies and different communication rules. For example, Ethernet, Wi-Fi, mobile networks, and cable Internet all work differently. Connecting all these different networks together is called internetworking.

It would be easier if everyone used the same kind of network, but that's unlikely to happen. Different networks solve different problems—what works well in an office might not work in a moving vehicle or on a satellite. Also, many systems were built long ago using phone lines

or power lines, and we still use them today. So, we have to accept that networks will always be different.

Even though networks are different, we still want them to work together. That's because the more computers that are connected, the more useful the whole system becomes. This idea is called Metcalfe's Law, which says that a network becomes more valuable as more devices are connected to it. That's why people want to connect small networks together to make one big network—like the Internet.

The Internet is the best example of many networks working together. When you get Internet service from an ISP, you're paying not just to connect to their network, but to be able to talk to any device in the world that's also online. This global connection is what makes the Internet so powerful.

But connecting different networks isn't easy. There are many challenges—like making different systems understand each other and handling the huge size of the global network. To solve these problems, we use the Internet Protocol (IP), which is specially designed to connect different types of networks. IP includes features like tunnelling, routing, and breaking large packets into smaller ones (called fragmentation), which help make the whole system work smoothly.

16.2 HOW NETWORKS DIFFER

Different networks can vary in many ways. Some differences, like how signals are sent (modulation) or the format of frames, happen at the lower layers (physical and data link layers) and don't usually affect how the network layer works. But other differences do matter at the network layer, and these are the ones that make internetworking—connecting different networks—more complicated than just working within one network.

When a packet travels from one network, through other networks, and finally to the destination on a different network, many problems can occur at the boundaries between these networks. One of the first issues is addressing—how can a computer on an Ethernet network send a packet to a device on a WiMAX network, for example? Even if the destination can be identified, there's a bigger challenge: Ethernet is connectionless (it just sends packets), while WiMAX might be connection-oriented (it sets up a path first). That means before the packet can move forward, a connection might need to be created, which takes time and adds extra steps—especially if only a few packets are being sent.

Item	Some Possibilities
Service offered	Connectionless versus connection oriented
Addressing	Different sizes, flat or hierarchical
Broadcasting	Present or absent (also multicast)
Packet size	Every network has its own maximum
Ordering	Ordered and unordered delivery
Quality of service	Present or absent; many different kinds
Reliability	Different levels of loss
Security	Privacy rules, encryption, etc.
Parameters	Different timeouts, flow specifications, etc.
Accounting	By connect time, packet, byte, or not at all

Figure 16.1 Some of the many ways networks can differ.

There are also many smaller technical issues to handle. For instance, what if we want to send a packet to a group of devices (multicast) but some of those devices are on a network that doesn't support multicast? These kinds of mismatches between network features are what make internetworking so tricky.

Different networks often support different maximum packet sizes. For example, an 8000-byte packet from one network may need to pass through another that only allows 1500 bytes. In such cases, the large packet must be broken into smaller pieces, sent separately, and reassembled later—a process called fragmentation.

Other problems arise when connection-oriented networks send packets through connectionless networks. Packets may arrive out of order, which can confuse receivers expecting them in sequence. These issues can be handled by gateways that split large packets, reorder them, or create multiple packets for destinations when multicast is not supported.

However, some differences are harder to fix. A major one is quality of service (QoS). If one network guarantees speed and reliability while another doesn't, it's impossible to ensure smooth, end-to-end performance—especially for real-time traffic like video or voice. Similarly, inconsistent security features across networks can create vulnerabilities, although encryption can be added later.

Finally, differences in accounting and billing between networks can lead to unexpected costs. For example, mobile users may face high charges when roaming, even if their usage seems normal.

16.3 HOW NETWORKS CAN BE CONNECTED

A data packet travels across different types of networks, such as Wi-Fi (802.11), MPLS, and Ethernet. Each network may have different addressing methods and limitations, so the packet carries a network layer address (like an IP address) that helps it reach the correct destination across these networks.

Initially, the packet is sent over a Wi-Fi (802.11) network, which is connectionless. It is encapsulated in a Wi-Fi frame and sent to the first router. At the router, the Wi-Fi header is removed, and the router examines the IP address to decide the next hop. The packet is then sent into an MPLS network, which is connection-oriented. To cross this network, a virtual circuit must be set up, and the packet is wrapped with MPLS headers. At the exit of the MPLS network, these headers are removed.

Next, the packet enters an Ethernet network. Since Ethernet may support smaller frame sizes than Wi-Fi, the packet might be too large and must be split into fragments. Each fragment is placed in a separate Ethernet frame and sent to the destination. At the destination, the Ethernet headers are removed, and the fragments are reassembled to reconstruct the original packet.

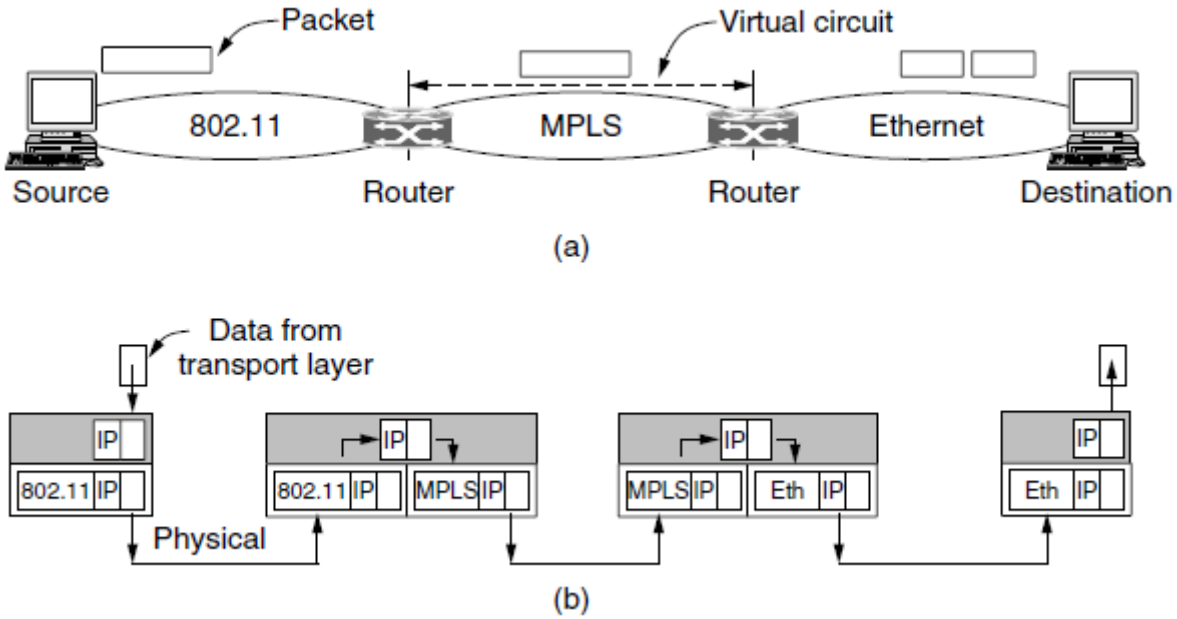


Figure 16. 2 (a) A packet crossing different networks. (b) Network and link layer protocol processing.

Finally, the routers use the IP address in the packet to make forwarding decisions, extracting the packet from its frame. In contrast, switches or bridges forward entire frames based on MAC addresses and do not need to understand the network layer.

Internetworking, which is the process of connecting different types of networks to allow smooth communication between them, is more difficult than it may initially seem. In the early days, bridges were introduced with the goal of connecting different LANs (Local Area Networks), even if they had different characteristics. The idea was that bridges would translate frames from one type of LAN to another. However, this approach proved to be ineffective due to significant differences among LANs, such as varying maximum packet sizes, frame formats, and features like priority classes. These differences made it hard to perform reliable and complete translations. As a result, bridges are now mainly used to connect similar networks at the link layer, while routers are used to connect different networks at the network layer.

Despite these challenges, internetworking has been largely successful due to the widespread adoption of a common network layer protocol—the Internet Protocol (IP). While there were once many competing network protocols, such as IPX, SNA, and AppleTalk, most of these have now faded away, with IP becoming the standard. However, even within IP, issues exist. For instance, IPv4 and IPv6 are two versions of IP, but they are not directly compatible. This incompatibility creates a challenge for seamless communication, making the transition to IPv6 more complex and slower than expected.

To support multiple protocols, multiprotocol routers were developed. These routers can either translate between different network protocols or pass the responsibility to higher protocol layers like TCP. But both options have downsides. Relying on TCP only works if all networks support it, and it limits communication to applications that use TCP, excluding many real-time services like voice or video. On the other hand, translating packets between different network protocols is very difficult, especially when their address formats and

structure differ significantly—for example, converting a 128-bit IPv6 address to a 32-bit IPv4 address is impossible without losing crucial information.

Due to these limitations, protocol translation is rarely attempted in practice. Instead, IP has succeeded globally by acting as a "lowest common denominator"—it demands very little from the networks it runs on and, in return, provides a best-effort service. This means it tries to deliver packets but does not guarantee delivery, order, or quality. While this approach has enabled massive global connectivity, it also reflects the compromises made to achieve interoperability across diverse systems.

16.4 TUNNELING

Internetworking between different types of networks is usually complex, but there is a special case that can be handled more easily. This occurs when the source and destination hosts are on the same type of network (e.g., IPv6), but the path between them includes a different network type (e.g., IPv4). For example, an international bank might have an IPv6 network in Paris and another in London, connected through the IPv4 Internet.

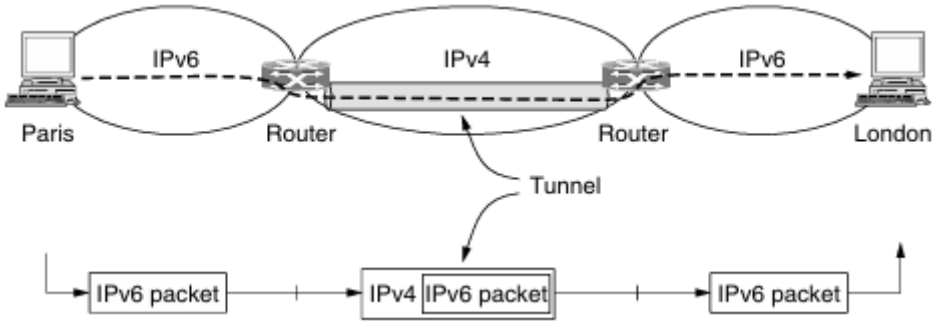


Figure 16.3 Tunneling a packet from Paris to London

To enable communication in this case, a technique called tunneling is used. Here, the source host in Paris creates a regular IPv6 packet addressed to the destination in London. When this packet reaches the multiprotocol router that connects to the IPv4 Internet, the router encapsulates the IPv6 packet inside an IPv4 packet. This new IPv4 packet is then sent across the IPv4 network.

Once the packet reaches the router in London that connects back to the IPv6 network, the outer IPv4 header is removed, and the original IPv6 packet is delivered to the destination. This entire process creates a "tunnel" through the IPv4 network, making it look like a direct IPv6 connection between the two locations. The hosts in Paris and London are unaware of the IPv4 network in between—only the routers need to understand both protocols.

Internetworking between different types of networks is usually complex, but there is a special case that can be handled more easily. This occurs when the source and destination hosts are on the same type of network (e.g., IPv6), but the path between them includes a different network type (e.g., IPv4). For example, an international bank might have an IPv6 network in Paris and another in London, connected through the IPv4 Internet.

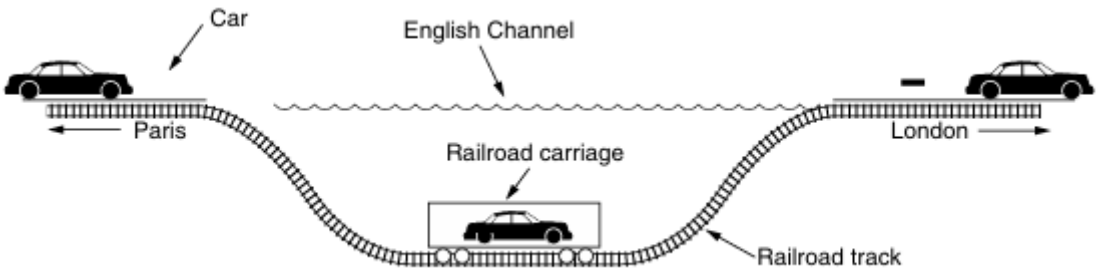


Figure 16.4 Tunneling a car from France to England

To enable communication in this case, a technique called tunneling is used. Here, the source host in Paris creates a regular IPv6 packet addressed to the destination in London. When this packet reaches the multiprotocol router that connects to the IPv4 Internet, the router encapsulates the IPv6 packet inside an IPv4 packet. This new IPv4 packet is then sent across the IPv4 network.

Once the packet reaches the router in London that connects back to the IPv6 network, the outer IPv4 header is removed, and the original IPv6 packet is delivered to the destination. This entire process creates a "tunnel" through the IPv4 network, making it look like a direct IPv6 connection between the two locations. The hosts in Paris and London are unaware of the IPv4 network in between—only the routers need to understand both protocols.

16.5 INTERNETWORK ROUTING

Routing across an internet, which is made up of many interconnected networks, is similar to routing within a single network but comes with extra challenges. Different networks may use different internal routing algorithms—for example, one may use link state routing while another uses distance vector routing. These differences make it difficult to compute consistent shortest paths across the entire internet.

Further complications arise when networks are operated by different organizations or Internet Service Providers (ISPs). Each operator may prioritize different routing goals—one might focus on minimizing delay, while another might aim to reduce cost. Since the criteria for choosing routes vary and are not comparable (like delay vs. money), defining a universal shortest path becomes impossible. Also, operators may not want to share details about their network structure or routing metrics, as that information can be commercially sensitive.

Another challenge is the sheer scale of the internet, which is much larger than any individual network. This requires routing algorithms that can scale well, often by organizing routing into a hierarchy. To handle all these issues, the internet uses a two-level routing approach. Within each network (also called an Autonomous System or AS), an intradomain routing protocol is used—such as OSPF or a link state protocol. Between different ASes, a separate interdomain routing protocol is used. On the Internet, this protocol is BGP (Border Gateway Protocol).

Each AS is an independently managed network—usually an ISP or part of one—and can use any internal routing method it prefers. Interdomain routing, on the other hand, must follow a common protocol like BGP. This structure allows for scalability, policy control, and privacy between networks. It avoids comparing routing metrics across different domains and prevents exposure of sensitive routing information.

Finally, route selection across ASes isn't only based on technical factors like performance. Business agreements between ISPs—such as who pays whom for traffic—play a major role. In addition, legal constraints, such as privacy laws in some countries, may restrict how traffic is routed internationally. All these influences—technical, business, and legal—are collectively managed through routing policies, which determine how routes are chosen in the real-world Internet.

16.6 PACKET FRAGMENTATION

In internetworking, one major challenge is dealing with packet size limitations across different types of networks. Each network or communication link has a Maximum Transmission Unit (MTU), which is the largest packet size it can handle. These MTU limits are imposed by several factors. For example, hardware like Ethernet restricts packet size to 1500 bytes, while 802.11 allows 2272 bytes. Operating systems might have internal buffer size limitations, some protocols define size limits within their headers, and international standards may set specific maximums. Additionally, smaller packet sizes may be chosen to reduce the impact of errors and avoid allowing a single packet to monopolize the transmission channel for too long.

In general, hosts prefer to send large packets because they reduce overhead: fewer headers are needed, and transmission becomes more efficient. However, when a large packet must travel through a network with a smaller MTU, it creates problems. This is especially difficult in a connectionless network like the Internet, where packets may take different paths to the same destination. As a result, a sender does not always know the smallest MTU along the path—a value known as the Path MTU. Even if the path MTU were known, dynamic routing means the path (and its MTU) might change unpredictably.

One solution is fragmentation, where routers split larger packets into smaller fragments so they can pass through networks with smaller MTUs. There are two main strategies for this. The first is transparent fragmentation, where a router fragments a large packet and reassembles it at the exit point of the small-packet network before forwarding it further. This method hides the fragmentation process from the rest of the network. However, it introduces several issues: routers need to keep track of fragments, know when all have arrived, and buffer them while waiting. It also forces all fragments to follow the same route to ensure proper reassembly, which restricts routing flexibility and can lead to inefficient paths.

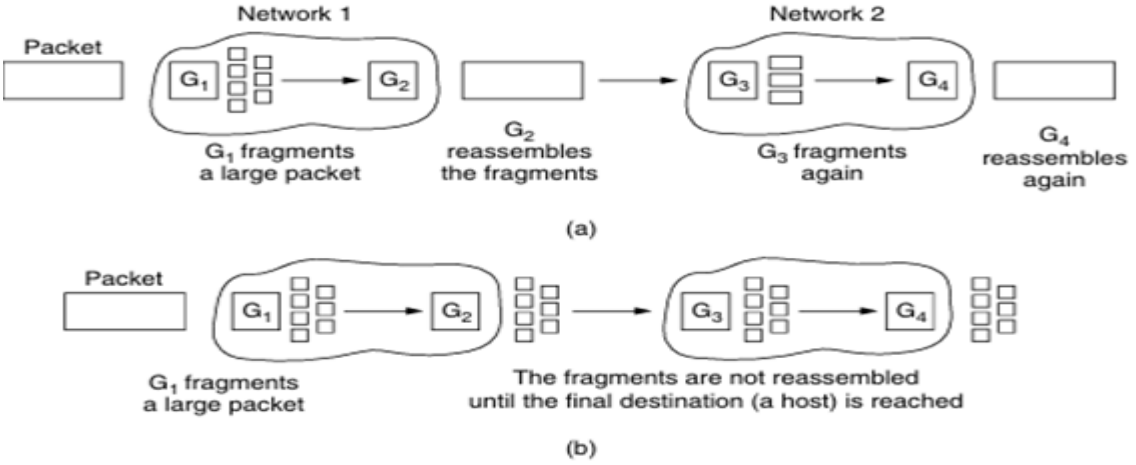


Figure 16.5 (a) Transparent fragmentation. (b) Nontransparent fragmentation.

The second method is nontransparent fragmentation, used by the Internet Protocol (IP). In this strategy, once a packet is fragmented, the fragments are treated as independent packets and travel separately to the final destination, where they are reassembled by the receiving host. This method simplifies router design since routers do not need to manage reassembly. IP assigns each fragment a packet number, a byte offset, and an end-of-packet flag. These allow the destination to reassemble the data correctly even if fragments arrive out of order. Fragments can be further fragmented if they encounter another network with a smaller MTU, and retransmissions can use different fragmentation patterns.

Despite its advantages, fragmentation has significant drawbacks. First, each fragment carries a header, increasing overhead. More importantly, if any single fragment is lost, the entire original packet must be retransmitted. This creates inefficiencies, especially for large packets, and places more processing burden on the receiving host. These issues led researchers to conclude that fragmentation should be avoided inside the network whenever possible.

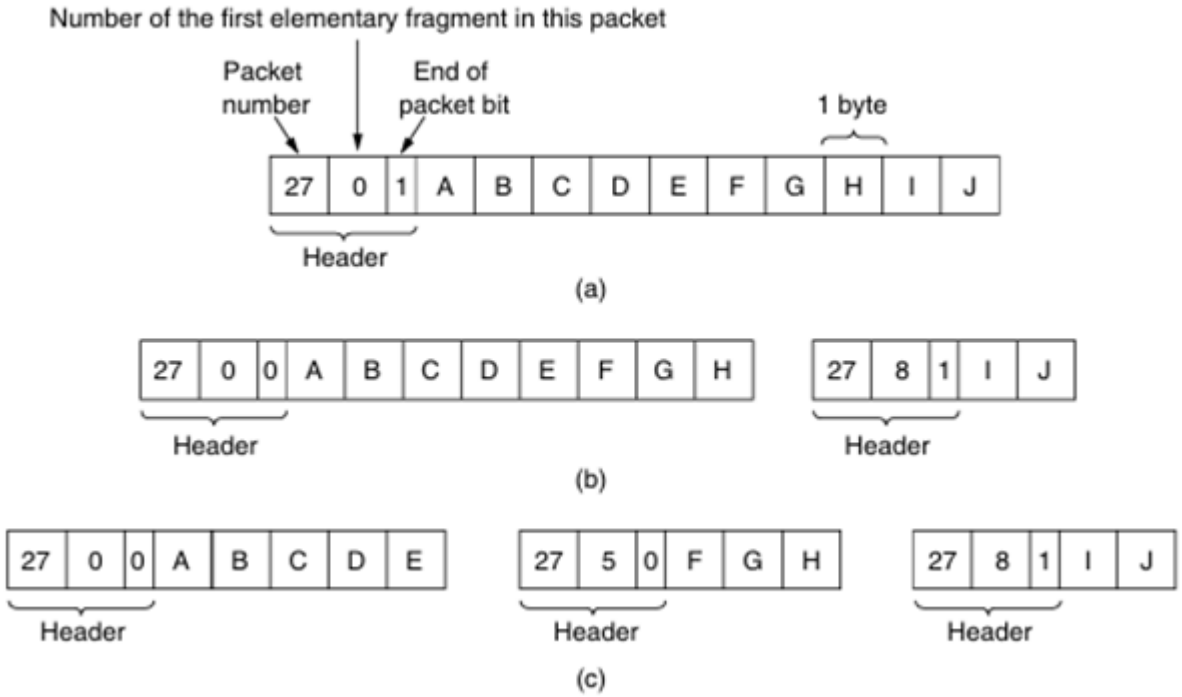


Figure 16.6 Fragmentation when the elementary data size is 1 byte. (a) Original packet, containing 10 data bytes. (b) Fragments after passing through a network with maximum packet size of 8 payload bytes plus header. (c) Fragments after passing through a size 5 gateway.

To address these problems, the Path MTU Discovery technique was developed. In this approach, the source sets the "Don't Fragment" (DF) flag in the IP header. If a router receives a packet that exceeds its MTU and the DF flag is set, it drops the packet and sends an ICMP error message back to the source, indicating the maximum size it can handle. The source then adjusts the packet size accordingly and tries again. If another router along the path has an even smaller MTU, it repeats the process. This continues until the packet is small enough to pass through the entire path without fragmentation.

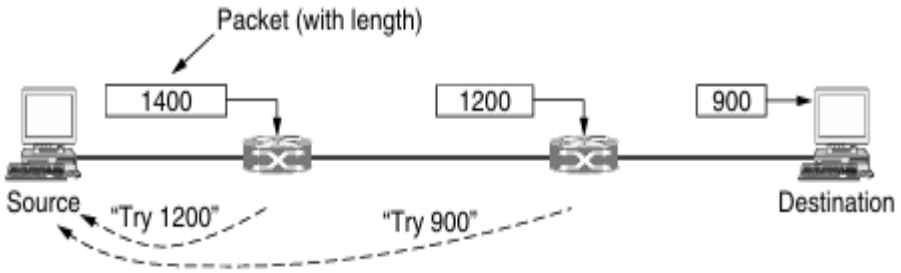


Figure 16.7 Path MTU discovery

The advantage of Path MTU Discovery is that it eliminates the need for routers to perform fragmentation, reducing their processing load and simplifying network operation. It also ensures that packets are sized correctly for the entire path, improving performance. This method is typically used in TCP/IP, where the transport and network layers work together to adapt packet sizes.

However, Path MTU Discovery has its own downsides. It can cause initial delays in data transmission because multiple trial-and-error attempts may be needed before the source finds the correct MTU size. Additionally, if any routers block ICMP messages (as some firewalls do), the source might never receive the needed error message, leading to failed transmissions. In theory, better designs might exist. For instance, one idea is to have routers truncate packets that exceed their MTU and forward the truncated data, allowing the destination to learn the MTU and receive at least part of the data immediately. However, such a design introduces its own complexities and has not been widely adopted.

In summary, managing packet sizes across diverse networks is a complex and persistent issue in internetworking. Fragmentation provides a way to cope with size mismatches but introduces overhead and performance concerns. Path MTU Discovery offers a cleaner, more modern approach by pushing the responsibility of adapting packet size to the sender, but it comes with its own set of limitations.

16.7 SUMMARY

This chapter covers internetworking, explaining how different networks with varying technologies can be connected. It discusses concatenated virtual circuits, connectionless internetworking, tunneling, and fragmentation to manage packet delivery across diverse networks. The chapter also examines the Internet network layer, focusing on the IP protocol, IP addressing, and control protocols that manage routing and reliability. Key routing protocols such as OSPF, BGP, and other exterior gateway protocols ensure efficient and scalable routing across the global Internet.

16.8 TECHNICAL TERMS

Internetworking, IP addressing, OSPF, BGP, gateway protcols.

16.9 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the differences between networks and the methods to connect them.
2. Describe connectionless internetworking and concatenated virtual circuits.
3. Discuss tunneling and fragmentation in internetworking.
4. Explain the IP protocol, IP addressing, and key Internet control protocols.
5. Describe OSPF, BGP, and the role of exterior gateway routing protocols in the Internet.

Short Questions:

1. What is internetworking?
2. Define concatenated virtual circuits.
3. What is tunneling in networks?
4. What is the purpose of the IP protocol?
5. Name one exterior gateway routing protocol

16.10 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008)

Dr. Vasantha Rudramalla

LESSON- 17
THE TRANSPORT LAYER SERVICES AND PROTOCOL MECHANISMS

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the role and functions of the transport layer.
- Learn the services provided by the transport layer to upper layers.
- Study transport service primitives and their operations.
- Understand the concept and usage of Berkeley sockets.
- Recognize how transport protocols support reliable communication

STRUCTURE OF THE LESSON:

17.1 THE TRANSPORT SERVICES

The transport layer, along with the network layer, forms the core of the protocol stack. While the network layer handles end-to-end packet delivery between machines, the transport layer ensures reliable data transfer between processes running on those machines. It provides essential services like reliability, flow control, and error handling, allowing applications to communicate smoothly without needing to know the details of the underlying network. This layer enables consistent and efficient communication, regardless of the physical networks in use. In this chapter, key transport layer topics such as TCP, UDP, reliability, congestion control, and API design will be explored.

The upcoming sections introduce the transport service by explaining the type of communication it offers to the application layer. To make this concept clearer, two sets of transport layer primitives will be discussed. The first is a simple, hypothetical set meant to illustrate the fundamental ideas behind transport services. The second is the actual interface used in the Internet, showing how real-world applications interact with the transport layer to send and receive data.

17.2 SERVICES PROVIDED TO THE UPPER LAYER

The main goal of the transport layer is to offer efficient, reliable, and cost-effective data transmission to its users, typically the processes in the application layer. To do this, the transport layer builds upon the services provided by the network layer. The part of the system that performs transport functions is called the transport entity, which can reside in the operating system kernel, in a user-space library, or even in the network interface card—though most commonly it's in the OS or a linked library.

There are two types of transport services: connection-oriented and connectionless, mirroring the two network service types. Connection-oriented services involve setup, data transfer, and connection release, and include mechanisms for addressing and flow control. Connectionless services send data without establishing a connection first. However, it's difficult to implement a connectionless transport service on top of a connection-oriented network service, because it's inefficient to set up and tear down connections just to send one packet.

A natural question arises: if transport services are so similar to network services, why are both layers needed? The key difference lies in where each layer operates. The network layer typically runs on routers owned by carriers, while the transport layer runs entirely on the end-user's machines. Users can't control or fix the network layer if it offers poor service (like dropping packets or crashing). Instead, they rely on the transport layer to detect and correct errors, such as through retransmissions. This is why a separate transport layer is essential—it compensates for the limitations of the network layer and ensures better communication quality for applications.

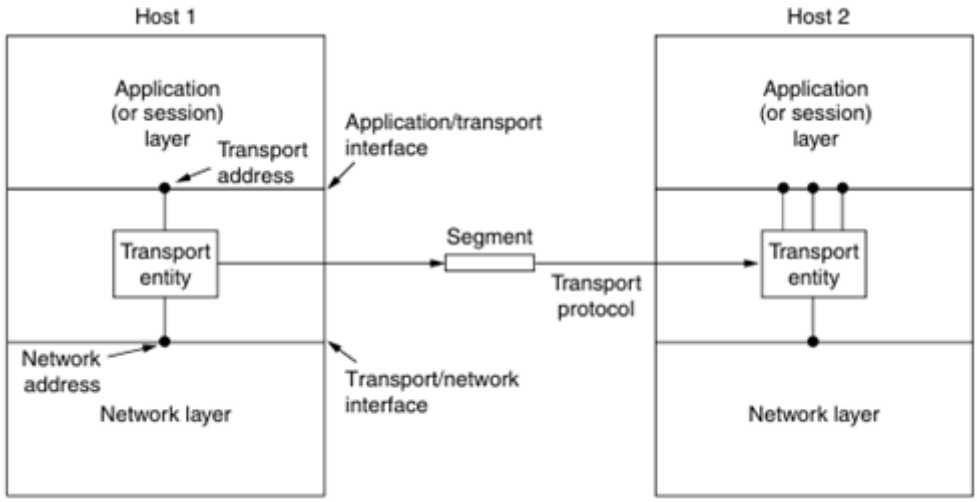


Figure 17.1 The network ,transport, and application layers.

The transport layer plays a critical role in ensuring reliable communication, especially when the underlying network layer is unreliable. For example, if a connection-oriented network fails during data transmission, the transport entity can establish a new connection and query the receiving end to find out which data was successfully received. This allows it to resume transmission without data loss, making the overall communication more robust than the network itself.

This added reliability is a key benefit of the transport layer. It acts as a buffer between applications and the network, providing a consistent interface regardless of what type of network is being used—be it connectionless like Ethernet or connection-oriented like WiMAX. By using a standard set of transport service primitives, application programs become independent of the specific network details, making them more portable and easier to develop and maintain.

In practice, networks are far from perfect, and their protocols and technologies vary widely. The transport layer shields the application layer from these complexities and imperfections. Without it, programmers would have to tailor their applications to every possible network environment. Thanks to the transport layer, they can write software once and run it anywhere. Because of this, networking professionals often draw a distinction between layers 1 to 4, which provide transport services, and the layers above 4, which use those services. The transport layer sits at the critical boundary, acting as the interface between service provider and user. It is the layer most visible to applications and plays a central role in enabling reliable, standardized communication across diverse networks.

17.3 TRANSPORT SERVICE PRIMITIVES

To enable application programs to communicate across a network, the transport layer offers a set of operations known as the transport service interface. This interface acts as a bridge between the application layer and the transport layer, giving programs the ability to interact with the transport services directly. While the network service aims to reflect the capabilities of real-world networks (including their flaws such as packet loss), the transport layer is designed to hide these imperfections and offer reliable communication. This reliability is especially crucial in connection-oriented transport services, which ensure data is delivered accurately and in the correct sequence, despite underlying network issues.

To illustrate this, consider two processes communicating via a UNIX pipe. These processes expect a flawless connection where data input at one end is received at the other without worrying about packet loss, errors, or retransmissions. Similarly, the transport layer emulates this kind of seamless communication between processes running on different machines by correcting errors, managing acknowledgements, and handling lost data behind the scenes.
While the transport layer can also provide unreliable datagram services (similar to what the network layer offers), such services are simpler and mostly used in specific scenarios like streaming multimedia or client-server communication, where speed is prioritized over reliability. However, the primary focus remains on the reliable, connection-oriented transport service, as it is more widely used and complex.

Another significant distinction lies in the intended users of the services. The network service is accessed by transport entities and is mostly hidden from end-user applications. On the other hand, the transport service is directly exposed to application programmers. This necessitates that transport interfaces be user-friendly and practical, offering essential operations to establish, utilize, and terminate connections easily.

A minimal transport interface might include five key primitives (as mentioned in the referenced Fig. 6-2): primitives for connection establishment, data transfer, and connection release. Although basic, such a set of operations is adequate to support many real-world applications that require reliable data communication.

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	Request a release of the connection

Figure 17.2 The primitives for a simple transport service

To understand the practical use of transport service primitives, consider a typical client-server application. The server begins by executing a LISTEN primitive, which usually involves calling a system-level function that blocks the server, keeping it idle and waiting for a client connection. This means the server is passively waiting for a connection request. When a remote client wants to initiate communication, it issues a CONNECT primitive. This action also invokes a library routine or system call, which causes the client process to block temporarily while the transport layer sends a connection request to the server. This request is encapsulated in a transport layer message, also known as a segment, and sent across the network to the server's transport entity.

At this point, it's important to clarify the terminology used in networking. The message exchanged between transport entities (i.e., the sender's and receiver's transport layers) is called a segment. This term is widely used in modern protocols such as TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). In contrast, older networking literature sometimes refers to such messages as TPDU (Transport Protocol Data Unit), though this term is now largely obsolete.

The transport segment is encapsulated within a network layer packet, which is, in turn, embedded within a data link layer frame. This hierarchical wrapping of data units forms a core concept of layered protocol design. When a frame arrives at a destination machine, the data link layer first examines its header to determine whether it is addressed to the local machine. If it is, the payload of the frame is extracted and passed up to the network layer, which performs a similar check on the packet. Once validated, the payload of the packet, which is the transport segment, is handed over to the transport layer. This step-by-step unwrapping of the communication unit illustrates the layered encapsulation model. It is typically shown in a diagram such as Fig. 6-3, where each layer encapsulates and decapsulates its respective protocol data units.

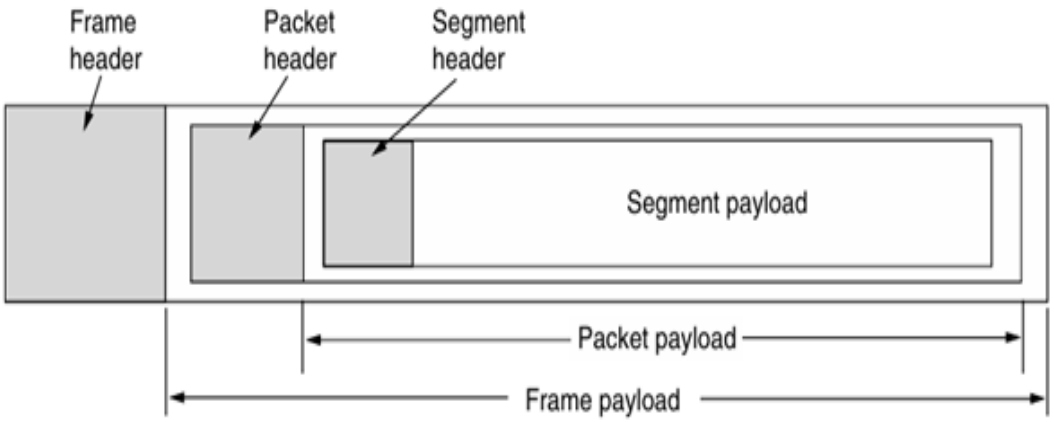


Figure 17.3 Nesting of segments , packets, and frames

Understanding Transport Layer Primitives Through a Client-Server Model

In the world of networking, the transport layer is crucial for enabling communication between processes on different machines. This is especially evident in client-server applications, where transport service primitives define how two processes initiate, manage, and terminate a connection reliably. Let's walk through a complete example—from connection establishment to termination—while emphasizing the internal workings of the transport layer.

1. Connection Establishment: LISTEN and CONNECT

The communication starts when a server process prepares itself to accept connections. It does this by issuing the LISTEN primitive. This is usually implemented as a blocking system call—meaning the server waits (or "sleeps") until a connection request arrives. The transport entity associated with the server stores this state internally and listens for connection attempts from any client.

On the client side, a process wishing to communicate with the server issues the CONNECT primitive. This is also typically a blocking system call. The client's transport layer builds a special segment called a CONNECTION REQUEST, which includes information such as the source and destination addresses, port numbers, and possibly some sequence numbers for reliability mechanisms.

This segment is passed down to the network layer, encapsulated in a packet, and further into a frame at the data link layer before being physically sent across the network. This encapsulation hierarchy follows the OSI model, where each layer adds its own header to the data.

2. Server Acknowledges: CONNECTION ACCEPTED

When the server's transport entity receives the CONNECTION REQUEST segment, it first verifies whether the server process is currently in the LISTEN state. If it is, the server accepts the connection. The transport layer then does two things:

1. It unblocks the server process, allowing it to continue and handle the client.
2. It sends back a CONNECTION ACCEPTED segment to the client.

This segment travels the same path in reverse—encapsulated in a packet and frame—until it reaches the client.

When the CONNECTION ACCEPTED segment arrives, the client is unblocked. Now, both the client and server know that the connection is officially established. Internally, both transport entities store connection state information, such as sequence numbers, acknowledgment status, and window sizes (in case of flow control).

3. Data Exchange: SEND and RECEIVE

Once the connection is up, the two parties can now exchange data using the SEND and RECEIVE primitives.

- The SEND primitive allows an application to push data into the transport layer.
- The RECEIVE primitive is used to extract incoming data from the transport layer.

In simple transport protocols, these primitives may block the calling process. For example, if a process calls RECEIVE but no data has arrived yet, it will wait until a DATA segment is received.

The data flow works like this:

1. Application A (client or server) calls SEND(data).
2. The data is broken into segments by the transport layer (if needed), and sent across the network.
3. When the segment reaches Application B's host, its transport layer verifies integrity (checksums), confirms the sequence number, sends back an acknowledgment (ACK), and finally delivers the data to the application via a RECEIVE.

Even though applications use high-level, clean primitives, underneath, the transport entities manage many tasks:

- Acknowledging received data
- Retransmitting lost segments
- Detecting duplicates
- Managing timers
- Preserving ordering of segments

All these complexities are invisible to the transport user, who simply sees a reliable, ordered stream of bytes, also known as a "reliable bit pipe." This abstraction is one of the greatest achievements of the layered protocol model—it hides the messy details of unreliable physical and network layers.

4. Disconnection: DISCONNECT Primitive

Once communication is complete, the connection must be gracefully terminated to reclaim resources (such as memory buffers and state tables) inside the transport entities. This process is initiated using the DISCONNECT primitive.

There are two disconnection models:

a. Asymmetric Disconnection:

- Either party (client or server) can call DISCONNECT.
- The transport layer sends a DISCONNECT segment to the other side.
- Upon receiving it, the peer transport entity releases the connection and unblocks its local process.
- This is simple but sudden—the connection ends as soon as one party initiates disconnection.

b. Symmetric Disconnection:

More graceful and controlled.

Each side independently signals that it has no more data to send by issuing DISCONNECT.

- However, it can still receive data until the peer also issues a DISCONNECT.
- The connection is only fully closed once both sides have called DISCONNECT.
- This model resembles TCP's half-close mechanism, where one side can finish sending while still accepting input from the other.

5. Connection State Diagram (Figure 6-4)

To clearly represent the flow of events during connection setup and teardown, we can refer to a state diagram like Figure 6-4. Here's a simplified explanation of what this diagram includes:

- *States like:* CLOSED, LISTEN, CONNECTING, ESTABLISHED, DISCONNECTING, and CLOSED.
- Transitions triggered by primitives (CONNECT, LISTEN, DISCONNECT) or incoming segments (CONNECTION REQUEST, CONNECTION ACCEPTED, DISCONNECT).
- In a symmetric disconnection model, the client initiates disconnection, and the server later follows. The connection only truly ends when both sides reach the CLOSED state.

17.4 BERKELEY SOCKETS

Sockets are a set of transport layer primitives introduced in Berkeley UNIX 4.2BSD in 1983 to support communication over the Internet. Since their release, they have become the standard programming interface for building networked applications. The socket interface provides a structured way for applications to create, configure, and manage connections over a network, particularly using TCP (Transmission Control Protocol) for reliable, connection-oriented communication. Unlike the simplified conceptual primitives like CONNECT, SEND, RECEIVE, and DISCONNECT discussed earlier, socket primitives offer more granularity, flexibility, and control, making them better suited for real-world use cases where multiple clients, error handling, and concurrent operations are common.

Creating a Communication Endpoint with SOCKET

The first step in any socket-based application is to create a socket, which is accomplished using the SOCKET primitive. This function call sets up a communication endpoint in the transport layer. It allocates internal resources within the transport entity and returns a file descriptor (an integer value) that uniquely identifies the socket within the process. This file descriptor can later be used with other primitives to configure the socket, send or receive data, or close the connection. The parameters passed to the SOCKET call specify the address family (like IPv4), the socket type (e.g., SOCK_STREAM for TCP), and the protocol (usually 0 to choose the default for the given type). Importantly, this primitive does not assign a network address to the socket—it simply creates an unbound endpoint that can later be configured.

Assigning an Address Using BIND

Once a socket is created, servers must bind the socket to a specific network address and port number, which is done using the BIND primitive. This step is crucial because it allows remote clients to know where to reach the server. The BIND primitive associates the socket with a well-known IP address and port number, essentially giving it a "mailing address" on the network. This separation between SOCKET and BIND is intentional. It allows developers to reuse socket creation code for different roles, and to provide flexibility—some applications require known addresses (like web servers on port 80), while others prefer dynamic or system-assigned addresses. Clients usually skip this step because they don’t care about their source port; the system assigns an ephemeral port for them during connection.

Preparing for Connections with LISTEN

After binding the address, the server uses the LISTEN primitive to indicate readiness to accept incoming connections. This primitive transforms the socket into a passive listening socket and allocates space for a queue of pending connection requests. This is especially useful when multiple clients attempt to connect simultaneously—the operating system can queue their requests and handle them in order. Unlike the abstract LISTEN primitive from earlier models, the socket version of LISTEN is not a blocking call. It simply prepares the socket for incoming connections and tells the operating system how many connections can be queued. The actual blocking behavior happens in the next primitive: ACCEPT.

Handling Clients with ACCEPT

To block and wait for a client connection, the server calls the ACCEPT primitive. When a CONNECTION REQUEST segment arrives from a client (which would have issued a CONNECT call), the transport layer checks the listening socket's queue and processes the request. Upon successful connection, the system creates a new socket—essentially a clone of the original one but dedicated to this specific client. The ACCEPT call then returns a new file descriptor, which the server uses for communicating with the client. This allows the original listening socket to remain open and continue accepting other incoming requests. A common practice in server applications is to create a new process or thread to handle each accepted connection using the new socket, allowing the server to handle many clients simultaneously.

Client Connection with CONNECT

On the client side, the process also begins with a SOCKET call, creating a communication endpoint. However, clients usually do not invoke BIND, as their local address is not important—they are interested only in reaching the server. To initiate a connection, the client uses the CONNECT primitive, which actively starts the connection setup process. This is a blocking call: the client process is suspended until the connection is either successfully established or fails. Internally, this results in a CONNECTION REQUEST segment being sent to the server. If the server accepts the request, it sends a CONNECTION ACCEPTED segment back to the client, which causes the CONNECT call to return and the client to continue execution with a fully established connection.

Data Exchange with SEND, RECEIVE, READ, and WRITE

Once a connection is established, full-duplex data communication can begin. Both the client and server can send and receive data simultaneously. The socket interface provides specific primitives: SEND to transmit data and RECEIVE to accept incoming data. These primitives are more feature-rich than their conceptual counterparts—they allow options such as sending out-of-band data, toggling blocking behavior, or specifying flags for behavior control. However, most socket implementations also support standard UNIX system calls—READ and WRITE—on socket file descriptors. This makes sockets behave like regular files, maintaining the UNIX philosophy of "everything is a file." In practice, developers can choose between the more flexible SEND/RECEIVE or the simpler READ/WRITE based on application needs.

Termination and Cleanup with CLOSE or SHUTDOWN

Though not explicitly mentioned in the previous passage, closing a socket connection is essential to free system resources and notify the peer that communication has ended. This is done using the CLOSE system call or the more selective SHUTDOWN call, which allows one to terminate just the sending, just the receiving, or both directions independently. This corresponds to the DISCONNECT primitive in theoretical models and supports both asymmetric and symmetric disconnection. TCP ensures a graceful connection teardown by exchanging FIN (finish) and ACK (acknowledgment) segments to confirm that both sides have closed properly.

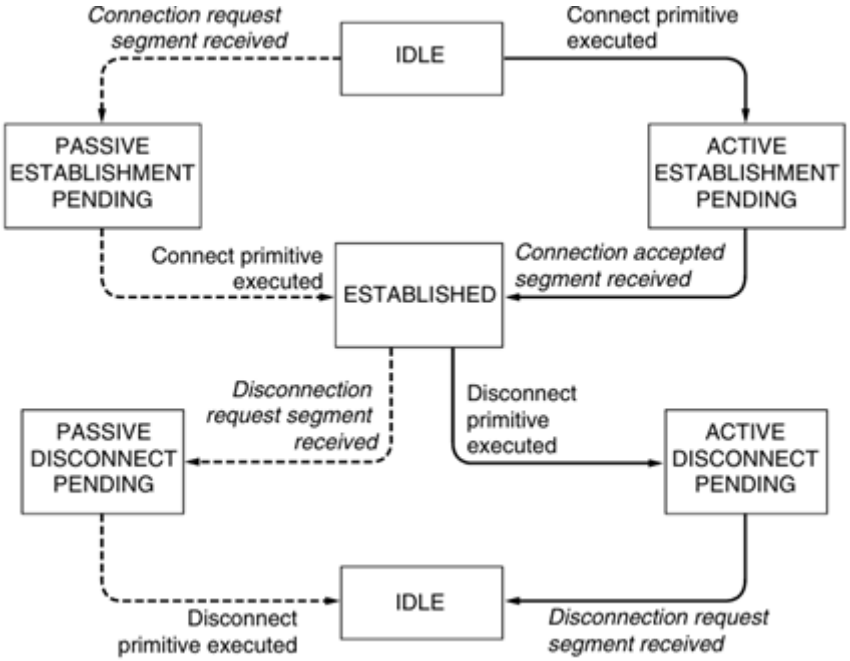


Figure 17.4 A state diagram for a simple connection management scheme. Transitions labeled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The dashed lines show the server's state sequence.

Sockets have become the foundational programming interface for network communication, particularly in applications that rely on the TCP/IP protocol suite. One of the core features of socket-based communication is the symmetric nature of connection release. In TCP, both communicating parties must explicitly issue a CLOSE primitive to fully terminate the connection. This symmetric closing ensures that all data has been properly transmitted, acknowledged, and that neither side is left uncertain about the connection's status. It adheres to TCP's four-step connection termination process, which is designed to maintain the integrity and order of data flow even during disconnection. This careful and deliberate closure mechanism further emphasizes the reliability that TCP—and by extension, sockets—promise to application developers.

The popularity of sockets stems from their ability to abstract the complexities of transport layer services into a consistent, flexible, and widely supported API. Originally introduced in Berkeley UNIX as part of the 4.2BSD release, the socket interface was designed to provide

access to TCP's connection-oriented services. These services manifest as a reliable byte stream, which can be thought of as a dependable, ordered, error-free pipeline between two processes, whether on the same machine or across the globe. What makes sockets particularly powerful is that this same API can be used with other transport protocols as well. For instance, while TCP uses sockets to establish a reliable stream, the same socket functions can be used with UDP to support connectionless communication. In such scenarios, a call to CONNECT does not establish a persistent session like in TCP, but rather sets the destination address for future SEND and RECEIVE operations. This model allows sockets to function efficiently even in lightweight communication tasks, such as DNS lookups or live media streaming.

Additionally, the socket API can be extended to support message-oriented communication through primitives like SENDTO and RECEIVEFROM. These primitives are particularly useful in connectionless services, allowing each message to specify its destination or source independently. This is ideal for applications that need to communicate with multiple peers without establishing dedicated connections for each one. The socket interface is also compatible with newer transport protocols that introduce additional features, such as congestion control without reliability, or message-based communication rather than byte streams. For example, the Datagram Congestion Control Protocol (DCCP) builds on UDP by adding congestion control mechanisms, making it suitable for applications like real-time video where timely delivery is more critical than guaranteed delivery. Despite sharing the same socket interface, the behaviors of these protocols differ significantly. Thus, it is crucial for developers to understand the nature of the transport service they are employing, even if the API appears uniform.

However, as the demands of modern networked applications evolve, the limitations of the traditional socket interface become apparent. One significant limitation is its approach to handling multiple simultaneous data streams between the same pair of hosts. In contemporary applications like web browsers, which often request numerous objects from a single server, each object typically necessitates a separate socket connection. This leads to inefficient use of resources and fragmented congestion control, since each connection is managed independently by the transport layer. Consequently, the burden of coordinating multiple streams falls on the application itself, making the overall design less efficient and more complex. This shortcoming has led to the development of newer transport protocols and APIs that better support grouped or structured streams.

Two notable examples that attempt to overcome these limitations are SCTP (Stream Control Transmission Protocol) and SST (Structured Stream Transport). SCTP, defined in RFC 4960, introduces the concept of multi-streaming within a single connection, allowing multiple data flows to be managed under a unified congestion control mechanism. This not only improves efficiency but also ensures faster recovery from individual stream failures. Similarly, SST extends the transport interface to support complex stream hierarchies, multiplexed connections, and even hybrid models that combine connection-oriented and connectionless traffic. These advanced capabilities require enhancements to the traditional socket API, as the one-stream-per-socket model is no longer sufficient to express such interactions. In addition, these protocols introduce support for features like multi-homing and multipath routing, enabling data to flow over multiple network interfaces for improved reliability and performance. Whether or not these newer models will gain widespread adoption remains to be seen, but they clearly indicate that while the socket interface has been remarkably successful, it is not the final word in the design of transport service APIs. As applications

grow increasingly complex and performance-sensitive, the need for more expressive and efficient interfaces will continue to shape the evolution of network programming.

17.5 AN EXAMPLE OF SOCKET PROGRAMMING: AN INTERNET FILE SERVER

a very basic client-server architecture using real socket function calls in C on a UNIX system, offering an excellent opportunity to understand how transport layer communication translates into actual code. The program is a primitive Internet file server coupled with a client that connects to it, requests a file, and receives the file contents. While the example is intentionally simplified to focus on the core idea rather than production-level robustness, it demonstrates the essential components of socket programming: creation, binding, listening, accepting, connecting, sending, and receiving data. The notable aspect of this example is its portability across UNIX-based systems on the Internet. A server running the program can be hosted on any such system, and a client on a remote machine anywhere in the world can connect to it, provided there are no firewall or network restrictions. When the client runs, it fetches the specified file and outputs the contents to standard output, allowing users to redirect the stream to a file or pipe, effectively simulating a basic download mechanism.

Examining the server code reveals several crucial steps that align closely with the theoretical understanding of socket primitives. The program starts by including standard C library headers, particularly those that handle system calls, data types, and socket structures (<stdio.h>, <stdlib.h>, <unistd.h>, <sys/socket.h>, <netinet/in.h>, and <arpa/inet.h>). These are essential for defining the networking elements such as socket addresses and data buffers. Following the headers is the definition of a server port, chosen arbitrarily as 12345. This number is critical because clients will use it to reach the server. Port numbers above 1024 are commonly used for user applications, while those below 1024 are reserved for system-level (privileged) services such as SSH or HTTP. The code assumes no conflict exists on port 12345. If another process is already bound to that port, the server would fail to bind, causing it to exit with an error. Therefore, selecting an available port is essential in real deployment.

After the port definition, two constants are set to guide how the server behaves. The first constant typically defines the buffer size for reading chunks of data during file transmission. This buffer controls how much data is read or written in one I/O operation. A standard chunk size might be 1024 bytes (1 KB), balancing performance with memory efficiency. The second constant determines the maximum number of queued incoming connections, corresponding to the backlog parameter in the LISTEN call. If more clients attempt to connect while this limit is reached, the server drops the new connections until space becomes available. This mechanism ensures that the server can control its workload and not be overwhelmed by simultaneous incoming requests. These two constants help structure how the server handles data and client load, even though the logic in this specific example is basic and lacks advanced features like concurrency, input validation, authentication, or logging.

Despite its limitations, the example serves as a fundamental template for learning how sockets function in a real application. It shows the transformation from abstract ideas like “connection,” “send,” and “receive” into tangible C function calls like socket(), bind(), listen(), accept(), connect(), read(), and write(). This is a valuable step in bridging the gap between network theory and programming practice. It also implicitly demonstrates the importance of choosing appropriate parameters, error checking, and the sequencing of socket operations to maintain a predictable and stable communication channel. Once compiled and executed, the server listens on a known port, the client initiates a connection with the server’s

IP and port, and upon successful establishment, the requested file is read in chunks by the server and sent to the client, which writes it to output. Though simple, this process embodies the essence of what transport services aim to offer—a reliable channel for sending and receiving data between distributed processes.

The provided example delves into the detailed workings of a very simple Internet file server and its corresponding client, implemented using sockets in C on a UNIX system. Despite its simplicity and numerous limitations, this implementation serves as an excellent educational tool to understand real-world usage of socket primitives and the lifecycle of a basic client-server interaction over TCP. The server begins by declaring necessary local variables and proceeds to initialize a data structure representing the server's IP address and port. The memset call is used to zero out the structure to ensure a clean initialization. The server's port is filled in using the htons function to ensure the correct byte order across different architectures (little-endian vs big-endian), ensuring network interoperability. The server then creates a socket with the socket() system call and checks for errors, a crucial step in real systems where robustness is key. To allow reuse of the server port for repeated use without delay, it uses the setsockopt() call. This is particularly useful for development or rapid restarts, preventing the "address already in use" error. After this, the socket is bound to the IP address using bind(), and any failure here leads to termination. Finally, the server calls listen() to begin waiting for incoming client requests, and it specifies a backlog size (queue limit) that determines how many incoming connections can wait while the server is busy.

Once initialization is complete, the server enters its main execution loop, which continues indefinitely until the server process is externally killed. Inside this loop, the accept() call is used to block the server until a client requests a connection. When this happens, a new socket descriptor is returned, distinct from the original listening socket, and this new socket is used for two-way communication with the client. Unlike traditional UNIX pipes which are one-way, sockets support full duplex (bidirectional) communication, enabling both reading and writing on the same descriptor. After the connection is established, the server reads the file name sent by the client. If the data is not immediately available, the read() call blocks. Once the file name is received, the server attempts to open the file. If successful, it enters a loop where it reads the file in chunks (as per the defined buffer size) and writes each chunk to the socket, thus transmitting the file to the client. Once the entire file is sent, the server closes both the file and the connection socket and then returns to the top of the loop to wait for the next client connection. This approach, although functional, is inherently sequential; the server handles only one client at a time and is blocked during all I/O operations. It lacks concurrency, multithreading, or asynchronous I/O, all of which are essential in real production-grade servers.

Turning to the client, it begins by checking if the program has received exactly three arguments: the program name, the server's hostname (like flits.cs.vu.nl), and the desired file path on the server machine. The client uses the gethostbyname() function to resolve the given hostname into an IP address using DNS, a service which will be explained further in the book's later chapter on DNS. Then, the client creates a socket and attempts to establish a TCP connection to the server using the connect() call. This operation blocks until the connection is successfully established or fails. If the server is running, listening on the correct port, and has room in its listen queue, the connection proceeds. The client then sends the file name to the server using a write() system call, sending one extra byte (a null character \0) to signal the end of the file name string, ensuring the server knows when the name ends. After this, the client enters a loop where it reads data from the socket in blocks and writes it directly to the

standard output (stdout). Because UNIX allows standard output to be redirected, the received file data can be saved to a local file or piped into another process, effectively making this client a basic file retrieval tool.

The fatal() function, defined in the client code but referenced in both, is a simple utility that prints an error message and exits the program. The server would also need such a function for consistent error handling, though it's not included in the printed snippet for brevity. It's important to understand that this example has many flaws. It lacks proper error checking and reporting, cannot handle concurrent clients, and doesn't include any form of access control, authentication, or security. It assumes the file name fits entirely within a fixed buffer and that the transmission is atomic, which may not be true in all environments or under all network conditions. These simplifications are intentional for educational clarity but would be unacceptable in a real-world application. Nonetheless, the program does illustrate the fundamental operations and structure of a basic TCP-based file server-client architecture. Readers are encouraged to improve upon it—by adding multithreading, better error handling, or protocol refinements—as part of their learning process. Further guidance on these topics is available in supplemental resources such as Donahoo and Calvert's socket programming books, which explore advanced socket programming practices in greater detail.

17.6 SUMMARY

This chapter introduces the transport layer, which provides reliable, end-to-end communication between processes on different hosts. It describes the services offered to upper layers, including connection establishment, error control, and flow control. Transport service primitives define operations like connect, send, receive, and disconnect, enabling communication between applications. The chapter also covers Berkeley sockets, a practical API that allows applications to use network services easily. Overall, the transport layer acts as a bridge between the network layer and application layer, ensuring efficient and dependable data transfer.

17.7 TECHNICAL TERMS

Transport layer, Sockets, Berkeley sockets

17.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the services provided by the transport layer to upper layers.
2. Describe transport service primitives and their role in connection management.
3. Discuss the concept and functions of Berkeley sockets.
4. Explain how the transport layer ensures reliable data transmission.
5. Compare the transport layer's role with that of the network layer.

Short Questions:

1. What is the main function of the transport layer?
2. Name two services provided by the transport layer to upper layers.
3. What are transport service primitives?
4. What is a Berkeley socket?
5. Why is the transport layer important for reliable communication?

17.9 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008).

Mrs. Appikatla Pushpa Latha

LESSON- 18
THE INTERNET TRANSPORT PROTOCOLS TCP AND UDP

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the features and purpose of UDP.
- Learn about Remote Procedure Call (RPC) and Real-Time Transport Protocol (RTP).
- Study TCP, its service model, and protocol structure.
- Learn TCP connection management, congestion control, and timer management.
- Understand variations like wireless TCP, UDP, and transactional TCP.

STRUCTURE OF THE LESSON:

18.1 INTRODUCTION

The Internet's transport layer includes two main protocols: UDP and TCP, which serve different purposes and complement each other. UDP (User Datagram Protocol) is a connectionless protocol that provides minimal services—mainly sending packets from one application to another without establishing a connection or guaranteeing delivery. It leaves reliability and flow control to the application itself, offering speed and low overhead for use cases like streaming or DNS.

On the other hand, TCP (Transmission Control Protocol) is connection-oriented and provides much more functionality. It ensures reliable data delivery through acknowledgements and retransmissions, manages flow control to prevent overwhelming the receiver, and controls congestion to avoid network overload. TCP handles these complexities so applications don't have to.

Because UDP is simpler, it is introduced first when studying transport protocols. Although UDP-based protocols often run in user space and may appear as applications, their mechanisms (like multiplexing or basic reliability) are fundamental and common enough to be treated as part of transport services.

18.2 INTRODUCTION TO UDP

The Internet protocol suite includes two primary transport layer protocols: one is connectionless, and the other is connection-oriented. These protocols serve different purposes and are designed to complement each other. The connectionless protocol is UDP, or User Datagram Protocol, which offers a minimal and lightweight service by allowing applications to send IP datagrams without the overhead of establishing and maintaining a connection. On the other hand, the connection-oriented protocol is TCP, the Transmission Control Protocol, which provides a comprehensive suite of services such as reliable data delivery, connection setup and teardown, flow control, congestion control, and ordered delivery. TCP handles all the complexities involved in reliable communication so that applications do not need to manage them themselves.

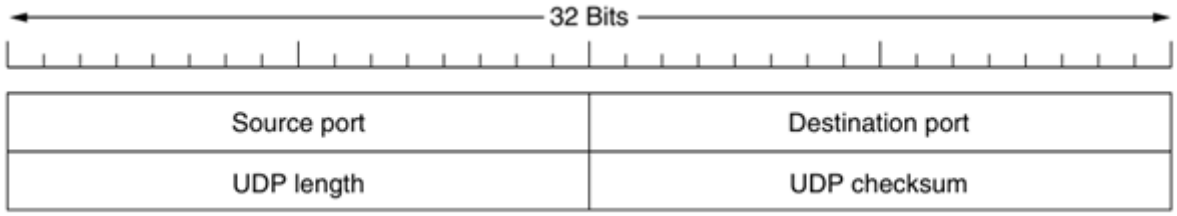


Figure 18.2 The UDP header

UDP, as described in RFC 768, is designed to be a very simple protocol that allows applications to send messages, called datagrams, to other hosts without the need for prior communications to set up special transmission channels or data paths. It transmits segments composed of an 8-byte header followed by the actual data payload. This header includes the source and destination port numbers, a length field, and a checksum. The ports are used to identify specific processes within the source and destination hosts, which allows multiple applications to use UDP simultaneously without interfering with each other. The ports serve as demultiplexing points so that when a datagram arrives, the operating system can examine

the destination port number and deliver the data to the appropriate receiving process. This mechanism is similar to each application having its own mailbox for receiving messages.

The length field in the UDP header specifies the total length of the segment, including the 8-byte header and the data portion. This length has a minimum value of 8 bytes and a maximum of 65,515 bytes. The optional checksum provides basic error detection for the segment. It is calculated over the entire UDP segment, including a pseudoheader that contains information from the IP layer such as source and destination IP addresses, the protocol number, and the UDP segment length. This inclusion of the pseudoheader enhances the reliability of detecting misdelivered packets, although it slightly violates the principle of strict layer separation in protocol design. The checksum is calculated using one's complement arithmetic, where the sum of all 16-bit words is complemented. If the result at the receiver is not zero, the packet is assumed to be corrupted. If no checksum is used, the field is set to zero; however, omitting the checksum is typically discouraged unless the application is tolerant of data corruption, such as in some forms of real-time multimedia transmission.

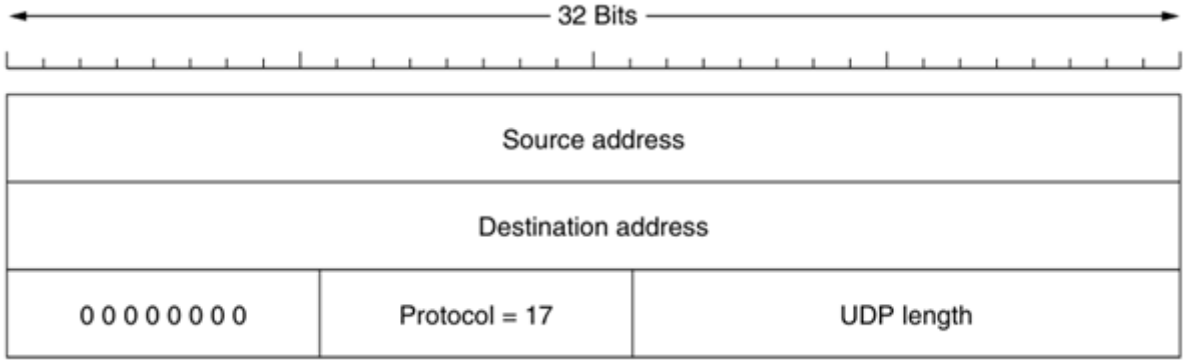


Figure 18.2 The IPv4 pseudoheader included in the UDP checksum

Despite the useful addition of ports and an optional checksum, UDP is a very limited protocol in terms of functionality. It does not include mechanisms for flow control, congestion control, data ordering, or retransmissions. This means that if a datagram is lost, duplicated, arrives out of order, or is corrupted, UDP does not correct the problem. These tasks are left entirely up to the application layer. The absence of these features makes UDP lightweight and efficient, especially suitable for applications where simplicity, speed, and low overhead are more critical than guaranteed delivery. This trade-off is especially useful in environments where some packet loss can be tolerated or quickly corrected at the application level.

UDP is particularly advantageous in client-server applications where only a small exchange of messages is required. A typical use case involves a client sending a single short request and receiving a short reply from a server. This kind of communication does not warrant the overhead of TCP’s connection setup and management. UDP enables such communication to occur with a minimum number of packets—often just one in each direction. If either packet is lost, the client can detect the problem via a timeout and simply resend the request. This simplicity reduces protocol overhead and often improves performance in such scenarios.

A well-known application of UDP is the Domain Name System (DNS), which translates human-readable domain names into IP addresses. When a client wants to resolve a domain like www.example.com, it sends a UDP packet containing the domain name to a DNS server. The server responds with a UDP packet containing the corresponding IP address. This exchange involves only two messages and avoids the overhead of a TCP connection, making

it fast and efficient. This use of UDP exemplifies its value in quick request-reply protocols, especially when reliability can be managed by higher-level logic or when minor data loss is acceptable.

In summary, UDP provides a minimal transport service that offers datagram transmission with port-based demultiplexing and optional error detection. Its simplicity and low overhead make it ideal for applications where speed and efficiency matter more than reliability, such as DNS lookups, real-time multimedia, or simple network services. However, for more demanding applications that require reliable data transmission, flow control, and congestion management, the connection-oriented TCP is the preferred choice. The existence of both UDP and TCP in the Internet protocol suite allows developers to choose the right tool for the specific needs of their applications.

18.3 REMOTE PROCEDURE CALL

Sending a message to a remote host and receiving a reply is conceptually very similar to invoking a function or procedure in a programming language, where you supply parameters and expect a result in return. This similarity has inspired the development of programming models that treat network communication like procedure calls, abstracting away the complexities of the underlying transport mechanisms. One significant advantage of this approach is that it greatly simplifies the development of network applications by hiding the intricate details of message construction, transmission, and reception from the programmer. For instance, instead of a developer explicitly crafting a UDP packet to query a DNS server, they could simply use a function like getIPAddress(hostname), which internally handles the network communication. If the reply does not arrive quickly enough, the function can automatically retry, making the entire interaction feel like a typical local function call.

The most influential contribution in this area was made by Birrell and Nelson in 1984. They proposed a mechanism that allows a process on one machine to call a procedure on another machine as if it were local. In their model, when a process on one computer invokes a procedure residing on another computer, the invoking process is temporarily suspended while the called procedure executes on the remote machine. Data can be transmitted from the client to the server in the form of parameters, and results are returned in the same fashion. This mechanism, known as Remote Procedure Call (RPC), hides the details of network communication from the programmer, providing a clean and intuitive programming model. Typically, the caller is referred to as the client, while the callee is called the server.

To implement RPC, both the client and server programs are supplemented with additional components known as stubs. The client stub resides in the client’s address space and acts as a proxy for the server procedure. It has the same name as the remote procedure and is responsible for packaging the parameters into a network message—a process known as marshaling. On the server side, a server stub receives the message, unpacks the parameters (unmarshaling), and invokes the actual server procedure with the correct arguments.

The flow of an RPC starts when the client program makes a call to the client stub, just as it would for a local procedure. The client stub then marshals the parameters and makes a system call to send the request message over the network. The client's operating system handles the transmission of the message to the server machine. Upon arrival, the server's operating system passes the message to the server stub, which unmarshals the parameters and calls the server procedure. The return path follows the reverse sequence, delivering the result back to

the original caller. This entire process is seamless to the programmer, who simply perceives it as a standard function call.

While RPC presents a very elegant abstraction, it also introduces certain challenges, one of the most significant being the handling of pointer parameters. In traditional local procedure calls, pointers can be safely passed because both the caller and callee reside in the same virtual address space. However, with RPC, the client and server operate in separate address spaces, often on different physical machines. As a result, pointers become meaningless outside their originating address space and cannot be directly passed. This limitation requires developers to avoid pointer-based data structures or to use serialization mechanisms that convert complex data into a transmittable format.

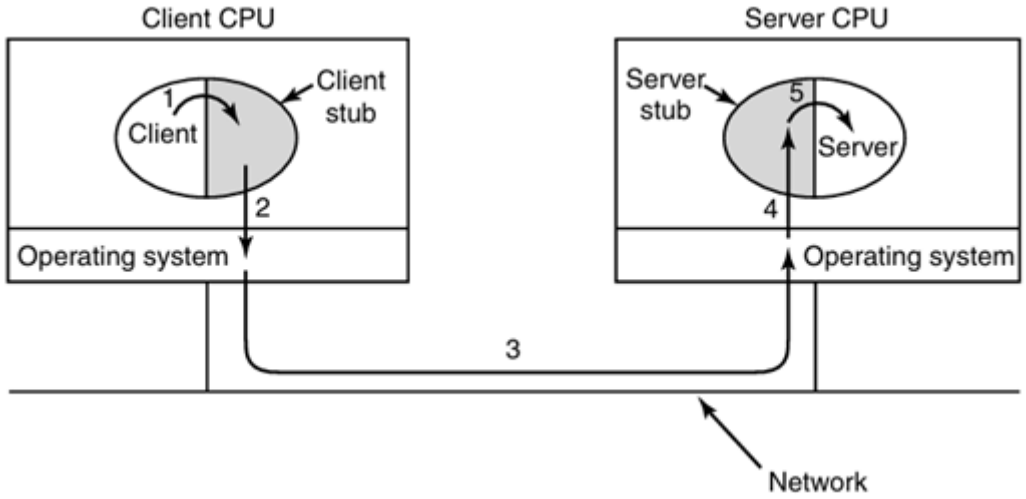


Figure 18.3 Steps in making a remote procedure call. The stubs are shaded.

In conclusion, RPC provides a powerful model for building distributed systems by enabling developers to write networked code in a style that mimics conventional local programming. Despite its simplicity and usefulness, it requires careful consideration of data formats and memory management to ensure correct operation across different systems and environments.

In some situations, clever methods can be employed to make it appear that pointer parameters can be passed across a network in remote procedure calls. For example, if the first parameter to a remote procedure is a pointer to an integer, the client stub can marshal the value being pointed to—say, the integer k—and transmit it to the server. Upon receiving it, the server stub reconstructs a pointer to this copied value and provides it to the server procedure, which then behaves as if it had received a legitimate pointer. After the server procedure completes its execution, any changes it made to k are copied back by the server stub and sent to the client stub, which overwrites the old value with the new one. This process essentially mimics call-by-reference behavior by using a call-by-copy-restore mechanism. Although this trick can successfully simulate pointer passing in simple cases, it does not scale well for complex data structures. If a pointer refers to a graph or another elaborate memory structure with dynamic references, this method breaks down, as it becomes extremely difficult to reconstruct the relationships and content remotely. Consequently, practical implementations of remote procedure calls must restrict the types of parameters that can be passed, especially when dealing with pointers.

Another significant challenge arises in languages that are not strongly typed, such as C. In such languages, it is common to write procedures that take arrays or vectors without specifying their length, instead relying on conventions such as special terminating values. These conventions work fine locally but pose serious problems for marshaling data in RPCs because the stub has no way to determine the actual size of the array. Without a known boundary or count, the client stub cannot decide how many bytes to send, potentially leading to incomplete or incorrect data transfers.

A third complication concerns procedures with variable or dynamic parameter types. A classic example is the printf function in C, which can accept a wide variety of parameter types and numbers—integers, floats, strings, and more—all based on the format string. Because of this flexibility, it is nearly impossible to automate the marshaling of parameters for such functions without knowing their exact types at runtime. This makes them unsuitable candidates for remote procedure calls unless the language or development environment imposes strict type constraints. While one could argue for using RPC only with strongly typed languages to avoid these issues, such a restriction would likely be unpopular, especially with the large developer base using C and C++.

There is also the problem of global variables. In traditional local function calls, procedures can communicate indirectly via global variables, which are accessible throughout the program. But when a function is called remotely on another machine, these global variables are no longer shared between the client and server, breaking the implicit communication pathway. This difference in memory scope necessitates avoiding or rethinking the use of global variables in programs designed to use RPC.

Despite these limitations, RPC remains a practical and powerful paradigm, widely adopted in distributed systems. The key to its success is in establishing certain constraints and adhering to disciplined programming practices. For simple and fast client-server interactions, especially those that fit into a single request-response cycle, UDP is a good transport protocol to use as a base for RPC. Its lightweight nature and minimal setup requirements allow requests and replies to be encapsulated in single packets, minimizing latency and overhead.

However, any real-world RPC implementation built on UDP must handle the potential for message loss. Since neither the request nor the reply is guaranteed to arrive, the client must be equipped with a timer to trigger retransmissions if no reply is received within a certain time frame. It is also worth noting that a successful reply acts as an implicit acknowledgment of the request, which simplifies the protocol. Yet, when requests or responses exceed the maximum UDP payload size, mechanisms are needed to handle message fragmentation and reassembly. Furthermore, in systems that support concurrent calls, each request must carry a unique identifier so that replies can be correctly matched to their originating request, especially when responses may arrive out of order.

A final and important consideration in RPC design is the nature of the procedure being called. Some procedures are idempotent, meaning they can be executed multiple times without adverse effects—for example, looking up a DNS record. These operations are safe to repeat if a reply is lost, as resending the same request results in the same answer, assuming the data hasn't changed in the meantime. But not all operations are idempotent. For example, a procedure that updates a record, increments a counter, or charges a customer account cannot be executed multiple times without causing inconsistencies or harm. For such operations, relying on UDP's best-effort delivery is risky. Instead, a more reliable protocol like TCP

should be used, as it ensures that data is delivered exactly once in the correct order, which is essential for operations with side effects. By using TCP, one can establish a connection, manage acknowledgements, and ensure that sensitive operations are performed safely and correctly.

18.4 REAL TIME TRANSPORT PROTOCOLS

Client-server RPC is a well-known area where UDP sees extensive use due to its speed and low overhead, especially for simple request-reply interactions. Another major domain in which UDP is widely used is real-time multimedia communication. Applications like Internet radio, video conferencing, video-on-demand, music streaming, and IP telephony all demand low latency and timely delivery of data. As these applications evolved, it became clear that they were all building their own versions of transport mechanisms tailored to real-time data. To streamline this process and avoid redundancy, a standardized protocol was developed—RTP (Real-time Transport Protocol), defined in RFC 3550.

RTP is now the de facto protocol used in multimedia applications for handling real-time data like audio and video streams. It runs in user space over UDP, giving developers flexibility while leveraging UDP's minimal transmission delay. In this setup, the multimedia application handles multiple streams (such as audio and video), which are then passed to the RTP library in user space. This library takes care of multiplexing the streams, formatting them into RTP packets, and sending them through the UDP socket to the network layer. The receiver side performs the reverse process: the RTP library extracts the audio and video streams, and the application handles the playback.

Although RTP is implemented at the application level, it functions like a transport protocol. It provides essential transport-like services such as sequencing, timestamping, and payload identification, all critical for correct media playback. Due to this hybrid nature, RTP is often described as a transport protocol implemented in the application layer. This design allows it to remain generic and reusable across various multimedia applications while taking advantage of UDP's speed and simplicity.

RTP: The Real-Time Transport Protocol

The main role of the Real-time Transport Protocol (RTP) is to multiplex multiple real-time data streams into a single stream of UDP packets, which can be delivered either to one destination (unicasting) or to multiple destinations (multicasting). Since RTP is built on top of standard UDP, it inherits UDP's simplicity and lack of reliability. As a result, RTP packets are treated just like any other UDP packets by the network, without any special quality-of-service guarantees unless explicitly enabled through IP-layer mechanisms. This means that RTP packets may suffer from the usual issues of network communication, such as delays, corruption, and loss, but these conditions are tolerated within the protocol design, especially considering the nature of multimedia traffic.

To help receivers manage the challenges of real-time multimedia transmission, RTP includes features that allow for better synchronization and error detection. One of these is the use of sequence numbers in the RTP header. Each RTP packet is assigned a number that is incremented by one from the previous packet. This makes it possible for the receiver to detect if any packets are missing. However, unlike reliable protocols, RTP does not support retransmissions, as retransmitted packets are often useless for real-time playback. Instead, the

application must decide how to handle losses—by skipping frames in video, or interpolating audio samples, for example.

RTP also allows the payload of each packet to contain one or more samples, with the format determined by the application. To support compatibility, RTP defines several profiles, each of which may allow multiple encoding formats. This means audio data, for instance, could be encoded using PCM, GSM, MP3, or other codecs. RTP itself doesn't manage encoding; it simply provides a way for the sender to indicate which encoding is being used via a header field.

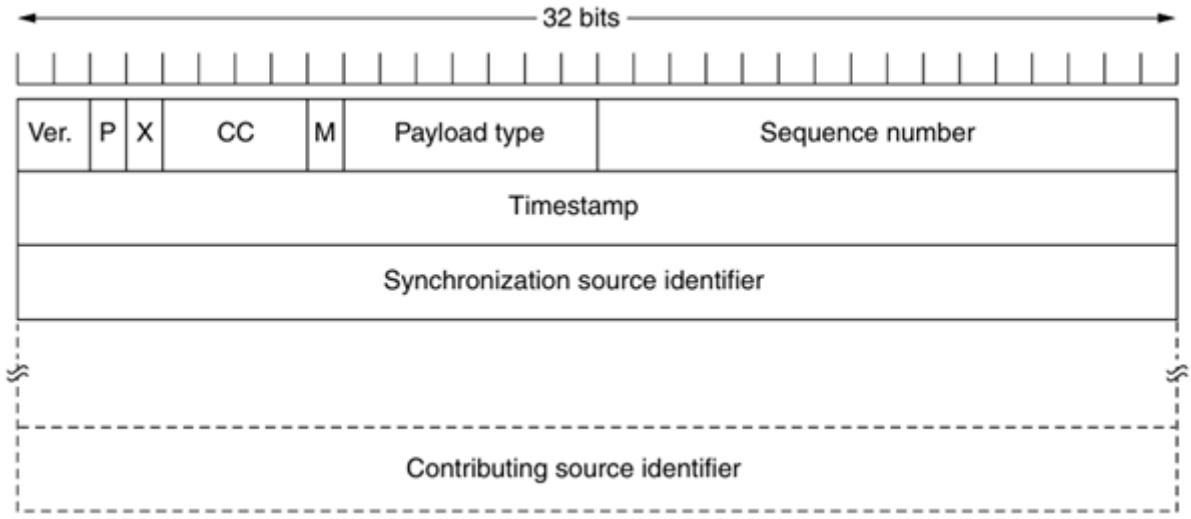


Figure 18.4 The RTP header

Timestamping is another essential feature provided by RTP, enabling the destination to play out audio or video streams at the correct time intervals. Each packet includes a timestamp indicating when the first sample in the packet should be played, relative to the start of the stream. This allows the receiver to buffer incoming packets and then reproduce the stream smoothly, even if the packets arrive irregularly. Furthermore, timestamping makes it possible to synchronize multiple streams, such as audio and video tracks or multiple language audio streams. If the timestamps are derived from a single common counter, playback can remain in sync even if the streams take different paths through the network or are received at different times.

The structure of the RTP header supports all these functions. It consists of three 32-bit words with optional extensions. The first word includes fields like the version number (currently 2), a padding bit (P), and an extension bit (X). Padding is used when the payload must align to a multiple of four bytes, and the X bit signals the presence of an additional header extension, whose format is flexible and meant to accommodate future needs. These design choices ensure RTP is both robust and extensible for the evolving demands of real-time multimedia communication.

RTCP: The Real-Time Transfer Control Protocol

RTCP, or the Real-time Transport Control Protocol, is a companion protocol to RTP and is defined alongside it in RFC 3550. Unlike RTP, which is responsible for transporting actual media data such as audio or video streams, RTCP serves auxiliary but essential functions like

feedback, synchronization, and user interface support. It does not carry any media samples itself, but it helps optimize the delivery and quality of those samples.

One of the key roles of RTCP is to provide feedback about the network conditions to the sources of media. This feedback can include measurements of delay, delay variation (jitter), available bandwidth, and network congestion. By receiving such continuous updates, the media source or encoder can dynamically adjust its behavior to match the current network situation. For instance, if the network bandwidth is high and stable, the sender might switch to a higher-quality encoding format like MP3. Conversely, during network congestion or reduced bandwidth, it may fall back to simpler, more compact formats such as 8-bit PCM or delta encoding. RTP includes a Payload type field to inform receivers of the encoding method used in each packet, enabling the seamless switch between formats as needed.

However, one complication with feedback in group settings is scalability. In multicast scenarios with many participants, if each receiver were to send RTCP reports at a fixed rate, the total control traffic would become excessive. To prevent this, RTCP includes mechanisms to throttle the rate of reports. The general guideline is to limit RTCP traffic to no more than about 5% of the total session bandwidth. Each participant adjusts its reporting rate based on the total estimated number of participants and the overall media bandwidth, which it learns from the session's sender and other RTCP messages.

RTCP also contributes to synchronizing multiple data streams. This is important because different media streams—such as audio and video—may be generated by different devices, using clocks with different granularities and drift characteristics. RTCP helps keep these streams aligned so that playback remains smooth and coordinated. For instance, in a video conference, it ensures that the speaker’s voice and facial expressions stay synchronized.

Lastly, RTCP enables sources to be identified using human-readable names, typically in ASCII text. This identification feature can be useful in user interfaces, for example, to show the name of the current speaker in a video meeting or to indicate the source of a stream in a multimedia application.

Altogether, RTCP complements RTP by improving the media delivery experience without introducing significant overhead, especially in large and dynamic communication environments.

Playout with Buffer and Jitter Control

Once media data reaches the receiver, it must be played at the correct time to maintain a smooth and synchronized user experience. However, due to the nature of packet-switched networks, packets often experience varying amounts of delay during transit, a phenomenon known as jitter. Even if packets are sent at regular intervals from the sender, they might arrive at irregular intervals at the receiver. This inconsistency in arrival times can cause issues like stuttering video or garbled audio if the data is played immediately upon arrival.

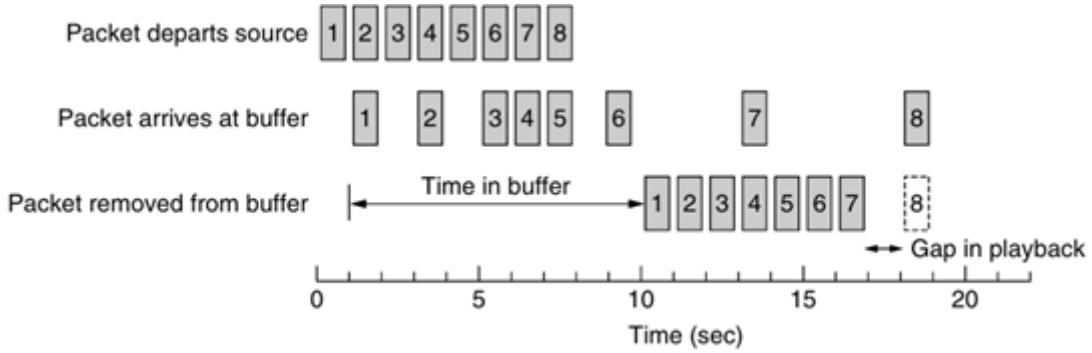


Figure 18.5 Smoothing the output stream by buffering packets

To overcome jitter, receivers use buffering, where incoming packets are temporarily stored before being played out. For example, a receiver might start buffering media from the moment the first packet arrives and delay actual playback by a few seconds. This buffer absorbs the variation in arrival times and ensures that media can be played at regular intervals, regardless of minor network delays. A diagram in the text shows packets being sent and received over time, with playback beginning only after several packets have arrived and are buffered. This creates a steady stream of data for playback, even if some packets arrive slightly later than others.

However, excessive delay can still cause problems. For example, if a packet is delayed too much and does not arrive in time for its scheduled playback, the application must decide whether to skip it or pause the playback. Skipping is common in real-time applications like voice or video calls, where waiting would be disruptive. In contrast, media players for streaming content might pause and resume once the delayed packet arrives. This can be managed by increasing the buffer size, commonly to about 10 seconds for non-live streaming, which allows the player to handle most network variations smoothly. In contrast, live applications need smaller buffers to maintain responsiveness, which makes them more susceptible to jitter.

Determining the playback point—how long the receiver should wait before starting playback—is crucial. This decision is based on measuring the amount of jitter. Applications can compare RTP timestamps with actual arrival times to calculate delay samples. These values can change over time as network conditions fluctuate, so applications often adapt the playback point dynamically. However, changes in playback timing must be handled carefully to avoid glitches or noticeable jumps in playback. For instance, adapting the playback point between talkspurts in a voice conversation, rather than during speech, can help mask any transitions from the user.

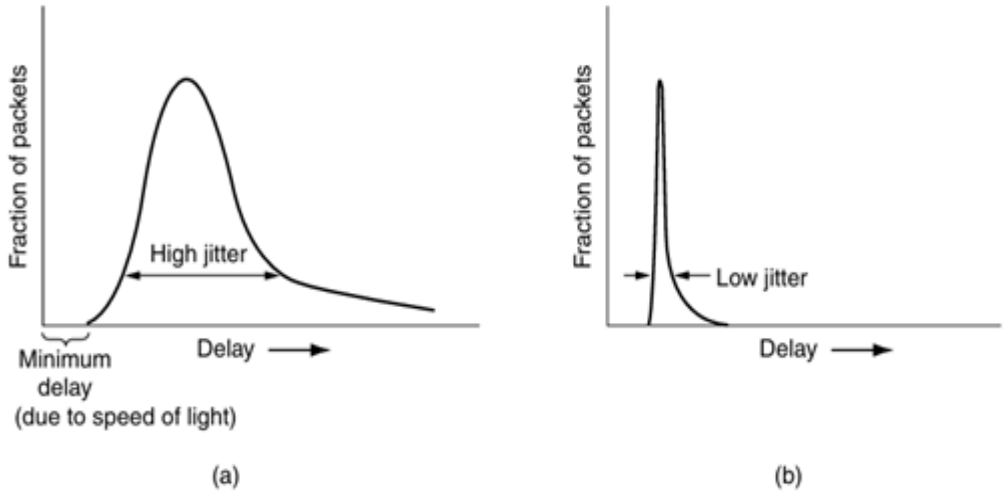


Figure 18.6 (a) High jitter (b) low jitter

If the overall network delay is too long, particularly for live applications, it negatively impacts performance. Since the actual propagation delay (based on the physical distance and speed of signal transmission) is unavoidable, reducing jitter becomes the primary means of minimizing the total delay. This may involve improving the underlying network quality, such as by using better Quality of Service (QoS) settings like expedited forwarding in differentiated services. In such cases, enhancing the network infrastructure is the only viable long-term solution for minimizing jitter and ensuring timely media playback.

18.5 THE INTERNET TRANSFER PROTOCOLS: TCP

UDP is a simple protocol and it has some very important uses, such as client server interactions and multimedia, but for most Internet applications, reliable, sequenced delivery is needed. UDP cannot provide this, so another protocol is re quired. It is called TCP and is the main workhorse of the Internet. Let us now study it in detail.

TCP, or Transmission Control Protocol, was designed with the goal of providing a reliable, end-to-end communication service across unreliable and heterogeneous internetworks. Unlike a single homogeneous network, an internetwork might consist of links and nodes with drastically varying characteristics in terms of topology, bandwidth, latency, and even the size of data packets it can handle. Given this variability, TCP was developed to be highly adaptable and capable of maintaining reliability despite differing and unstable conditions across the path. It is a protocol specifically engineered to detect, handle, and recover from various forms of network failure, ensuring data delivery in the correct order without duplication or loss.

The original definition of TCP was given in RFC 793 back in 1981. Since its introduction, it has undergone numerous enhancements and refinements to improve functionality, robustness, and performance. These enhancements have been documented in a series of additional RFCs (Request for Comments), which together outline the modern TCP standard. These include clarifications and bug fixes (RFC 1122), improvements for high-speed networks (RFC 1323), support for selective acknowledgements (RFC 2018), introduction of congestion control algorithms (RFC 2581), repurposing TCP header fields for quality of service (RFC 2873), improved handling of retransmission timers (RFC 2988), and support for explicit congestion

notification (RFC 3168). Because of the large number of updates and related documents, a guide to TCP's many RFCs was created in the form of yet another RFC—RFC 4614.

Every computer that supports TCP includes a TCP transport entity. This component might exist as a library function, a user-space process, or more commonly, a part of the operating system's kernel. The TCP entity is responsible for managing TCP connections, interfacing with the IP layer, and providing reliable delivery of data to applications. When a user application passes data to TCP, the transport entity splits this data stream into manageable segments—typically around 1460 bytes to fit within a single Ethernet frame—and then encapsulates each segment in a TCP header before handing it over to the IP layer for delivery as an individual IP datagram.

Upon receiving incoming datagrams from the network, the TCP entity at the destination machine reassembles them into the original byte stream in the correct order, even if some packets arrive out of sequence. This is necessary because the underlying IP layer offers no guarantees of delivery, order, or duplication control. IP does not manage congestion either, and it has no built-in mechanism to notify higher layers when datagrams are lost. All of this responsibility falls to TCP, which implements features like timeouts, retransmissions, sequencing, acknowledgement, and congestion control.

As a result, TCP delivers the reliable, ordered, and congestion-aware service that is required by most networked applications, compensating for the deficiencies of IP. It is this reliability and adaptive nature that has made TCP the backbone of many crucial Internet services such as web browsing, email, and file transfers.

18.6 THE TCP SERVICE MODEL

To make use of TCP services, both the sender and receiver must create endpoints known as sockets. A socket is defined by a combination of an IP address and a 16-bit port number that is unique to the host. In essence, this socket acts as the address at which a process can send or receive data. The communication between two machines using TCP requires an explicit connection to be established between a socket on the source machine and a socket on the destination machine. These sockets act as the communication interface, and the connection is uniquely identified by the combination of both sockets' identifiers—specifically, the pair (socket1, socket2). No additional identifiers such as virtual circuit numbers are needed to distinguish connections.

Interestingly, a single socket can handle multiple connections simultaneously. This is possible because TCP distinguishes connections not just by the local socket, but by the entire socket pair—meaning it includes both local and remote IP addresses and ports. This ability allows servers to manage numerous concurrent sessions with different clients, even if all the sessions connect to the same port on the server.

Port	Protocol	Use
20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

Figure 18.7 Some assigned ports

TCP uses port numbers to identify different services on a host. Ports numbered below 1024 are known as well-known ports and are reserved for standard services such as web browsing, email, file transfer, and remote login. These services include ports like 20 and 21 for FTP, 22 for SSH, 25 for SMTP, 80 for HTTP, and 443 for HTTPS. Since these ports are typically associated with critical services, they are usually only accessible by privileged users (such as the root user in UNIX systems). The Internet Assigned Numbers Authority (IANA) maintains the registry of these well-known ports, and hundreds have been officially designated.

For services or applications developed by non-privileged users, ports in the range of 1024 to 49151 can be used. Some applications may choose their ports independently of IANA registration; for instance, BitTorrent commonly uses ports 6881–6887, though it may operate on others as well.

Instead of having every service-specific daemon running and listening on its designated port at all times—which would consume unnecessary memory and CPU resources—UNIX systems often employ a centralized service called inetd (Internet daemon). This daemon listens on multiple ports simultaneously. When a connection request is received on one of the ports, inetd forks a new process and launches the appropriate daemon (such as the FTP or SSH daemon) to handle the request. This method keeps the system resource-efficient by activating specific daemons only when necessary. The configuration of inetd is typically controlled via a configuration file where system administrators define which ports inetd should monitor.

TCP connections are full duplex, meaning that data can flow in both directions simultaneously. Furthermore, TCP connections are point-to-point, involving exactly two endpoints. Unlike UDP, TCP does not support multicasting or broadcasting, so it is limited to unicast communication, which fits its purpose of reliable, end-to-end, connection-oriented data transfer.

A notable characteristic of TCP is that it operates as a byte stream rather than a message stream. This means that when an application writes data into a TCP connection, the boundary of each message or write operation is not preserved by TCP. For example, if a sender writes four separate chunks of 512 bytes each, TCP might deliver them to the receiver as four 512-byte chunks, two 1024-byte chunks, or even one 2048-byte chunk. The receiving process cannot detect how the data was originally divided by the sender. The data is treated as a

continuous sequence of bytes, much like reading from a UNIX file, where there's no inherent way to know how the data was originally written.

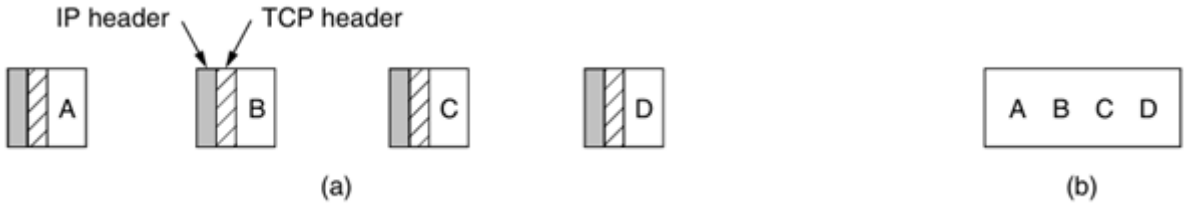


Figure 18.8 (a) Four 512-byte segments sent as separate IP datagrams. (b) The 2048 byts of data delivered to the application in a single READ call.

TCP has no understanding of the content of the byte stream—it doesn't care whether the bytes represent text, numbers, or anything else. Its only concern is to ensure the reliable, ordered delivery of the byte stream from the sender to the receiver. This abstraction simplifies the protocol and provides flexibility, but also shifts the responsibility of message parsing to the application layer.

When data is passed from the application to TCP, TCP has discretion over when to actually send the data. It might send it immediately or buffer it to accumulate a larger segment, aiming for efficiency in network usage. However, there are cases where applications require data to be sent instantly—such as real-time games or interactive applications—where delays can disrupt the user experience. For such needs, TCP introduced the PUSH flag, which was originally designed to prompt TCP to send data without delay. In practice, applications cannot directly set this flag. Instead, modern operating systems offer options like TCP_NODELAY to disable TCP's Nagle algorithm and force immediate transmission of small packets.

TCP also includes a lesser-known and rarely used feature called urgent data. This mechanism allows an application to mark certain bytes as urgent (such as a CTRL-C key press to interrupt a remote operation) using the URGENT flag. When urgent data is sent, TCP halts buffering and transmits all accumulated data immediately. Upon arrival, the receiving side is notified—typically via an interrupt or signal—so it can handle the urgent data promptly. The end of the urgent data is marked in the stream, but the beginning is not, requiring the application to locate it on its own.

Despite its design, urgent data never gained widespread use due to inconsistencies across implementations and the lack of strong use cases. As a result, its usage is now generally discouraged. Developers are instead encouraged to implement application-level signaling when urgent or control information needs to be transmitted out-of-band. Future transport protocols may offer more robust and consistent mechanisms for signaling and control.

18.7 THE TCP PROTOCOL

TCP is fundamentally designed around the concept of reliable data delivery, and at the heart of this reliability lies the use of 32-bit sequence numbers assigned to each byte transmitted over a TCP connection. This approach ensures that every byte can be tracked individually as it moves through the network. Originally, when the Internet's backbone links operated at just 56 kbps, it would take more than a week for a sender to exhaust the sequence number space at full speed. However, with the massive increase in data rates on modern networks, sequence

numbers can now be consumed extremely rapidly, making their management even more critical.

Data is transmitted in segments, with each TCP segment consisting of a 20-byte header and an optional data field that may include zero or more bytes of payload. The size of these segments is determined by the TCP software, which may choose to collect several writes from the application into one segment or split a single large write across multiple segments. However, segment sizes are subject to two major constraints: they must fit within the 65,515-byte IP payload limit and within the MTU of the network path. The MTU—Maximum Transmission Unit—is typically 1500 bytes on Ethernet networks, and this effectively limits how large a TCP segment can be without risking fragmentation.

Fragmentation is highly undesirable in TCP, as it degrades performance and introduces complexity. To avoid it, modern TCP implementations utilize path MTU discovery, a technique that relies on ICMP error messages to identify the smallest MTU along the transmission path. Once this value is known, TCP adjusts its segment size accordingly, thereby ensuring that all segments fit through the smallest link on the route without being broken apart.

The underlying mechanism TCP uses for flow control and reliability is a sliding window protocol with a variable window size. When a segment is sent, the sender sets a timer and waits for an acknowledgement from the receiver. This acknowledgement includes the sequence number of the next expected byte and information about the remaining window size. If the sender does not receive an acknowledgement before the timer expires, it retransmits the segment.

Managing TCP transmissions becomes complex because segments can arrive out of order or be delayed in transit. For example, a segment carrying bytes 3072–4095 might arrive before the segment carrying bytes 2048–3071. In such cases, TCP cannot acknowledge the received data until the missing earlier bytes arrive. Delays can also result in unnecessary retransmissions, possibly with slightly different byte ranges. The uniqueness of each byte's sequence number allows TCP to handle these situations gracefully, but it requires meticulous bookkeeping to ensure data integrity and to avoid duplicate processing.

Considerable engineering effort has gone into making TCP robust and high-performing, even under challenging network conditions. Numerous refinements and algorithms have been incorporated into modern TCP implementations to address these real-world issues, many of which are built upon the basic principles discussed here.

18.8 THE TCP SEGMENT HEADER

Every TCP segment begins with a fixed-format 20-byte header that contains crucial information necessary for reliable communication between sender and receiver. Following this fixed header, additional header options may be included, and then the actual data, if any, is appended. The total size of a TCP segment is limited to a maximum of 65,535 bytes, out of which 20 bytes are reserved for the IP header and another 20 for the fixed TCP header, leaving up to 65,495 bytes for the payload.

At the beginning of the header are two 16-bit fields identifying the source and destination ports. These ports represent the endpoints of the connection, where a port number combined with the host's IP address uniquely defines a connection endpoint. Together, the source and destination IP addresses and ports, along with the protocol identifier (TCP), form a 5-tuple that uniquely identifies a TCP connection.

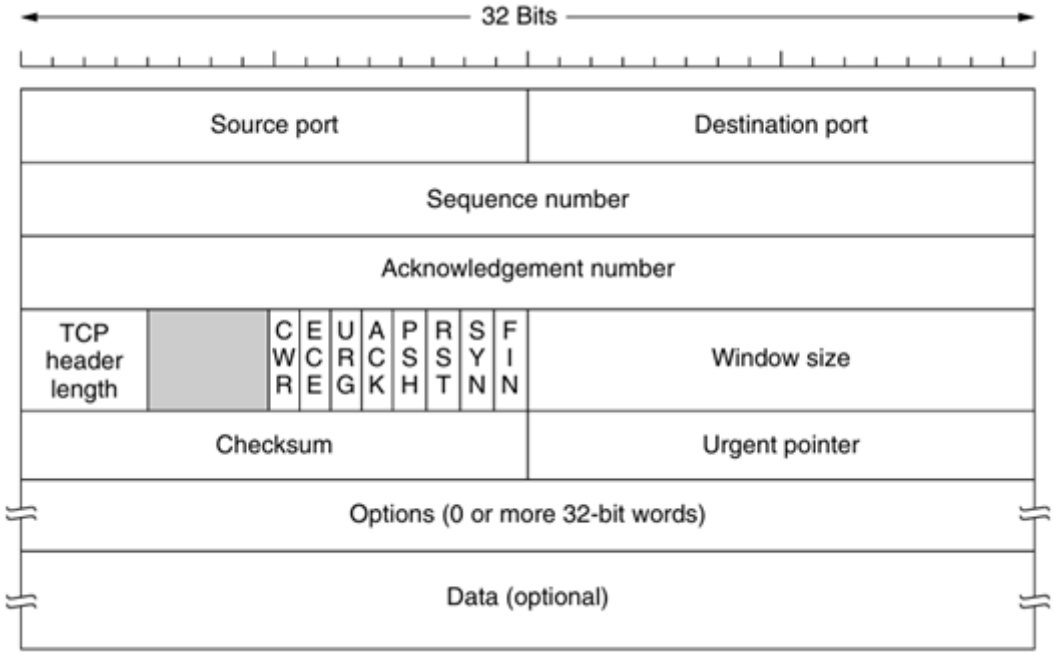


Figure 18.9 The TCP header

The next two fields are the 32-bit sequence number and acknowledgement number. The sequence number represents the byte number of the first byte in the segment's data payload. Every byte transmitted over a TCP connection is numbered individually, which enables precise tracking and reliable data transfer. The acknowledgement number indicates the next expected byte from the sender, thus serving as a cumulative acknowledgement. If this field is set, it confirms the successful receipt of all prior bytes in sequence.

The header length field, expressed in 32-bit words, tells where the actual data starts within the segment. This is necessary because the options field is of variable length, so the overall TCP header size is not fixed beyond the initial 20 bytes. There are four reserved bits immediately following the header length field, which have largely remained unused for over three decades, showcasing the robustness and foresight in the original TCP design.

Following the reserved bits are eight 1-bit flags that control various aspects of TCP operation. These include the CWR (Congestion Window Reduced) and ECE (ECN Echo) flags, which are used in Explicit Congestion Notification. The ECE flag informs the sender that congestion has been detected, prompting it to reduce its sending rate, while the CWR flag tells the receiver that the sender has done so. The URG flag indicates that the urgent pointer field is being used to mark urgent data within the segment. However, this mechanism is rarely used in practice due to implementation differences.

The ACK flag, which is set in most TCP segments, signals that the acknowledgement number field is valid. The PSH (push) flag requests that the receiver immediately pass the data to the application without waiting for the buffer to fill up. The RST (reset) flag is used to abruptly

terminate a connection, often in response to errors or invalid segment receipt. SYN is used to initiate a connection, typically sent with ACK set to 0 in the initial request and then with ACK set to 1 in the corresponding reply. Lastly, the FIN flag indicates that the sender has finished sending data and wishes to gracefully close the connection.

TCP implements flow control through a variable-size sliding window mechanism. The window size field in the header specifies how many bytes beyond the acknowledged byte the sender is allowed to transmit. A window size of zero is valid and tells the sender to stop sending data until further notice, providing a mechanism for the receiver to manage its buffer. This decoupling of acknowledgement and flow control, unlike in earlier protocols with fixed window sizes, allows for more flexibility and efficiency in managing data streams.

The checksum field provides error detection by verifying the integrity of the TCP segment. It includes not just the header and data but also a pseudoheader derived from IP fields like source and destination addresses. This ensures that accidental corruption during transmission can be detected.

The urgent pointer field, only relevant when the URG flag is set, indicates the byte offset of the last urgent byte in the data stream. While originally intended to support interrupt-style communication, its use has declined due to its limited utility and inconsistent implementations.

Finally, the options field offers a way to extend TCP functionality. Many useful features, such as Maximum Segment Size (MSS), window scaling, and timestamping, are implemented using this field. These options vary in length and follow a Type-Length-Value format. Together, these header fields make TCP a highly reliable, flexible, and efficient protocol for managing end-to-end communication in an internetworked environment.

A number of TCP options have been introduced to enhance performance and efficiency, especially on modern high-speed networks. One commonly used option is the Maximum Segment Size (MSS) option. This allows each host to specify the largest segment of data (excluding the header) it can accept. Larger segments are more efficient because they reduce header overhead, but smaller or resource-limited devices might require smaller segments. If MSS is not explicitly specified during connection setup, it defaults to 536 bytes of data. However, all TCP/IP hosts must accept at least 556-byte segments (including the TCP header).

Another useful option is window scaling, which addresses the limitation of TCP’s 16-bit window size field. A 16-bit field can represent up to 65,535 bytes, which is too small for high-bandwidth or long-delay networks. For example, on a fast optical link with a large round-trip time, the sender would spend most of its time waiting for acknowledgements unless the window size is increased. The window scale option allows the window size to be expanded up to 2302^{30} bytes by shifting the window value left by up to 14 bits. This negotiation happens during the connection setup phase.

The timestamp option adds a timestamp to each segment and echoes it back in acknowledgements. This helps measure the round-trip time more accurately and detect lost packets. It also assists with sequence number wrapping in high-speed environments, using a method called PAWS (Protection Against Wrapped Sequence numbers) to reject old, potentially duplicated packets.

Finally, Selective Acknowledgement (SACK) improves TCP's reliability and efficiency when packet loss occurs. Normally, TCP's acknowledgement only tells the sender which data was received in order. SACK goes further by allowing the receiver to inform the sender of exactly which blocks of data have arrived, even if they're out of order. This lets the sender retransmit only the missing parts, not everything after a lost packet, which saves bandwidth and improves performance. SACK is now widely implemented in modern TCP stacks.

18.9 TCP CONNECTION ESTABLISHMENT

TCP establishes connections using a process known as the three-way handshake. In this process, one machine (often the server) waits passively for incoming connections by invoking the LISTEN and ACCEPT primitives. It may be open to connections from any client or be configured to accept only from a specific address. On the other hand, the client actively initiates a connection using the CONNECT primitive. This primitive specifies the destination's IP address and port number, the maximum segment size the client can accept, and may include some optional data, such as a password or login credentials. As part of this process, the client sends a TCP segment with the SYN (synchronize) flag set and waits for a response.

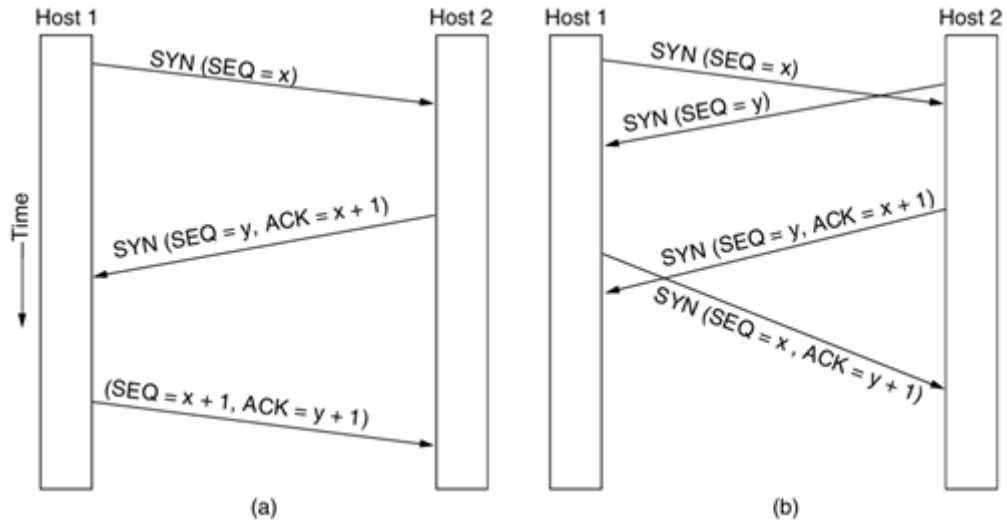


Figure 18.10 (a) TCP connection establishment in the normal case (b) Simultaneous connection establishment on both sides

When the SYN segment arrives at the server, the TCP entity checks if there is a process listening on the specified port. If no such listener exists, the server responds with a segment containing the RST (reset) flag to reject the connection attempt. If a listening process is found, the server may accept the connection, in which case it responds with a segment containing both the SYN and ACK (acknowledgement) flags set. This segment acknowledges the client's initial SYN and initiates its own SYN request to the client. The client then completes the handshake by sending a final segment with the ACK flag set, acknowledging the server's SYN. At this point, the connection is fully established. It's important to note that even the SYN segments use up one byte in the sequence number space to ensure proper sequencing and acknowledgement.

In rare scenarios, both hosts may try to initiate a connection with each other simultaneously using the same port numbers. In this case, each sends a SYN segment, and both respond with SYN-ACKs. The result is a single, shared connection, because TCP connections are uniquely

identified by a 5-tuple that includes the protocol (TCP), source IP and port, and destination IP and port.

The selection of the initial sequence number (ISN) is critical. Rather than using a fixed value like 0, TCP employs a mechanism where the ISN is derived from a clock that ticks every few microseconds. This approach helps prevent issues caused by delayed duplicate packets from old connections that could otherwise interfere with new ones.

However, this handshake mechanism can be exploited in what is known as a SYN flood attack. In such an attack, a malicious client sends many SYN segments to a server without completing the handshake. The server, trying to be helpful, allocates resources for each partial connection and waits for the final ACK, which never arrives. This can exhaust the server's resources, preventing it from handling legitimate connections. To mitigate this threat, TCP can use SYN cookies. With SYN cookies, the server encodes the initial sequence number using a cryptographic function that incorporates the client's IP, port number, and a secret key. The server does not store any state. When the client responds with an ACK, the server recalculates the expected sequence number using the same function to verify the response. If it matches, the connection proceeds. This technique allows the server to verify and establish connections without allocating memory until it is sure the client is legitimate, although it may not support all TCP options during such verification.

18.10 TCP CONNECTION RELEASE

Although TCP connections operate in full duplex mode—meaning that data can flow simultaneously in both directions—it is helpful to conceptualize them as two independent simplex connections (one for each direction) when examining how connections are terminated. Either side of the connection can initiate the closure by sending a TCP segment with the FIN (finish) flag set, which indicates that it has no more data to send. Once the receiving end acknowledges this FIN segment, that direction of the data flow is considered closed. However, the opposite direction remains open and data can continue to flow until it too is explicitly closed with a FIN.

Typically, closing a TCP connection involves four segments: the initiating side sends a FIN, the other side acknowledges it with an ACK, then sends its own FIN, which is subsequently acknowledged. This four-step process ensures a graceful shutdown where both parties confirm that all pending data has been transmitted and acknowledged. However, in some cases, the ACK of the first FIN and the second FIN itself can be combined into a single TCP segment, reducing the total number of packets exchanged during closure to three.

There are also scenarios where both sides send their FIN segments at the same time, similar to both parties in a phone conversation saying "goodbye" simultaneously. These FINs are independently acknowledged, and the connection is closed just as effectively. This simultaneous release is handled gracefully by TCP and is not fundamentally different from the sequential release.

To guard against issues like the two-army problem—where both sides wait indefinitely for final confirmation—TCP employs timers. If no response is received for a FIN within twice the maximum expected packet lifetime, the sender assumes the connection is no longer active and proceeds to release it. Eventually, the other side will also detect the silence, time out, and close its end. While not foolproof (since theoretical perfection is impossible), this pragmatic

solution works reliably in practice and rarely causes problems in real-world network communication.

18.11 TCP CONNECTION MANAGEMENT MODELLING

In TCP (Transmission Control Protocol), managing a connection between two hosts involves a precise sequence of events that ensure reliable and ordered communication. This entire process is represented by a finite state machine (FSM), where each TCP connection is associated with a particular state that evolves based on the events occurring at each end. The FSM consists of 11 distinct states, each representing a step in the connection's life cycle.
At the beginning, every TCP connection starts in the CLOSED state. This means that there is no active or pending connection. A connection can move out of this state in one of two ways, depending on whether the host is acting as a client or a server.

If the host is a server, it performs a passive open operation by executing the LISTEN system call. This transition changes the state from CLOSED to LISTEN. At this point, the server is waiting for incoming connection requests on a specific port.

State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIME WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

Figure 18.11 The states used in the TCP connection management finite state machine

On the other hand, a client wishing to initiate a connection performs an active open by calling CONNECT, specifying the destination's IP address and port number. This causes the client's TCP state to move from CLOSED to SYN SENT, and it sends a SYN (synchronize) segment to the server. This is step one of the three-way handshake, which is the standard mechanism TCP uses to establish a connection.

When the server receives this SYN segment, if it is listening on the requested port, it sends back a SYN + ACK segment (step two of the handshake) and transitions to the SYN RCVD state. The client, upon receiving the SYN + ACK, replies with an ACK (step three), completing the handshake. The client's state then transitions to ESTABLISHED, and so does the server's, after it receives the final ACK. At this point, the connection is fully established and data transfer can begin. This entire handshake process ensures both parties agree to the connection parameters, including sequence numbers and optional settings.

Once in the ESTABLISHED state, both ends can send and receive data freely. TCP ensures reliability through acknowledgements, retransmissions, and proper sequencing of data using the sequence number and acknowledgement fields in the TCP header.

Eventually, one of the hosts decides to terminate the connection. Let's say the client wants to close the connection. It does so by calling the CLOSE primitive. This sends a FIN (finish) segment to the server and transitions the client to the FIN WAIT 1 state. The FIN segment indicates that the sender has finished sending data but is still willing to receive data.

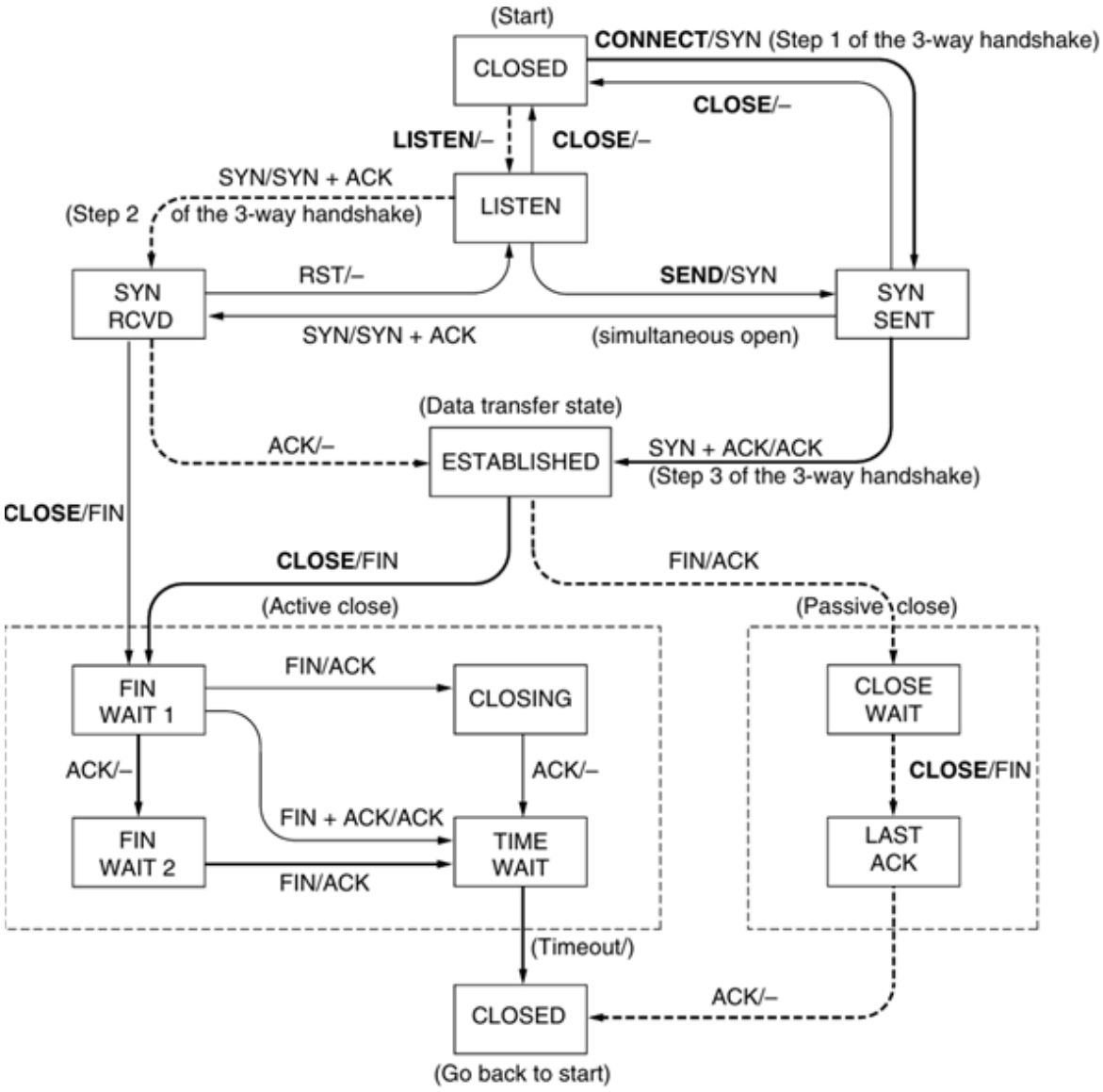


Figure 18.12 TCP connection management finite state machine. The heavy solid line is the normal path for a client. The heavy dashed line is the normal path for a server. The light lines are unusual events. Each transition is labelled with the event causing it and the action resulting from it, separated by a slash.

When the server receives this FIN, it sends back an ACK and moves to the CLOSE WAIT state. The client, having received this ACK, now moves to FIN WAIT 2, meaning it has finished sending and is waiting for the server to finish as well. The server, once it's ready,

calls its own CLOSE and sends a FIN back to the client. This causes the server to enter the LAST ACK state, where it waits for an acknowledgement of its FIN.

When the client receives this second FIN, it responds with an ACK and enters the TIME WAIT state. This is a crucial step: the TIME WAIT state ensures that any delayed or duplicate packets from the connection are allowed to expire in the network before the connection is considered truly closed. The client remains in this state for twice the maximum packet lifetime (2 MSL) before finally transitioning to the CLOSED state and deleting the connection record. The server, upon receiving the final ACK, also moves to CLOSED and clears the connection.

There are special cases TCP must handle. One such situation is simultaneous connection establishment, where both sides attempt to connect to each other at the same time using the same sockets. Both ends send SYNs simultaneously and receive SYNs from each other. TCP handles this by creating a single connection identified by the same 5-tuple (protocol, source IP, source port, destination IP, destination port). This ensures that no duplicate connections are created.

Another special case is simultaneous close, where both hosts attempt to close the connection at the same time. Both send FINs, which are acknowledged normally, and the connection is closed properly. TCP is designed to handle such symmetrical behavior robustly.

TCP also uses timers to handle scenarios where segments might be lost. For instance, if a FIN segment does not receive an ACK within a certain timeout period (twice the maximum packet lifetime), TCP assumes the other end has died or the segment is lost and proceeds to close the connection anyway. This use of timers addresses the two-army problem, where perfect reliability in connection termination is theoretically impossible.

To protect against denial-of-service attacks like the SYN flood—where attackers send large numbers of SYN segments without completing the handshake—TCP can employ SYN cookies. This mechanism allows the server to respond to a SYN without committing resources, by encoding state information in the sequence number and verifying it only if the handshake completes. While this makes it difficult to support TCP options (which are not remembered in this approach), it allows a server to continue functioning under attack.

In summary, the TCP finite state machine provides a precise and reliable mechanism to manage the entire life cycle of a connection—from establishment to data transfer and finally to safe termination. Each transition is governed by specific events and responses, with built-in protection against network failures, attacks, and errors. This model has made TCP one of the most robust and enduring protocols in networking.

18.12 TCP SLIDING WINDOW

TCP’s window management mechanism plays a crucial role in regulating the flow of data between a sender and receiver. It decouples two important concerns: confirming the correct receipt of segments through acknowledgements and managing the availability of the receiver's buffer. For instance, suppose a receiver has a 4096-byte buffer. If it receives a 2048-byte segment correctly, it will acknowledge that segment but advertise a window size of only 2048 bytes, which reflects the remaining space in the buffer. This ensures the sender does not overwhelm the receiver.

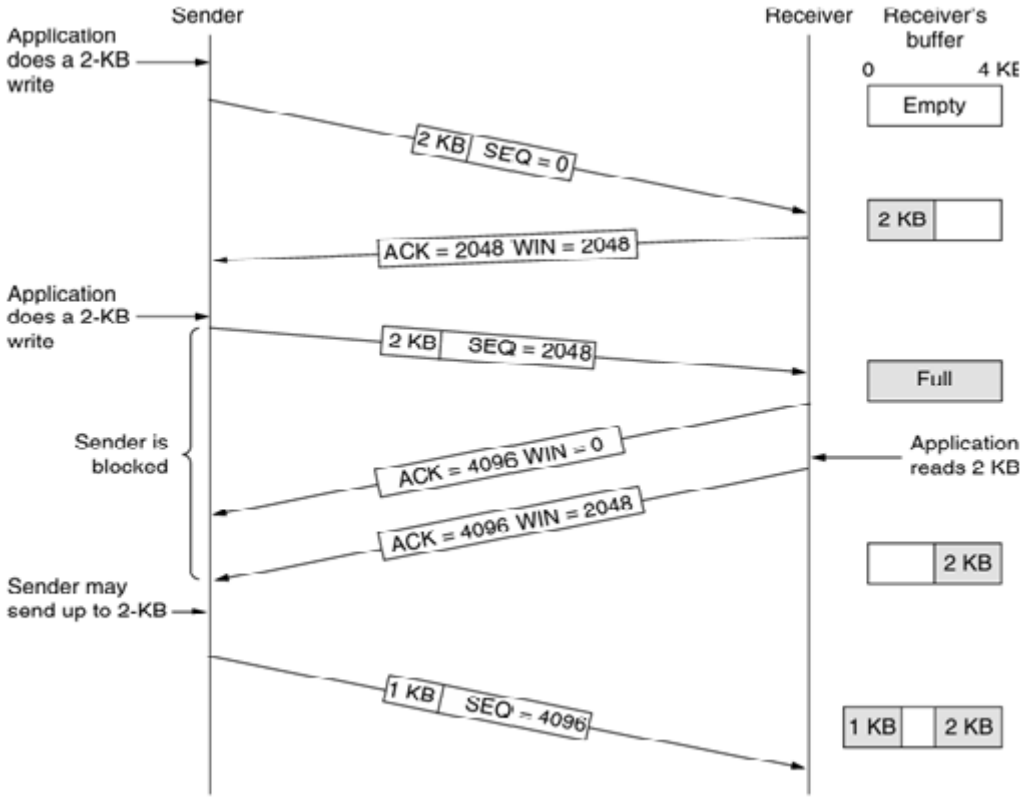


Figure 18.13 Window management in TCP

As data continues to be sent and acknowledged, the receiver updates the advertised window. Once the receiver's buffer becomes full—say, after another 2048-byte segment—the window size drops to zero, and the sender must pause. The sender cannot resume transmission until the receiver's application reads data from the buffer, creating free space. This pause is crucial to preventing data loss. To avoid deadlocks when a zero-window condition exists, TCP allows two exceptions: sending urgent data and sending a 1-byte window probe segment to force the receiver to resend the window size. This ensures communication continues even if a window update is lost.

To improve efficiency, TCP implementations are not obligated to send data or acknowledgements immediately. For example, a sender may choose to wait until enough data has accumulated to fill a large segment, rather than sending small ones. Similarly, a receiver might delay sending acknowledgements for up to 500 milliseconds in hopes of piggybacking them on a segment carrying data in the opposite direction. This optimization is known as delayed acknowledgement, and it helps reduce overhead, especially in interactive applications like SSH or Telnet where data is typically sent a byte at a time. Without optimization, typing a single character could trigger four separate packets totaling 162 bytes—a huge inefficiency for just 1 byte of useful data.

To further reduce overhead caused by many small packets, Nagle's algorithm is used. It ensures that when data comes in byte-by-byte, the sender transmits the first byte immediately and then buffers any subsequent bytes until the first one is acknowledged. This drastically cuts down the number of small segments and improves bandwidth utilization. However, Nagle's algorithm may introduce latency in real-time or interactive applications like online games, where rapid communication is essential. In such cases, Nagle's algorithm can be

turned off using the TCP_NODELAY option. Still, care must be taken because Nagle's algorithm can interact negatively with delayed acknowledgements, causing temporary deadlocks.

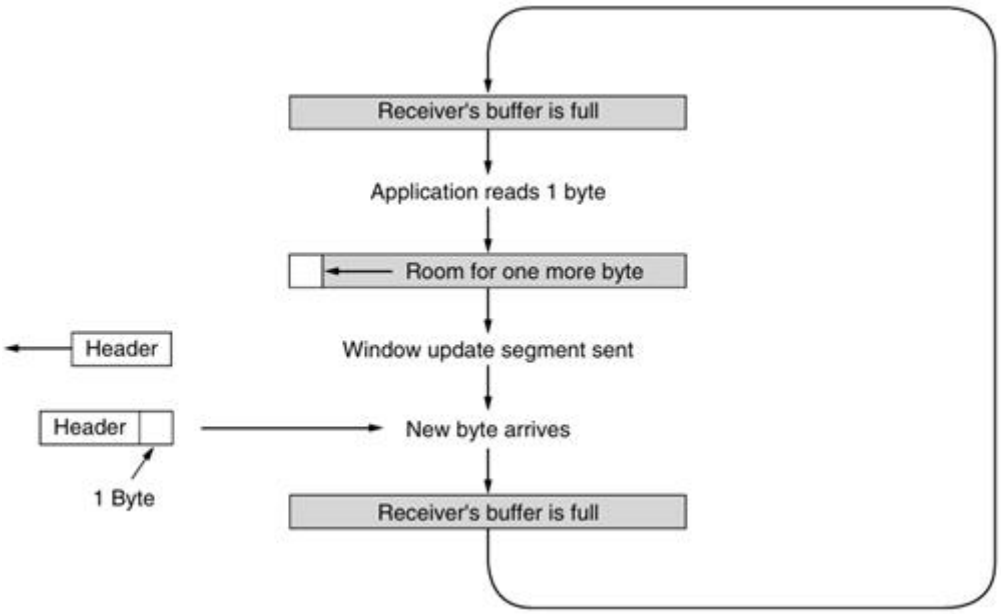


Figure 18.14 Silly window syndrome

A related issue is the silly window syndrome, where the receiver's application reads only a tiny amount of data at a time, say 1 byte. As soon as a byte is read from the full buffer, the receiver sends a window update indicating that it can accept just 1 byte, prompting the sender to transmit another single byte. This cycle can repeat indefinitely, causing an inefficient trickle of tiny segments. To mitigate this, Clark's solution suggests that receivers should not send window updates unless they can accept a sizable amount of data—typically half the buffer size or one maximum segment size. Senders, in turn, should refrain from sending data until they can transmit either a full segment or at least one that fills half of the receiver's buffer.

Together, Nagle's algorithm (for the sender) and Clark's solution (for the receiver) complement each other, tackling inefficiencies from both ends. The sender avoids flooding the network with small packets, while the receiver avoids requesting tiny segments unnecessarily. These measures ensure efficient use of bandwidth and reduce overhead, especially on networks with limited capacity.

Furthermore, the receiver may implement its own data buffering strategies. Instead of handing off data immediately to the application, it can accumulate a larger chunk before doing so. This reduces the number of READ system calls and processing overhead, especially in applications like file transfers, where throughput is more important than latency. For interactive applications, however, responsiveness is prioritized over efficiency.

Lastly, TCP must also deal with out-of-order segment arrival. If segments 0, 1, 2, 4, 5, 6, and 7 arrive, the receiver can only acknowledge data up to segment 2, because cumulative acknowledgements require all prior data to be present. Segments 4–7 are buffered but not acknowledged until segment 3 arrives. This approach prevents gaps in the data stream passed to the application and avoids unnecessary retransmissions while ensuring correctness.

Altogether, TCP's window management, acknowledgement strategies, congestion controls, and optimization techniques are designed to balance efficiency, reliability, and responsiveness. These features allow TCP to perform well across a wide range of network types and usage scenarios.

18.13 TCP TIMER MANAGEMENT

TCP employs a sophisticated system of timers to ensure reliable data transmission over unpredictable and variable networks like the Internet. The most critical of these is the Retransmission TimeOut (RTO) timer. When a segment is transmitted, TCP starts a timer to track how long it takes for an acknowledgment to return. If the acknowledgment arrives before the timer expires, the timer is stopped. However, if it doesn't arrive in time, TCP assumes the segment was lost and retransmits it, restarting the timer. The main challenge lies in deciding the appropriate timeout interval. If the timeout is too short, TCP might retransmit segments unnecessarily, leading to increased network congestion. On the other hand, if the timeout is too long, the connection's performance can degrade due to the delay in retransmissions.

This problem is more complex at the transport layer than at the data link layer, like in 802.11 protocols. At the data link layer, delays are generally short and predictable, so timers can be set just slightly above the expected round-trip time. In contrast, TCP experiences much higher and more variable round-trip times. Delays can change rapidly due to varying levels of congestion, making it difficult to determine a fixed timeout. Therefore, TCP adopts a dynamic algorithm to adapt the timeout interval based on ongoing network performance.

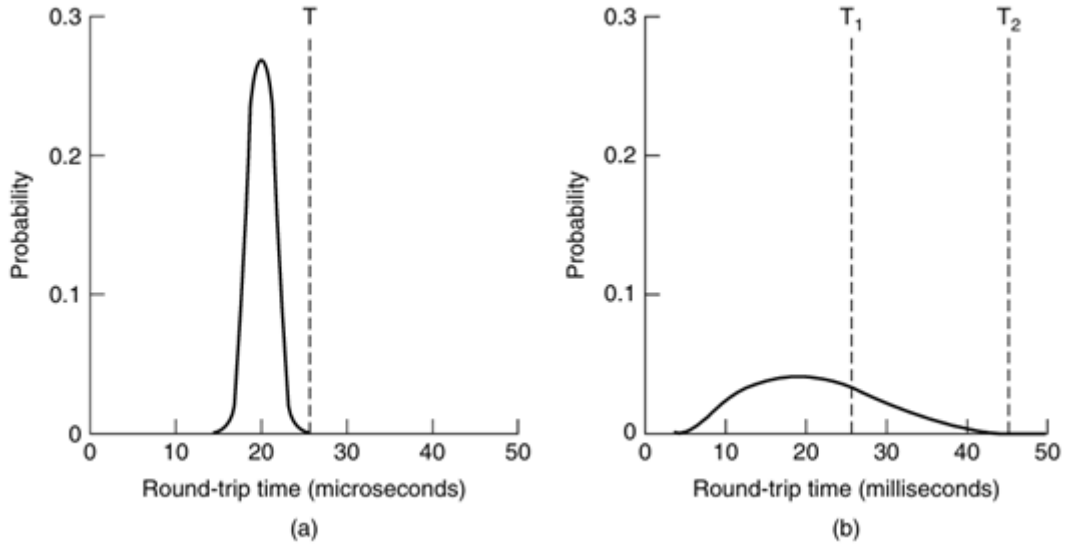


Figure 18.15 (a) Probability density of acknowledgment arrival times in the data link layer. (b) Probability density of acknowledgement arrival times for TCP

The algorithm TCP uses was developed by Jacobson and relies on measuring and updating the Smoothed Round-Trip Time (SRTT). Each time an acknowledgment is received, TCP measures the round-trip time (R) and updates SRTT using an exponentially weighted moving average (EWMA) formula:

$$SRTT = \alpha \times SRTT + (1 - \alpha) \times R,$$

where α is typically set to 7/8. This approach smooths out short-term fluctuations in delay, providing a stable average over time. But since the round-trip time may vary widely, Jacobson's method also tracks the Round-Trip Time Variation (RTTVAR) using another EWMA:

$$RTTVAR = \beta \times RTTVAR + (1 - \beta) \times |SRTT - R|,$$

where β is typically 3/4. TCP then sets the timeout value as:

$$RTO = SRTT + 4 \times RTTVAR.$$

This formula accounts not only for the average delay but also for its variability, reducing the risk of premature or delayed retransmissions.

One practical issue is distinguishing whether an acknowledgment corresponds to the original transmission or a retransmission. If TCP uses that acknowledgment to update SRTT incorrectly, it can corrupt future timeout estimates. To handle this, Karn's algorithm proposes ignoring round-trip samples from retransmitted segments. Instead, the timeout value is exponentially backed off (usually doubled) with each retransmission until an acknowledgment comes back from a non-retransmitted segment. This approach avoids contaminating the timing data and adjusts more cautiously in uncertain situations.

TCP uses several other timers for robust connection management. One of them is the persistence timer, which helps resolve deadlock situations when the receiver's advertised window size drops to zero. If the receiver later increases the window size but the window update is lost, both sender and receiver might wait indefinitely. To break this deadlock, the sender uses the persistence timer to periodically send a window probe to the receiver. If the receiver replies with a nonzero window size, the sender resumes data transmission; otherwise, the timer is reset to try again later.

Another useful mechanism is the keepalive timer, which checks whether an idle connection is still active. If a connection remains silent for too long, the timer triggers the sending of a keepalive probe. If the peer doesn't respond after a few retries, TCP assumes the connection is broken and terminates it. This feature helps identify half-open connections (e.g., due to crashed hosts or severed links), though it's considered controversial because it consumes extra resources and may terminate functioning connections during temporary network issues.
Finally, TCP employs a TIME-WAIT timer during connection termination. After both sides have agreed to close the connection, TCP keeps the connection state alive for twice the maximum segment lifetime (typically about 2 minutes). This precaution ensures that any delayed packets still circulating in the network won't accidentally interfere with new connections that might reuse the same port numbers and IP addresses. Only after this waiting period does TCP fully delete the connection record and release all associated resources.

Together, these timers make TCP robust and adaptive in handling retransmissions, preventing deadlocks, verifying connection health, and safely terminating sessions. The elegant blend of statistical estimation, adaptive timeout strategies, and defensive mechanisms makes TCP one of the most reliable transport protocols in networking history.

18.14 TCP CONGESTION CONTROL

TCP plays a crucial role in congestion control, ensuring that the Internet remains stable and efficient under varying load conditions. When a network is overwhelmed with more data than it can handle, congestion builds up, leading to increasing queue lengths at routers and ultimately packet loss. While the network layer detects and attempts to handle congestion by dropping packets, it is up to the transport layer—specifically TCP—to respond to these signals and reduce its transmission rate accordingly. Thus, TCP is not only responsible for providing reliable data transport but also for regulating the flow of data to prevent congestion.

The foundational principle behind TCP's congestion control is the AIMD (Additive Increase, Multiplicative Decrease) algorithm. This method ensures a fair and efficient distribution of bandwidth across multiple connections. In practice, TCP maintains a congestion window that dictates the number of bytes it can have in the network at any given moment. This window size, combined with the round-trip time, determines the effective data rate. TCP dynamically adjusts this window: it increases the window size incrementally to probe for available bandwidth and decreases it rapidly when signs of congestion, such as packet loss, are detected.

The congestion window operates alongside the flow control window, which reflects the receiving host's ability to accept more data. TCP transmits data based on the smaller of these two windows. For example, if the receiver can handle 64 KB but the network begins dropping packets after 32 KB, TCP limits itself to 32 KB. Conversely, if the sender can transmit 128 KB smoothly and the receiver allows 64 KB, it sends up to the permitted 64 KB. This dual-window mechanism ensures both receiver limitations and network conditions are respected.

Modern congestion control in TCP began with the work of Van Jacobson in the late 1980s. Following a period of congestion collapse caused by rapidly increasing Internet use, Jacobson proposed incorporating congestion avoidance mechanisms into TCP without altering its packet formats. He recognized that packet loss was a reliable indicator of congestion. While the signal is slightly delayed—arriving only after the network is already under strain—it is consistent, especially on wired links where transmission errors are rare. The assumption that packet loss signifies congestion underlies TCP's design.

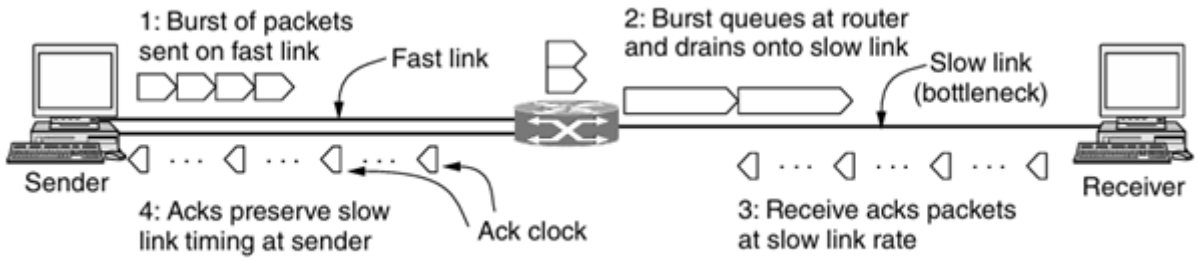


Figure 18.16 A burst of packets from a sender and the returning ack clock.

To act on congestion signals, TCP needs an accurate retransmission timer. This timer relies on precise measurements of round-trip times. Jacobson improved this process by incorporating both the average and variation in round-trip time measurements to calculate a reliable retransmission timeout. Once equipped with an accurate timeout, TCP could monitor

how many bytes were outstanding (sent but not yet acknowledged) and adjust its sending rate accordingly. This adjustment follows the AIMD pattern: TCP gradually increases its congestion window with each successful acknowledgement and reduces it sharply when a packet loss is detected.

Despite the simplicity of AIMD in principle, its implementation is complex. One challenge is the need to match the sender's burstiness with the characteristics of the entire network path. If a sender transmits a large burst over a high-speed link that feeds into a much slower link, such as a 1-Gbps Ethernet into a 1-Mbps DSL, the burst could overwhelm the slower link, causing delays or packet loss. To avoid this, TCP uses small packet bursts, which allow routers to queue and forward data more gracefully, thereby preventing congestion from escalating.

In such a setup, sending a burst of a few packets allows for controlled queuing at the bottleneck router without causing it to overflow. The slower link naturally stretches the packets in time due to its lower transmission rate, giving the network more breathing room. This practice maintains high throughput while minimizing the risk of congestion. Hence, TCP's congestion control adapts both to network feedback and to variations in network speed across different segments of the path. By combining these techniques—accurate timers, adaptive windowing, and intelligent pacing—TCP provides robust, scalable, and fair congestion control across the diverse and expansive environment of the Internet.

When TCP sender packets finally reach the receiver, they are acknowledged in the order and timing in which they arrived after traversing the slowest link in the network path. This arrival timing is reflected in the acknowledgements themselves. As these acknowledgements make their way back to the sender, they carry this time-spacing information, essentially reproducing the delay characteristics of the network path. The most critical insight from this process is that the acknowledgements return to the sender at approximately the rate allowed by the slowest link in the path. This returning rate acts as a natural pacing mechanism for the sender, guiding it on how quickly new packets can be injected into the network without causing congestion. This mechanism is known as "ack clocking," and it is a fundamental behavior of TCP. By aligning its transmission rate to the rate of returning acknowledgements, TCP manages to keep queues in the network shallow and avoids unnecessary packet loss due to congestion.

However, simply relying on additive increase behavior from a small congestion window can be slow to reach an optimal operating point, especially in fast networks. For instance, consider a connection with a 10 Mbps capacity and a round-trip time of 100 milliseconds. The ideal window size for such a connection—determined by the bandwidth-delay product—is 1 megabit, or roughly 100 packets of 1250 bytes each. If TCP started with a window of one packet and increased it by just one packet per round-trip time, it would take 100 round-trip times (or 10 seconds) to reach the optimal window size. This slow ramp-up time can significantly delay data transfers. Increasing the initial window to something large, like 50 packets, would reduce startup latency but would overwhelm slower links, causing immediate congestion. Thus, the solution Jacobson designed balances these needs using a hybrid strategy known as slow start.

In slow start, when a TCP connection begins, the sender initializes the congestion window to a small value—typically no more than four segments, as outlined in RFC 3390. The sender transmits this initial burst of packets and then waits for acknowledgements. For each

acknowledgement received, the congestion window is increased by one segment's worth of data. Since each acknowledgement both permits a new packet to be sent and triggers the window to grow, the number of packets in the network doubles every round-trip time. Though it is called "slow start," this algorithm actually increases the congestion window exponentially—far faster than linear growth—and allows the connection to quickly ramp up toward its optimal transmission rate without immediately overloading the network.

The timing of packet arrivals and the corresponding acknowledgements play a crucial role in regulating slow start. When the sender is connected to a high-speed network, it may transmit packets very closely spaced. But as those packets move through the network, especially across slower links, the spacing between them widens. For example, on a 100 Mbps Ethernet, each 1250-byte packet takes only 100 microseconds to send. But on a 1 Mbps ADSL line, it takes 10 milliseconds. This means that packets originally sent back-to-back will arrive at the receiver with much greater spacing. This widened spacing is preserved in the acknowledgements, and the sender uses this spacing as a guide—thanks to the ack clock—to inject new packets into the network at a sustainable rate.

Despite its benefits, slow start's exponential growth cannot continue indefinitely. Eventually, the congestion window will become too large, causing more packets to be sent than the network can handle. When this happens, queues will fill up and packets will be dropped. If the TCP sender fails to receive an acknowledgement for a packet—indicating that it was lost—it will trigger a retransmission timeout. This packet loss is a clear signal that the congestion window has grown beyond the path's capacity. In the example given, after a few round-trip times, the congestion window may reach eight packets, even though the network path might only be able to handle four packets per round-trip time without queuing. The overflow of packets builds up in router queues and causes delays or loss, thereby triggering TCP's congestion response.

To prevent such uncontrolled growth, TCP implements a safeguard in the form of a threshold called the slow start threshold. Initially set to a large value (typically the flow control window size), this threshold limits how much the congestion window is allowed to grow during slow start. Once the congestion window exceeds this threshold, TCP switches to a different phase called congestion avoidance, where the window grows linearly rather than exponentially.

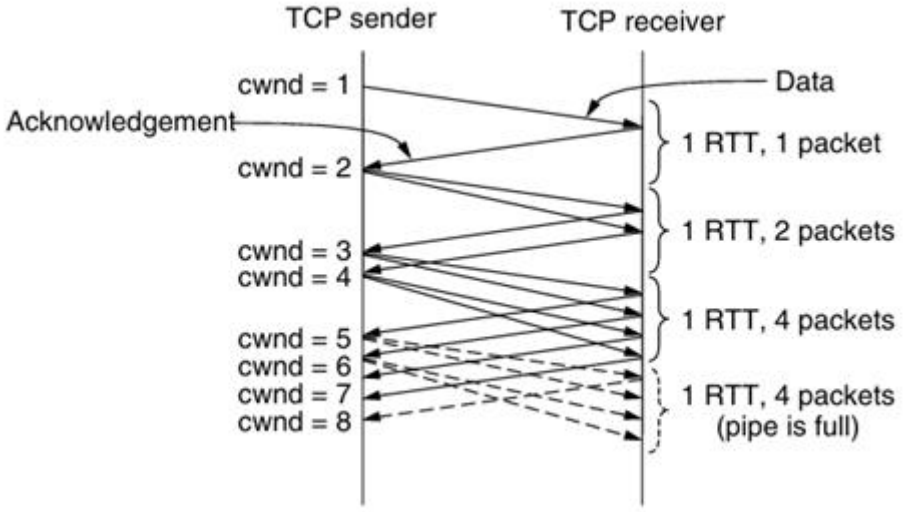


Figure 18.17 Slow start from an initial congestion window of one segment.

If a timeout occurs—indicating packet loss due to congestion—the slow start threshold is halved, and the congestion window is reset to its initial value. This adaptive response ensures that TCP probes for the network's capacity, backs off when congestion is detected, and grows cautiously thereafter. The threshold ensures the window doesn't expand too aggressively, balancing the need for rapid ramp-up with the necessity of avoiding congestion collapse. Thus, slow start and the threshold mechanism work together to allow TCP to efficiently utilize available bandwidth while reacting sensitively to changing network conditions.

Once the slow start threshold is exceeded, TCP shifts from exponential growth to linear growth, known as additive increase. In this phase, the congestion window grows more slowly—by roughly one segment per round-trip time—based on how many packets are acknowledged. Rather than waiting an entire RTT to make this adjustment, TCP incrementally increases the window by a small amount for each acknowledgement it receives. A common method is to increment the window by (MSS × MSS)/cwnd for each segment acknowledged, which results in a linear overall increase. This ensures that TCP remains close to the optimal window size: large enough to maintain high throughput, but not so large as to risk congestion. Compared to the rapid pace of slow start, additive increase proceeds cautiously and is much slower to reach large window sizes, which is ideal once the network is operating near capacity.

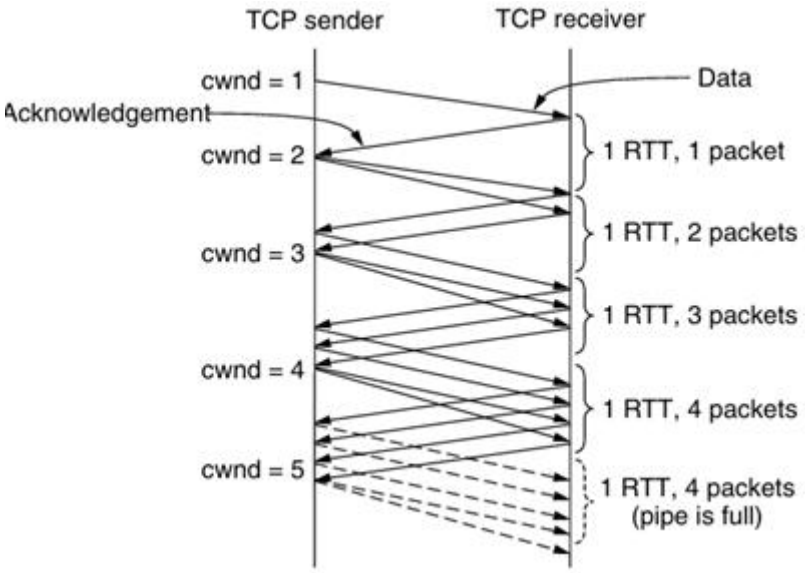


Figure 18.18 Additive increase form an initial congestion window of one segment

Despite this careful behavior, waiting for a timeout to detect packet loss can significantly impact performance. Retransmission timeouts are necessarily conservative and can cause TCP to pause transmission for an extended period. This is problematic because, during this pause, the sender is essentially idle while waiting for the timer to expire. An improvement over relying solely on timeouts is the use of duplicate acknowledgements. When a packet is lost but others beyond it continue to arrive at the receiver, the receiver sends back multiple acknowledgements for the same last correctly received packet. These are known as duplicate acknowledgements. TCP uses a heuristic: upon receiving three duplicate acknowledgements for the same data, it infers that a packet has likely been lost and retransmits it immediately—before the timer expires. This method, known as fast retransmission, allows TCP to recover more quickly from a single lost packet.

After fast retransmission, TCP adjusts its congestion control parameters. The slow start threshold is reduced to half the current congestion window, and the congestion window itself is reset to a small value. Traditionally, this would cause TCP to re-enter slow start, increasing its congestion window exponentially again. This behavior forms the basis of TCP Tahoe, named after the 4.2BSD Tahoe release where it was introduced. As shown in Tahoe's operation, whenever packet loss occurs and is detected via timeout or duplicate acknowledgements, the window is reduced dramatically and starts over. The window then increases rapidly up to the threshold, after which it grows linearly. This cycle repeats, forming a kind of sawtooth pattern in window size over time, with alternating exponential growth, packet loss, and recovery.

Although Tahoe was a significant improvement, Jacobson recognized it could be made even more efficient. He introduced a method called fast recovery to avoid restarting from a tiny window after every packet loss. During fast recovery, when duplicate acknowledgements are received, they are interpreted as signs that packets are still flowing—just not in the correct order. TCP can use this stream of duplicates to keep the ack clock running and maintain a steady flow of data. Instead of reducing the congestion window all the way to one, fast recovery sets it to the new threshold (half of the previous congestion window) and continues sending new packets for each duplicate acknowledgement received. This keeps the pipeline of packets moving, shortens the pause in transmission, and avoids unnecessary slow starts.

This enhanced version of TCP is called TCP Reno, named after the 4.3BSD Reno release that introduced it. Reno incorporates fast retransmission and fast recovery on top of Tahoe's base. The result is a more efficient and responsive congestion control mechanism. Reno's behavior can be visualized as a smooth sawtooth pattern: the congestion window increases linearly over time, until a loss is detected via duplicate acknowledgements. At that point, Reno performs a fast retransmission, enters fast recovery, and then resumes from the new threshold, avoiding slow start unless a timeout occurs. Thus, Reno spends most of its time near the optimal congestion window, making efficient use of available bandwidth.

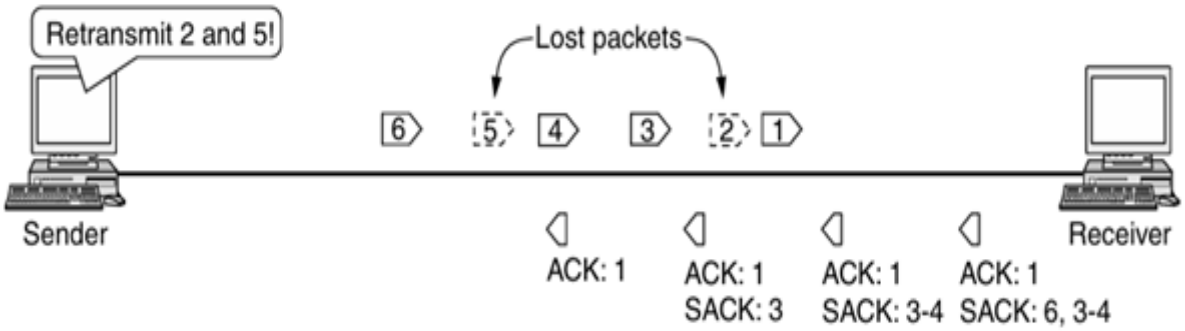


Figure 18.19 Selective acknowledgements

Over the years, TCP Reno has been extended and refined. One important enhancement is Selective Acknowledgements (SACK), which improves TCP's ability to recover from multiple losses in a single window. Standard cumulative acknowledgements do not tell the sender which specific packets were lost, only the last one that arrived in sequence. With SACK, the receiver includes additional information about which out-of-order packets it has received. This allows the sender to pinpoint exactly which packets were lost and retransmit them selectively. To use SACK, both sender and receiver must agree during connection setup. Once enabled, SACK options are included in acknowledgements, listing up to three

byte ranges of received data. These byte ranges help the sender identify which packets to retransmit, improving recovery from complex loss scenarios. SACK is now widely supported and has become an essential part of modern TCP implementations.

Another major addition is Explicit Congestion Notification (ECN). Instead of waiting for packets to be dropped, ECN allows routers to signal impending congestion by marking packets. When the sender and receiver support ECN, they signal this capability during connection setup. Once enabled, ECN-capable packets are flagged in the IP header. When a router experiences congestion, it marks the packet instead of dropping it. The receiver, upon detecting the mark, informs the sender using the ECN Echo (ECE) flag. The sender then acknowledges this signal by setting the Congestion Window Reduced (CWR) flag and responds by reducing its congestion window, just as it would in response to packet loss. The advantage of ECN is that it allows for early congestion detection without losing packets, improving performance especially in networks where packet loss is expensive or undesirable.
Together, these mechanisms—slow start, additive increase, fast retransmission, fast recovery, SACK, and ECN—form the backbone of TCP's congestion control. They allow TCP to adapt dynamically to changing network conditions, avoid congestion collapse, and maintain high throughput while minimizing packet loss. Over decades, TCP's congestion control has evolved into a sophisticated system that balances performance, fairness, and stability across a vast, heterogeneous Internet.

18.15 THE FUTURE OF TCP

TCP has long been the foundational protocol for reliable transport on the Internet, powering countless applications ranging from web browsing to email and file transfers. Over the decades, it has been refined to adapt to changing network conditions and extended to maintain strong performance across various types of links and environments. Nonetheless, as the Internet continues to evolve, so do the demands placed on transport protocols like TCP. Developers have introduced various tweaks and versions of TCP to better handle specific challenges, particularly around congestion control and resilience to network anomalies or attacks. Yet, even with these advances, TCP is not without limitations, and it is increasingly clear that it cannot perfectly address all modern networking needs.

One area where TCP falls short is in the transport semantics it offers to applications. TCP provides a reliable byte stream, but this abstraction does not always align well with what applications require. Some applications are designed to work with discrete messages or records and would benefit from having their boundaries preserved, something that TCP does not inherently support. Others involve multiple parallel conversations, such as a web browser fetching different parts of a webpage, which are not efficiently handled by TCP's stream-oriented model. Additionally, applications that wish to exert finer control over the network paths their traffic takes cannot do so easily through TCP's standard socket interface. These limitations have led to the development of alternative transport protocols designed to offer different interfaces or behaviors. Notable examples include SCTP (Stream Control Transmission Protocol), which provides message-oriented delivery and multihoming, and SST (Structured Stream Transport), which addresses some of TCP's structural constraints. However, deploying new protocols faces strong resistance due to TCP's widespread adoption and proven reliability. There is an ongoing debate between those advocating for newer, more flexible transport layers and those who believe the current system works well enough to avoid risking destabilization.

The second major issue facing TCP is congestion control. Despite the sophisticated mechanisms TCP has developed—like slow start, additive increase, fast retransmission, and fast recovery—congestion control remains an active area of research and improvement. Traditional TCP congestion control relies on packet loss as the main indicator that congestion is occurring in the network. This approach works reasonably well under many conditions, but it begins to falter in high-speed networks. Studies have shown that for TCP to maintain high throughput on such networks, packet loss must be exceedingly rare. For instance, sustaining a 1 Gbps connection over a link with a 100 ms round-trip time and typical 1500-byte packets requires that fewer than one packet be lost every 10 minutes. This loss rate—around 2×10^{-8}—is so low that even minimal transmission errors on the network can cause TCP to mistakenly assume congestion is occurring, throttling the connection unnecessarily and preventing full bandwidth utilization.

This situation has led many in the networking community to reconsider how congestion should be signaled and managed. Alternative signals to packet loss are being explored, such as using changes in round-trip time (RTT) as an early indicator of congestion. As RTT increases, it typically means queues are forming in the network, which is a precursor to packet loss. This principle is employed by protocols like FAST TCP, which adapts its transmission rate based on RTT measurements rather than waiting for losses to occur. Other strategies also exist, including variants of TCP like CUBIC and Compound TCP, which attempt to address these scaling limitations through different congestion window growth functions. Ultimately, while TCP remains a robust and highly adaptable protocol, its limitations—especially in handling ultra-high-speed links and evolving application requirements—ensure that the pursuit of better transport solutions remains an ongoing and dynamic area of innovation.

18.6 SUMMARY

This chapter explains the Internet transport protocols, focusing on UDP and TCP. UDP provides a simple, connectionless service for applications like DNS and streaming, supporting Remote Procedure Call (RPC) and Real-Time Transport Protocol (RTP). TCP offers a reliable, connection-oriented service with mechanisms for connection establishment and release, segmentation, flow and congestion control, and timer management. Variants such as wireless TCP/UDP and transactional TCP address specific challenges in mobile and transaction-based communications, ensuring efficient and reliable data transport across the Internet

18.7 TECHNICAL TERMS

User Datagram Protocol, Transmission Control Protocol, Remote Procedure Call, Real-Time Transport Protocol

18.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain UDP and its applications in network communication.
2. Describe Remote Procedure Call (RPC) and the Real-Time Transport Protocol (RTP).
3. Explain the TCP service model and the structure of a TCP segment.
4. Describe TCP connection establishment, release, and modeling of TCP connections.
5. Discuss TCP transmission policies, congestion control, timer management, and variations like wireless TCP/UDP and transactional TCP.

Short Questions:

1. What is UDP and its main characteristics?
2. Define Remote Procedure Call (RPC).
3. What is the purpose of TCP?
4. Name two TCP connection management processes.
5. What is TCP congestion control?

18.9 FURTHER READINGS

1. Andrew S. Tanenbaum, "Computer Networks", Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, "Computer Networking", Third Edition, Pearson Education
3. Behrouz A Forouzan, "Data Communications and Networking", Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, "Computer Communications and NetworkingTechnologies", Cengage Learning (2008).

Mrs. Appikatla Pushpa Latha

LESSON- 19

DOMAIN NAME SYSTEM

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the purpose and structure of the Domain Name System (DNS).
- Learn about the DNS name space and hierarchical organization.
- Study DNS resource records and name servers.
- Understand the architecture and services of electronic mail.
- Learn about message formats, transfer, and final delivery in email systems.

STRUCTURE OF THE LESSON:

19.1 INTRODUCTION

In the early days of the Internet, although it was technically possible for programs to refer to web pages, email addresses, and other online resources by using numerical network identifiers such as IP addresses, this method posed significant usability challenges. IP addresses are inherently difficult for humans to remember, making them impractical for widespread use. Moreover, if a server changed its IP address—for example, due to a company migrating its website to a different machine—every user would need to be informed of the new address. To address these limitations, more intuitive, human-readable names were introduced to provide a level of abstraction between machine names and their actual IP addresses. This allowed people to access resources through easily memorable names like www.cs.washington.edu, regardless of the physical machine or IP address hosting the service.

Despite the convenience of human-readable names, the underlying network infrastructure operates solely on numerical IP addresses. Therefore, a system was required to bridge the gap between user-friendly names and machine-understandable IPs. Initially, this was accomplished

through a simple mechanism: a file called hosts.txt that listed all known computer names and their corresponding IP addresses. Every host on the network downloaded this file from a central location each night. While this solution was manageable for a small network comprising a few hundred machines, it was clearly not scalable to the millions of devices that would later join the Internet. As the network expanded, maintaining a centralized file became impractical, and issues such as name conflicts and update latency became increasingly problematic.

To overcome these challenges, the Domain Name System (DNS) was developed in 1983. DNS introduced a hierarchical, distributed naming architecture that could scale with the growing Internet. Rather than depending on a centralized text file, DNS employs a decentralized database to manage the mapping of names to IP addresses. This system not only enhances scalability and efficiency but also reduces the risk of name conflicts through coordinated domain registration. DNS operates using a domain-based naming structure in which each domain name corresponds to a specific path in the hierarchy, allowing for organized and systematic address resolution.

In practice, when an application needs to resolve a hostname into an IP address, it invokes a library function known as a resolver, such as the gethostbyname function seen in earlier discussions. This resolver constructs a DNS query with the target name and sends it to a local DNS server, typically provided by the user's internet service provider or institution. The server then searches its database—or queries other DNS servers if necessary—and returns the appropriate IP address. This result is then relayed back to the application by the resolver. The entire exchange typically takes place over the User Datagram Protocol (UDP), which is suitable for the quick, small messages used in DNS lookups. Once the application has obtained the IP address, it can proceed with its intended operation, whether that means opening a TCP connection or sending a UDP packet to the resolved address. This elegant and efficient architecture makes DNS an indispensable component of modern networking.

19.2 THE DNS NAMESPACE

Managing a vast and ever-evolving collection of names in a global network like the Internet is a complex task. Drawing a parallel with the postal system, which avoids confusion by using a hierarchical structure—such as country, city, and street—the Internet adopted a similar naming scheme through the Domain Name System (DNS). This system ensures uniqueness and clarity, so different individuals or entities can coexist with similar names but in distinct contexts, much like people with the same name living in different cities.

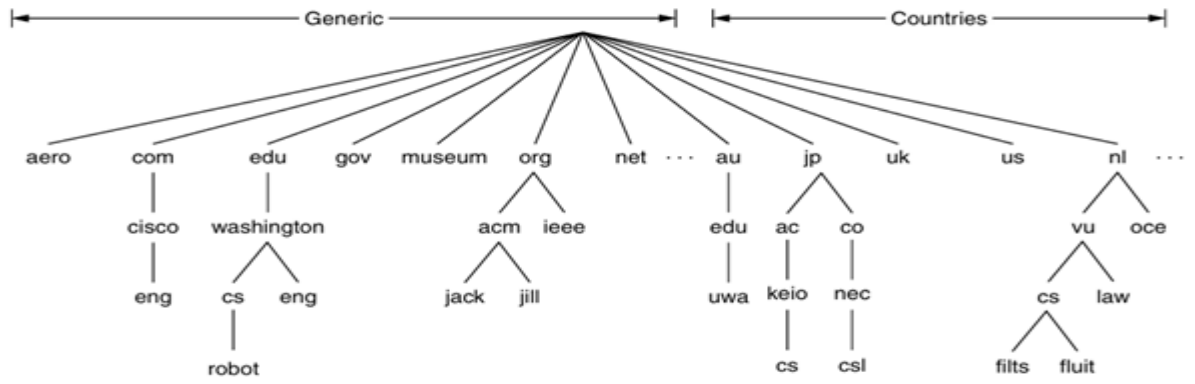


Figure 19.1 A portion of the Internet domain name space

At the highest level of DNS management is ICANN, the Internet Corporation for Assigned Names and Numbers, established in 1998 to oversee the global coordination of the DNS. The DNS organizes names into a tree structure composed of over 250 top-level domains, each of which can be further divided into subdomains. These top-level domains fall into two categories: generic domains like .com or .org, and country-specific domains like .us, .jp, or .nl. This structure allows the Internet to scale efficiently, supporting millions of hosts and users across the world. Each leaf in this tree represents a domain that contains individual machines or services.

Obtaining a second-level domain—such as example.com—is relatively straightforward. Registrars authorized by ICANN manage these top-level domains and can assign names to entities upon request, provided the desired name is available and does not infringe on existing trademarks. However, as the Internet became more commercial, naming disputes became increasingly common. The process of assigning names has sparked debates over who qualifies for restricted domains (e.g., .pro for professionals), and even geopolitical and ethical controversies have emerged—such as those surrounding the .xxx domain for adult content.

In addition to ethical and procedural issues, domain names have become valuable assets. Some entities, such as the country of Tuvalu, have profited immensely by leasing their country code top-level domain—.tv—which is ideal for television websites. This commercial value has led to practices like cybersquatting, where individuals register popular or trademarked names solely to sell them at a profit. Despite being legally permissible under certain conditions, these practices have necessitated ongoing policy development to manage conflicts and protect legitimate interests.

The hierarchical nature of DNS also allows for clear and conflict-free domain naming. Each domain is identified by tracing its path upward from its position in the DNS tree, with components separated by periods. For example, the domain eng.cisco.com refers to the engineering department of Cisco and is distinct from similarly named domains under other parent nodes, such as eng.washington.edu. Domains can be absolute, ending with a period, or relative, depending on the context in which they're used. Domain names are not case-sensitive, and while individual components can be up to 63 characters long, the entire name must not exceed 255 characters.

Domain	Intended use	Start date	Restricted?
com	Commercial	1985	No
edu	Educational institutions	1985	Yes
gov	Government	1985	Yes
int	International organizations	1988	Yes
mil	Military	1985	Yes
net	Network providers	1985	No
org	Non-profit organizations	1985	No
aero	Air transport	2001	Yes
biz	Businesses	2001	No
coop	Cooperatives	2001	Yes
info	Informational	2002	No
museum	Museums	2002	Yes
name	People	2002	No
pro	Professionals	2002	Yes
cat	Catalan	2005	Yes
jobs	Employment	2005	Yes
mobi	Mobile devices	2005	Yes
tel	Contact details	2005	Yes
travel	Travel industry	2005	Yes
xxx	Sex industry	2010	No

Figure 19.2 Generic top-level domains

Globally, domain names can reside under generic or country-specific domains. In practice, most U.S.-based organizations use generic domains like .edu or .com, whereas entities in other countries often fall under their national domains. Nevertheless, there's flexibility, and large organizations frequently register under multiple top-level domains—for instance, Sony may own sony.com, sony.net, and sony.nl.

Control over subdomains lies with the parent domain. If a new group within an organization—like a VLSI research group at a university—wants its own subdomain, it must obtain permission from the domain's manager. However, once a subdomain is created, it can independently manage its own namespace. This distributed control structure ensures both scalability and autonomy while preserving the integrity of the overall naming system.

Finally, the logical structure of domain names is based on organizational relationships, not physical network topology. This means that departments within the same building may belong to separate domains, while geographically distributed units of the same department may share a domain. This reflects the abstract nature of domain naming, which focuses on administrative and functional boundaries rather than physical infrastructure.

19.3 DOMAIN RESOURCE RECORDS

On the Internet, each domain name—whether it's a simple host like a single machine or a broad domain like .com—has a set of associated resource records. These records form the contents of the DNS (Domain Name System) database, which is the core of how name resolution happens on the Internet. When an application or user wants to access a resource by its domain name (like visiting www.google.com), the DNS helps resolve that name into something the network can understand, like an IP address. What DNS essentially does is return these resource records when given a domain name, allowing computers to map friendly names to actionable network data.

A resource record is made up of five key fields: the domain name, time to live (TTL), class, type, and value. While these are stored in a binary format for computers to process efficiently, they are typically displayed as human-readable ASCII text when viewed in configuration files or documentation. The domain name field indicates which name this record belongs to and serves as the main key for looking up the record in the DNS database. Each domain can have many such records—for instance, one record for the IP address, another for the mail server, and others for different services or aliases.

Type	Meaning	Value
SOA	Start of authority	Parameters for this zone
A	IPv4 address of a host	32-Bit integer
AAAA	IPv6 address of a host	128-Bit integer
MX	Mail exchange	Priority, domain willing to accept email
NS	Name server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
SPF	Sender policy framework	Text encoding of mail sending policy
SRV	Service	Host that provides it
TXT	Text	Descriptive ASCII text

Figure 19.3 The principal DNS resource record types

The time to live field defines how long this record can be kept in a DNS cache before it should be discarded or re-validated. This is critical for optimizing performance and reducing the load on DNS servers. If a record rarely changes—like the IP address of a large website's home server—it can have a high TTL value, such as 86400 seconds (which is one day). But for records that are subject to frequent updates, such as temporary services or development servers, a shorter TTL (like 60 seconds) is more appropriate to ensure clients always get the latest information.

The class field of the record specifies the protocol family to which the record belongs. In nearly all common DNS uses today, this class is simply "IN," which stands for "Internet." Other classes exist for non-Internet uses, such as "CH" for Chaosnet or "HS" for Hesiod, but they are largely obsolete or used in very niche environments. So, in practice, the class field almost always has the value IN.

The type field is one of the most important parts of the resource record. It specifies what kind of information is being stored. The most frequently encountered type is the A record, which maps a domain name to a 32-bit IPv4 address. For example, www.example.com might map to 93.184.216.34. For IPv6 addresses, there is the AAAA record ("quad-A"), which stores the 128-bit address. Both are essential because every host on the Internet needs at least one IP address to be reached.

One especially useful type of record is the MX (Mail Exchange) record, which identifies mail servers that handle emails for a domain. For example, emails sent to user@example.com are routed based on the MX records of example.com. These records not only indicate the hostnames of mail servers but also assign them priorities using numerical values. The mail-sending system tries the lowest-numbered priority server first; if it's unavailable, it tries the next, and so on. This redundancy ensures more reliable email delivery.

Another important record is the NS (Name Server) record, which identifies the authoritative name servers for a domain. These are the servers that hold the actual DNS records for the domain and respond to queries from other DNS servers. If you register a domain, your domain registrar will ask you for your NS records so the Internet knows where to look when resolving your domain name.

The CNAME (Canonical Name) record allows domains to act as aliases for others. This is helpful in many scenarios. For instance, www.example.com might just be a CNAME pointing to webhost123.example.com, meaning that whenever someone accesses www.example.com, the DNS system internally redirects them to the canonical domain. This enables organizations to change server details in the background without affecting public-facing URLs.

The PTR (Pointer) record is used in reverse DNS lookups, where the goal is to resolve an IP address back into a domain name. This is the reverse of what a standard DNS query does. PTR records are especially useful for verifying the source of incoming emails or for security logging. More modern types of DNS records have been introduced to accommodate advanced networking needs. The SRV (Service) record allows specification of a particular server that offers a certain service, such as SIP or LDAP. This is more flexible than older methods, as it can include both hostname and port information. The SPF (Sender Policy Framework) record is crucial in the fight against email spam and spoofing. It defines which servers are allowed to send email on behalf of a domain, helping receiving mail servers detect forged messages.

The TXT (Text) record was originally designed to let administrators include arbitrary text in DNS entries, such as contact information or server descriptions. However, in modern usage, TXT records are most often used to store structured data like SPF policies or domain ownership verification codes for services like Google Workspace or Microsoft 365.

To illustrate how these records appear in real DNS configurations, consider a simplified DNS database for the domain cs.vu.nl. This domain might have several entries in its DNS zone file. There would be an SOA (Start of Authority) record providing the administrative information and serial numbers used for synchronization among DNS servers. There might be two MX records specifying that email for cs.vu.nl should go first to the host zephyr and then to top if zephyr is unavailable. An NS record would list star as the authoritative name server.

Following this, there would be A records mapping the domain names star, zephyr, and top to their respective IP addresses. These ensure that other computers can contact them over the network. For user convenience, the domain could have CNAME records for www.cs.vu.nl and ftp.cs.vu.nl pointing to specific machines, so users don't have to remember complicated or changeable machine names.

A host like flits.cs.vu.nl might have two A records if it has two network interfaces (or belongs to two networks), and it could also have multiple MX records indicating mail delivery preferences. Other hosts like rowboat.cs.vu.nl would have a single A record for its IP address and perhaps MX records for mail routing. Even devices like printers, such as laserjet.cs.vu.nl, might have A records so users can print over the network.

Altogether, these records work in harmony to provide a flexible, scalable, and robust naming infrastructure for the Internet. DNS allows domain names to be assigned and resolved in a way that is understandable to humans, yet highly efficient for computers. Its design supports a massive, decentralized, and continually evolving Internet, while also ensuring backward compatibility and consistent global behavior.

19.4 NAME SERVERS

In an idealized world, a single name server could maintain the entire Domain Name System (DNS) database and respond to all queries. However, this setup would be highly impractical in reality. Such a central server would be overwhelmed with traffic, and any failure in it would cripple the Internet's naming infrastructure. To address this, DNS is designed to be distributed and hierarchical. The name space is divided into nonoverlapping zones, each managed by its own administrative authority. These zones correspond to subtrees of the DNS hierarchy and may cover as little as a single machine or as much as an entire top-level domain. The decision about how to divide these zones and where to place their boundaries is made by administrators and often depends on operational and administrative needs.

Each zone in the DNS is managed by one or more name servers. These servers hold the database for their respective zones. Typically, a zone has a primary name server, which holds the original data files, and one or more secondary servers that replicate the primary's data. These secondary servers fetch zone data from the primary through a process called zone transfer. To ensure high reliability and accessibility, some name servers may be located outside the zone they serve. When a domain name needs to be resolved, the query process begins by consulting a local name server, which may either have the answer or must look it up remotely.

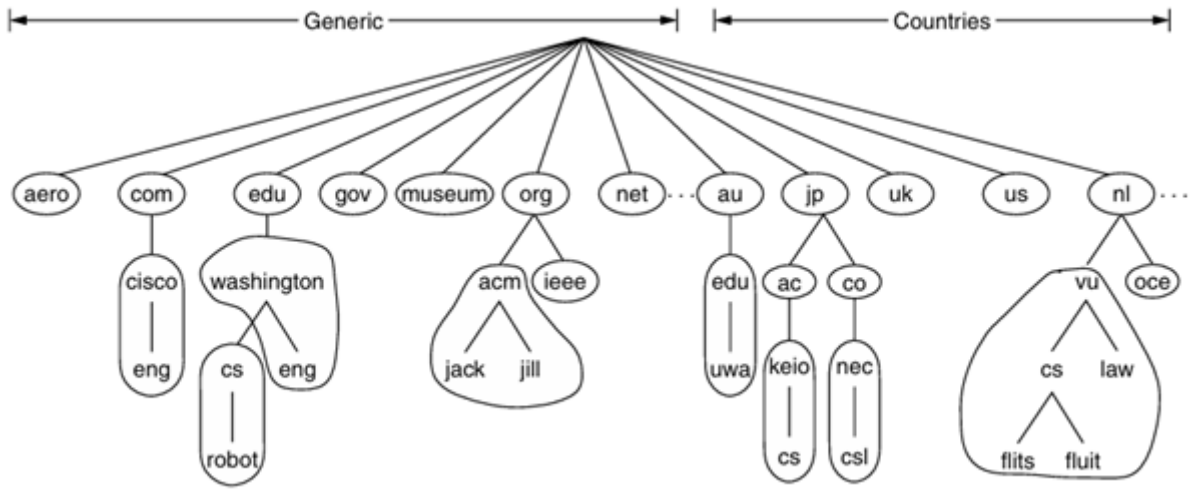


Figure 19.4 Part of the DNS name space divided into zones (which are circled)

When a name like robot.cs.washington.edu is queried from a host such as flits.cs.vu.nl, and no local cached information is available, the resolution must proceed through a chain of DNS queries. The process starts with a query from the originator to its local name server, including the domain name, query type (such as A for address), and class (IN for Internet). If the local name server cannot resolve the name directly, it begins by contacting a root name server. These root servers are crucial because they know the authoritative name servers for all top-level domains like .com, .edu, or country-specific domains like .nl or .uk.

Each name server holds a configuration file containing information about the root servers. These are not single machines but are actually replicated systems using anycast routing to direct queries to the nearest available instance. There are thirteen named root servers (a.root-servers.net through m.root-servers.net), and each is backed by dozens of physical instances around the world. This replication strategy enhances both performance and fault tolerance.

After contacting a root server, the local name server receives a referral to the name server for the .edu domain. The local server then contacts the .edu name server, which in turn refers it to the washington.edu name server, and then to cs.washington.edu, which finally provides the authoritative IP address for the host robot. This entire multi-step journey is necessary because the DNS system is a distributed and delegated system; no single server holds all the information. Each step returns part of the answer, gradually narrowing down the search until the exact host is located.

There are two key mechanisms used in these queries: recursive and iterative. When the original host sends a request to its local name server, it expects a complete answer in return. This is known as a recursive query. The local name server takes on the responsibility of continuing the resolution process through multiple iterative queries to other name servers, each of which returns only partial information—such as the next server to contact. Root and intermediate servers typically do not handle recursive queries due to the high load it would impose; they rely on the querying server to perform subsequent steps.

Caching plays a vital role in improving the performance and efficiency of DNS. Each time a name server receives a response—whether a full or partial one—it stores that result in a cache for later use. This reduces the number of external queries that must be made for repeated

requests, especially when multiple queries concern hosts in the same domain. If the information is already cached, future queries can be answered immediately. However, caching introduces the risk of outdated information being used. To manage this, each resource record includes a Time To Live (TTL) value, which specifies how long the information should be retained before being considered stale. Volatile data may have a TTL of just a few seconds or minutes, while stable data may be cached for hours or even days.

All DNS queries and responses are transmitted over the UDP protocol. This choice is made for performance reasons since UDP is connectionless and incurs less overhead than TCP. DNS messages are small enough to fit within the size limits of UDP packets. To maintain robustness, DNS clients are designed to retry queries using different servers if no response is received within a short time. Each query includes a unique 16-bit identifier that helps match responses to the correct outstanding query, especially when multiple queries are sent simultaneously.

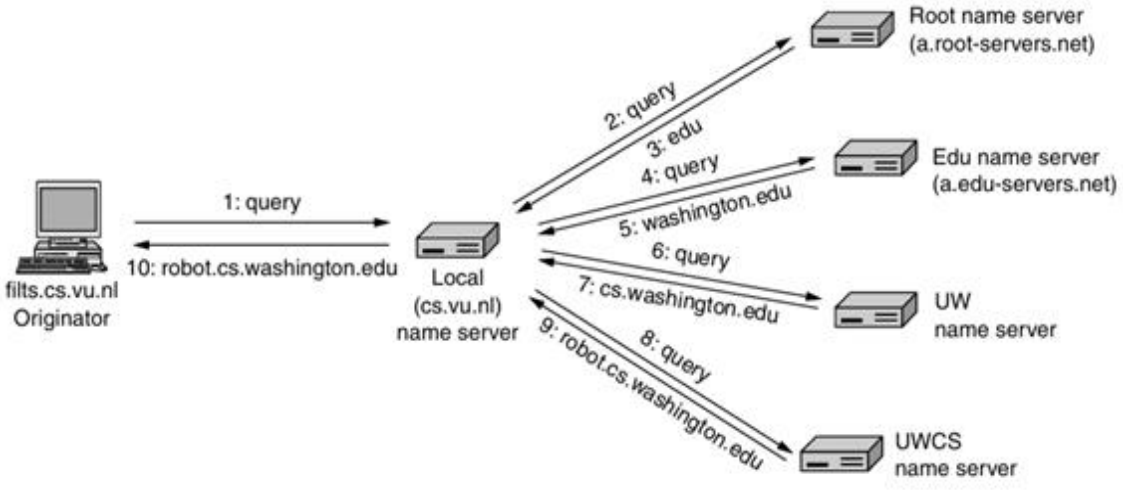


Figure 19.5. Example of a resolver looking up a remote name in 10 steps.

DNS, though conceptually simple, is a massive and intricate distributed system composed of millions of name servers globally. It serves as the critical bridge between user-friendly domain names and machine-readable IP addresses. Its hierarchical structure, combined with replication and caching, ensures that the system is scalable, fault-tolerant, and fast. However, because DNS is so central to Internet functionality, it also poses significant security challenges. A malicious actor who can alter DNS responses could redirect users to fraudulent sites. To address such threats, security extensions like DNSSEC have been developed, which add cryptographic signatures to DNS data to ensure its authenticity and integrity.

Additionally, modern Internet applications demand even more flexible name resolution mechanisms. For example, rather than locating a specific machine, users might want to find the nearest server that hosts a specific piece of content, like a movie. This approach shifts the focus from machine identity to content availability. Systems like Content Distribution Networks (CDNs) build upon DNS to provide such functionality, directing users to geographically optimal servers that host the desired content, thereby enhancing speed and reducing network load. DNS, therefore, continues to evolve to meet the growing and changing demands of global Internet users.

19.5 EMAIL

Electronic mail, or email, has been one of the most popular Internet applications for more than three decades. It is faster and cheaper than traditional paper mail and has been widely used since the early days of the Internet. Before 1990, email was mainly limited to academic use, but during the 1990s, it became a global communication tool. Today, the number of emails sent daily far exceeds that of regular mail. Despite the rise of other communication methods like instant messaging and voice-over-IP, email remains the backbone of Internet communication, especially for business and organizational use across the world. Unfortunately, a major issue with email is spam—unwanted junk messages—which make up nearly 90% of all emails. Email communication is informal and accessible to everyone, often breaking social barriers related to rank or position. It is rich in abbreviations (e.g., BTW, ROTFL, IMHO) and uses emoticons like ":-)" to express tone and emotion, helping users convey meaning beyond plain text.

Over time, email protocols and systems have evolved significantly. Early email systems were simple file transfer mechanisms where the recipient's address appeared on the first line of a message. Gradually, more advanced features were introduced, such as sending messages to multiple recipients and supporting multimedia attachments like images and documents. As email became mainstream in the 1990s, user interfaces improved from text-based to graphical formats, making email easier and more interactive. Modern email systems also enable users to access their mail from any location or device. With the growing problem of spam, both mail clients and mail transfer protocols now incorporate filtering and security mechanisms to detect and remove unwanted messages. In this context, understanding how mail messages move between users is more essential than focusing on the interface design of email programs.

19.5.1 Architecture and Services

Email systems consist of two main subsystems — User Agents (UAs) and Message Transfer Agents (MTAs) (Fig. 19.). User Agents allow users to compose, send, receive, and organize emails.

Message Transfer Agents move messages between mail servers using the Simple Mail Transfer Protocol (SMTP).

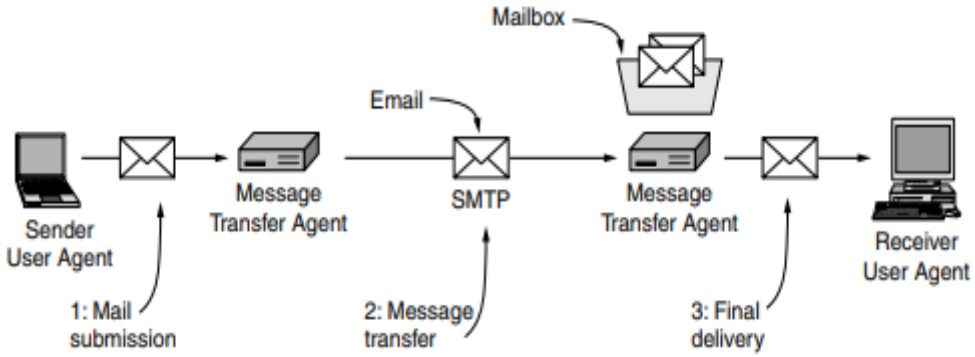


Figure 19.6 Architecture of the email system User Agents

A User Agent provides either a graphical or text-based interface for sending and reading emails. It enables message composition, replying, organizing (by filing, searching, or deleting), and sending (mail submission).

Modern user agents may include features like spam filtering, auto-replies, and message prioritization. They run on the user's local system and interact with mail servers for sending and retrieving messages.

Message Transfer Agents

Message Transfer Agents (MTAs) are background processes running on mail servers that handle message movement across the network. Using SMTP (defined in RFC 5321), they ensure reliable message transfer and delivery status reporting.

MTAs support additional features like mailing lists, carbon copies (CC), blind carbon copies (BCC), encryption, priority levels, and alternate recipients.

Mailboxes and Message Format

Mailboxes are maintained by mail servers to store received emails. User agents communicate with these servers to view, delete, or organize emails—this step is called final delivery. A single mailbox can be accessed through multiple user agents.

Email mesages follow a standard format (originally RFC 822, now RFC 5322) and support multimedia through MIME. Messages have two key parts: the envelope and the message.
The Envelope contains transport information (destination, priority, and security).
The Message has two components:
Header – control information for the
user agent (e.g., sender, receiver, subject).
Body – the actual message content for the recipient.

Figure 19.7 illustrates the difference between an envelope and a message for both paper and electronic mail.

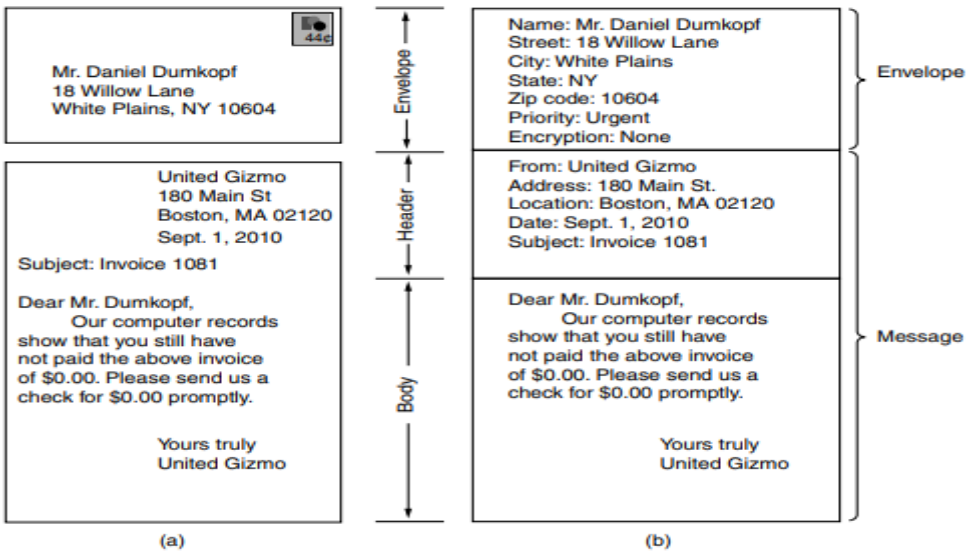


Figure 19.7 Envelopes and messages. (a) Paper mail. (b) Electronic mail.

19.5.2 The User Agent

A User Agent (UA), also called an email reader, is a program that allows users to compose, receive, reply to, and organize email messages. Popular examples include Gmail, Microsoft Outlook, Mozilla Thunderbird, and Apple Mail. While modern user agents have graphical or touch-based interfaces, older ones like Elm, mh, and Pine used text-based interfaces.

Interface Features

A typical user agent displays a summary of messages in the user's mailbox (Fig. 7-9). Each line usually shows the From, Subject, and Received fields, allowing users to identify the sender, topic, and time. Icons may indicate unread mail, attachments, or high-priority messages. Messages can be sorted by time, sender, or subject, and users can customize these preferences.

Message Viewing and Disposition

User agents let users preview and read messages, reformat content, and convert formats (e.g., audio to text). After reading, users can decide on message disposition — such as deleting, replying, forwarding, or saving a message. Saved messages are usually organized in folders like Inbox, Junk Mail, or custom folders.

Automatic Filing and Spam Filtering

Many user agents can automatically classify incoming mail based on predefined rules or spam filters. Internet service providers and organizations assist this process by labeling spam using collective data across users. Automatic filing helps separate legitimate mail from junk efficiently.

Mailbox Search

Modern user agents provide search tools to quickly locate messages by keyword, sender, or date. These advanced tools help manage large mailboxes that can store gigabytes of data and thousands of emails.

Automatic Responses

Some messages trigger automatic responses. For example, forwarding agents can redirect emails to other addresses, while vacation agents send auto-replies like "I'm away until August 24." These responses usually run on mail servers since the user agent may not always be active.

Mail Composition

Mail composition involves creating and sending new messages or replies. User agents assist with addressing, filling header fields, appending signatures, checking spelling, and adding digital signatures for authenticity. The most important part of the outgoing message is the envelope, which includes the destination address in the format user@domain.

Addressing Formats

Besides DNS-style addresses, some systems use X.400 addresses (ISO standard), written as attribute-value pairs, e.g.:

`/C=US/ST=MASSACHUSETTS/L=CAMBRIDGE/PA=360 MEMORIAL DR./CN=KEN SMITH/`

Although less convenient, user agents simplify this through aliases or nicknames, allowing users to send mail without typing long addresses.

Mailing Lists
Mailing lists allow sending a single message to multiple recipients. Lists may be:
Local lists, maintained by the user agent (sending separate messages to each recipient), or
Remote lists, maintained by a mail server, where messages sent to a list address (e.g., birders@meadowlark.arizona.edu) are automatically distributed to all members.

19.5.3 Message Formats

After studying the user agent interface, we now focus on the format of email messages themselves. Messages created by the user agent must follow a standardized format so that the message transfer agents (MTAs) can correctly deliver them. Two major standards govern message formats:

1. RFC 5322 – The Internet Message Format (for text-based messages), and
2. MIME – Multipurpose Internet Mail Extensions (for multimedia and non-ASCII content).

(a) RFC 5322 — The Internet Message Format

RFC 5322 is the current standard defining the structure of Internet email messages. It is an updated version of the original RFC 822.
A standard email message consists of:

1. An envelope (defined by SMTP – RFC 5321)
2. Header fields
3. A blank line
4. The message body

Each header field contains a field name, a colon, and a value (e.g., Subject: Meeting Agenda).
Message Transport Headers
The main transport-related headers are shown in Figure 19.8

Header	Meaning
To:	Email address(es) of primary recipient(s)
Cc:	Secondary recipients (Carbon copy)
Bcc:	Blind carbon copy recipients (not visible to others)
From:	Person(s) who created the message
Sender:	Actual sender (may differ from the author)
Received:	Added by each transfer agent; shows routing path
Return-Path:	Path back to the sender or return address

Figure 19.8 RFC 5322 header fields related to message transport

The *To*, *Cc*, and *Bcc* fields specify recipients. While *Cc* is visible to all recipients, *Bcc* hides its addresses.
The *From* and *Sender* fields identify the author and the person who actually transmitted the message. Each mail server that handles the message adds a Received line with time and server

details. The *Return-Path* field is typically filled by the last server to indicate how replies or delivery errors can reach the sender.

User and Display Headers
RFC 5322 also defines headers used mainly for user-level processing (Figure 19.9).

Header	Meaning
Date:	Date and time the message was sent
Reply-To:	Address for sending replies
Message-ID:	Unique identifier for the message
In-Reply-To:	Message-ID of the message being replied to
References:	Related message identifiers
Keywords:	User-chosen keywords
Subject:	Short summary of the message

Figure 19.9 Some fields used in the RFC 5322 message header

The Message-ID uniquely identifies a message and helps in organizing threads of replies. The Reply-To field can direct replies to a different address (e.g., for marketing or multi-account scenarios).

Additionally, users may define custom headers prefixed with "X-" (e.g., X-Priority: or humorous fields like X-Fruit-of-the-Day:).

After all headers, a blank line separates them from the message body, which can contain text, signatures, or other user content.

(b) MIME — Multipurpose Internet Mail Extensions

Originally, email systems supported only plain ASCII text messages. With the rise of the Internet and global communication, users needed to send messages in multiple languages, character sets, and media formats (such as images, audio, and documents).
To address this, MIME was developed. It extends RFC 822/5322 to support multimedia email content while remaining compatible with existing email protocols.
MIME is defined in RFCs 2045–2049 and 4288–4289.

MIME Message Headers
MIME adds five additional headers (Figure 19.10).

Header	Meaning
MIME-Version:	Identifies MIME version used
Content-Description:	Short description of message content
Content-ID:	Unique identifier for the content
Content-Transfer-Encoding:	Encoding method for message body
Content-Type:	Type and format of the message content

Figure 19.10 Messages headers added by MIME

Content-Transfer-Encoding defines how non-ASCII data is encoded for safe transmission over SMTP (which was originally 7-bit).

Common encoding methods include:

- **7-bit ASCII:** For plain text
- **8-bit:** For extended character sets
- **Binary:** For raw binary files (e.g., executables)
- **Base64:** Encodes binary data as ASCII characters (common for attachments)
- **Quoted-Printable:** Efficient for mostly text messages with occasional special characters

The **Content-Type** header describes the nature and structure of the message body

Type	Example Subtypes	Description
text	plain, html, xml, css	Text content in various formats
image	gif, jpeg, tiff	Still images
audio	basic, mpeg, mp4	Sound data
video	mpeg, mp4, quicktime	Moving pictures
model	Vrml	3D models
application	pdf, javascript, zip	Application-specific data
message	http, rfc822	Encapsulated messages
multipart	mixed, alternative, digest	Messages with multiple parts

Figure 19.11 Message headers added by MIME

Multipart Messages

MIME's **multipart** type enables a single message to contain multiple parts, such as a text message with image or document attachments.

- **multipart/mixed:** Combines different types (e.g., text + attachments).
- **multipart/alternative:** Same message in different formats (e.g., plain text and HTML).
- **multipart/parallel:** For synchronized data (e.g., audio and video).
- **multipart/digest:** Collection of multiple messages (e.g., mailing list summaries).

Example

A birthday greeting email might be sent as a multipart/alternative message — one part containing an HTML message and another an audio file (audio/basic). If the recipient's system supports audio playback, it plays the sound; otherwise, the HTML text is displayed.

19.6 SUMMARY

This chapter covers DNS and electronic mail, two critical Internet services. DNS translates human-readable domain names into IP addresses, using a hierarchical name space, resource records, and distributed name servers for resolution. Electronic mail involves a layered architecture with a user agent, mail servers, and defined message formats. Messages are transferred between servers and ultimately delivered to recipients, providing reliable and standardized communication across the Internet.

19.7 TECHNICAL TERMS

Domain Name System, IP address, Email

19.8 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the DNS name space and hierarchical structure.
2. Describe DNS resource records and the function of name servers.
3. Discuss the architecture and services of electronic mail.
4. Explain message formats and the process of message transfer.
5. Describe final delivery of email and the role of various mail servers.

Short Questions:

1. What is the purpose of DNS?
2. Define the DNS name space.
3. What are resource records in DNS?
4. Name the main components of an email system.
5. What is the role of the user agent in email?

19.9 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008)

Mrs. Appikatla Pushpa Latha

LESSON- 20

THE WORLD WIDE WEB AND MULTIMEDIA

OBJECTIVES:

After going through this lesson, you will be able to

- Understand the architecture of the World Wide Web (WWW).
- Learn the difference between static and dynamic web documents.
- Study the HTTP protocol and web performance enhancements.
- Understand the basics of digital audio and video in networking.
- Learn multimedia applications such as streaming, Internet radio, VoIP, and video on demand

STRUCTURE OF THE LESSION:

20.1 INTRODUCTION

The Web, often referred to as the World Wide Web, is a vast architectural framework designed to allow seamless access to interconnected content distributed across millions of machines around the globe. In a remarkably short span of just ten years, it transformed from a tool to support scientific collaboration in high-energy physics research to the primary application that many people now equate with the entire Internet. Its widespread adoption and massive popularity are largely due to its user-friendly interface and the ability it provides users to navigate an immense wealth of information in a highly visual and intuitive manner. Regardless of the topic—ranging from animals like aardvarks to distant cultures such as the Zulus—the Web offers accessible and engaging content to users with even minimal technical expertise.

The origin of the Web can be traced back to 1989 at CERN, the European Center for Nuclear Research. At the time, the primary motivation was to facilitate collaboration among large,

geographically dispersed teams of scientists working on complex particle physics experiments. These teams produced and needed to access a growing collection of evolving documents such as reports, diagrams, drawings, and photographs. Tim Berners-Lee, a physicist at CERN, proposed a system of linked hypertext documents to address this challenge. Within 18 months, a basic, text-only prototype of the system was operational, allowing users to navigate linked documents with ease. The concept was formally presented at the Hypertext '91 conference, where it captured the interest of researchers beyond the physics community. This exposure eventually led to the development of Mosaic—the first graphical web browser—by Marc Andreessen at the University of Illinois, which was released in February 1993.

The release of Mosaic marked a major turning point in the Web's evolution. Its graphical interface made the Web dramatically more appealing to the general public and significantly lowered the barrier to entry for new users. Mosaic's success led Andreessen to leave the university and establish Netscape Communications Corporation, a company aimed at commercializing Web software. What followed was the infamous "browser war" of the mid-1990s, a fierce competition between Netscape Navigator and Microsoft Internet Explorer. Both companies raced to release new versions of their browsers, each introducing a flood of features—often at the expense of stability and security—in their bid to dominate the new digital frontier.

As the Web matured through the late 1990s and 2000s, the number of Web pages and websites expanded exponentially. What had started as a relatively modest system of linked documents quickly exploded into a sprawling ecosystem of billions of pages. Among these, a select group of websites gained immense popularity and helped shape how users interacted with the Web. Companies such as Amazon, which began in 1994 as an online bookstore and later expanded to become a global e-commerce giant, and eBay, founded in 1995 as a digital flea market, emerged as early success stories. Google, which was founded in 1998 by two Stanford students, Sergey Brin and Larry Page, revolutionized online search and quickly became one of the most valuable companies in the world. Social networking also found its footing on the Web, with Facebook—launched in 2004 by Harvard student Mark Zuckerberg—quickly growing into a global social media powerhouse. These platforms not only defined the user experience of the Web but also shaped global communication, commerce, and culture in profound ways.

The late 1990s also saw the rise and fall of countless Web-based startups in what became known as the dot-com era. During this time, companies with little more than an idea and a catchy domain name attracted enormous investment and skyrocketed in value, only to collapse soon after when their business models proved unsustainable. Despite this turbulent period, the Web continued to evolve, and new innovations continued to emerge—often from student-led initiatives and academic projects. The Web proved to be an open arena for creativity and disruption, with the potential to turn a simple idea into the next big thing on a global scale.

To ensure the continued growth and interoperability of the Web, CERN and the Massachusetts Institute of Technology (M.I.T.) signed an agreement in 1994 to establish the World Wide Web Consortium (W3C). This organization, directed by Tim Berners-Lee, was tasked with developing Web standards, promoting universal protocols, and fostering compatibility across the global Web ecosystem. Over the years, W3C has grown to include

hundreds of universities, research institutions, and commercial companies, all contributing to the ongoing development of the Web.

20.2 ARCHITECTURAL OVERVIEW

The Web presents itself to users as an enormous and interconnected repository of information, encompassing millions of web pages hosted across servers all over the world. Each of these pages, often referred to simply as "pages," can include embedded references, or links, to any other pages, whether they reside on the same server or one located on the opposite side of the globe. Users interact with this system through a simple mechanism: clicking on hyperlinks, which initiates the process of retrieving and displaying the linked page, enabling a seamless browsing experience. This model of interconnected pages, now called hypertext, was conceived by Vannevar Bush in 1945, decades before the actual implementation of the Internet. His visionary idea was based on the notion of information retrieval and navigation through related documents, which became a reality only with the development of modern computer networks.

Users access and navigate web content using programs known as browsers—examples include Firefox, Chrome, and Internet Explorer. These browsers are responsible for fetching requested pages from remote servers, interpreting their contents, and rendering them in a visually coherent manner for users. The content retrieved might consist of a mixture of text, images, video, and interactive scripts, giving each web page a rich and dynamic nature. A graphical representation of a page—such as the homepage of the Computer Science & Engineering department at the University of Washington—typically displays a combination of text and images, some of which are configured as hyperlinks. When users click on these hyperlinks, the browser initiates a request to retrieve the corresponding target page, providing an interactive and intuitive method of exploration. The hyperlink could appear as underlined text, a button, an image, or even a dynamic visual element that changes when hovered over with a mouse. The choice of how the link appears is entirely up to the web page designer, and thoughtful design is crucial to ensure users understand what is clickable.

Web navigation is designed to be transparent to the user. A page might contain content that originates from multiple servers, and the browser's job is to integrate all this data into a single unified display. This process is illustrated in an example involving the cs.washington.edu homepage, where the page includes not just university content, but also an embedded video served by youtube.com and analytics information fetched from google-analytics.com. All these elements come together in the browser's rendering process without requiring the user to know or do anything about their distinct origins. This integration of content from multiple servers is made possible through a well-defined communication protocol known as HTTP—HyperText Transfer Protocol. HTTP operates over TCP and is a simple, text-based request-response mechanism. When a user navigates to a page, the browser sends an HTTP request to the relevant server(s), which then respond with the requested content. This content might be static or dynamic. A static page is one that remains the same for all users, whereas a dynamic page changes based on user interaction, preferences, or data retrieved at the time of the request.

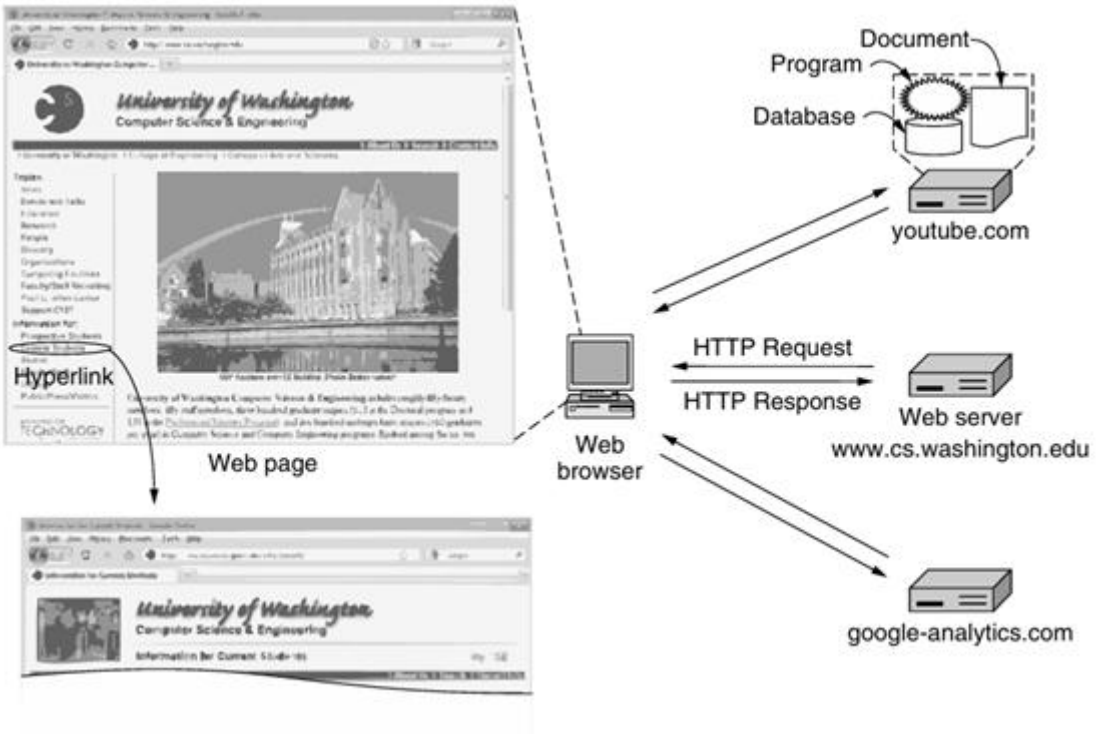


Figure 20.1 Architecture of the Web

Dynamic pages enable personalized experiences, which is increasingly important in the modern Web. For instance, when a user visits an online bookstore, the front page might adapt itself based on their past purchases. A reader who frequently buys mystery novels will likely see new thrillers featured prominently, while another interested in cooking might be greeted by the latest cookbooks. This customization is achieved through mechanisms such as cookies, which help the website remember user preferences and tailor content accordingly. This personalization capability is a key reason behind the Web's effectiveness as a medium for e-commerce, education, and entertainment.

As web pages become more complex and feature-rich, the browser must handle not just the basic task of rendering static documents, but also executing scripts, interacting with media content like videos, and incorporating data from third-party services. For instance, an embedded video from YouTube might play directly within a page hosted by a university, while analytics scripts running invisibly in the background collect data about how the user interacts with the site. These behind-the-scenes processes contribute to both the functionality and the business models of many modern websites. The seamless integration of varied content and the use of user behavior tracking tools exemplify the evolving sophistication of the Web. In this way, the Web remains not only a platform for accessing information but also a dynamic environment for interaction, customization, and innovation.

Client side

A Web browser is a specialized program designed to display web pages and manage user interactions such as mouse clicks on hyperlinks. When a user clicks on a link, the browser interprets the selected element as a command to fetch the target page, initiating a series of behind-the-scenes processes. These processes ensure that the appropriate content is retrieved from the correct location and presented in a readable, interactive format. This seemingly simple operation involves several critical steps and systems working in unison, especially

given the complexities of naming and locating information in a globally distributed network like the Web.

At the core of web navigation is the need to name and locate web pages unambiguously. This challenge arises because merely assigning unique identifiers to web pages is insufficient; one must also be able to locate the resource and determine how to access it. To illustrate, a person in the United States may have a unique social security number, which serves as an identifier. However, without an address or communication language, knowing this number alone provides little actionable information. The Web faces a parallel dilemma. It's not enough to uniquely identify a page; there must also be mechanisms to locate it and determine the protocol for retrieval and communication.

The solution to this tri-fold problem lies in the concept of the Uniform Resource Locator, or URL. A URL encapsulates all three critical components necessary to locate and retrieve a web resource. First, it includes the protocol to be used for accessing the resource, such as HTTP. Second, it specifies the DNS name of the machine hosting the resource. Third, it includes a path to the specific file or application on the host machine. For instance, the URL http://www.cs.washington.edu/index.html breaks down into the HTTP protocol, the domain name of the host server, and the path to the file being requested. The path may appear hierarchical, reflecting a directory structure, though this is not strictly enforced and its interpretation is left to the server's configuration.

When a user selects a hyperlink, the browser undergoes a structured sequence of actions to retrieve the associated page. First, it extracts the URL of the selected hyperlink. Then, it contacts the Domain Name System (DNS) to resolve the domain name—like www.cs.washington.edu—into its corresponding IP address. Upon receiving the IP address (e.g., 128.208.3.88), the browser initiates a TCP connection to that address on port 80, which is the standard port for HTTP. Once the connection is established, the browser sends an HTTP request to the server, asking for the specific resource—in this case, /index.html.

The server responds by transmitting the requested page as an HTTP response. If the retrieved page contains embedded references to other resources, such as images, scripts, or videos, the browser initiates additional HTTP requests to fetch those as well. These additional resources might come from the same server or entirely different domains. For example, the page might include images hosted on the same university server, a video from YouTube, and tracking scripts from Google Analytics. All of these are fetched in parallel by the browser and integrated seamlessly into the final display. Once all necessary components are retrieved, the browser renders the page for the user to view. After a brief idle period without further requests, the TCP connections are typically released to free up resources.

Modern browsers often display real-time feedback about these steps in a status bar or loading indicator. This transparency helps users understand performance issues, such as whether delays are due to DNS lookup problems, server unavailability, or slow data transfer. The structured and modular nature of URL design makes it easy to support multiple protocols, not just HTTP. Over time, the URL schema has been extended to accommodate a variety of resource types and access methods.

In addition to HTTP, other common URL protocols include HTTPS for secure communication, FTP for file transfers, FILE for accessing local files, and MAILTO for initiating email composition. For instance, HTTPS provides encrypted data transfer and is used extensively by banks and e-commerce platforms to protect sensitive information. FTP is a legacy protocol used for downloading files from public repositories, and the Web provides a graphical interface to these resources, simplifying access compared to traditional command-line clients. The FILE protocol allows browsers to render local HTML files directly from a user's device, bypassing the need for a web server. MAILTO links facilitate the sending of emails directly from web pages by launching the user's default mail client pre-filled with a recipient address.

There are also protocols designed for media streaming and communication. RTSP (Real-Time Streaming Protocol) is used for establishing and controlling media streams, such as video or audio, while SIP (Session Initiation Protocol) is used for initiating and managing multimedia communication sessions, including voice and video calls. Browsers also use special schemes like ABOUT to display internal information. For example, about:plugins shows details about installed plugins that handle specific types of content. These additional protocols extend the capabilities of URLs far beyond simple page retrieval, unifying many internet-based services within the familiar interface of a web browser.

The Web's reliance on URLs as a universal addressing system integrates a wide array of services and protocols into a single, cohesive user experience. This unified approach eliminates the need for separate user interfaces for various services, such as email clients, FTP applications, or streaming players. The browser becomes a one-stop portal to the entire Internet, dramatically enhancing usability and accessibility. This elegant design was the brainchild of Tim Berners-Lee, a British physicist working at CERN in Switzerland, whose idea ultimately revolutionized the way information is accessed and shared.

Despite its strengths, the URL system has a notable limitation: it ties a resource to a specific location, which can be inefficient or impractical in certain contexts. For example, high-demand pages might benefit from being hosted in multiple locations worldwide to balance traffic and reduce latency. However, traditional URLs cannot express a request for a resource irrespective of its location. This leads to a challenge where users are forced to access a specific server, even if a nearby server could serve the same content more efficiently.

Name	Used for	Example
http	Hypertext (HTML)	http://www.ee.uwa.edu/~rob/
https	Hypertext with security	https://www.bank.com/accounts/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Local file	file:///usr/suzanne/prog.c
mailto	Sending email	mailto:JohnUser@acm.org
rtsp	Streaming media	rtsp://youtube.com/montypython.mpg
sip	Multimedia calls	sip:eve@adversary.com
about	Browser information	about:plugins

Figure 20.2 Some common URLschemes

To address this limitation, the concept of Uniform Resource Identifiers (URIs) was introduced. URIs generalize URLs by distinguishing between two classes: locators and names. URLs, which we've been discussing, are a subset of URIs that specify how to locate a resource. Uniform Resource Names (URNs), on the other hand, provide a unique name for a resource without specifying where or how to retrieve it. This abstraction allows for more flexible and distributed architectures where the resolution of a name to a location can happen dynamically. The formal syntax and usage of URIs are specified in RFC 3986, and the list of registered URI schemes is maintained by the Internet Assigned Numbers Authority (IANA). Although many URI schemes exist, the protocols listed earlier continue to dominate the Web due to their widespread support and integration.

In summary, the Web browser operates as a sophisticated and flexible client capable of retrieving, interpreting, and displaying a wide variety of content from diverse sources. Its reliance on the URL system enables precise, universal access to resources while remaining open to further extensions and protocols. This architecture supports a broad range of use cases—from simple static pages to dynamic, interactive applications—and continues to evolve as the foundation of the modern Internet experience.

The MIME type

To ensure that browsers can correctly display web pages, it is crucial that they understand the format in which the content is delivered. This is achieved through a standardized language known as HTML (HyperText Markup Language), which has become the universal language of the web. HTML defines the structure and layout of a web page using various tags and elements. Although web browsers primarily serve as HTML interpreters, they are equipped with many additional features to facilitate smooth navigation. For example, users can easily move back to previously viewed pages, advance to the next pages, return to their home page, or manage bookmarks for future reference.

Web pages are no longer restricted to just text and hyperlinks. They may also contain videos, PDFs, images, songs, and many other types of multimedia files. As a result, browsers often encounter files they do not inherently know how to display. To manage this, web servers send supplementary information along with each file, including the MIME (Multipurpose Internet Mail Extensions) type. The MIME type tells the browser what kind of file it is dealing with, such as text/html for an HTML document, image/jpeg for a photo, or application/pdf for a PDF. If a browser encounters a MIME type it cannot interpret on its own, it refers to an internal table to find the correct way to handle the file. This table maps MIME types to either plug-ins or helper applications that extend the browser's capabilities.

Plug-ins are small software modules that integrate directly into the browser and run within its process space. They allow browsers to display or interact with content types they do not natively support. For instance, plug-ins exist for displaying PDF documents or playing embedded videos. These plug-ins must implement a specific set of functions that the browser can invoke to pass the data for rendering. Conversely, the browser provides certain services that plug-ins can call, such as memory allocation or user messaging. Plug-ins are typically installed manually by users, who download them from the web and register them with the browser. Many browsers come with commonly used plug-ins pre-installed to save users time and effort.

Helper applications, unlike plug-ins, operate as separate programs outside the browser. When a browser encounters a file it cannot display directly, it may launch a helper application to handle the content. For example, clicking on a PowerPoint file might open Microsoft PowerPoint to display it. Helper applications receive the name of a temporary file from the browser and handle the rendering independently. These applications often correspond to specific MIME types, such as application/vnd.ms-powerpoint for PowerPoint files. Since these helpers are independent programs, they are usually more powerful and complex than plug-ins and are not limited by the browser's environment.

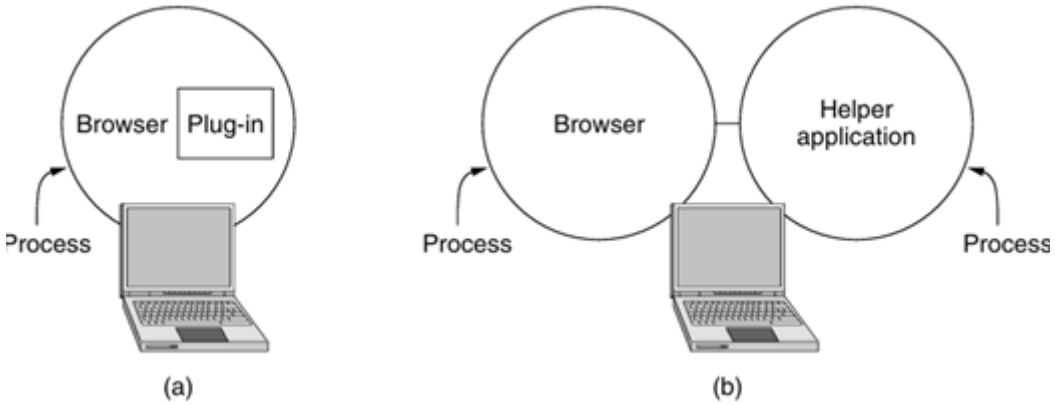


Figure 20.3 A browser plug-in A helper application

Browsers can support an unlimited variety of file types through the use of MIME-type associations. Web servers are typically configured with hundreds of type and subtype combinations to accommodate the wide variety of content available online. However, complications can arise when multiple programs register for the same MIME type. In such cases, the last program installed often overwrites the previous association, potentially changing how the browser handles a particular type of file. This can be inconvenient and confusing for users, especially if a preferred application is suddenly replaced.

Browsers are also capable of opening local files, without involving any network communication. In such cases, the browser relies on the file extension to determine the MIME type, which is mapped to a specific plug-in or helper application. For instance, a file named report.pdf might automatically open in the browser using a PDF plug-in, while document.doc might launch Microsoft Word as a helper application. However, the same issues with conflicting file associations can arise here as well, with newly installed programs potentially hijacking file types during installation. Advanced installation programs often allow users to select which MIME types they wish to associate with the application, while simpler installations aimed at non-technical users may seize control of all supported types without asking.

While the flexibility to extend browser functionality is valuable, it also introduces significant security risks. For example, if a browser automatically executes an .exe file upon download, it could open the door to malicious attacks. A deceptive web page could present seemingly harmless images or videos that, when clicked, actually trigger the download and execution of harmful software. To mitigate this, most modern browsers, especially Firefox and Chrome, are configured to be cautious when dealing with unknown file types. They generally require explicit user confirmation before executing any downloaded program. Nonetheless, some users may prioritize convenience over caution, potentially exposing their systems to malware and other threats.

This complex yet powerful system of MIME types, plug-ins, and helper applications allows modern browsers to handle an incredibly diverse range of content types seamlessly. However, it also demands awareness and careful configuration from users to maintain a secure and reliable browsing experience.

The Server Side

When a user enters a URL or clicks on a hyperlink in a web browser, the browser first parses the URL and identifies the section between "http://" and the next slash as the domain name. It then performs a DNS lookup to retrieve the IP address corresponding to that domain. Using this IP address, the browser establishes a TCP connection to the web server, typically on port 80. It sends an HTTP command containing the rest of the URL, which is essentially the path to the specific file or page on the server. Upon receiving this command, the server processes it and returns the requested content, allowing the browser to display it. This process is fundamentally similar to a simple server loop that accepts TCP connections, retrieves files based on provided paths, sends them over the network, and then releases the connection. For dynamic content, instead of fetching a static file, the server may execute a program to generate the content dynamically. While this simple model works, modern web servers are designed with more sophistication to handle the heavy demand of multiple concurrent requests and to optimize performance.

One of the main limitations of the basic design is that disk access tends to be a major bottleneck. Disk operations are far slower compared to the speed of CPU executions, and if the same files are accessed repeatedly, relying on the disk for every request becomes highly inefficient. Additionally, handling one request at a time is inadequate for high-traffic environments, especially if a large file is being transmitted and other requests are blocked until the transfer completes. To address this, modern servers implement caching mechanisms that store the most frequently or recently accessed files in main memory. Before attempting a disk read, the server checks the cache. If the requested content is found in memory, it is served directly, which significantly reduces response time and system load. Although maintaining and managing a cache requires additional memory and logic, the performance benefits far outweigh the associated costs.

To further enhance performance and support multiple concurrent clients, web servers often adopt a multithreaded architecture. In one such design, the server consists of a front-end module responsible for accepting requests and several processing modules that handle the actual request processing. All these threads run within the same process and can access shared memory resources, such as the cache. When a new request arrives, the front end accepts it and hands off a descriptor of the request to an available processing module. This module then checks the cache, retrieves the file if it is cached, or initiates a disk read if it is not. Once retrieved, the content is both added to the cache and sent back to the client. The major advantage of this multithreaded setup is that while some threads are blocked waiting for disk or network operations to complete, others can continue processing new incoming requests. With multiple processing threads, the server's throughput can improve significantly, although actual performance gains also depend on hardware constraints such as disk speed and network bandwidth.

Modern web servers go beyond merely fetching files from disk and returning them. The processing of HTTP requests can be quite involved, depending on the nature of the request. Once a TCP connection is established, often accompanied by secure transport mechanisms such as SSL or TLS, the server executes a sequence of processing steps. The first task is resolving the name of the requested resource. This may not be a literal filename but a symbolic URL that requires translation into an actual file path using configured rules. For instance, a request to a domain with an empty path often defaults to loading a file like "index.html," and URLs referencing a user directory might be mapped to that user's web folder. Some web servers also tailor content delivery based on browser preferences and language settings to serve a more customized experience.

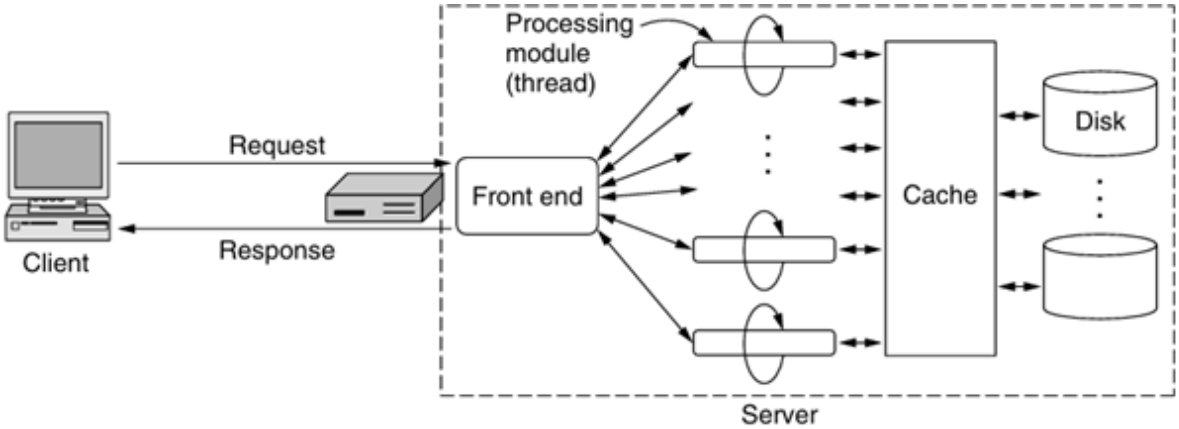


Figure 20.4 A multithreaded Web server with a font end processing modules

Access control is another important aspect of request processing. Not all resources are meant to be publicly available. The server evaluates whether the requesting client is authorized to access the requested content. This decision might rely on authentication methods such as usernames and passwords, or be based on the client's IP address or domain. Configuration files like ".htaccess" are commonly used in systems like Apache to define access rules for directories and pages. After confirming access rights, the server determines whether the requested page can be served from the cache. For dynamically generated pages, caching might be avoided to ensure fresh output is delivered. In such cases, input parameters for dynamic programs are extracted from the URL or HTTP request.

Following this, the server decides on additional elements of the HTTP response. One such element is the MIME type, which indicates the nature of the content being served. The MIME type can be inferred from the file extension, initial content of the file, configuration settings, or other heuristics. Once this metadata is determined, the server sends the response back to the client. To improve efficiency, a persistent TCP connection may be used for multiple requests between the client and server, requiring the server to map responses appropriately. Finally, the server logs the transaction, capturing details that can later be analyzed for system management, security, or understanding user behavior. These logs are invaluable for tracking usage patterns and optimizing content delivery strategies.

Cookies

In the early stages of the web, each interaction between a user's browser and a web server was entirely independent, with no mechanism for maintaining continuity between visits. When a user requested a web page, the server would respond with the requested data and then forget

everything about the client, effectively treating each request as if it came from a completely new visitor. This stateless model was sufficient for browsing publicly available documents but proved inadequate for more complex interactions that required maintaining context across multiple requests. For example, websites that required user registration or payment needed a way to distinguish between returning users and new visitors. Similarly, e-commerce sites had to keep track of what a user placed in their shopping cart as they browsed, and personalized portals needed a mechanism to show tailored content based on user preferences. This raised the need for a method to persist user information between multiple interactions, without relying on unreliable or insecure methods like tracking IP addresses, which could be shared among multiple users or change frequently due to technologies like NAT and DHCP.

To address this challenge, web developers introduced cookies—small pieces of data sent from a web server to a user's browser, which the browser stores and sends back with future requests to the same server. Cookies enable servers to maintain stateful information about users across multiple interactions. These cookies are just strings, limited in size (usually up to 4 KB), and stored on the client's disk, often persisting even after the browser is closed, unless they are explicitly deleted or marked as nonpersistent. While they are not executable and generally considered safe, malicious actors can exploit browser vulnerabilities to misuse them. A typical cookie includes several fields such as the domain from which it originated, a path indicating the portion of the site it applies to, a name-value pair for storing custom information, an expiration date that determines whether it is persistent or not, and a secure flag indicating that it should only be sent over encrypted connections like HTTPS.

Cookies serve a wide variety of purposes. For instance, a gambling site might assign a unique customer ID to each user through a cookie, enabling the server to present personalized game options on return visits. An online store could use cookies to track items placed in a shopping cart as the customer browses, adding each item's product code to the cookie and updating it dynamically. When the user proceeds to checkout, the full shopping list is already known to the server via the cookie. A web portal could store user preferences for news categories, stock tickers, or sports teams in a cookie so that a personalized homepage can be quickly generated on subsequent visits. Since cookies are returned with every request to their originating domain, they provide a convenient way for servers to recall previous interactions without storing large amounts of user-specific data on the server itself.

However, the power of cookies also leads to significant privacy concerns, particularly when they are used for tracking users across multiple websites. Advertising networks often embed cookies in seemingly unrelated websites by including their content—like image ads—hosted on their own servers. When a user visits any such website, their browser fetches the ad image and receives a tracking cookie from the advertising server. As the user browses to different sites using the same advertising network, the cookie is sent again, allowing the ad network to build a comprehensive profile of the user's browsing habits. Over time, this can reveal a surprising amount of personal information. If the user ever provides identifying information, such as their name or email, to any site cooperating with the tracker, the anonymous browsing data can be linked to a real-world identity and potentially sold or misused.

One particularly stealthy tracking method involves invisible elements such as 1×1 pixel images (often called tracking pixels) that are the same color as the background and thus not visible to users. Despite being imperceptible, these elements function like any other content:

they must be fetched from a server, which enables that server to drop or read cookies from the user's browser, further enhancing its ability to track user behavior. Because of this, cookies—especially third-party cookies set by domains other than the one the user is actively visiting—have become a flashpoint in discussions about digital privacy. Many users are unaware that such extensive tracking is even happening, leading to calls for greater transparency and control.

To mitigate these issues, browsers offer various privacy settings. Some users choose to block all cookies, though this can interfere with legitimate website functionality such as login sessions or shopping carts. A more common approach is to block only third-party cookies, which helps to prevent cross-site tracking while allowing necessary cookies from the main site to function properly. Additionally, browser extensions and privacy tools give users more granular control over which cookies are accepted and retained. In response to growing concerns, companies have begun drafting privacy policies that purport to limit data sharing, though such policies are often vague and self-serving. The statement that a company may use collected data "in the conduct of our business" can include a wide range of activities, including selling personal information to third parties. Ultimately, while cookies play a critical role in enabling many of the web's most useful features, their potential for abuse continues to fuel ongoing debates about how to balance functionality with user privacy.

20.3 STATIC WEB PAGES

The fundamental operation that underlies the World Wide Web is the transfer of web pages from servers to clients, typically initiated when a user clicks a hyperlink or enters a URL in their browser. This client-server interaction enables users to access a vast array of online content. In the simplest and earliest implementation of this model, the pages delivered from the server are static. A static web page is a fixed file residing on a web server that is transmitted to the client exactly as it is stored, without any modification or interaction with back-end databases or scripts. When a user requests such a page, the server simply reads the file from its disk and sends it to the browser, where it is rendered and displayed. Every time the page is requested, the server provides the same content, making the experience identical for all users and consistent across time—unless the actual file on the server is manually updated by the webmaster or administrator.

Despite the term "static," these pages are not necessarily dull or limited in functionality. Static web pages can include rich media elements such as images, animations, audio, and even video content that is embedded within the page. These elements can be interacted with or played back on the client side through the browser's built-in capabilities or plugins. Therefore, even though the underlying page does not change dynamically with user input or real-time data, it can still offer an engaging experience. Many educational, informational, and personal websites rely on static content for simplicity, security, and ease of maintenance, especially when the content does not require frequent updates or user interaction.

The primary language used to build and structure these static pages is HTML, or HyperText Markup Language. HTML is the foundational markup language of the Web, used to define the structure, layout, and content of web documents. It allows developers to organize text, insert links, include images, and apply formatting using tags and attributes. For instance, a basic HTML document includes elements like headings, paragraphs, tables, and lists, all of which help to present information in a readable and organized manner. HTML is

complemented by CSS (Cascading Style Sheets) for styling and JavaScript for adding interactivity, but in a purely static site, JavaScript is often minimal or completely absent.

Static HTML pages are especially common among personal websites, academic homepages, and other sites where the content remains relatively unchanged over time. For example, a university professor's homepage might consist of a static HTML file listing their research interests, published papers, office hours, and contact information. Such a page doesn't require server-side logic or databases because the content is straightforward and updates only occasionally. Static pages are also favored for their simplicity, faster loading times, and better security posture compared to dynamic pages, which involve more complex backend infrastructure and are more susceptible to vulnerabilities like SQL injection or cross-site scripting.

On the other hand, corporate websites, commercial platforms, and modern web applications typically rely on dynamic content to deliver personalized, interactive, and real-time experiences. These pages are generated on the fly by server-side scripts and can change based on user behavior, preferences, and current system data. While static HTML is sufficient for many use cases, it forms only the foundational layer of web development. As user expectations and the complexity of online services have evolved, the need for dynamic and responsive content has become more pronounced, leading to the widespread adoption of dynamic web technologies. Nonetheless, understanding static HTML remains essential for grasping how the web functions at its most basic level and serves as a critical stepping stone toward learning more advanced topics such as dynamic content generation, scripting, and web services.

HTML - Hyper Text Markup Language

HTML, or HyperText Markup Language, was developed specifically for the Web to enable users to create documents that are not only text-based but also rich with multimedia and interactivity. These documents, called web pages, can include a variety of elements such as formatted text, graphics, videos, and links to other web resources. As a markup language, HTML focuses on structuring content and defining how it should be displayed by a browser. The term "markup" itself originates from the traditional publishing industry, where editors would annotate manuscripts with instructions for typesetters on how the text should appear in print. Similarly, HTML embeds formatting instructions within the document itself. For example, placing before and after a word will instruct the browser to render it in bold. This structured formatting makes it easy for any web browser to interpret the HTML content and render it appropriately, regardless of the device or display size.

One of the primary strengths of HTML is that it separates content from its presentation. Instead of hardcoding visual details directly into the content, HTML uses tags that tell the browser how to present the material. This allows for greater flexibility and consistency, especially as web pages must adapt to a variety of screen sizes and devices—from high-resolution desktops to compact mobile phones. Because HTML uses standard tags and follows a universal structure, any modern browser can read and display content accurately even if it was written years ago or by someone using a completely different device. This compatibility is essential for the Web to function smoothly on a global scale. Writing HTML can be done manually using a simple text editor, but there are also specialized HTML editors

and web design tools that simplify the process by generating code automatically as the user designs a page visually.

HTML tags can also include attributes, which provide additional information about how the element should behave or appear. For instance, the <img> tag is used to embed images and often includes attributes like src for the image source URL and alt for alternate text in case the image cannot be displayed. Attributes are always named and can be listed in any order within the tag. Because the HTML standard allows for flexibility, tags can be written in either uppercase or lowercase, although lowercase is generally preferred for compatibility. Extra spaces, line breaks, and indentation in the HTML code do not affect how the browser renders the content, which makes it easier for developers to format their code for readability.

HTML also includes tags for creating lists and hyperlinks, two features that are central to the Web's functionality. Unordered lists, which appear as bulleted points, are created using the <ul> tag with individual items wrapped in <li> tags. Ordered lists, which appear numbered, use the <ol> tag instead. Hyperlinks, which allow users to navigate from one page to another, are created using the <a> tag with the href attribute pointing to the destination URL.

The clickable text appears between the opening <a> and closing tags. Hyperlinks can even be wrapped around images or other elements to create interactive buttons and visual links. Additional tags like <h1> to <h6> define heading levels, while <b> and <i> are used for bold and italic text, respectively. Tags like begin a new paragraph, and although an ending is technically required, many browsers will handle its omission gracefully.

```
<html>
<head> <title> AMALGAMATED WIDGET, INC. </title> </head>
<body> <h1> Welcome to AWI's Home Page </h1>
<img src="http://www.widget.com/images/logo.gif" ALT="AWI Logo"> <br>
We are so happy that you have chosen to visit <b> Amalgamated Widget's</b>
home page. We hope <i> you </i> will find all the information you need here.
<p>Below we have links to information about our many fine products.
You can order electronically (by WWW), by telephone, or by email. </p>
<hr>
<h2> Product information </h2>
<ul>
   <li> <a href="http://widget.com/products/big"> Big widgets </a> </li>
   <li> <a href="http://widget.com/products/little"> Little widgets </a> </li>
</ul>
<h2> Contact information </h2>
<ul>
   <li> By telephone: 1-800-WIDGETS </li>
   <li> By email: info@amalgamated-widget.com </li>
</ul>
</body>
</html>
```

Figure 20.5 (a) The HTML for a sample Web page(b)The formatted page

Over time, HTML has evolved through several versions to support new capabilities. Early versions such as HTML 1.0 and 2.0 provided only basic functionality—text formatting, links, and images. With HTML 3.0, tables and more advanced formatting became possible. HTML 4.0 added features like accessibility support for users with disabilities, object embedding (extending beyond simple images), and scripting capabilities using JavaScript. HTML 5, the most recent major standard, marked a significant shift. It introduced support for rich media, including native video and audio playback without requiring external plugins. It also added capabilities for interactive graphics through vector-based drawing (using <canvas>), offline storage, background computational threads, and better support for complex application-like behavior directly in the browser. HTML 5 reflects the web's evolution from static document sharing to fully interactive applications that run in the browser and rival traditional desktop software.

One particularly valuable aspect of HTML is its backward compatibility and openness. A web page created years ago using basic HTML will still display correctly in modern browsers, thanks to the consistent and well-documented behavior of HTML tags. For users or developers interested in learning HTML, most browsers include a “View Source” option, allowing anyone to examine the HTML code behind any webpage. This transparency and simplicity helped HTML become one of the most widely understood and utilized languages in the world. Its open nature, combined with continued improvements and the ability to integrate with technologies like CSS and JavaScript, ensures HTML remains the foundation of modern web development, empowering both novice users and professional developers to build and share content across the globe.

Inputs and Forms

HTML initially allowed users to passively view content, but as the web evolved, it became necessary for users to interact with web pages, sending information such as search queries, registration data, or purchase orders back to the server. This functionality marked the

transition from a one-way to a two-way communication model. To enable this, two key elements were introduced: first, the HTTP protocol was enhanced to support methods like POST, which allows data to be submitted to a server; second, HTML was expanded to include forms, which provide graphical user interface elements such as text boxes, checkboxes, radio buttons, and submit buttons. Forms are enclosed within <form> tags, and although they are part of static HTML content, they enable interactive user behavior. When a user fills in a form and submits it, the browser collects all the data, encodes it using a simple scheme (where & separates fields and + represents spaces), and sends it to a designated server URL for processing.

A typical HTML form might include several input elements for collecting user information. For example, text input boxes allow users to enter their name, address, city, and other personal data. These are created using the <input> tag with type set to "text" and can include size attributes to define how much space is displayed. For choices among options, radio buttons are used, which belong to the same group if they share the same name attribute—clicking one will deselect the others, just like the buttons on a car radio. For example, users may choose between Mastercard or Visa for payment using such buttons. If there is a binary option, like whether to use express shipping, checkboxes are employed. These allow for independent selection, meaning multiple boxes can be checked or left unchecked as desired.

```
<html>
<head> <title> AWI CUSTOMER ORDERING FORM </title> </head>
<body>
<h1> Widget Order Form </h1>
<form ACTION="http://widget.com/cgi-bin/order.cgi" method=POST>
<p> Name <input name="customer" size=46> </p>
<p> Street address <input name="address" size=40> </p>
<p> City <input name="city" size=20> State <input name="state" size =4>
Country <input name="country" size=10> </p>
<p> Credit card # <input name="cardno" size=10>
Expires <input name="expires" size=4>
M/C <input name="cc" type=radio value="mastercard">
VISA <input name="cc" type=radio value="visacard"> </p>
<p> Widget size Big <input name="product" type=radio value="expensive">
Little <input name="product" type=radio value="cheap">
Ship by express courier <input name="express" type=checkbox> </p>
<p><input type=submit value="Submit order"> </p>
Thank you for ordering an AWI widget, the best widget money can buy!
</form>
</body>
</html>
```

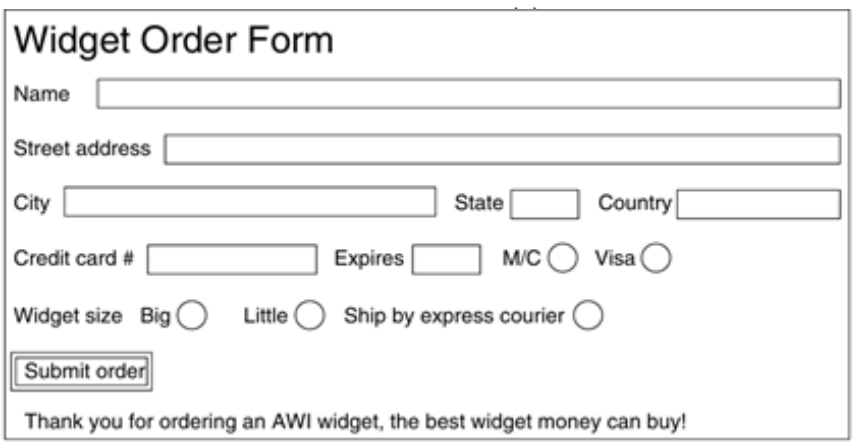


Figure 20.6 (a)The HTML for an order form (b)The formatted page

At the bottom of the form is typically a submit button, created using <input type="submit">. This sends all the form data to the server. When clicked, the browser packages the user's inputs into a long string and transmits it to the action URL specified in the form's attributes. For instance, an order form might result in a string like customer=John+Doe&address=100+Main+St.&city=White+Plains, and so on. The server then interprets this string and passes it to backend logic for processing, such as storing the data in a database or confirming an order. Beyond basic elements, HTML also includes support for more advanced input types. A password input works like a text box but masks the typed characters. A textarea allows for multiline input, useful for comments or messages. When selecting from a list of options, the <select> tag is used, creating dropdown menus, which can function like either radio buttons (single selection) or checkboxes (multiple selections) depending on whether the multiple attribute is included. Forms may also include default values that appear in the input fields when the page loads, allowing users to edit or accept pre-filled information. This rich system of form elements, even within static HTML pages, significantly advanced the Web's functionality, transforming it into a powerful medium for user interaction and input.

CSS- Cascading Style Sheets

Initially, HTML was created with the intention of specifying the structure and logical meaning of content rather than its exact visual appearance. For instance, using a tag like <h1>Deborah's Photos</h1> merely tells the browser that the enclosed text is a primary heading, without providing details on how it should look in terms of font, size, or color. The browser, in turn, decides the exact presentation based on its own defaults and the capabilities of the display system. However, as web design evolved, many developers demanded more control over how their content appeared across different devices and browsers. This led to the inclusion of tags in HTML that allowed designers to dictate specific visual properties. For example, the tag Deborah's Photos provides explicit instructions on the font face, text size, and color. While this allowed for finer control over appearance, it had significant drawbacks. The HTML became cluttered and harder to maintain, and the results were often inconsistent across different browsers or screen sizes. Pages that looked perfect on one browser could render poorly on another, especially if browser versions differed or if users had different screen resolutions.

To address these limitations and promote a cleaner separation of content and presentation, style sheets were introduced. Borrowed from traditional publishing tools, style sheets allowed authors to define how elements should appear based on their logical roles rather than embedding styling directly into the HTML. For instance, instead of specifying "italic blue text" for an introductory paragraph, an author could simply assign it the logical style "initial paragraph" and define the desired appearance for that style separately. This modular approach made it far easier to maintain and update the look of a website. If a designer wanted to change all introductory paragraphs from blue italic text to bold pink text, they only needed to change the style definition once instead of editing every instance throughout the HTML. Cascading Style Sheets (CSS), introduced in HTML 4.0, gave developers the tools to implement this approach effectively. A CSS rule like body {background-color:linen; color:navy; font-family:Arial;} sets default styles for the body of the document, affecting all enclosed content. Additional rules such as h1 {font-size:200%;} and h2 {font-size:150%;}

define how headers should be sized relative to normal text, contributing to a consistent visual hierarchy throughout the page.

One of the powerful features of CSS is the concept of cascading rules and inheritance. Style properties defined at higher levels in the HTML hierarchy automatically apply to nested elements unless explicitly overridden. This approach streamlines style management and ensures consistency. Moreover, CSS definitions can be embedded directly within HTML documents using the <style> tag or, more commonly, placed in separate external files and referenced via the <link> tag in the document's <head> section. For example, adding <link rel="stylesheet" type="text/css" href="awistyle.css" /> connects the HTML document to an external style sheet named "awistyle.css". This practice offers significant advantages. First, it allows for a uniform appearance across multiple web pages, even when they are created by different authors or at different times. A single change to the CSS file can update the look of an entire website, just as modifying a shared header file in C can change behavior across multiple source files. Second, it reduces the size of individual HTML files, since style rules do not need to be repeated within each file. The browser can download the CSS file once and reuse it for all pages that reference it, improving load times and bandwidth efficiency. This separation of content and style not only simplifies the development process but also enhances accessibility, maintainability, and scalability of web content.

20.4 DYNAMIC WEB PAGES AND WEB APPLICATIONS

In the early days of the World Wide Web, the static page model was sufficient because it served the purpose of presenting multimedia documents that could be easily linked to one another. This approach made it possible to publish vast amounts of information online in a manner that was accessible and straightforward. However, as the Web evolved, its utility shifted dramatically from merely displaying content to becoming a platform for full-fledged applications and services. Today, people use the Web for activities such as shopping on e-commerce platforms, searching library databases, exploring interactive maps, managing emails, and even collaborating in real time on shared documents. These functionalities mirror those of traditional desktop applications like email clients and word processors, but with a crucial difference — they operate within the browser environment. This paradigm shift means that users no longer need to install dedicated software for each application. Instead, applications reside on remote servers and use web technologies to deliver interactive user experiences directly through the browser. One major benefit of this model is the ability to access applications and data from multiple devices without needing separate installations, all while ensuring data is securely stored and backed up by the service providers themselves. This centralized approach to computing is a core component of cloud computing, where services and resources are delivered from large clusters of servers over the Internet rather than from local machines.

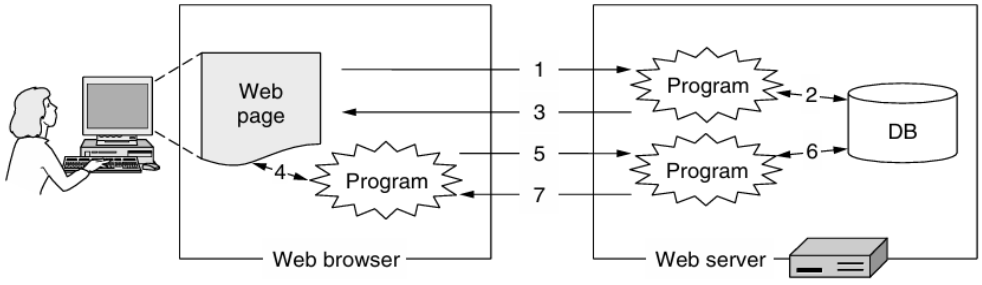


Figure 7-29. Dynamic pages.

Figure 20.7 Dynamic pages

For web applications to be truly interactive and responsive, static content is no longer adequate. What is needed is dynamic content — content that adapts in real-time to reflect user interactions or current data. For instance, an online library catalog must display accurate availability information for books, and a finance website must allow users to view stock trends over time and calculate investment outcomes. These examples highlight the necessity for web pages that change based on input or data, either from user interactions or backend systems. Dynamic content is generated either on the server side, on the client side (within the browser), or through a combination of both. When the server generates dynamic content, it uses web applications to process user requests and pull data from databases in order to craft custom web pages. For example, in an interactive mapping service, when a user enters a street address, the server-side application processes the input, queries a geographic database for the relevant map section, and dynamically generates a web page that displays the map. This process, shown in the early steps of an interaction diagram, involves receiving a request, running a server-side program, and returning the customized page to the client.

The interaction does not end there. The returned page may itself contain client-side scripts — small programs that execute directly in the user's browser. These scripts enhance interactivity by enabling real-time updates to the web page without needing to reload it entirely. Taking the map service example further, the client-side program might allow the user to zoom, pan, or search for nearby points of interest. As the user interacts with the map, the program might need additional data, such as detailed images or route calculations. In such cases, it sends asynchronous requests to the server behind the scenes, retrieves the necessary information, and updates the display accordingly. These interactions occur so seamlessly that users often remain unaware that additional communication with the server is taking place. The page's URL and title typically stay the same, maintaining a continuous and smooth browsing experience. This model of combining server-side and client-side programming allows web applications to be more powerful, efficient, and user-friendly, offering a level of responsiveness that static content alone could never achieve. It marks a major step in the evolution of the Web, turning it into a platform for dynamic, rich, and interactive applications.

Server Side Dynamic Web Page Generation

When server-side content generation is required, especially in the context of web forms, the process becomes more dynamic and tailored to user input. For example, when a user fills out a form like an online order and clicks the submit button, the browser sends a request to a specific URL defined in the form’s configuration, often using the HTTP POST method. This request includes the data filled out by the user, which must be processed by a backend program or script. That URL doesn’t simply retrieve a stored file but actually calls a program to process the incoming data. In a commercial scenario, such as an online widget store, this process would include logging the order into the company’s internal system, updating customer records, and processing the payment. The output—what the user sees as a response page—is dynamically generated based on what happens during this processing. For instance, if the order is successful, the returned page might display a shipping date; if there is a failure, such as an invalid credit card or out-of-stock items, the message would reflect that accordingly. Importantly, the actual mechanism of executing server-side programs is determined by the design of the server software and is not standardized by web protocols,

because from the browser's perspective, the request is simply fetching a page, regardless of how it was generated.

Over time, developers have adopted standard APIs to simplify how web servers invoke these backend programs. One of the earliest and most common is the Common Gateway Interface, or CGI, which is defined in RFC 3875. CGI provides a standard method for web servers to execute backend programs or scripts, allowing them to take input—like form data—and output HTML content in response. These scripts can be written in various languages such as Python, Perl, or Ruby, depending on developer preference. By convention, these CGI scripts are stored in a directory like cgi-bin, which is visible in URLs, and when a request targets that directory, the server executes the relevant script. The input from the form submission is passed to the script, and the HTML output generated by the script is what gets sent back to the user's browser. Often, CGI programs are designed to show the form itself if no user input is present, allowing a single script to handle both displaying the form and processing its results.

Another approach to server-side programming is to embed scripts directly within HTML pages that are processed by the server before being sent to the client. A widely used language for this is PHP (PHP: Hypertext Preprocessor). PHP pages are usually identified by the .php file extension instead of .html, signaling to the server that it must interpret the file using a PHP engine. In this model, when a user submits a form, the server processes the incoming data using PHP commands embedded in the HTML. For example, a form that requests a name and age from the user might send that data to a file named action.php, which then uses PHP code to generate a personalized response. If the user inputs "Barbara" and "32" into the form, the resulting web page might greet her by name and predict her age next year as 33. This is made possible because the PHP code dynamically inserts the submitted values into the HTML being returned to the browser. PHP simplifies dynamic page generation significantly compared to CGI because it allows developers to write and manage logic and presentation in the same file. It supports robust programming constructs like variables, arrays, and control structures, and includes strong capabilities for working with server-side databases. PHP is open-source and was designed to integrate tightly with the Apache web server, which also enjoys wide usage in the industry.

While CGI and PHP are two prominent technologies for dynamic content generation, there are several other options as well. JavaServer Pages (JSP) is a technique similar to PHP but uses Java instead of PHP for its embedded code. Pages developed using JSP typically have the .jsp extension and are used in Java-based server environments. Microsoft's answer to this paradigm is ASP.NET, which uses the .NET framework to power its server-side logic. These pages carry the .aspx extension and are typically deployed on Microsoft's IIS servers. In essence, all three—PHP, JSP, and ASP.NET—enable the same core functionality: dynamically generating web pages based on user input or other data sources. The choice among them often boils down to organizational preference or platform alignment rather than technical superiority, as each offers similar capabilities in terms of dynamic content generation.

Client Side Server Dynamic Web Page Generation

PHP and CGI scripts are essential technologies that address the fundamental need for server-side interactivity on the Web. They handle tasks such as receiving input from users via web forms, performing queries or updates on one or more databases, and then constructing and

sending back dynamically generated HTML pages based on those results. These technologies allow developers to create content that changes based on the user's input, enabling functionality like login authentication, shopping cart management, and data analytics. However, while they are powerful for server-side data processing, these tools lack the ability to directly respond to user actions like mouse movements or clicks without a round-trip to the server. This is where client-side scripting enters the picture. Starting with HTML 4.0, the <script> tag was introduced to allow embedding of scripts that are executed on the user's browser rather than on the web server. This capability opened the door to what is known as Dynamic HTML (DHTML), a collection of technologies that work together to create interactive and responsive web experiences directly within the browser.

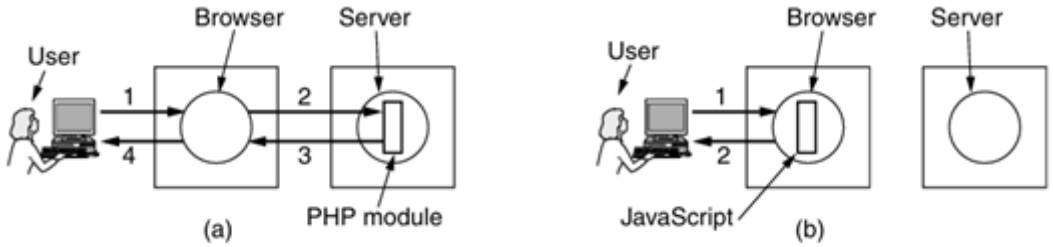


Figure 20.8 (a)Server-side scripting with PHP(b)Client-side scripting with Java Script

The most popular language used for client-side scripting is JavaScript. Despite its name, JavaScript is unrelated to Java and is instead a lightweight, high-level programming language designed for embedding directly into HTML pages. Its high-level nature allows for impressive functionality in minimal code; for instance, a single line can open a dialog box, prompt the user for input, and store that input for further processing. This simplicity and power make JavaScript especially suited for adding interactivity to web pages, such as form validation, dynamic content updates, and responsive user interfaces. However, JavaScript's rapid evolution and inconsistent implementation across browsers have historically made it difficult to ensure consistent behavior on all platforms, though this situation has improved in recent years.

An illustrative example of JavaScript's capabilities is seen in a simple form-handling application. Just like the PHP-based example, a user is asked to input their name and age, but unlike PHP, JavaScript processes this information entirely within the browser. When the user clicks the submit button, a function written in JavaScript is invoked. This function retrieves the values entered into the form, performs basic computation (such as incrementing the age), and then generates new HTML content on the fly to display a personalized message. The function uses methods like document.open() and document.writeln() to construct the output page directly within the browser, without ever needing to contact the server. This kind of local processing results in near-instant feedback to the user, unlike server-side approaches which typically involve a delay due to the network round trip and server processing time.

Although PHP and JavaScript may seem similar because both are embedded within HTML, they are executed in entirely different environments. In the PHP model, once the form is submitted, the browser packages the data and sends it to the server, which then processes the PHP code, interacts with databases if needed, and finally returns a completely new HTML page to be displayed. The browser has no idea how the page was generated; it simply renders the HTML it receives. This server-client interaction cycle introduces a delay and depends on

the availability and responsiveness of the server. By contrast, in the JavaScript model, no data is sent to the server; the browser handles everything locally by executing the embedded JavaScript code as soon as the user interacts with the page. This makes JavaScript especially useful for real-time validation and user interaction.

JavaScript is not the only option for client-side scripting. On Windows-based systems, another option is VBScript, a scripting language derived from Visual Basic. Though less commonly used today, VBScript allows similar interactivity for Internet Explorer browsers. A more powerful and historically significant method for creating interactive content was the use of Java applets. These are small applications written in Java, compiled into bytecode, and run in the browser through the Java Virtual Machine (JVM). Applets are embedded in web pages using the <applet> tag and provide more computational power than JavaScript, since they are interpreted in a secure sandbox environment designed to restrict malicious behavior. However, applets have fallen out of favor due to security concerns, browser incompatibility, and the rise of more secure and modern technologies.

Microsoft's response to Java applets was the creation of ActiveX controls. These are small programs compiled into native x86 machine code and executed directly by the browser on Windows platforms. Because they run on bare hardware rather than in a virtual machine or interpreter, ActiveX controls offer unmatched performance and flexibility, capable of performing complex tasks like accessing hardware devices or modifying system settings. However, this same power introduces enormous security risks, since downloading and running compiled code from untrusted websites could allow malicious programs to harm the user's system. As such, ActiveX is now largely deprecated and is rarely used outside of specific enterprise contexts.

Web developers today must often choose among JavaScript, Java applets, and sometimes ActiveX when they aim to create highly interactive web pages. JavaScript remains the most widely used due to its simplicity and native browser support, though its portability issues persist because different browsers implement JavaScript standards in subtly different ways. Java applets, while more portable across platforms thanks to the uniform behavior of JVMs, are heavier and require additional setup. ActiveX, while the fastest and most flexible, is tied to Windows and Internet Explorer and carries high security risks. Ultimately, the choice of technology depends on the specific needs of the application, the target user base, and the desired trade-off between performance, portability, and security.

AJAX- Asynchronous Javascript and Xml

Modern web applications are expected to provide rich user experiences that are both dynamic and interactive, mirroring the responsiveness and power of traditional desktop applications. To achieve this, developers rely on a variety of technologies, both on the client and server side. Technologies like JavaScript on the client and PHP on the server are fundamental tools, but when combined with other elements, they become part of a more powerful toolkit known as AJAX, or Asynchronous JavaScript and XML. Although often thought of as a technology on its own, AJAX is not a single language or system—it is a synergistic combination of several key technologies that together enable highly responsive web applications such as Google Maps, Gmail, and Google Docs. AJAX integrates HTML and CSS for structuring and presenting content, DOM (Document Object Model) for manipulating page elements dynamically, XML for data formatting and exchange, asynchronous communication for non-

blocking data transfer, and JavaScript as the binding agent that enables everything to work together seamlessly.

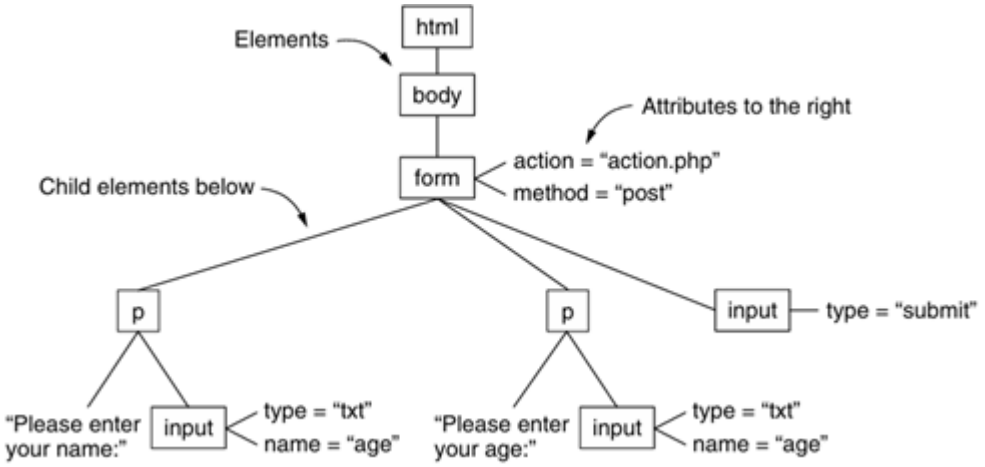


Figure 20.8 The DOM tree for the HTML in Fig.20.7(a)

HTML and CSS form the foundational structure of any web page by defining the content and its visual presentation. These technologies are widely supported and allow any software that can generate HTML and CSS to use the web browser as a rendering engine. However, dynamic web applications go beyond static page content. To allow parts of a web page to be updated without reloading the whole page, the Document Object Model (DOM) is used. The DOM is a programmatically accessible tree representation of the HTML page. Each HTML element is represented as a node in the DOM tree, with relationships that reflect the nesting and attributes of elements in the original HTML. This structure makes it easy for scripts, particularly JavaScript, to modify the page's content and appearance on-the-fly. When a specific DOM node is changed, the browser only re-renders that part of the page, preserving other content and offering a smoother, more efficient user experience. This model enables highly interactive applications, such as forms that update dynamically or new content being loaded without a full page refresh.

Structured data communication between the browser and the server is another critical piece of AJAX, and XML plays a central role in this. XML, or eXtensible Markup Language, is a flexible data format developed to enable automated processing of structured data. Unlike HTML, which mixes formatting with content, XML focuses purely on the structure of the information. This separation is crucial for applications that need to interpret the data itself rather than just display it. For example, a program searching multiple online bookstores for the lowest price on a book needs a consistent and understandable data structure to locate titles and prices effectively. XML allows the definition of custom tags, enabling highly specific data representations such as book titles, authors, and publication years. These tags can also be nested and subdivided, allowing for complex, hierarchical data structures. Although XML is more rigid and syntactically strict than HTML—which historically was often written with minor errors—this strictness results in cleaner, more machine-readable content. XML files must close tags properly, respect case sensitivity, and quote attribute values, all of which help in error-free parsing and consistent interpretation across different systems.

In addition to XML, a subset of HTML known as XHTML has been developed to bring the rigor of XML to web page design. XHTML pages must conform strictly to XML rules, which makes them more reliable for applications that need consistent processing. Although the adoption of XHTML has been gradual due to legacy habits and uneven browser support, newer specifications like HTML5 aim to bridge the gap, allowing pages to be interpreted as either HTML or XHTML depending on context. XML has also proven invaluable for enabling interoperability between programs, especially in the context of web services. These services use HTTP as a transport protocol and rely on XML-formatted messages to facilitate communication between heterogeneous applications. For example, SOAP (Simple Object Access Protocol) allows remote procedure calls between programs regardless of language or platform, by structuring requests and responses in XML and transmitting them via HTTP. This makes it possible for systems running in different environments to interact reliably and efficiently.

A crucial characteristic of AJAX-powered applications is their use of asynchronous communication. This means that scripts can send or receive data from the server without halting the user's interaction with the page. For instance, if a user scrolls through a web-based map, the script might detect that the user is approaching the edge of the currently loaded map data. Without freezing the interface, it can request additional data from the server, allowing the map to scroll smoothly while the new content loads in the background. When the requested data arrives, it can be incorporated into the DOM, and the display updates automatically—giving users the impression of a continuous, responsive application. This kind of non-blocking input/output is critical for maintaining smooth and interactive user experiences and is a key reason why AJAX-based applications feel more like desktop software than traditional web pages.

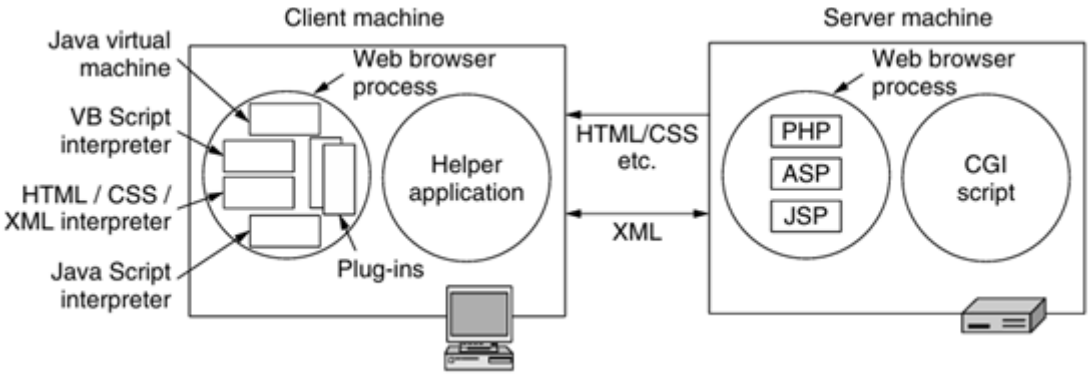


Figure 20.9 Various technologies used to generate dynamic pages

To bring all these technologies together, developers use scripting languages—chiefly JavaScript. JavaScript is a powerful and versatile language that enables interaction with all aspects of a web page, including the DOM, HTML, CSS, and server communication via HTTP. Despite its occasionally quirky syntax, JavaScript is a fully-featured programming language, capable of complex logic and operations equivalent to what can be done in languages like Java or C. It supports variables, arrays, objects, functions, and control structures, making it suitable for building complete applications within the browser. JavaScript can respond to user actions, modify page content, and even track cursor movements, which opens the door for rich interactions such as dynamic menus, interactive graphics, and complex form validation. Moreover, its built-in support for asynchronous HTTP requests—via technologies like XMLHttpRequest or the more modern fetch API—enables seamless data exchange without page reloads, which is the backbone of AJAX.

Dynamic content generation is not limited to the client side. Server-side scripting also plays a significant role in constructing web applications. Web pages can be generated dynamically on the server using languages and platforms such as PHP, JSP (Java Server Pages), ASP.NET, or CGI scripts written in a wide range of programming languages. These server-side scripts process user input, query databases, and generate HTML pages that are sent back to the client. Once the dynamic pages are delivered to the browser, they are treated like any other web content and rendered accordingly. Browsers can be extended through plug-ins and helper applications to handle a variety of content types. Simultaneously, scripts embedded in the delivered HTML—written in languages like JavaScript, VBScript, or Java—can perform complex tasks on the client side, including interface updates and communication with the server. With AJAX, these embedded scripts can continuously exchange data with the server, often using XML or other lightweight formats like JSON, resulting in web applications that not only look and feel like traditional software but also provide the added benefit of being accessible through any modern browser with an internet connection. This powerful model underpins the modern Web and continues to evolve, enabling developers to create increasingly sophisticated and user-friendly applications.

20.5 HTTP- THE HYPERTEXT TRANSFER PROTOCOL

HTTP, or HyperText Transfer Protocol, is the fundamental protocol used to transmit web content between servers and clients. Defined in RFC 2616, HTTP functions as a simple, text-based request-response protocol that typically operates over the Transmission Control Protocol (TCP). In its most basic form, a client—usually a web browser—sends a request to a server for a resource, and the server returns a response containing the requested content or an error message. Both the request and response messages are composed using ASCII text, with headers providing metadata about the transaction and the content itself formatted similarly to MIME (Multipurpose Internet Mail Extensions). This simplicity and use of human-readable text were major contributors to the early success of the World Wide Web, as they made it easy for developers to build and debug web applications.

As the internet has evolved, so too has the role of HTTP. While it is technically classified as an application-layer protocol—because it operates above TCP and is most closely associated with web browsing—it has begun to take on characteristics that resemble those of a transport protocol. This evolution reflects its widespread adoption as a general-purpose communication protocol for a variety of applications that extend far beyond traditional web browsing. For instance, HTTP is now commonly used by media players to request album information from servers, by antivirus software to download definition updates, and by developers to retrieve project files from online repositories. The use of HTTP by consumer electronic devices, such as digital photo frames that embed lightweight HTTP servers, demonstrates its adaptability and growing pervasiveness.

Machine-to-machine communication is also increasingly being conducted over HTTP, further cementing its role as a versatile conduit for data exchange across the internet. A practical example of this is seen in service integration scenarios, where an airline server might use SOAP—a protocol that transmits XML-formatted data over HTTP—to interact with a car rental company's server to reserve a car as part of a broader travel package. These behind-the-scenes interactions are transparent to the end user but rely heavily on HTTP's ability to carry structured data reliably and universally. This trend illustrates how HTTP has grown

beyond its original scope, becoming a backbone not just for human-facing web pages, but for automated, cross-service communication across the internet. Its ongoing adoption in both consumer and industrial applications suggests that its role will only continue to expand in the future.

Connections

When a browser wants to retrieve information from a web server, it typically initiates a TCP connection to the server, usually targeting port 80, the standard port for HTTP. Though not a mandatory rule, this approach is widely adopted because TCP ensures reliable delivery, handles message fragmentation and reassembly, and manages congestion control—meaning that neither the browser (client) nor the server needs to worry about the complexities of low-level data transmission. In the early days of the Web, HTTP/1.0 operated under a simple model: once the TCP connection was established, the client would send a single HTTP request, receive a single response, and then immediately close the connection. While this approach sufficed when most web pages were small and primarily consisted of simple HTML text, it quickly became inefficient as web content grew more complex. Modern web pages often contain numerous embedded resources like images, scripts, fonts, and stylesheets. Opening a new TCP connection for each of these elements created substantial overhead, drastically affecting performance and scalability.

This inefficiency led to the development of HTTP/1.1, which introduced persistent connections—a major enhancement to the protocol. With persistent connections, a single TCP connection can be reused for multiple HTTP requests and responses. This approach eliminates the need to repeatedly set up and tear down connections, thereby reducing latency and lowering the overhead associated with TCP handshakes. Additionally, connection reuse enables TCP congestion control to operate more effectively. When a TCP connection is reused, the slow-start mechanism—which gradually increases data transfer rates as the path becomes better understood—has a longer duration to build throughput. In contrast, repeated short-lived connections never stay active long enough to reach optimal transmission speed. Persistent connections also support HTTP pipelining, where multiple requests can be sent in quick succession without waiting for prior responses. This reduces idle time on the server side and improves overall page load performance.

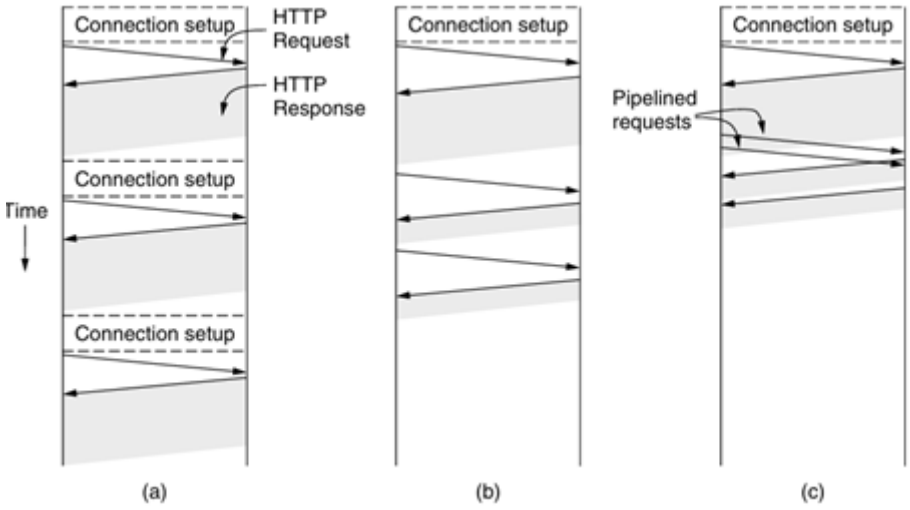


Figure 20.10 HTTP with (a)multiple connections and sequential requests (b) A persistent connection and sequential requests (c) A persistent connection and pipelined requests

To illustrate this evolution, three usage patterns are typically compared. In the first scenario, under HTTP/1.0, each resource (such as the main HTML page and each image it references) is retrieved using a separate TCP connection. This results in repeated connection setup delays and slow data transfers due to the TCP warmup process. In the second scenario, HTTP/1.1 allows for a persistent connection, where all resources are fetched sequentially over the same connection. This avoids repeated setup times and allows TCP to optimize throughput. In the third and most efficient case, pipelining is used with a persistent connection. Here, the browser sends multiple requests as soon as it knows what resources are needed—without waiting for earlier responses—allowing the server to begin processing them earlier and reducing total load time.

However, persistent connections also introduce new challenges. The main question is: when should a connection be closed? If the server closes it too soon, the client might have to reopen it to make subsequent requests. If it remains open too long, server resources might be unnecessarily consumed, especially if many clients are idle. In practice, browsers and servers implement timeouts, typically keeping connections alive for a brief idle period (e.g., 60 seconds). If no further requests are received during this window, the connection is closed to free up resources. This balances user experience with system efficiency.

Another approach once used to improve performance was parallel connections—opening multiple TCP connections simultaneously to the same server and fetching resources in parallel, even if each connection handled only one request. While this method does hide some latency and can improve page load speed, it introduces serious network inefficiencies. Since TCP congestion control operates independently for each connection, these parallel streams compete with each other and can lead to increased packet loss and network congestion. This not only wastes bandwidth but can also degrade overall network stability. For these reasons, persistent connections with pipelining are now preferred over parallel connections, as they offer better performance while being more network-friendly and resource-efficient.

In conclusion, HTTP's transition from one-request-per-connection in HTTP/1.0 to persistent connections with pipelining in HTTP/1.1 marks a major evolution in how web communication is handled. By minimizing overhead, improving TCP performance, and optimizing server response time, persistent connections have significantly enhanced the efficiency and speed of web browsing—especially as the complexity and size of web content have grown.

Methods

HTTP, or HyperText Transfer Protocol, though originally intended for delivering simple web pages, was deliberately designed with general-purpose capabilities in mind to allow future extensibility and support object-oriented applications. Its flexibility has enabled a wide range of applications beyond traditional web browsing, such as communication between services and embedded systems. HTTP uses operations known as methods, which define the kind of action a client wants to perform on a given resource.

Method	Description
GET	Read a Web page
HEAD	Read a Web page's header
POST	Append to a Web page
PUT	Store a Web page
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Connect through a proxy
OPTIONS	Query options for a page

Figure 20.11 The built-in HTTP requests methods

Each HTTP request begins with a method name in uppercase, followed by other request details, all composed in readable ASCII text. The most common method is GET, which asks the server to return the contents of a specified page. This is used in nearly every browser interaction, from accessing websites to retrieving assets. In practice, the format of the request is typically written as GET /filename HTTP/1.1, where the filename is the resource path and 1.1 represents the protocol version.

Another method, HEAD, is similar to GET but requests only the metadata (i.e., headers) of a page, without the actual content. It is useful for validating URLs or gathering information for search engine indexing. POST is a method typically used when a form is submitted from a browser. Instead of simply retrieving data, POST sends data (such as form entries or parameters) to the server, which then processes it and responds with a result page. This method is also commonly used in SOAP-based web services, where it helps invoke procedures remotely by passing structured data. PUT is used to upload or modify a web page on the server; it essentially stores data at the specified resource path. This method can be helpful in collaborative editing or content publishing environments. However, because it writes to the server, it often requires authentication headers to verify that the client is authorized to perform the action.

The DELETE method, as the name suggests, is used to remove a page or resource from the server. It too generally requires appropriate authentication. TRACE is primarily used for diagnostic purposes. When invoked, it causes the server to return the exact request it received. This feedback helps developers identify how their HTTP requests are interpreted, especially when intermediaries like proxies are involved. CONNECT is meant to establish a tunnel to a remote server through an intermediary, typically used to enable secure HTTPS connections through a proxy server. OPTIONS allows a client to inquire about the communication options available for a resource, such as which methods and headers it supports, thereby allowing clients to adjust their behavior dynamically based on server capabilities.

Code	Meaning	Examples
1xx	Information	100 = server agrees to handle client's request
2xx	Success	200 = request succeeded; 204 = no content present
3xx	Redirection	301 = page moved; 304 = cached page still valid
4xx	Client error	403 = forbidden page; 404 = page not found
5xx	Server error	500 = internal server error; 503 = try again later

Figures 20.12 The status code response groups

Every HTTP request receives a response from the server, which includes a status line followed by optional content such as headers or a page body. The status line includes a three-digit code that indicates whether the request was successful or if some issue occurred. These codes are divided into five categories. The 1xx class provides informational responses but is rarely used in real-world applications. The 2xx class indicates success, with code 200 meaning the request was successful and the content is returned, and 204 meaning the request succeeded but there is no content to deliver. The 3xx class covers redirection, guiding the client to a different location for the requested resource; for example, 301 means the resource has permanently moved, while 304 indicates that the resource in the cache is still valid and does not need to be re-fetched. The 4xx class denotes client errors; for instance, 403 indicates that access is forbidden and 404 signifies that the requested page does not exist. Finally, the 5xx class identifies server-side problems. These include situations where the server has encountered an unexpected condition, such as code 500 for internal server errors or 503, which suggests the server is temporarily unavailable, typically due to overload or maintenance. These structured response codes provide a robust mechanism for diagnosing issues and guiding client behavior across a wide variety of web applications and services.

Message Headers

In the process of communication between web clients and servers, HTTP message headers play a crucial role by supplying additional information about the request or response, often in the form of metadata. When a client makes a request, the request line can be followed by a number of headers that detail the client's capabilities, preferences, or intentions. Similarly, when a server responds, it can include headers that describe the nature of the content being returned, how it should be processed, or provide instructions related to caching and access. These headers resemble the parameters in a procedure call, supplying important context that determines how the message is to be interpreted or acted upon.

One common header included in client requests is the User-Agent, which identifies the browser type and platform the client is using. This allows the server to tailor its responses appropriately since different browsers may support different features. In cases where the client has limitations or preferences, several Accept headers are used. These include headers for specifying acceptable MIME types (Accept), character sets (Accept-Charset), encodings (Accept-Encoding), and natural languages (Accept-Language). These headers help the server decide the most suitable version of a resource to serve back. In situations where a client already has a cached copy of a page, the If-Modified-Since and If-None-Match headers can be used to request the resource only if it has changed, thus saving bandwidth and improving efficiency.

The Host header, which is mandatory in HTTP/1.1, tells the server which hostname the client wants to reach. This is essential when multiple domain names are served from the same IP address, a practice known as virtual hosting. If the requested resource requires authentication, the Authorization header is used to pass credentials. Additionally, the Referer header (intentionally misspelled due to a historical typo) indicates the URL of the page that led the client to make the current request, often helping servers analyze browsing paths or apply security measures.

Cookies are another critical feature managed through headers. Servers use the Set-Cookie header to send a cookie to the client, which the client must store and return in future requests using the Cookie header. These headers are used for session management, tracking user activity, and personalizing content. Though newer standards for cookies exist, such as RFC 2965, these are not widely adopted, and the original RFC 2109 format remains dominant.

On the server's side, response headers provide detailed information about the resource. The Server header identifies the server software. The various Content- headers—such as Content-Encoding, Content-Language, Content-Length, Content-Type, and Content-Range—describe the format, language, size, and encoding of the content being sent. This helps the client understand how to display or process the content correctly. The Last-Modified and Expires headers are particularly important for caching mechanisms, indicating when the page was last changed and when it will become stale.

The Location header is used when the server wants to redirect the client to a different URL, either because the resource has moved or to route users to localized versions of a site. For handling large files, the Accept-Ranges header tells clients that the server supports partial content delivery, enabling them to fetch resources in chunks, which is useful for streaming media or resuming downloads.

Some headers can appear in both requests and responses. The Date header provides a timestamp indicating when the message was sent. The Range header specifies which part of a resource is being requested or returned. The ETag (Entity Tag) header is another tool for caching, providing a unique identifier for a specific version of a resource so clients can check whether their cached version is up to date. The Cache-Control header gives instructions on how resources should be cached by clients and intermediary caches. Lastly, the Upgrade header allows clients and servers to negotiate a protocol change, enabling transitions to newer versions of HTTP or to secure protocols like HTTPS. All these headers collectively ensure that the HTTP protocol remains flexible, extensible, and capable of supporting the complex and varied needs of the modern web.

CACHING

Caching plays a fundamental role in improving the efficiency and performance of web browsing by minimizing redundant data transfers and reducing the load on both the network and servers. When users frequently revisit websites or navigate through related pages within a site, many of the embedded resources—such as images, stylesheets, and scripts—remain unchanged. Retrieving these elements afresh with each request would be inefficient and wasteful, especially considering that the browser already holds a copy from previous visits. This concept of storing and reusing previously fetched resources is referred to as caching. The principal benefit of caching is that it can significantly decrease the time it takes for web pages to load by eliminating the need to re-download content that hasn’t changed. This not only lowers network traffic but also reduces latency, offering a faster user experience. Given the low cost and ample availability of local storage, typically in the form of disk space, this trade-off—allocating space to store web content—is almost always advantageous.

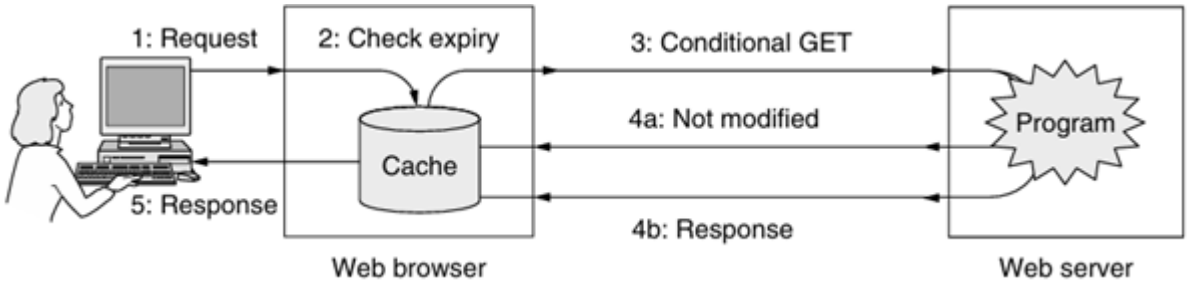


Figure 20.13 HTTP caching

However, caching introduces a complex challenge: determining whether a previously stored version of a resource is still valid and identical to the version that would be retrieved from the server if requested again. This is particularly difficult because a URL alone is not a reliable indicator of a page's freshness. For instance, a single URL may always refer to the latest news article, which is updated frequently, or it may point to a static page like a list of mythological figures, which changes infrequently or not at all. To address this challenge, HTTP employs two primary strategies. The first is page validation. In this strategy, when a client receives a request for a page, it first checks its local cache to see if it has a copy that is known to still be valid. If so, it can immediately serve that cached page without contacting the server. This determination is made using headers like Expires, which indicate when the page is expected to become stale. If the current date and time precede the Expires value, the cached copy is reused. Unfortunately, not all web pages are accompanied by an Expires header because predicting the future accuracy of content is inherently uncertain. In such cases, browsers often apply heuristics—rules of thumb—to decide whether a page is likely to remain unchanged for a short period. For example, if a page hasn't been modified in over a year, it's probably safe to assume it won't change in the next few minutes or hours. But heuristics aren't foolproof; some content may change rapidly despite having long periods of apparent stability, like a stock market page that only updates when the market is open.

When validation based on expiry fails or is inconclusive, HTTP resorts to the second strategy: conditional requests. Here, the client sends a conditional GET request to the server, asking whether the cached version is still up to date. This is accomplished using headers like If-Modified-Since, which passes the timestamp from the Last-Modified header of the cached page. If the server determines that the content has not changed since that date, it responds with a short confirmation rather than the full content, saving bandwidth. Another mechanism used is entity tags, or ETags, which act like unique fingerprints for a specific version of a page. These can be cryptographic hashes or other identifiers that change whenever the content changes. The client includes these tags in the If-None-Match header of its request. If the server finds that the ETag still matches the current version of the resource, it allows the cached copy to be used. This method is particularly useful when freshness cannot be determined based solely on modification dates—for instance, when the server customizes the content based on language or content type preferences.

Caching behavior can be explicitly controlled using the Cache-Control header. This header allows servers to define rules about how their responses should be cached. For instance, sensitive or frequently updated pages can be marked as no-cache, indicating that they should not be stored. Dynamic content or pages requiring user authentication typically use such directives to prevent outdated or unauthorized data from being inadvertently reused.

Furthermore, it's important to recognize that caching isn't limited to just browsers. Proxy caches, which are intermediary servers that cache content for multiple users, play a significant role in reducing the overall demand on origin servers. Organizations like Internet Service Providers (ISPs) and large enterprises commonly deploy proxy caches to serve frequently accessed content more efficiently across users, further optimizing network performance.

Nevertheless, caching—despite all its advantages—has limitations. While it greatly improves performance for popular and frequently accessed resources, it is less effective for the vast number of less commonly accessed items. This phenomenon, often referred to as the "long tail," describes the distribution where a small number of web documents receive a large portion of the traffic, while the majority receive very little. Many of these unpopular documents, such as long videos or obscure articles, take up significant cache space without being frequently requested. As a result, even with large caches, the hit rate—the proportion of requests that can be fulfilled from the cache—does not increase proportionally with cache size. Research has shown that caches typically handle fewer than half of all requests. Thus, while HTTP caching is a powerful and essential mechanism for enhancing web performance, it is not a panacea and must be complemented by other techniques like content distribution networks to achieve optimal results across the global web.

Experimenting With HTTP

Since HTTP is an ASCII-based protocol, it is quite straightforward for a person to interact with web servers directly from a terminal, without needing a web browser. This is possible because HTTP messages are simply plain-text commands and headers that follow a predictable format. All a user needs is a TCP connection to the server, typically on port 80, which is the default port for HTTP communication.

To experiment with this, one can use the telnet command in a UNIX shell or a Windows command window. For example, by typing telnet www.ietf.org 80, the user initiates a TCP connection to the IETF's web server. Once connected, the user can type an HTTP request manually, such as GET /rfc.html HTTP/1.1, which asks the server to send back the content at the specified path. This must be followed by a Host header, like Host: www.ietf.org, to tell the server which domain the request is intended for, especially when multiple websites are hosted on the same server.

After entering these lines, the user must add a blank line to indicate the end of the HTTP request. This lets the server know that the message is complete and that it should now respond. Depending on the specific server and the requested content, a variety of headers and webpage data will be returned. This process illustrates how transparent and accessible HTTP is, allowing direct interaction with servers for testing or educational purposes.

20.6 THE MOBILE WEB

Web browsing from mobile phones has become increasingly common due to the convenience of accessing information while on the move. However, it also introduces technical challenges because most traditional web content was originally designed for large desktop screens, powerful processors, and broadband connectivity. Mobile devices such as smartphones, in contrast, have relatively small screens, limited input capabilities, lower network bandwidth—especially on older 3G cellular networks—intermittent connectivity, and constrained

computing power due to factors like battery life and device size. As a result, directly displaying desktop-designed websites on mobile devices often results in a frustrating user experience.

Initially, a completely different protocol stack was developed to cater to mobile devices, with WAP (Wireless Application Protocol) being the most notable example. WAP was introduced in 1997 by major mobile phone manufacturers like Nokia, Ericsson, and Motorola. It aimed to provide a lightweight protocol suite that matched the limitations of early mobile devices. However, technological advancements over the following decade brought significant improvements in mobile hardware and networks. With the rollout of 3G services, more powerful processors, high-resolution color displays, and Wi-Fi capabilities became standard in many mobile phones. These improvements reduced the necessity for a completely separate mobile web infrastructure and allowed mobile devices to begin using standard web browsers. Given this progress, the modern approach has shifted toward using the same web protocols for both mobile and desktop environments. Websites now typically serve device-specific content by detecting the type of device making the request. This detection is often done through HTTP request headers, particularly the User-Agent header, which identifies the client's browser and platform. Using this information, web servers can deliver optimized versions of pages tailored to mobile devices—for instance, with smaller images, simplified layouts, and streamlined navigation—while providing full-featured pages to desktops and laptops. This strategy offers a seamless experience without requiring separate technologies for different devices.

To promote this unified approach, the World Wide Web Consortium (W3C) has issued guidelines and best practices for mobile web content. These practices focus on reducing page size through measures like image compression and effective caching because the cost of data transmission is typically higher than computation. W3C's guidelines encourage large websites to create mobile-specific versions of their pages to accommodate users browsing on phones. Additionally, W3C introduced a recognizable logo for mobile-friendly websites to help users identify pages that perform well on mobile devices.

Another important development is XHTML Basic, a simplified version of the standard XHTML language designed specifically for constrained devices like mobile phones, televisions, game consoles, and even watches. XHTML Basic includes a subset of standard HTML tags, organized into eleven modular groups such as structure, text, hyperlinks, lists, forms, and images. Some of these modules are required (e.g., structure, text, and hyperlinks), while others are optional. Because XHTML Basic omits style sheets, scripts, and frames, it is easier for low-powered devices to render the pages efficiently, yet still supports meaningful web content.

Despite improvements in mobile browser support and XHTML Basic, not all websites are designed with mobile users in mind. To address this, another strategy called content transformation or transcoding is used. This involves placing an intermediary system between the mobile device and the web server. This intermediary takes the content returned from the server and transforms it into a more mobile-friendly format before delivering it to the user. For instance, large images might be reformatted into smaller, lower-resolution versions. Other transformations might include simplifying page layout or converting complex elements into more basic ones. Transcoding has been helpful in making traditional web content

accessible on mobile devices, but it can sometimes conflict with mobile-specific content already being served by the original website, leading to redundant or contradictory optimizations.

Although the main challenges of mobile web development are content-related, the underlying network protocols can also present inefficiencies. Standard protocols like HTTP, TCP, and IP tend to include a significant amount of overhead in the form of headers, which can consume precious bandwidth—especially over cellular or other low-bandwidth networks. In the early mobile web era, special-purpose protocols were introduced to reduce these overheads, but such measures are now less necessary. Instead, header compression technologies such as ROHC (Robust Header Compression) have emerged as effective solutions. These technologies significantly reduce the size of protocol headers, allowing standard protocols to be used efficiently over both high-speed and low-speed links. By enabling header compression, developers can maintain a single set of protocols—HTTP, TCP, and IP—while still meeting the performance needs of both mobile and desktop users.

20.7 WEB SEARCH

The development of web search stands out as one of the most transformative successes in the history of the World Wide Web. Initially pioneered in 1998 by Sergey Brin and Larry Page while they were graduate students at Stanford University, Google revolutionized the concept of search by introducing a fundamentally different ranking algorithm. Unlike earlier methods that simply counted keyword matches, Google's approach assessed a page's importance based on how many other pages linked to it. This methodology more accurately reflected a page's relevance and usefulness, since popular and trusted pages naturally attracted more inbound links. For instance, the official Cisco homepage is inherently more authoritative for a query like "Cisco" than a lesser-known page that simply repeats the keyword multiple times. This novel approach proved highly effective, leading to a surge in Google's popularity and growth. The company quickly attracted venture capital, became publicly traded in 2004 with a substantial market valuation, and by 2010 had established a vast infrastructure of over one million servers spread across data centers worldwide.

Web search may seem like a straightforward web application, but its influence extends far beyond that. As one of the most widely used and mature applications on the internet, search engines handle over a billion queries daily. For many users, search is the gateway to discovering information, products, services, or places that they otherwise wouldn't know how to find. For example, someone looking to buy Vegemite in Seattle might not know where to start—but a quick search often provides an accurate and fast result. The typical search process involves directing a browser to a search engine's website, such as Google, Yahoo!, or Bing, entering a query through a form, and receiving a dynamic results page with relevant links. From this point, the user can explore any of the listed pages. While this process seems simple on the surface, it is supported by a massive and complex infrastructure of crawling, indexing, and data analysis.

At the heart of every search engine lies the mechanism of web crawling. To answer user queries, a search engine must first maintain a database of web pages. Since most pages link to others, theoretically all public pages on the web can be discovered through recursive traversal of links—a process that crawlers perform continually. However, challenges arise with dynamic content. Many modern websites are generated on-the-fly from databases in response to user input. This makes them inaccessible to crawlers that rely on static links. Such content

is referred to as the "deep web," a vast expanse of data that remains largely invisible to traditional search engines. Researchers are actively exploring ways to access this hidden portion of the web. Furthermore, websites can use a special file called robots.txt to instruct crawlers which parts of the site to index and which to ignore, offering site owners some control over how their content is accessed.

Beyond crawling, the storage and processing of the web's content is a monumental task. The scale of this data is immense—estimated to include tens of billions of pages, each averaging about 320 KB in size. The resulting index requires roughly 20 petabytes of storage space. Although this is a massive volume, it remains manageable for tech giants like Google and Microsoft due to declining storage costs and the availability of large-scale data centers. For instance, even at a cost of $20 per terabyte, storing the entire web would cost under half a million dollars—a minor investment for such companies. Nevertheless, managing this data involves more than just storage. Understanding and interpreting the data presents even greater challenges. While structured data formats like XML can assist with organizing information, much of the web consists of unstructured or semi-structured content. This makes it difficult for algorithms to extract meaning, especially when dealing with varied formats and languages. The ultimate goal is to develop search systems that don’t just locate information, but interpret it—such as identifying where to buy a high-quality, budget-friendly toaster oven nearby.

Search engines have also evolved into powerful tools for simplifying how people navigate the internet. Instead of memorizing lengthy URLs, users can now simply type in the name of a person, company, or topic and be guided to the appropriate page. In this way, search engines are beginning to replace URLs in the same way that DNS replaced IP addresses—providing a more human-friendly naming system. Moreover, search engines help correct user errors. If a user makes a spelling mistake or typo in their query, the search engine often still provides the correct result. This user-friendliness has made search engines indispensable tools in daily digital life.

Perhaps most importantly from a business standpoint, web search has become a significant driver of revenue through targeted advertising. Unlike traditional print advertising, online ads can be tailored to individual users based on their search queries. This makes advertisements more relevant and effective, thereby increasing their value to advertisers. Most major search engines use auction-based models to determine which ads appear alongside which searches, maximizing profit while ensuring user relevance. However, this system has also given rise to issues such as click fraud, where automated scripts mimic human behavior to generate ad revenue illegitimately. Despite such challenges, advertising remains the financial engine powering the continuous growth and innovation in web search technology.

20.8 SUMMARY

This chapter covers the World Wide Web and multimedia applications. The WWW is based on a client-server architecture, delivering static and dynamic web documents using the HTTP protocol. Performance improvements include caching, compression, and the wireless web for mobile access. Multimedia networking introduces digital audio and video, compression techniques, and applications like streaming audio, Internet radio, VoIP, and video on demand, enabling rich, interactive, and real-time content delivery over networks.

20.9 TECHNICAL TERMS

World Wide Web, HTTP, multimedia, audio streaming, MIME, VoIP

20.10 SELF ASSESSMENT QUESTIONS

Essay questions:

1. Explain the architecture and working of the World Wide Web.
2. Describe static versus dynamic web documents and their applications.
3. Discuss the HTTP protocol and performance enhancements for the web.
4. Explain digital audio and video concepts, including compression and streaming.
5. Describe multimedia applications such as Internet radio, VoIP, and video on demand.

Short Questions:

1. What is the architecture of the World Wide Web?
2. Define static and dynamic web documents.
3. What is HTTP?
4. Name one audio and one video compression technique.
5. What is the purpose of VoIP?

20.11 FURTHER READINGS

1. Andrew S. Tanenbaum, “Computer Networks”, Fourth Edition, PHI.
2. James F.Kurose, Keith W.Ross, “Computer Networking”, Third Edition, Pearson Education
3. Behrouz A Forouzan, “Data Communications and Networking”, Fourth Edition, TMH (2007)
4. Michael A. Gallo, William M. Hancock, “Computer Communications and NetworkingTechnologies”, Cengage Learning (2008)

Mrs. Appikatla Pushpa Latha