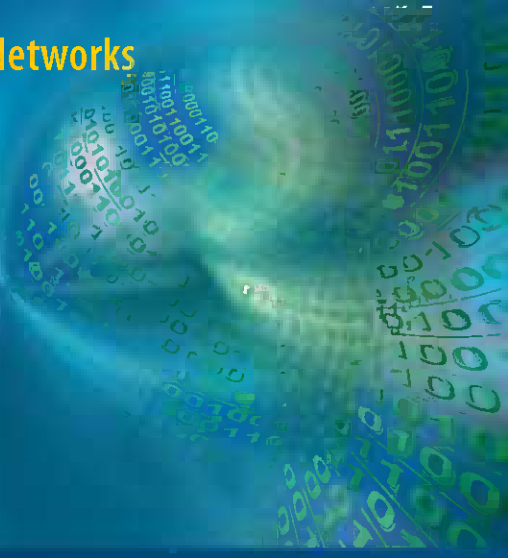
Computer Communications and Networks



Bogdan Ciubotaru
Gabriel-Miro Muntean

Advanced Network Programming – Principles and Techniques

Network Application Programming with Java

Springer

Computer Communications and Networks

For further volumes:
www.springer.com/series/4198

The Computer Communications and Networks series is a range of textbooks, monographs and handbooks. It sets out to provide students, researchers and non-specialists alike with a sure grounding in current knowledge, together with comprehensible access to the latest developments in computer communications and networking.

Emphasis is placed on clear and explanatory styles that support a tutorial approach, so that even the most complex of topics is presented in a lucid and intelligible manner.

Bogdan Ciubotaru • Gabriel-Miro Muntean

Advanced Network Programming – Principles and Techniques

Network Application Programming with Java

Springer

Bogdan Ciubotaru
School of Electronic Engineering
Dublin City University
Dublin, Ireland

Gabriel-Miro Muntean
School of Electronic Engineering
Dublin City University
Dublin, Ireland

Series Editor
A.J. Sammes
Centre for Forensic Computing
Cranfield University
Shrivenham campus
Swindon, UK

ISSN 1617-7975 Computer Communications and Networks
ISBN 978-1-4471-5291-0 ISBN 978-1-4471-5292-7 (eBook)
DOI 10.1007/978-1-4471-5292-7
Springer London Heidelberg New York Dordrecht

Library of Congress Control Number: 2013944962

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

***Bogdan Ciubotaru**:*
*This book is dedicated to my wonderful daughter **Ilinca-Meda** and my lovely wife **Madalina** who have supported me throughout this effort, encouraged me, and blessed me with their love.*

***Gabriel-Miro Muntean**:*
*This book is dedicated to my wonderful children **Daniel-Sasha** and **Alexandra-Nadia** who are smart, playful and happy, and make me feel very proud being their father, to my parents **Dora-Aurelia** and **Ivo** who gave me the most important gifts of wisdom and knowledge and are always encouraging me, and last, but not least, to my lovely wife **Cristina**, a true life partner of mine.*
Thank you very much!

Preface

This book on Advanced Network Programming Principles and Techniques covers in detail network architectures, including the latest wireless heterogeneous networks, communication protocol models, and protocols and support for communication-based services. Network programming techniques are introduced in this book, including server-side and client-side programming solutions, advanced client–server communication models (i.e., socket-based, Remote Method Invocation, applet–servlet communication), network-based data storage, and multimedia transfer.

Advanced Network Programming Principles and Techniques is a useful asset for any reader interested in computer networking whether they are interested in understanding the underlying architectures and paradigms or are application developers looking for useful examples to build communication-based programs. Additionally, this book is an excellent companion to any network programming module taught at the third level institutions worldwide.

To all the readers of this book, the authors hope it will be of great help and wish them "happy reading".

Dublin
Ireland
March 2013

Bogdan Ciubotaru
Gabriel-Miro Muntean

Acknowledgements

Many thanks to Irina Tal and Cristina Muntean who have extensively contributed with their comments which helped make this book better.

Authors

Bogdan Ciubotaru received his Ph.D. degree from Dublin City University, Ireland in 2011 for research in the area of quality-oriented mobility management for multimedia applications and B.Eng. and M.Sc. degrees from "Politehnica" University of Timisoara, Romania in 2004 and 2005, respectively. Dr. Bogdan Ciubotaru was an IRC Postdoctoral research fellow with the Performance Engineering Laboratory, School of Electronic Engineering, Dublin City University (DCU), Ireland. Currently he is with Everseen Ltd, Ireland. His research interests include wireless mobile networks, multimedia streaming over wireless access networks as well as wireless sensor networks and embedded systems. He is a member of IEEE and ACM Institute, Ireland.

Gabriel-Miro Muntean received his Ph.D. degree from Dublin City University (DCU), Ireland in 2003 for research in the area of quality-oriented adaptive multimedia streaming and B.Eng. and M.Eng. degrees from "Politehnica" University of Timisoara, Romania in 1996 and 1997, respectively. He is Senior Lecturer with the School of Electronic Engineering at Dublin City University, Ireland, co-Director of the DCU Performance Engineering Laboratory, Director of the Network Innovations Centre, RINCE Institute, Ireland, and Consultant Professor with Beijing University of Posts and Telecommunications, China. His research interests include quality-oriented and performance-related issues of adaptive multimedia delivery, performance of wired and wireless communications, energy-aware networking and personalised e-learning. Dr. Gabriel-Miro Muntean has published over 180 papers in prestigious international journals and conferences, has authored two other books and 12 book chapters and has edited four other books. Dr. Muntean is an Associate Editor of the IEEE Transactions on Broadcasting, Associate Editor of the IEEE Communications Surveys and Tutorials, and reviewer for other important international journals, conferences and funding agencies. He is a member of ACM, ACM SIGMOBILE, IEEE, and IEEE Broadcast Technology Society.

Contents

Chapter 1
Introduction

Abstract Currently, computer networking has already become ubiquitous, the number of diverse devices is increasing constantly, as are also their capabilities, the range of applications and network-based services is expanding, and user expectations are rapidly evolving. This is the context in which the authors set the scene for this network programming book in its introductory chapter.

The past decades have seen an unprecedented evolution in computer networks. If originally a network has interconnected few computers in a research lab and then has linked computing machines across several university campuses, nowadays the Internet interconnects network devices worldwide. In the developed world, wired broadband Internet access is available in most homes and office buildings and diverse wireless broadband and cellular network technologies enable network access anywhere and anytime, in private and public places alike. Although lagging behind in developing countries or rural areas, network connectivity is becoming available in wireless forms (terrestrial or satellite) to an increasing population, even in the most remote places.

Due to the wide availability of the Internet access, both the range and popularity of communicating network applications has increased dramatically. Applications such as simple Web browsing or file transfer, although still used today, have been shadowed by the increasingly popular rich-media-based applications, ranging from video conferencing to video on demand, IP television, and online gaming.

Services such as electronic mail, online data storage, virtual servers, and workstations, as well as a wide range of utility and entertainment applications, are also growing in popularity among the Internet users.

Furthermore, mobile and hand-held devices are becoming increasingly capable both in terms of computational power and communication capabilities. Smartphones and light portable PCs such as netbooks are highly attractive to all users, including very young ones. As these devices are usually equipped with multiple technology wireless interfaces, they can easily communicate over the Internet, opening the door for a wide range of applications.

This book approaches the very active field of computer networks and network application programming. This field is extremely vast from both theoretical and practical points of view. The amount of information available to a reader willing to

B. Ciubotaru, G.-M. Muntean, *Advanced Network Programming – Principles and Techniques*, Computer Communications and Networks,
DOI 10.1007/978-1-4471-5292-7_1, © Springer-Verlag London 2013

explore this field of computer networks and network programming is overwhelming and any help in filtering or organizing the information is highly useful.

This is the context in which this book proposes a novel practical approach in which the reader is introduced gradually to basic and more advanced computer networking concepts. Side-by-side there are theoretical descriptions of these concepts and practical examples and step-by-step discussions.

An extensive and comprehensive set of practical code examples are presented with detailed comments and explanations. The reader benefits from a well organized approach to teaching computer network concepts and network programming techniques which is useful for both readers with a more theoretical interest and readers mostly interested in practical aspects.

The authors have a vast research and development experience in the area of wired and wireless networking. They have been involved in various research projects in the area of wired and wireless networks with focus from low power wireless sensor networks to high performance state-of-the-art wireless heterogeneous environments. The authors have almost 200 top international publications, including books, book chapters, and journal and conference papers addressing various aspects of networking starting from low layer protocol design to high layer application development. They have also been involved in application development projects using both wireless and wired network infrastructure for communication.

Noteworthy is that the authors are teaching various courses in the area of computer networks to both undergraduate and postgraduate students. They have designed this book in order to act as a significant reference to network programming modules taught at their university, and also at other third level institutions worldwide.

Advanced Network Programming Principles and Techniques introduce you to the most up-to-date network architectures, protocols, and paradigms, as well as network programming techniques. This book discusses basic and advanced principles of computer networking, including architectures, communication protocols, and network programming techniques and models. The code examples are extremely useful for understanding the practical aspects of computer networking and of communication services offered by various operating systems, and for learning how to develop network-based applications.

Chapter 2
Network Architectures

Abstract The networks have evolved significantly since the first network architecture has been proposed. Lately, the architecture is seen more as a framework which specifies not only the network topology, network type, network components, and their functionality, but also presents data communication protocols, data formats used, and supported services. This chapter introduces network topologies, network types, and network components, and discusses several network communication technologies.

2.1 Introduction

Designing network architectures and proposing or improving various data communication protocols were at the center of extensive research and development interest. Various network architectures have been proposed since 1950s when the first architecture involving several communication links only used to connect central processors to remote peripherals (e.g., printers). The networks have evolved significantly since, and currently a network architecture is seen as a framework which specifies not only network topology, network type, network components, and their functionality, but also presents data communication protocols available, data formats employed, and a set of services supported. Often billing aspects are also considered.

The first two chapters of this book discuss network architectures and data communication protocols focusing on two directions. This chapter details network topologies, types, components, and communications technologies, and the next chapter presents communication protocols and services, respectively.

Network components include many network devices which enable data exchange between different network parts alongside end-user devices. *Network topologies* indicate how network devices are interconnected by links and how all these are arranged to form a functional communication network. When discussing *network types,* one refers to the classification of networks based on various aspects, including size, communication technology, etc., and when mentioning *network components,* the focus is on both network links and network devices. *Communication technologies* are concerned with the mechanisms employed to exchange data between interconnected network or user devices via the communication links, whereas *protocols* are seen as formal mechanisms to exchange messages between network compo-

B. Ciubotaru, G.-M. Muntean, *Advanced Network Programming – Principles and Techniques*, Computer Communications and Networks,
DOI 10.1007/978-1-4471-5292-7_2, © Springer-Verlag London 2013

nents. A protocol architecture includes all the protocols used to transport messages over a certain network infrastructure and indicates the way these protocols interact with each other. Although there is a thin line separating *services* from protocols, the latter are seen mostly application-linked and related to the network interface with end-users or devices.

All these aspects are of extreme importance for application developers, especially when performance constraints are involved. This chapter introduces network architectures' major aspects with the focus on existing and future network technologies.

2.2 Network Topologies

A network topology refers to the arrangement of nodes (i.e., network devices, servers, and host machines) and links between them to form a computer network. Nowadays, various types of topologies have been proposed and are in use. Among these topologies, most known are *ring*, *star*, *bus*, *tree*, *mesh*, and *ad-hoc*. These will be discussed in detail next.

2.2.1 Ring Topology

In a ring topology, each node is connected with exactly two other nodes forming a single data path in a form of a ring. Such a network arrangement is presented in Fig. 2.1.

In the basic ring network topology, the messages (data bits) travel in one direction only. Each node has a dual role, as a host and as a relay. As a host, each node will send data messages to other nodes and will receive messages addressed to it. As a relay, each node forwards messages addressed to other nodes to the next node on the ring.

The main issue concerning ring networks is their reliability. If a single link is broken, the communication between certain nodes is impeded. Dual ring solutions, where communication is possible both clockwise and anticlockwise, have been proposed to improve reliability through redundancy. The increase in redundancy comes with higher deployment and maintenance costs.

Standardization related to the ring topology includes the Token Ring protocol (IEEE 802.5), initially proposed by IBM. Apart from the specifications of the protocol, IEEE 802.5 also includes details on the data formats.

2.2.2 Star Topology

In a star topology, every host is connected to a central network component (denoted as *hub*), which may be a network hub, a switch, or a router, as illustrated in Fig. 2.2.

Fig. 2.1 Ring topology

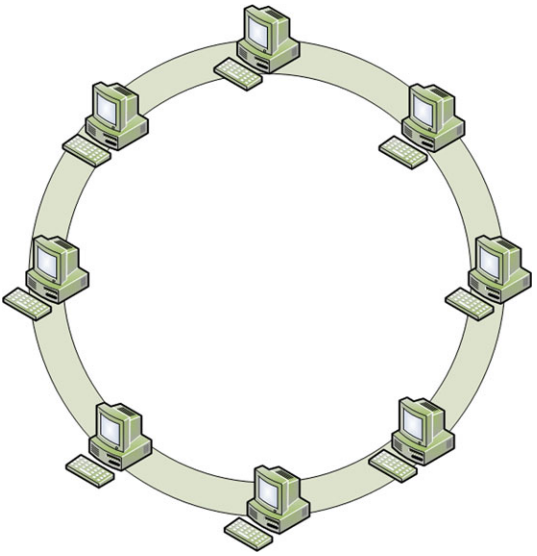
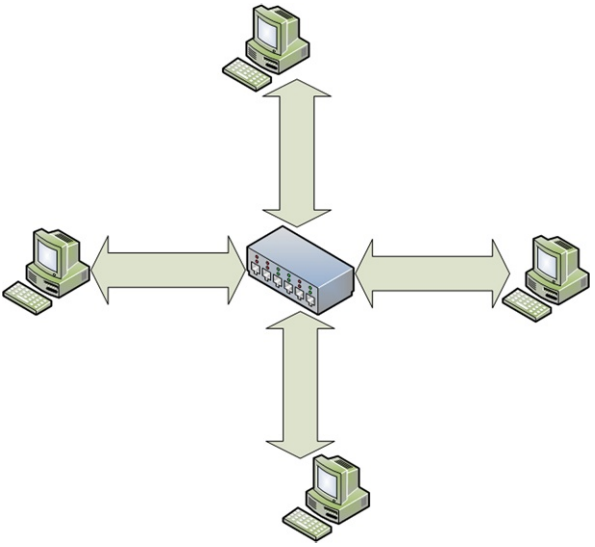


Fig. 2.2 Star topology



This topology is very popular for home networks where various devices such as desktop PCs, laptops, and mobile devices are connected to a local router, which is further connected to the broadband modem.

In terms of link failure, star topologies are more robust. If a certain link fails, only the hosts using those links will be disconnected from the network, while all the other hosts will not experience any disruptions in communications. The negative aspects of a star topology include the existence of a single point of failure and increased

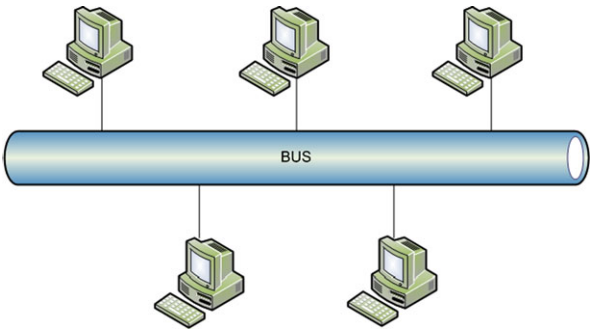


Fig. 2.3 Bus topology

deployment costs. The latter has been mitigated with the latest advancements in wireless networking.

2.2.3 Bus Topology

In a bus topology, a common backbone link is used to connect all the devices in the network with each other, as presented in Fig. 2.3. The hosts compete for accessing the backbone (a single cable) for data transmissions, which is a common communication medium.

When a host gains access to the medium, it sends data messages which are then received by all the hosts connected to the same backbone. However, only the host to which the messages are addressed will react to these messages, while the rest of the hosts will discard them.

The bus-based interconnection of hosts in a local network has been highly popular in the past when a small number of devices have required wired network connectivity. Today there are many diverse devices in need for network connectivity. However, bus networks work the best when a limited number of hosts are connected to the common bus and their efficiency is affected severely when a large number of stations require network access. This is mainly determined by the contention-based access to the common medium. As a consequence, bus topologies are less popular nowadays, in the context of the increasing demand for network connectivity and large growth of data traffic.

Standardization efforts related to the bus topology include the Token Bus protocol (IEEE 802.4) and the Fiber Distributed Data Interface (RFC 1188), which extends the token bus approach.

2.2.4 Tree Topology

The tree topology consists of a combination of bus and star topologies. As it can be seen in Fig. 2.4, the hosts are connected to a network hub which is further connected

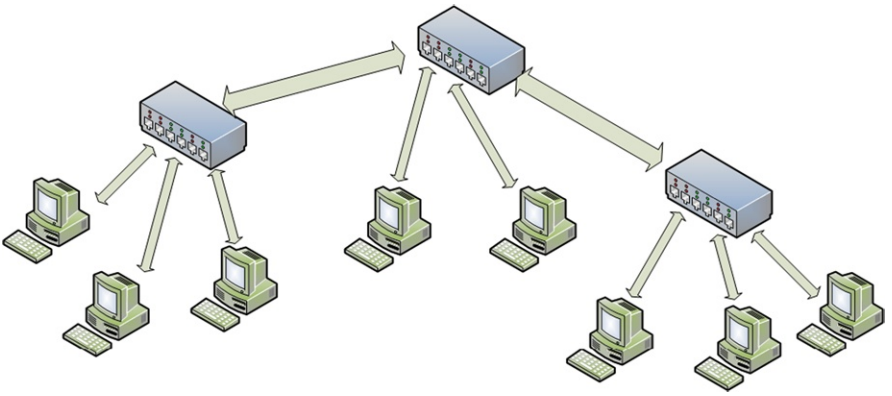


Fig. 2.4 Tree topology

to other hubs in a tree-like structure. Each hub acts as a root and router for a tree of hosts.

Routing messages in ring, bus, and star topologies is performed by broadcasting the messages to all hosts connected in the network. When tree topologies are used, messages originating at a host travel up the tree as far as necessary and then down the structure towards the destination host. Routing solutions become more important when tree topologies are involved, as efficiency is of high importance. In general, tree topologies support more scalable networks than bus and ring topologies. However, their maintenance may incur higher costs.

2.2.5 Mesh Topology

In a full mesh network topology, each host or network device is directly connected to any other device or host within that network. Although extremely robust, in general mesh topologies are very expensive, as they involve a high level of redundancy. This makes them less used for wired connectivity.

However, mesh topologies are most popular for wireless networks, as wireless links can be easily and cost effectively established and maintained. Full mesh topologies are also used for backbone networks.

Using partial mesh topologies is a more cost effective option. In such a topology, some of the devices are connected in a full mesh manner, while others are only connected to one or two devices.

There are several advantages brought by mesh topologies. Mesh networks can withstand high data traffic, as multiple independent paths can be formed to connect different devices within the network. Robustness is another advantage of mesh networks. Expansion and modification of the networks can also be done with minimum traffic disruption.

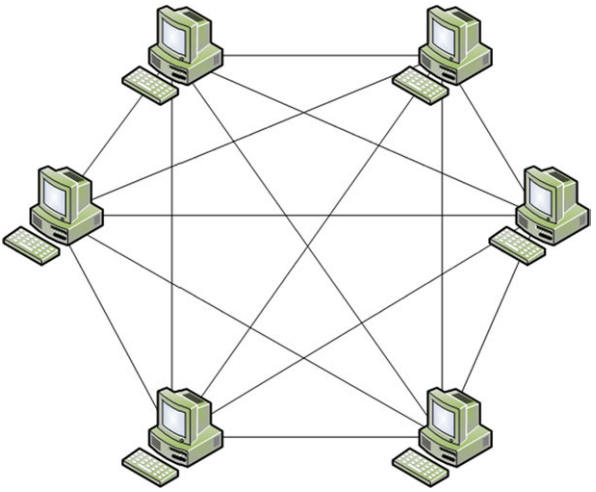


Fig. 2.5 Mesh topology

However, as already mentioned, the main disadvantage of the mesh networks is related to the high redundancy which leads to high costs of deployment and maintenance.

A full mesh topology is presented in Fig. 2.5.

2.2.6 Ad-Hoc Topology

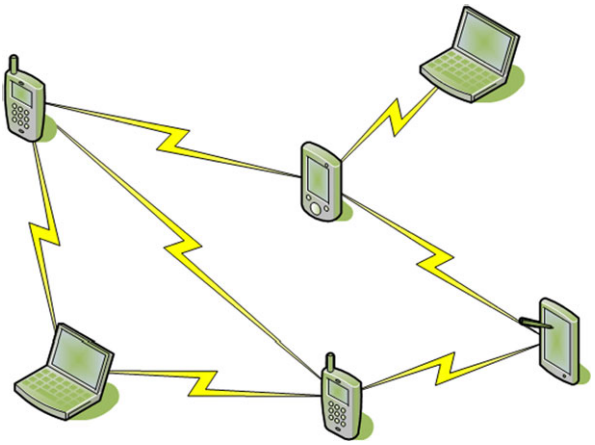
Lately there is an increased effort put on providing support for user mobility, and wireless connectivity already enables this. A step further is performed by wireless ad-hoc networks in which each node (potentially mobile in this case) dynamically establishes a communication link with the devices in its proximity. Each mobile node has a dual role, both as a mobile host and as a mobile router.

Ad-hoc networks do not rely on any infrastructure. Remote hosts communicate over dynamically formed paths based on links established between neighboring nodes. The messages travel over multiple links in an multi-hop manner in order to reach their destination. Such a network is graphically depicted in Fig. 2.6, but its topology is dynamically changing.

The main advantage of this type of network is its ease of deployment, low cost, and flexibility. As there is no previously deployed infrastructure, the network is formed on the go, as mobile hosts come and go. As each host in the network also acts as a router, the network range is also variable, adding scalability to the list of advantages.

Despite the advantages, ad-hoc networks suffer from unpredictable routes and data throughput. Due to host/router mobility, each route can be broken at any time due to a mobile device on the route moving away or going off-line.

Fig. 2.6 Ad-hoc topology



Furthermore, host mobility complicates paths formation, maintenance, and routing messages between senders and receivers, affecting both delivery efficiency and performance.

2.3 Network Components

Regardless of the network architecture employed, the major network components are their nodes and the inter-connecting links.

Based on the physical media used for data transmission between devices, the networks links may use: twisted pair, coaxial cable, fiber optics, as well as wireless media such as radio waves, microwaves, infra-red, and even visible light waves. Note that all these media have different characteristics which highly influence the communication properties and consequently determine their usage.

A twisted pairs cable consists of two insulated copper wires twisted together in a helical form. This cable was at the base of the first widely distributed network which enabled both telephony and later on basic data communications at very low bitrates.

A coaxial cable consists of a stiff copper core covered in a insulating material. The insulator is further surrounded by a cylindrical conductor, usually in the form of a mesh. This outer conductor is further protected by a plastic insulator. By making use of coaxial cables, the data transmission rate was improved, the interference was reduced and networks offering richer services such as cable TV were supported.

Fiber communications are very popular mainly due to their large bandwidth and low effect of interferences. They are performed over fiber optic cables which consist of three elements: a glass core, a glass cladding and some plastic cover. The glass core is the main light propagation medium and is at the center of the fiber cable. The plastic cover is like a shell and is used to protect the fiber. The glass cladding has a lower refraction index and is introduced to keep the light within the core and the plastic cover.

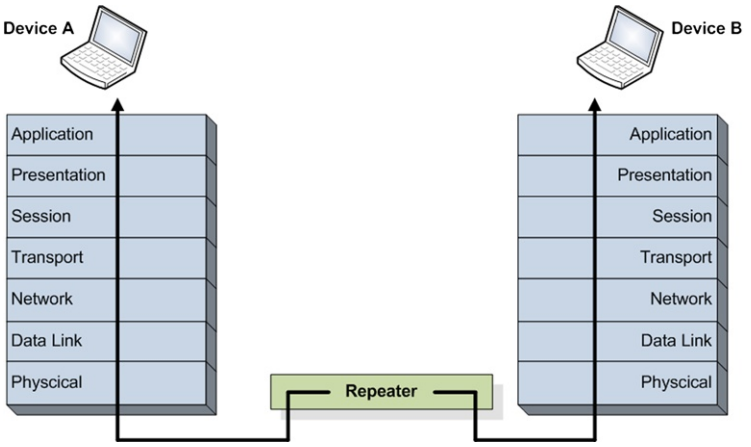


Fig. 2.7 Repeaters operate at physical layer

Wireless communication networks use modulated electromagnetic waves to send messages between directly linked devices. These devices can communicate directly among themselves in a distributed manner, forming ad-hoc networks or rely on a centralized network device to handle inter-end-device communication in the infrastructure mode. Among wireless networks, some use line-of-sight, others non-line-of-sight transmissions; some use low-latency channels (e.g., satellite communications), others fast communication channels; some use low frequency channels, despite the low bandwidth (e.g., military use), others high frequency-high bandwidth, etc.

In terms of network nodes, most visible are the end-user devices which range from smartphones, netbooks, and laptops to desktops and even servers. Lately, diverse consumer devices have also been enabled to exchange data via the networks. This is in the context of smart homes, but the trend is set to continue, supporting also networked device control.

The classic network nodes, also known as inter-networking devices, consist of intermediate devices which provide various support for data exchange and enable networking. Each type of inter-networking device is deployed at different network layers and provides different services. The most known are repeaters, bridges, routers, and gateways.

A repeater is a network device which amplifies, reshapes, and/or retimes the input signal in order to increase the distance, improve the signal quality, and boost efficiency of transmitted data. As repeaters do not attempt to make sense of the content of the data transmitted in any way, performing on the physical signal only, they are seen as operating at the physical network layer, as shown in Fig. 2.7. Repeaters' reshaping function is illustrated in Fig. 2.8.

A bridge is a network device which reduces the amount of traffic on a LAN by dividing it into two segments or enables communication between two LANs by inter-connecting them. Bridges filter data traffic at network boundary and take

Fig. 2.8 Repeaters operate at physical layer

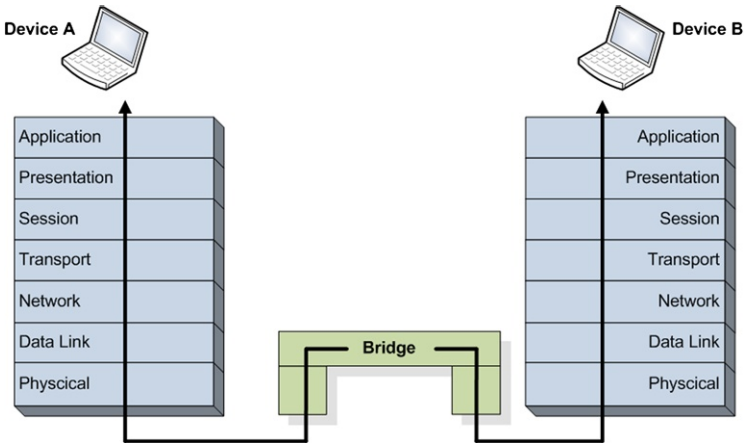


Fig. 2.9 Bridges operate at data link layer

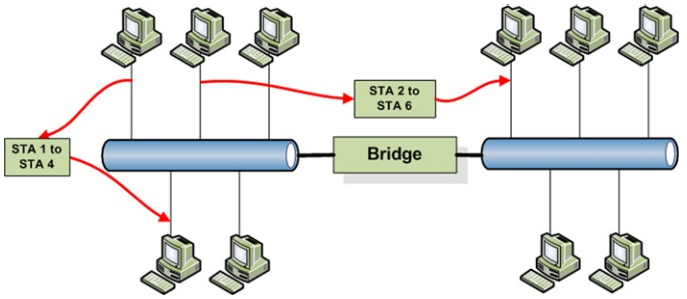


Fig. 2.10 Bridges filter the traffic between network segments

decisions whether or not to allow traffic passage. As bridges require some network-related information, they operate at the level of frames at the data link network layer, as illustrated in Fig. 2.9. A very important task bridges do when dividing networks into segments is confining local traffic to the various network segments, supporting overall network scalability and increasing communication efficiency. An equally important task bridges do when enabling inter-LAN communication is accommodating data exchange despite having different frame formats, payload sizes, data rates, bit order of addresses, usage of priority bits, existence of acknowledgments or negative acknowledgments (ACK/NACK), etc. The principle of bridges performing traffic

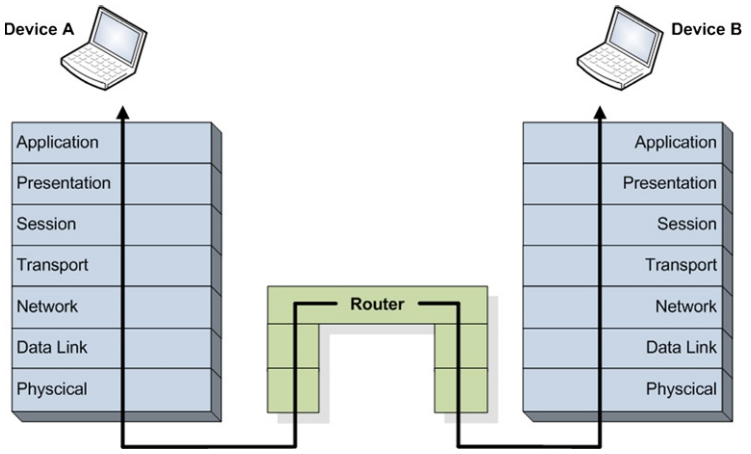


Fig. 2.11 Routers operate at network layer

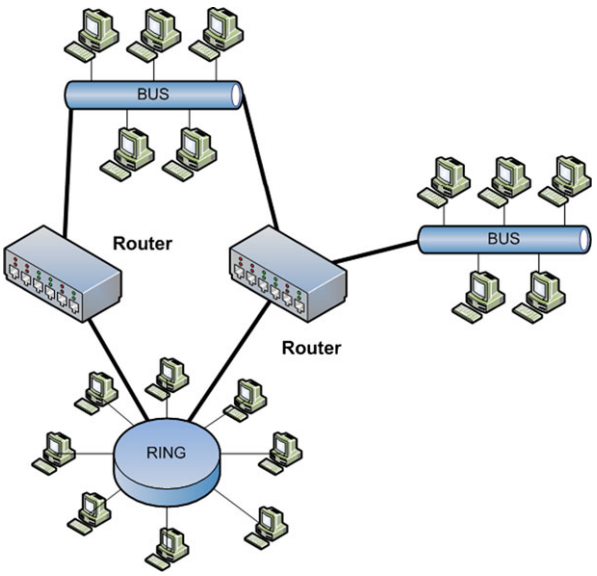


Fig. 2.12 Routers interconnect and enable data exchange between different networks

filtering and reducing the amount of data exchanged across two network segments is shown in Fig. 2.10.

A router is a network device which inter-connects different networks and relays packets from a network to another according to their destination address. Routers communicate with each other and are involved in network information collection which they store in forwarding tables. Based on this information, the routers run routing algorithms to determine the best path between any two hosts and forward the data packets on those paths. Routers are active at the network layer as shown in Fig. 2.11 and are deployed as illustrated in Fig. 2.12.

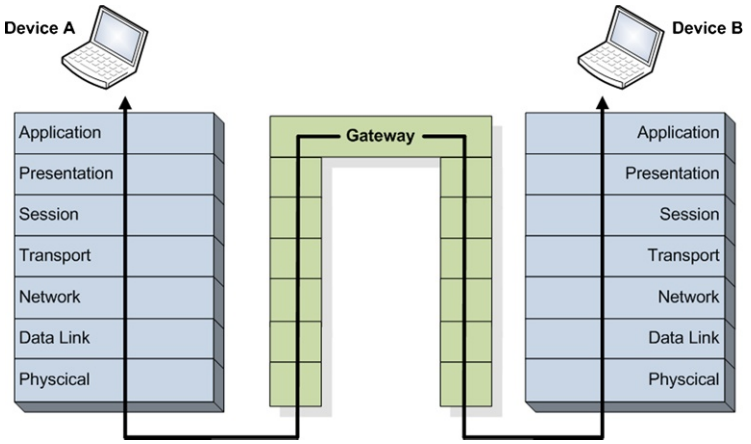


Fig. 2.13 Gateways operate at application layer

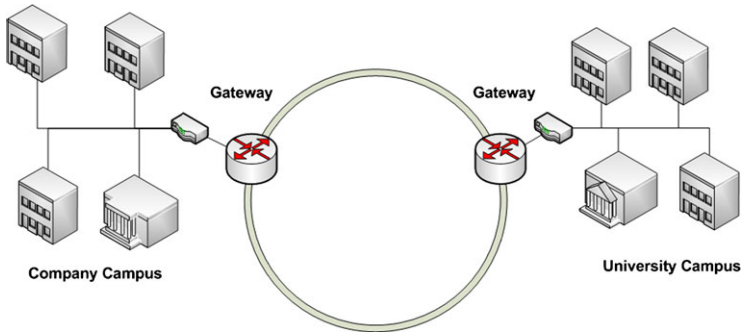


Fig. 2.14 Gateways interconnect and control data exchange between different networks

A gateway is a network device which extends the functionality of a router to include the application layer as illustrated in Fig. 2.13. Modifications of the data packets could include filtering or blocking certain type of traffic, changing values in the header and/or trailer fields, adjustments of data rates, modifications in the size of packets, applying security, etc. An example of gateway deployment is presented in Fig. 2.14.

2.4 Network Types and Communication Technologies

Networks differ in many aspects, not only in their topology, from communication technology to range. In this context, there are many criteria which can be used to classify the networks.

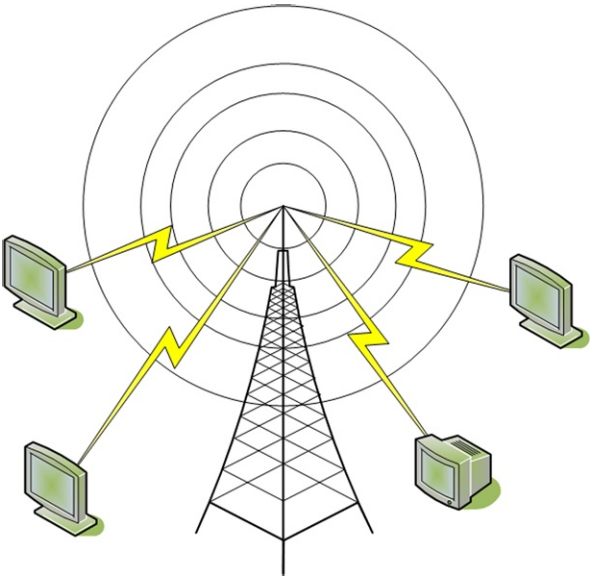


Fig. 2.15 Broadcast networks

Based on their *transmission technology*, the networks can be classified as broadcast or point-to-point networks.

In a broadcast network, all nodes share the same communication medium. A message sent by a node is heard by all other nodes connected to the network. This constitutes a major advantage of the broadcast networks as it allows the possibility to send the same message to all receivers attached to the network in the most efficient manner. A well known example of a broadcast network is the television network as presented in Fig. 2.15. The same content (TV channels) is delivered to all devices attached to the network, a mechanism suitable for distribution of highly popular non-interactive services.

As opposed to broadcast networks, point-to-point networks use many connections to link individual pairs of devices. A message travels from the source to its destination by traversing multiple interconnected devices. All these intermediate devices and the links connecting them form a communication route. A source node may be connected to a destination node by multiple routes, as presented in Fig. 2.16. Choosing the right route for message transportation is very important in point-to-point networks. These networks are suitable for delivering differentiated content based on various requests.

However, potentially the most important criterion for classifying networks is their scale. In general, the network scale dictates the transmission technology used and often the corresponding communication protocols.

Based on their scale, networks can be classified as personal area networks, local area networks, metropolitan area networks, wide area networks, and the Internet. Next these network categories are discussed in detail.

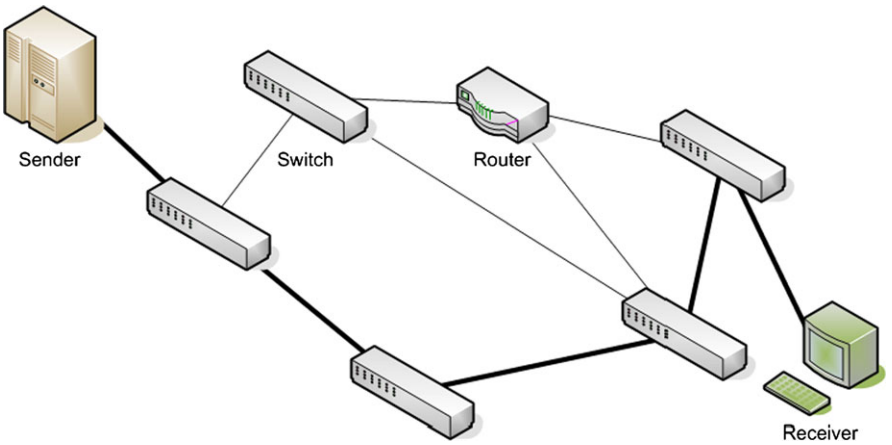
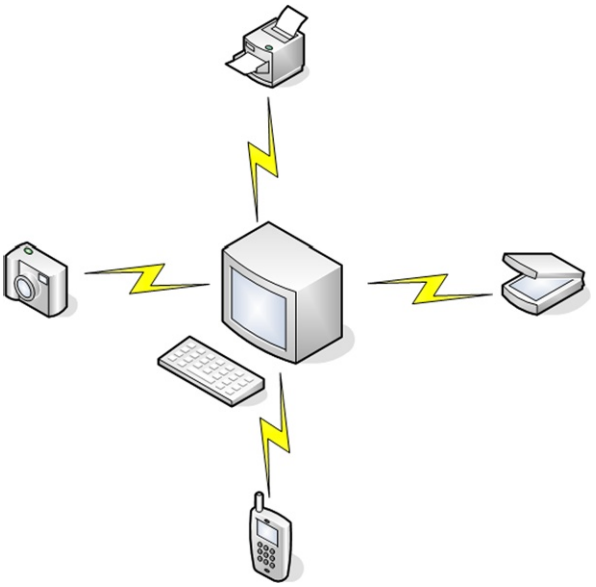


Fig. 2.16 Point-to-point networks

Fig. 2.17 Personal area network



2.4.1 Personal Area Networks

Personal Area Networks, or PANs, use short range transmission technologies (1 m) and are usually intended to serve one person, hence their name.

An example of a PAN is presented in Fig. 2.17. In this case, wireless communication technology is used to link various peripherals, such as a printer, scanner, as well as keyboard and mouse with the computer. Moreover, devices such as smartphones and video cameras can also be connected to computers forming PANs.

Wireless Personal Area Networks (WPANs) are increasingly popular, and the IEEE 802.15 Working Group has been established especially in order to standardize WPAN technologies. Their work has resulted in several standards, among which most important are briefly introduced next.

IEEE 802.15.1 (2002, 2005) standardizes the well known Bluetooth wireless communication technologies used by many portable devices to interconnect or communicate with peripherals or personal computers.

IEEE 802.15.2 (2003) address the coexistence of WPANs with other wireless networks such as wireless local area networks.

IEEE 802.15.3 (2003), IEEE 802.15.3b (2005), IEEE 802.15.3c (2009) address the physical and MAC layers for high-rate WPANs.

IEEE 802.15.4 (2011) specifies the MAC and PHY layer for low-rate, low-range, and low-power wireless network communications. Based on this standard, protocols such as Zigbee and 6LoWPAN define the network layer specialized on ad-hoc networking and the application layer targeting WPAN networks.

IEEE 802.15.5 (2009) provides an architectural framework for mesh networks deployed on low-power wireless communication technologies.

IEEE 802.15.6 (2012) is focused on low-power and short-range wireless technologies to be used around the human body or even in the human body for specific medical applications.

IEEE 802.15.7 (2011) targets the standardization of short-range wireless optical communication based on visible light.

2.4.2 Local Area Networks

Local area networks (LANs) are usually contained within a single building, campus or geographical area, up to a few kilometers in size. LANs are usually privately owned and their main purpose is to interconnect computers and resources such as printers and data storage units belonging to a single functional unit such as an office building, factory, school or university.

LANs are usually small in size, and LAN communications benefit from short delays and reduced error rates. Typical data transmission rates range between 10 and 100 Mbps with newer technologies reaching transmission speeds of up to 10 Gbps.

The most popular technology for LANs is Ethernet, standardized as IEEE 802.3. Other technologies such as token ring, token bus, and FDDI can also be used.

Often Ethernet uses a star topology, where multiple computers are interconnected using wires (usually twisted pairs) or fiber optics to a central active network device.

Fast, Gigabit, and 10 Gigabit Ethernet refer to Ethernet networks capable of reaching transmission speeds of up to 100 Mbps, 1 Gbps, and 10 Gbps, respectively, over twisted wired cables or fiber optics.

Figure 2.18 illustrates three typical LAN topologies.

Wireless Local Area Networks (WLANs) are increasingly popular, mostly due to the reduced cost of deployment and maintenance and their support for mobility.

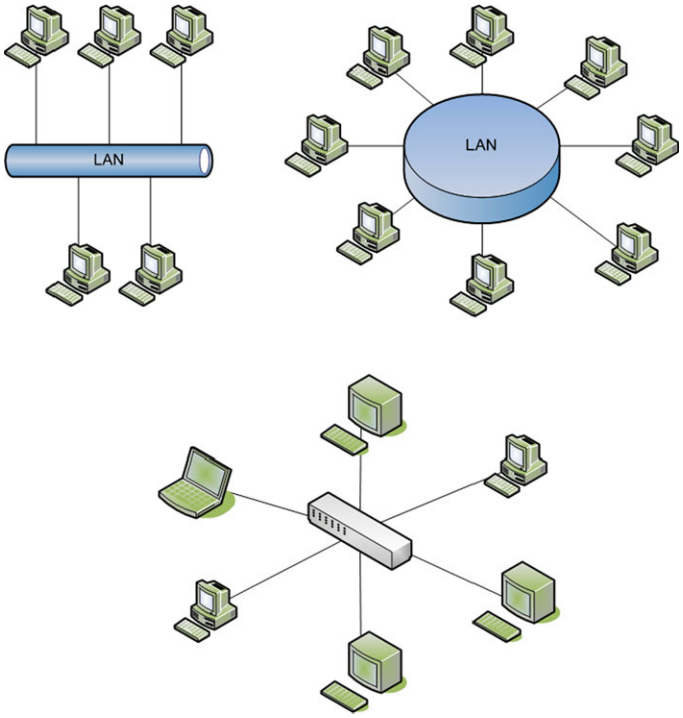


Fig. 2.18 Local area network

Currently, the IEEE 802.11 family of standards has been widely adopted and is being heavily used worldwide for WLANs. This family (also known as WiFi) includes the original standard and various extensions which address different issues including higher bit rates, QoS support, security, etc.

The standards for wireless access networks usually cover the physical layer and the medium access control protocol (MAC) sub-layer. The original IEEE 802.11 standard first released in 1997 [1] supports data rates up to 2 Mbps and was initially developed for best effort traffic only.

Each host connected to a certain IEEE 802.11 access point shares the wireless medium with the other mobile hosts associated with the same access point. This leads to race conditions for medium access which determine high collision rates and consequently low data rates, especially when the number of mobile hosts involved in simultaneous data communications increases.

The IEEE 802.11 MAC layer provides mechanisms for medium access coordination, including the Distributed Coordination Function (DCF) and the partially centralized Point Coordination Function (PCF).

A group of mobile stations connected to a single Access Point (AP) form the basic building block defined by this standard as a Basic Service Set (BSS). The geographical area covered by a BSS is called a Basic Service Area (BSA). Connect-

ing several BSSs through a Distribution System (DS) determines the creation of an Extended Service Set (ESS).

The first IEEE 802.11 extension, IEEE 802.11b [2] increased the maximum data rate to 11 Mbps, which was a huge step forward. Following additional efforts, the data rate was further increased to 54 Mbps in the IEEE 802.11a and IEEE 802.11g standard extensions [3, 4].

Maintaining high QoS levels by using the two coordination methods, DCF and PCF, is difficult, thus novel QoS enhancements for IEEE 802.11 MAC layer were standardized by IEEE 802.11e [5].

Consequently, two new mechanisms are described by the new standard, namely the Hybrid Coordination Function (HCF) and the Enhanced Distributed Coordination Function (EDCF). HCF is based on PCF, and EDCF relies on its implementation on DCF. Further enhancements brought by this standard extension are block acknowledgments which allows acknowledging more then one MAC frame by sending only one acknowledgment packet and *No Ack* which allows time critical data frames not to be acknowledged. To enhance QoS provisioning for time sensitive and bandwidth hungry applications, traffic prioritization was proposed for IEEE 802.11 [6]. Four traffic categories are defined: voice, video, best effort, and background, and in this order, IEEE 802.11e offers prioritization support.

The emerging IEEE 802.11n standard [7] aims at providing even higher bitrates, of up to 600 Mbps. The data rate enhancement approach of IEEE 802.11n is oriented on improving MAC layer techniques, unlike other IEEE 802.11 which aim at increasing the data rates at the physical layer. IEEE 802.11n uses the same QoS support techniques proposed for IEEE 802.11e.

The currently under study IEEE 802.11 VHT (Very High Throughput) [8] aims at offering data rates of up to 1 Gbps for low velocity mobile hosts.

The IEEE 802.11 family supports limited host mobility except for the IEEE 802.11s standard [9, 10] which specifies support for wireless mesh networks and which addresses host mobility within the wider range mesh network.

IEEE 802.11p standardizes wireless access in vehicular environments which represents a short to medium range communication service providing high data transfer rates for roadside-to-vehicle or vehicle-to-vehicle data communications.

The IEEE 802.11 family groups several other standards addressing various aspects of wireless data networks, including security, management, and compatibility. A more detailed overview of IEEE 802.11 family of standards can be found in [11].

Tables 2.1 and 2.2 summarize the characteristics of the most important IEEE 802.11 standards and extensions, including maximum data rates and frequencies.

2.4.3 Metropolitan Area Networks

Metropolitan Area Networks (MANs) usually cover an area the size of a city. Figure 2.19 graphically depicts a MAN interconnecting various areas of a city. Originally, MANs have been developed to distribute television services over the cable TV

Table 2.1 IEEE 802.11 family of standards

Standard	Bitrate	Frequency	Description
802.11	1 Mb/s (2 Mb/s)	2.4 GHz	Initial standard
802.11b	11 Mb/s	2.4 GHz	Data rate enhancement
802.11a	54 Mb/s	5 GHz	Data rate enhancement
802.11g	54 Mb/s	2.4 GHz	Backward compatibility
802.11n	600 Mb/s	2.4 and 5 GHz	Data rate enhancement
802.11p	27 Mb/s	5.9 GHz	Vehicular communication
802.11ac (VHT)	1 Gb/s	<6 GHz	Data rate enhancement
802.11ad (VHT)	1 Gb/s	60 GHz	Data rate enhancement

Table 2.2 IEEE 802.11 family of standards

Standard	Description
802.11e	Extension for QoS support
802.11aa	Extension for audio/video streaming
802.11r	Handoff support
802.11s	Transparent multi-hop operation (Mesh)
802.11u	Interworking with external networks (cellular)

network. The development and increased popularity of the Internet has determined the operators to adapt the cable TV network for the delivery of Internet services.

Several technologies have been used for implementing MANs. These technologies include Asynchronous Transfer Mode (ATM), Fiber Distributed Data Interface (FDDI), and Switched Multi-megabit Data Service (SMDS). These technologies are currently in the process of being replaced by Ethernet-based solutions.

Wireless MAN links interconnecting local area networks have been built based on either microwave, radio, or infra-red laser communication technologies.

Distributed Queue Dual Bus (DQDB), standardized as IEEE 802.6, has been developed specifically for MANs. This technology offers communication infrastructure over long distances, up to 160 km. The operating speed ranges from 34 to 155 Mbps.

Wireless Metropolitan Area Networks (WMANs) were developed to cover whole cities and to interconnect LANs or WLANs as well as individual users, both static and mobile. WMANs use two types of connectivity: *line of sight*, when there is a requirement for communication success such as no obstacles between senders and receivers can exist, and *non-line of sight,* when senders and receivers are not required to see each other in a straight line for communications.

Companies producing equipment for WMANs have formed the Worldwide Interoperability for Microwave Access (WiMAX) forum concerned with the standardization and technology development in this area of wireless communications.

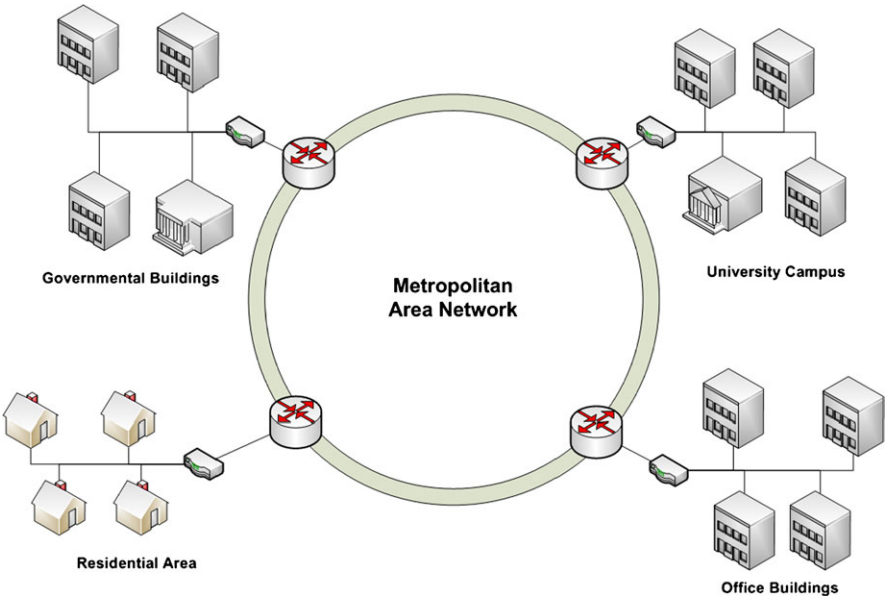


Fig. 2.19 Metropolitan area network

Specific to WMANs is the IEEE 802.16 family of standards. The IEEE 802.16 is based on two systems: the Multichannel Multipoint Distribution System (MMDS) and Local Multipoint Distribution System (LMDS) [12].

The MMDS system offers better coverage (i.e., typical cell radius is 50 km), but the throughput is quite low, between 0.5 and 30 Mbps. LMDS has lower coverage (e.g., 3 to 5 km radius), but provides higher bandwidth (e.g., 34 to 38 Mbps with an increase to 36 Gbps for the newer versions).

IEEE 802.16 provides QoS provisioning support. This is achieved mainly trough connections, service flows, and service scheduling. QoS provisioning is negotiated at the initiation of the session, and QoS requirements are mapped on the QoS parameters in the IEEE 802.16 MAC layer. Mobility is supported in the new IEEE 802.16e standard which permits mobile hosts to change their base station while the data connection is still active. Both soft and hard handover mechanisms are supported, while several enhancement solutions are being proposed [13].

WiMAx is relatively popular as a wireless broadband solution, with several types of mobile devices already having WiMAX interfaces. However, new technologies are already threatening WiMAX.

High Performance Radio Access (HiperACCESS) standardized by ETSI offers non-line of sight broadband wireless access using frequencies between 11 and 43.5 GHz. The typical cell radius is 5 km, and the data rates per cell ranges between 25 and 100 Mbps [14].

High Performance Radio Metropolitan Access Network (HiperMAN), also standardized by ETSI, offers broadband connectivity targeting residential and small of-

fice areas. HiperMAN works in the frequency bands below 11 GHz and offers non-line of sight connectivity with aggregated data rates of up to 25 Mbps [15].

WiBro is another WMAN solution developed in Korea which offers broadband connectivity to both stationary and mobile users. WiBro operates in the 2.3–2.4 GHz frequency band offering data rates of up to 50 Mbps [16]. The major advantage of WiBro over the other WMAN technologies is the mobility feature which is very well developed.

High Altitude Platforms (HAP) [17] use a quasi-stationary aerial platform equipped with wireless transceivers offering broadband wireless access with data rates of 120 Mbps or up to 10 Gbps in some configurations. This type of wireless technology offers good coverage with better line of sight connections.

IEEE 802.22 Wireless Regional Area Network (WRAN) offers data rates up to 18 Mbps for rural and remote areas using the unoccupied TV channels between 54 and 862 MHz [18].

Cellular networks which initially offered only voice services are already offering broadband Internet access through the current third generation (3G) and the future fourth generation (4G) networks.

The first to provide mobile communication services were the first generation (1G) cellular networks which supported only analog voice calls and very limited data applications. This technology was replaced by the second generation cellular networks (2G) which is entirely digital and apart from voice communication also supports low bit rate data communication in the form of Short Message Service, Multimedia Message Service.

The current cellular network technologies can be grouped in two main families: Global System for Mobile Communications (GSM) based on time division, multiple access (TDMA), and code division multiple access (CDMA) [19].

The maximum bit rate in GSM was 9.6 kbps; however, throughput enhancement solutions have been developed for this standard including the 2.5G General Packet Radio Service (GPRS) and the 2.75G Enhanced Data Rates for GSM Evolution (EDGE).

GPRS supports theoretical data rates around 114 kbps, but in reality the throughput reaches values around 40 kbps only. EDGE is the first to open the door for multimedia applications over cellular networks. It supports theoretical throughputs around 400 kbps.

The third generation cellular network (3G) supports voice and continues the improvement of the data communication rates.

In the GSM category, the Universal Mobile Telecommunications System (UMTS) makes use of wideband CDMA (WCDMA) and High-Speed Packet Access (HSPA) technologies in order to support bit rates of up to 2 Mbps.

The CDMA-based standards for 3G networks include the CDMA2000 family among which CDMA 1xRTT, supports average data rate of 40–80 kbps with peak data rate of 150 kbps. CDMA 2000 1xEV-DO supports only data communications with maximum data rates of 2.4 Mbps.

As the demand for higher bandwidth and QoS support is increasing with the increased popularity of bandwidth-hungry, real-time applications, the forth generation network (4G) is in the process of being defined and standardized.

The technologies which are principal candidates for 4G networks are Long-Term Evolution (LTE), Ultra Mobile Broadband (UMB), and 802.16m (WiMAX II) [19].

LTE is developed based on the GSM technology with data rates around 250 Mbps. LTE will support QoS provisioning for real-time applications like multimedia streaming [20].

UMB is developed based on the CDMA technology and provides data rates up to 288 Mbps. UMB incorporates control mechanisms which optimize data transmission in order to meet the QoS requirements of various user applications [21]. UBM also supports inter-technology handover with CDMA2000 standards [21].

IEEE 802.16m (WiMAX II) is developed based on the WiMAX standard with adaptation for cellular networks. 802.11m aims at supporting higher data rates and QoS support for various multimedia services. The data rate is expected to reach 100 Mbps for mobile users and 1 Gbps for static users.

2.4.4 Wide Area Networks

Wide Area Networks (WANs) usually cover larger geographical areas such as a whole country or even a continent. The biggest WAN known today is the Internet, spanning the whole globe. However, a typical WAN may interconnect several LANs, MANs, or even other WANs, providing the backbone infrastructure to transport data between the interconnected networks.

As it can be seen in Fig. 2.20, a WAN may use several technologies for the communication subsystem.

Wired infrastructure, including fiber optics or telephone lines, as well as wireless technologies, including terrestrial or satellite-based communication systems, can be used for data transfer within a WAN.

In general, a WAN consists of two basic elements: communication lines (i.e., copper wires, optical fibers, radio links) and switching elements (i.e., routers).

The switching element connects two or more communications lines. Whenever data is received by the switching element on a communication line, it decides on which line the data should be forwarded and transmits the messages on that particular line.

For long distance communications over wired links, WANs tend to use technologies such as Multiprotocol Label Switching (MPLS), Asynchronous Transfer Mode (ATM), Frame Relay, and X.25.

Similar to the wired WANs, the Wireless Wide Area Networks have the largest coverage area among the wireless networks. WWANs can be used as separate networks or as interconnection backbones for MANs.

WWANs are usually satellite networks, but terrestrial versions are also considered. A terrestrial WWAN is standardized by the IEEE 802.20 [22]. This standard targets high mobility users with speeds of up to 250 km/h. QoS preservation methods as well as handover management schemes are supported by this technology.

Satellite WWANs have the advantages of global coverage, high mobility support and broadcast capabilities [12]. Initially satellite networks had only broadcast

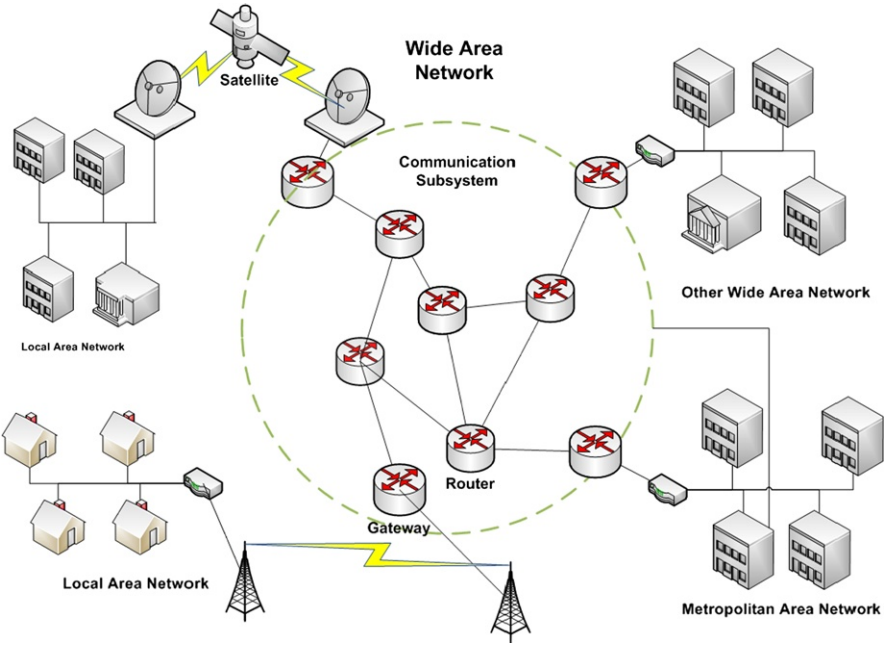


Fig. 2.20 Wide area network

capabilities, but within the Next Generation Satellite System (NGSS) unicast and multicast is also provided.

The Digital Video Broadcasting (DVB) standard family started first by supporting digital video and data broadcasting through the satellite networks. DVB-S (satellite) enables down-link data transfer with rates of up to 45 Mbps only. The newer DVB-S2 increases the downlink rate to 60 Mbps. For uplink DVB-RCS (return channel satellite) standard was developed supporting rates of up to 2 Mbps.

Apart from the satellite versions (DVB-S) DVB has also standardized a terrestrial wireless data service through the DVB-T, and more recently DVB-T2.

DVB-T offers much flexibility in terms of data rates. Depending on the particular configuration of the various parameters specific to the wireless transmission it offers a wide range of bitrates starting from 3.7 up to 31 Mbps [23].

Although DVB-T broadcasts multimedia content to static and mobile users, including vehicular receivers, it is not optimized for highly mobile handheld devices.

Consequently, DVB team has developed DVB-H (handheld) [24] for multimedia content delivery to mobile devices. DVB-H is developed based on the DVB-T (terrestrial), whose infrastructure it uses. Similar to DVB-T, DVB-H offers one way (downlink) point-to-multipoint data communication over wireless links with indoor and outdoor coverage. Considering the limited radio capabilities of a mobile handheld device as well as the higher error rates due to device mobility, DVB-H incorporates powerful error correction mechanisms. Time-multiplexing technologies are used to improve power consumption to cope with the energy constraints of battery

powered handheld devices. Seamless handover between base stations is also supported, and loss is highly reduced due to the time-slicing techniques used for power efficiency even with only one radio interface [25].

DVB-H supports mainly downlink communication, interactivity being achieved through separate backward point-to-point channels using other wireless data communication technologies like GPRS or UMTS. Supporting mainly broadcast services, DVB-H scales well offering downlink data rates between 3.3 and 31.6 Mbps. DVB-H specifies only the protocol layers below the network layer.

DVB-H provides an Internet Protocol (IP) interface for higher transport layers which is defined by the IP-based Data Broadcast (IP Datacast) specification. IP Datacast also offers the option of accessing an external cellular network for the backward channels and to create the so-called hybrid networks [26].

2.4.5 The Internet

The Internet can be best described as a network of networks. The Internet is not a single network, but instead a collection of a vast diversity of networks in terms of topologies and communication technologies which use, however, a common set of protocols to offer certain services.

Figure 2.21 schematically presents an overview of the Internet structure. As it can be seen in the figure, networks such as LANs owned by universities or small communities, regional Internet Service Provider (ISP) distribution networks, cellular networks, offering also data services, can be interconnected via backbones allowing for the creation of a global inter-network.

To describe how user hosts are interconnected and are allowed to communicate over the Internet, we will start from the client location. The client PC or home LAN router will be connected to the ISP modem/router which is designed to interconnect the user's LAN with the ISP Point of Presence (PoP) over the telephone lines or cable network. At the PoP level, the signals originating at the home are sent to the ISP's regional network.

Often, the local telecommunication company or the cable TV operator is also the ISP, so the telephone or cable networks and ISP regional networks are overlapping.

Except for the cable and telephone lines, home users my be offered access to the ISP core network using fiber or wireless links such as WiMAX or cellular.

The ISP's regional network consists of interconnected routers and links spread across the area served the ISP. The ISP regional network is further connected to the backbone network owned by a backbone operator. Backbone operators are companies owning and operating large international networks consisting of thousands of routers interconnected by high-bandwidth fiber optical links. These backbone networks can transport huge amounts of traffic and usually link countries and even continents.

The end user usually does not get direct access to a backbone. The ISP regional networks or distribution networks are connected to the backbones. However, large

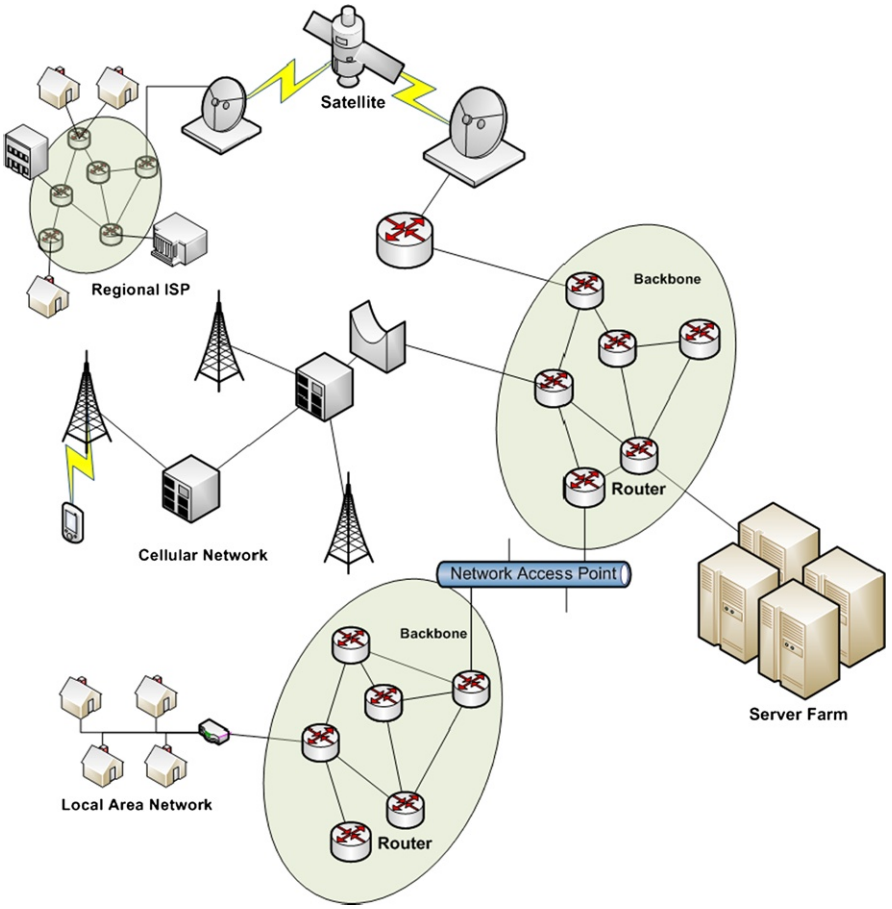


Fig. 2.21 Overview schematic of the Internet

corporations may be connected directly to the backbone, especially those operating high capacity server farms capable of handling millions of service requests and high amount of data traffic.

Various backbones exist, interconnecting all regions of the world, and being operated by various companies. In order to reach a global coverage, all these backbones are interconnect at Network Access Points (NAP). These NAPs basically consist of a high speed LAN interconnecting routers corresponding to different backbones.

Moreover, NAPs are not the only technique to interconnect backbones. Private peering is a well known technique where various routers belonging to distinct backbones have direct links between them allowing data packets to be exchanged between distinct backbones.

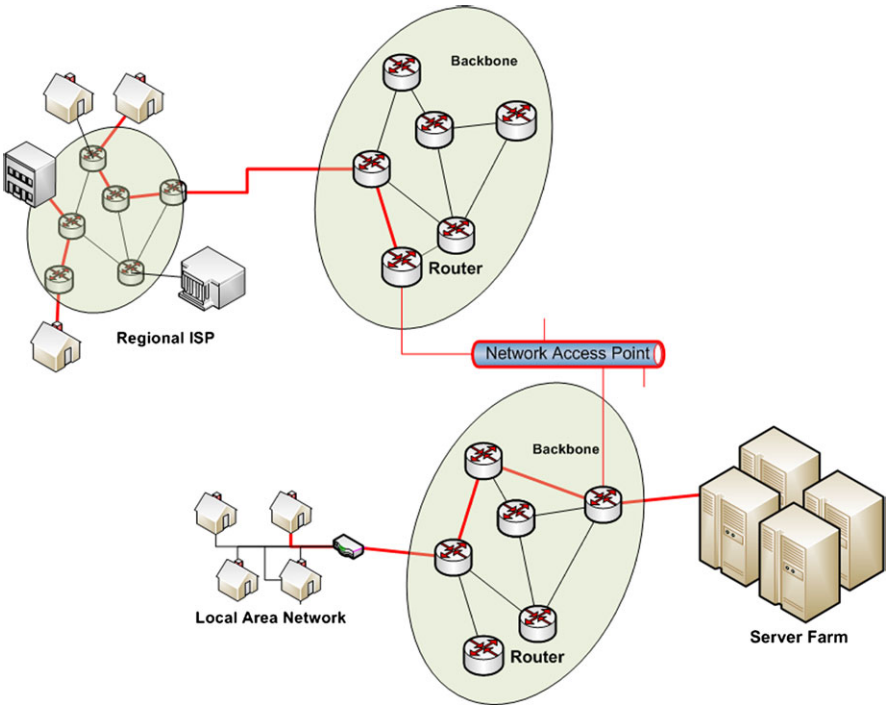


Fig. 2.22 Data communication in the Internet

Figure 2.22 describes how data is exchanged between two hosts over the Internet. As it can be observed in the figure, if two hosts communicate and are connected to the same ISP regional network then the traffic is routed within the ISP network only.

If, for example, a host accesses a service (e.g., a website) located on a server farm, the traffic will be routed from the ISP's network to the corresponding backbone and then through the farm's local network to the destination server.

If two hosts are connected to distinct ISP networks which are further connected to distinct backbones, the data packets will travel from the ISP regional network to the backbone, and then over the NAP to the other backbone and further to the destination ISP's regional network.

2.5 Conclusions

This chapter introduced the various network topologies used today, presented the major network components, and discussed various criteria used to classify the networks. Among the criteria identified, coverage area is accepted as one of the most relevant and with the greatest impact on network cost, complexity, and performance. Various network types identified based on size have been discussed along with the specific communication technologies used by each type of network.

Although the technologies and network characteristics discussed so far represent the foundation of any network, there is still a need for additional support to provide robust and performance-oriented network communications.

There is a need for a set of protocols to govern the way data is produced, formatted, transported, and consumed by various interconnected nodes communicating to each other and a set of services to be offered to the end-users.

The next chapter introduces these protocols and presents major network-based services.

References

1. IEEE (June 1999) IEEE standard for local and metropolitan area networks specfic requirements—Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specfications
2. IEEE (September 1999) IEEE standard for local and metropolitan area networks specfic requirements—Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specfications high speed physical layer extension in the 2.4 GHz band
3. IEEE (1999) IEEE standard for local and metropolitan area networks specific requirements—Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications high speed physical layer in the 5 GHz band
4. IEEE (June 2003) IEEE standard for local and metropolitan area networks specific requirements—Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 4: further higher data rate extension in the 2.4 GHz band
5. IEEE (2005) IEEE standard for local and metropolitan area networks specific requirements—Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications MAC enhancements for QoS
6. Xiao Y (2005) Performance analysis of priority schemes for IEEE 802.11 and IEEE 802.11e wireless LANs. IEEE Trans Wirel Commun 4(4):1506–1515
7. IEEE (September 2008) IEEE draft standard for local and metropolitan area network-specific requirements—Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications mendment 5: enhancements for higher throughput
8. Eastwood L, Migaldi S, Xie Q, Gupta V (2008) Mobility using IEEE 802.21 in a heterogeneous IEEE 802.16/802.11-based, IMT-advanced (4G) network. IEEE Wirel Commun 15(2):26–34
9. IEEE (December 2009) IEEE draft standard for information technology—telecommunications and information exchange between system—LAN/MAN specific requirements—Part 11: Wireless medium access control (MAC) and physical layer (PHY) specifications: amendment 10: mesh networking
10. Hiertz G, Denteneer D, Max S, Taori R, Cardona J, Berlemann L, Walke B (2010) IEEE 802.11s: the WLAN mesh standard. IEEE Wirel Commun 17(1):104–111
11. Hiertz G, Denteneer D, Stibor L, Zang Y, Costa X, Walke B (2010) The IEEE 802.11 universe. IEEE Commun Mag 48(1):62–70
12. Kuran MS, Tugcu T (2007) A survey on emerging broadband wireless access technologies. Comput Netw 51(11):3013–3046
13. Lee DH, Kyamakya K, Umondi J (2006) Fast handover algorithm for IEEE 802.16e broadband wireless access system. 6 pp
14. ETSI (March 2002) Broadband radio access net-works (BRAN) HIPERACCESS system overview
15. ETSI (March 2001) Broadband radio access networks (BRAN); Functional requirements for fixed wireless access systems below 11 GHz: HIPERMAN

16. Kim D (2005) Wibro overview and tta activities. Technical report, TTA
17. Cianca E, Prasad R, De Sanctis M, De Luise A, Antonini M, Teotino D, Ruggieri M (2005) Integrated satellite-hap systems. IEEE Commun Mag 43(supl 12):33–39
18. Chouinard G (2005) Status of work in the IEEE 802.22 WG. Technical report
19. Ortiz S (2007) 4G wireless begins to take shape. Computer 40(11):18–21
20. Anas M, Rosa C, Calabrese F, Michaelsen P, Pedersen K, Mogensen P (2008) Qos-aware single cell admission control for utran LTE uplink, pp 2487–2491
21. Gozalvez J (2007) Ultra mobile broadband [mobile radio]. IEEE Veh Technol Mag 2(1):51–55
22. Bolton W, Xiao Y, Guizani M (2007) IEEE 802.20: mobile broadband wireless access. IEEE Wirel Commun 14(1):84–95
23. Ladebusch U, Liss C (2006) Terrestrial DVB (DVB-T): a broadcast technology for stationary portable and mobile use. Proc IEEE 94(1):183–193
24. DVB (November 2004) Transmission system for handheld terminals (DVB-H), ETSI EN 302304 v1.1.1
25. Kornfeld M, Daoud K (2008) The DVB-H mobile broadcast standard [standards in a nutshell]. IEEE Signal Process Mag 25(4):118–122, 127
26. Kornfeld M, May G (2007) DVB-H and IP datacast mdash; broadcast to handheld devices. IEEE Trans Broadcast 53(1):161–170

Chapter 3
Network Communications Protocols and Services

Abstract As the previous chapter has introduced network topologies, types, components, and major communication technologies, this chapter completes the network architecture description by presenting network protocols and various services supported by the current networks. The hierarchical organization of network protocols is detailed focusing on the most known reference models and the layered communication paradigm. Furthermore, the various protocol layers are detailed, especially at transport and application layers which involve protocols and services mostly detailed in this book. Last, but not least, the principles of the most popular network-based services are summarized, including electronic mail, Web, and the increasingly popular multimedia-based services.

3.1 Introduction

The previous chapter has introduced existing network topologies and communication technologies used to enable data exchange between network-interconnected remote hosts. Although communication technologies and network infrastructures are at the basis for message exchange between nodes, in order to fully support data exchange, a set of protocols has to govern the way messages are sent, routed, received, and interpreted by the communicating parties and the network devices. This chapter presents some of the most important communication protocols and discusses major network services.

3.2 Protocol Hierarchy

3.2.1 Network Reference Models

In order to reduce design complexity and allow for a better standardization process, network protocols are organized in layers (or levels), each layer providing a set of services to the layer immediately above and relying on services from the layer below.

B. Ciubotaru, G.-M. Muntean, *Advanced Network Programming – Principles and Techniques*, Computer Communications and Networks,
DOI 10.1007/978-1-4471-5292-7_3,

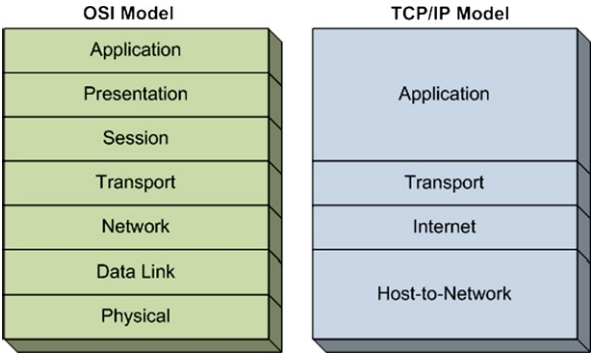


Fig. 3.1 OSI and TCP/IP reference models

The layered network architecture is organized in reference models; among these the most well known are the ISO Open System Interconnection (OSI) reference model and the TCP/IP reference model.

The OSI model is a theoretical model, and the protocols associated with its layers are rarely used. However, the model itself is widely used to present the concepts used in networking.

As opposed to the OSI, the TCP/IP model is less used for theoretical purposes, but the protocols associated with it are widely used in practice.

Figure 3.1 graphically shows the layers included in each reference model.

The OSI reference model includes seven layers: Physical, Data Link, Network, Transport, Session, Presentation, and Application. There layers are briefly introduced next.

- The *Physical layer* is responsible for transmitting raw bits over a communication channel.
- The *Data Link layer* is in charge of several tasks such as reliability, flow control, and medium access control for point-to-point data communication.
- The *Network layer* is mainly in charge of routing packets through sub-nets.
- The *Transport layer* offers end-to-end data communication services to upper layers.
- The *Session layer* allows users to establish sessions between them, each session offering services such as dialog control and synchronization.
- The *Presentation layer* is concerned with the syntax and semantics of the information (data) exchanged.
- The *Application layer* contains a variety of protocols specific to user applications.

Unlike OSI, the TCP/IP reference model has only four layers: Host-to-Network, Internet, Transport, and Application.

- The *Host-to-Network layer* corresponds to the Data Link and Physical layers from the OSI model, but the TCP/IP reference model does not detail this layer. However, the protocols used at this layer are specific to the network technology used to interconnect the physical user devices and network devices.
- The *Internet layer* corresponds to the Network Layer of the OSI reference model and similar to it, it is in charge of routing data packets through the sub-nets to

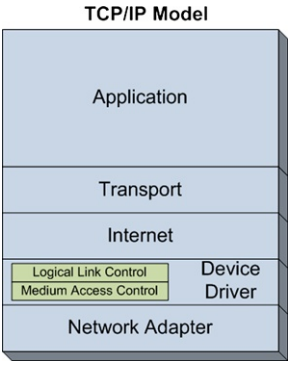


Fig. 3.2 A more realistic TCP/IP reference model

their destination. The widest used protocol residing at this layer is the Internet Protocol (IP).

- The *Transport layer* of the TCP/IP reference model corresponds with the same layer of the OSI model and offers similar services. The protocols residing at this layer are the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).
- The *Application layer* of the TCP/IP model is similar to the corresponding layer of the OSI model. Protocols residing at this layer include but are not limited to File Transport Protocol (FTP), electronic mail protocols (SMTP, IMAP, POP), Hypertext Transfer Protocol (HTTP), Domain Name System (DNS), Secure Shell (SSH), etc.

A more realistic reference model for TCP/IP is presented in Fig. 3.2. Although controversial, the host-to-network layer of the initial TCP/IP reference model has been split in two by some network specialists. These sub-layers are the Device Driver and Network Adapter sub-layers.

The Network Adapter layer corresponds to the physical layer of the OSI reference model and mainly consists of the hardware implementation of network interfaces.

The Device Driver layer contains two sub-layers, namely the Logical Link Control and the Medium Access Control. The Logical Link Control (LLC) offers the upper layers and the operating system access to the device driver. The Medium Access Control (MAC) is responsible for reporting and setting the device status, package outgoing data received from LLC in the format required by the network adapter, sending outgoing data at the appropriate time, receiving incoming data and unpacking it before verifying its integrity, and delivering it to the LLC sub-layer.

Figure 3.3 schematically presents the structure of the TCP/IP reference model and some of the network technologies and protocols involved.

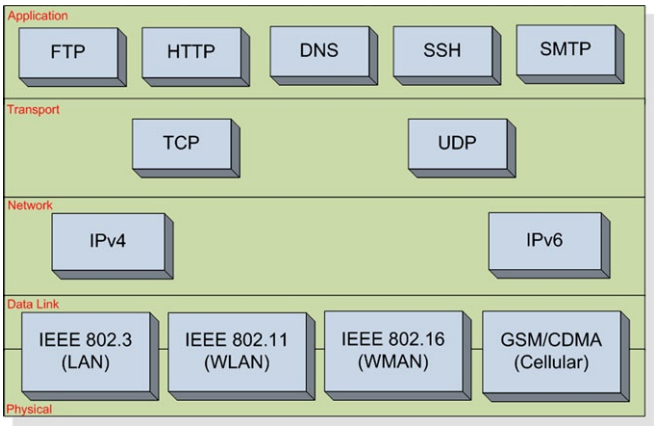


Fig. 3.3 Protocols and Networks specific to TCP/IP reference models

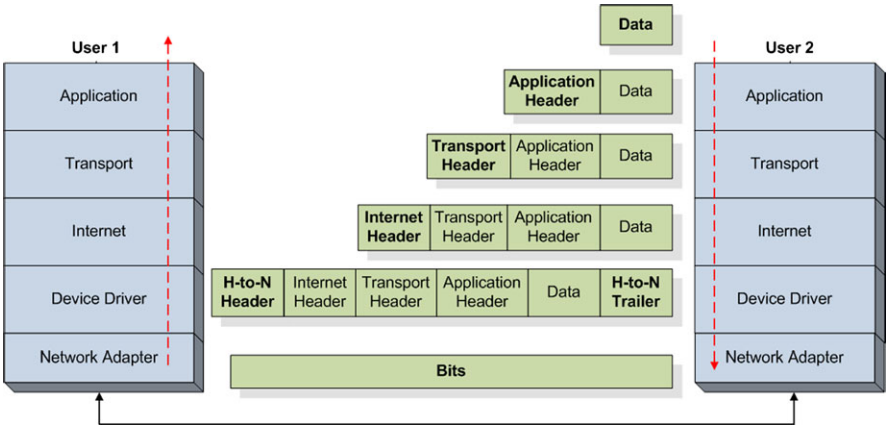


Fig. 3.4 TCP/IP reference model, data flow diagram

3.2.2 Layered Communication Paradigm

As mentioned in the previous sections, the layers of any reference model, including the TCP/IP model, rely on the services provided by the layers above and provide a set of services to the upper layer.

For example, the application layer protocols such as FTP use transport layer protocols such as TCP to carry the content of the files being transferred. The interaction between layers is done using dedicated interfaces which advertise the services provided by the particular layer.

Figure 3.4 schematically presents the data flow through the TCP/IP protocol hierarchy.

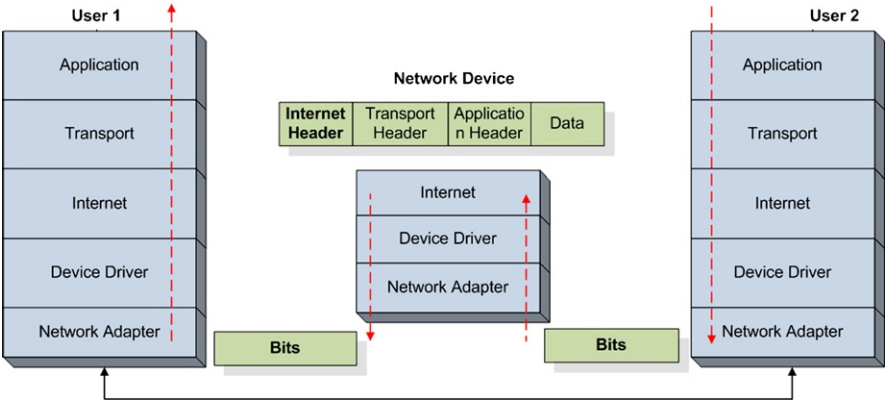


Fig. 3.5 TCP/IP reference model, data flow diagram through network devices

To exemplify, we consider two users, User 1 and User 2, exchange files using FTP. We assume that User 2 has an FTP server and the file repositories and User 1 has an FTP client and requests a certain file to be retrieved.

Figure 3.4 illustrates the file transfer process. The content of the file represents the Data. It is handed over to the application layer FTP protocol which adds its specific headers (Application Header). The FTP protocol may split the original file into chunks for transmission. FTP then hands over the data including the FTP headers to the transport protocol (i.e., TCP). The transport protocol splits the application data into packets, adds its own headers (transport layer headers) and hands over the packets to the Internet Protocol (i.e., IP). The Internet Protocol further adds its specific headers and injects the packets in the network, where the packets are routed towards the destination. The packets are sent via the network by being handed over to the Host-to-Network layer. This layer further adds its headers and trailers and manages the transmission of the raw bits representing the data packets to the next neighbor machine. The next neighbor machine is usually a network device mainly the LAN router. At the router level, the TCP/IP reference model is deployed up to the Internet level. As it can be seen in Fig. 3.5, at the network device level, the raw bits are decoded up to the Internet protocol level where routing is performed. The Internet layer changes the headers accordingly and re-injects the packets in the network by sending them to the next hop on the path towards the destination. At the receiver side, as it is presented in Fig. 3.4, the raw bits are received and are delivered from the physical layer across the layers to the user application. It can be seen that each layer removes its own headers before sending the data to the immediate upper layer.

From an application network programming perspective, all the details concerning layers below the application layer are hidden. For example, as it can be seen in Fig. 3.6, applications can use Sockets for accessing the data transport services offered by the transport layer. In these circumstances, next we discuss application and transport layer issues, ignoring the lower layers.

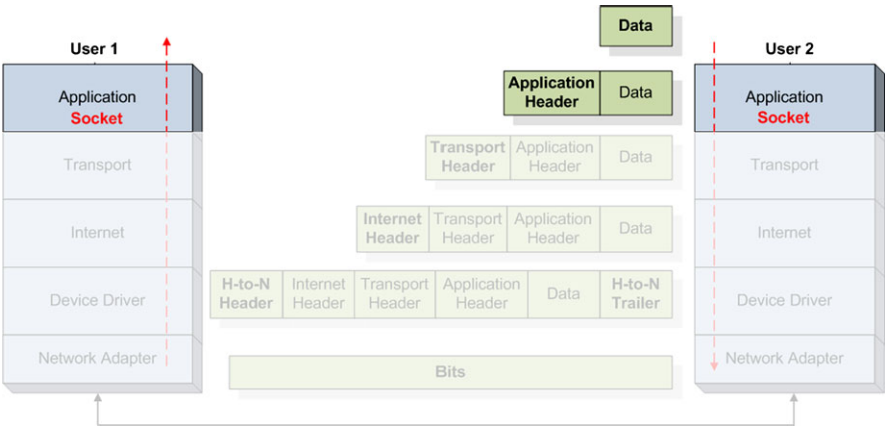
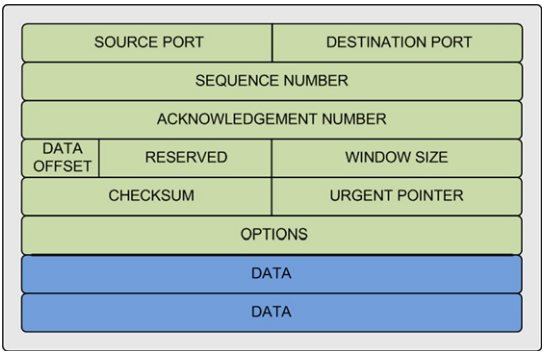


Fig. 3.6 TCP/IP reference model, transport layer programming interface

Fig. 3.7 TCP packet structure



3.2.3 Transport Layer

Transport layer protocols provide end-to-end data transmission and optionally provide functions such as congestion avoidance, reliability and flow control. Transport Control Protocol (TCP) [1] and User Datagram Protocol (UDP) [2] are two de-facto protocols employed at the TCP/IP transport layer. These two protocols are designed and widely deployed in wired network communication environments.

3.2.3.1 Transport Control Protocol

Transport Control Protocol (TCP) [1, 3] is a reliable, connection-oriented, congestion controlled byte stream data transfer protocol. A TCP packet consists of a 20 byte header followed by a payload as illustrated in Fig. 3.7.

The header includes a number of fields that enable the provision of key services. TCP uses 16 bit source and destination port number fields for multiplexing data to

various sending and receiving processes. The 32 bit sequence number field identifies the byte in the stream that the first byte of data in the segment represents. This field enables the reordering of out-of-order packets. The 32 bit acknowledgment field contains the sequence number of the next data segment the receiver expects to receive. This allows the sender to identify packets that have not been received, yet. These two fields are essential for providing a reliable delivery service. The 4 bit data offset/header length field specifies the length of the header. The 6 bit field is reserved for future use. Next, there are 6 flag bits. URG (U) is used to determine if the value in the urgent pointer field is valid. If set, the urgent pointer contains a sequence number offset, which corresponds to a TCP segment that contains urgent data and it should be expedited to its destination. ACK indicates if the acknowledgment number field is significant. It is used to by the receiver to inform the server that the packets it received are in order and intact. PSH is used to minimize the amount of buffering used before passing the data in this packet to the receiving process. The RST flag used to reset the connection, while the SYN and FIN flags are used for establishing and closing the TCP connection. The 16 bit window size field specifies the number of bytes each end of the connection is willing to accept, beginning with the one specified by the acknowledgment number. This field enables connection flow control. Finally, a checksum field covers the header and payload of the TCP segment.

Flow control is achieved by TCP using the window size field. This field identifies the number of bytes, starting with the byte acknowledged, that the receiver is willing to accept. If a receiver is busy or does not want to receive more data from the sender, this value can be set to 0. In addition to the flow control based on the window size, the current TCP standard (RFC 2581 [4]) uses a complex congestion control mechanism which involves four algorithms: Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery. The slow start algorithm employed by TCP tries to avoid congestion by starting the transmission at a low rate and fast increasing the rate until there is the first indication that the available bandwidth limit is being reached. Congestion avoidance further increases the rate gradually to a level acceptable given the existing bandwidth resources. Both slow start and congestion avoidance employ an Additive Increase Multiplicative Decrease (AIMD) approach, enabling the rate of transmitted data to increase incrementally, while the network is still capable of sustaining the current rate (i.e., no packet loss occurs). As soon as this rate exceeds the available network bandwidth (i.e., lost packets are detected), the sender dramatically reduces the data rate. Fast retransmit and fast recovery algorithms were introduced in order to speed up data delivery following loss and the consequent TCP drastic reduction in transmission rate.

TCP is used for a number of best effort applications, which rely on application-layer protocols such as the Hypertext Transfer Protocol (HTTP) for Web browsing and File Transfer Protocol (FTP) for file transfer. These applications are not time critical, but require guarantees that the integrity of the received data is maintained. For this reason TCP is not the preferred choice for streaming media. Streaming media requires video to be delivered in a timely manner and maintain relatively stable throughput, while also tolerate some loss. Some researchers proposed using

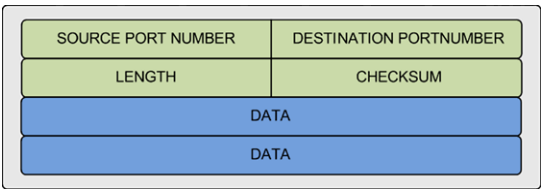


Fig. 3.8 Datagram structure

TCP for streaming media more than 10 years ago [5], but did not receive great attention at the time. However, the latest developments in network technologies have made TCP to be considered again for streaming multimedia, including as part of commercial implementations such as Apple's HTTP Live Streaming (HLS) [6] or the latest standard on Dynamic Adaptive Streaming over HTTP (DASH) [7].

3.2.3.2 User Datagram Protocol

User Datagram Protocol (UDP) [2] is a connectionless transport protocol. It provides the basic functionality required for applications to send encapsulated IP datagrams without having to establish a connection. A UDP datagram (see Fig. 3.8) consists of an 8 byte header followed by a payload.

The header consists of four 2 byte fields: source port, destination port, length, and checksum. The source and destination port fields provide required information to allow transport layer daemon processes to route packets to their correct destination application. This multiplexing/demultiplexing feature is the main benefit UDP has over raw IP datagrams. The 16-bit length field specifies the length of the datagram in bytes of the entire datagram (header and data). The field size sets a theoretical limit of 65,527 bytes for the data carried by a single UDP datagram. Finally, a 16-bit checksum field is used for error-checking of the header and data. UDP does not provide any reliability or congestion control features. As a result applications using this protocol must generally be willing to accept or deal with loss, duplication or out-of-order delivery and rely on network-based mechanisms to minimize potential of congestion collapse. The majority of applications using UDP often do not require reliability mechanisms and may even be hindered by them. Applications requiring high degrees of reliability should use a reliable protocol (e.g., TCP). These characteristics make UDP well suited for real-time multimedia streaming applications.

3.2.3.3 TCP/IP and Wireless Networks

As opposed to wired communications where packets not acknowledged by recipient within the expected deadline are supposed to be lost due to network congestion and buffer overflow, packet loss in wireless communications may be caused by interference, noisy channel, etc., which does not necessarily imply congestion.

UDP is a datagram-oriented protocol that provides no delivery guarantee to upper layers, and does not provide any support mechanism for congestion detection or

reliability control. This is why UDP is not suitable for use for services requiring transport reliability such as e-mail or file transfer applications.

Several studies [8–10] on the performance evaluation of these two protocols have shown that there are various performance issues when using them for data transport over wireless communication networks.

Several variants of TCP have been proposed, each making improvements in terms of energy consumption, network throughput, and reliability.

TCP Tahoe [4, 11] mainly contributes in the design of slow start, congestion avoidance, and fast retransmission, and is the first protocol to include congestion control and thus is energy efficient for bursty error which happens quite often in wireless sensor networks. TCP Reno [12] implements the three functions of Tahoe and adds additional fast recovery mechanism. TCP New-Reno [13] modifies the fast recovery scheme. The fast recovery function detects packet loss and initiates retransmission without the timeout signal required by traditional retransmission policies. In this case, it provides shorter delay and better quality for multimedia streaming applications. SACK [14] uses selective ACK instead of cumulative ACK to indicate successful transmission of specific packet, thus the sender is able to figure out which packets are lost and save the energy for redundant retransmission; and simulation results show that incorporating SACK in TCP achieves better performance in terms of packet delay and throughput [15]. SACK is supposed to be energy efficient as it decreases the number of unnecessary retransmissions; however, the study in [16] points out that the energy gain is neutralized by the extra overhead. Vegas [17] modifies the congestion control scheme and adapts the transmission rate at the sender side according to the observed Round Trip Time (RTT), and WestwoodNR [18] differentiates the causes of packet loss, i.e., traffic congestion or error-prone wireless channel, and adapts the congestion window size at the sender side accordingly.

3.2.4 Application Layer

The application layer provides most of the functionality required by the user in terms of its direct interaction with the network-based services. The lower layers, including the transport layer, provide data transport services to the application layer. Although the transport layer protocols support data exchange services, their functionality is still too basic for the applications layer. As a consequence various protocols have been developed and deployed at the application layer in order to support the requirements of the highly diverse user applications. Some of these protocols will be discussed in this section.

3.2.4.1 The Domain Name System

In order to initiate a TCP connection or to send UDP datagrams to a host in the network, one needs to know the host's IP address. The main disadvantages of addressing a host by its IP address include the less user-friendly format of IP addresses

Fig. 3.9 DNS resource record structure

(i.e., numerical form) and the possibility that a host will have its address changed by the network administrator.

To solve this issue, textual names (i.e., domain names) have been introduced to decouple the name of a host from its IP address. Consequently, user applications will address a specific host by its name and not its IP address. However, the underling network still uses IP addresses to exchange and route data packets.

The Domain Name System, best known as DNS, is a distributed system storing records about domain names and host machines. A standard DNS resource record contains the fields presented in Fig. 3.9, which are briefly introduced next.

The *Domain Name* represents the domain the record refers to and is the main query parameter when a DNS server is interrogated for particular records.

The *Time to Live* parameter is an indicator of how stable the record is expected to be. This is mainly relevant for cached records which may soon become outdated.

Class is usually "IN" for Internet information. Other codes can be used for non-Internet information.

The *Type* denotes the king of recorder, among other one of the most important is "A" which represents the IP address.

The *Value* can be a number, a domain name, or an ASCII string.

The following is an example of a DNS record specifying the IP of the host to which the domain *pel.eeng.dcu.ie* refers to. The time to live for this record is 86400 seconds which represents 24 hours. This is a stable record. For less stable records, the time to live field may be set to 60 seconds.

pel.eeng.dcu.ie 86400 IN A 10.10.105.189

Theoretically, a single server could store the DNS records for the whole Internet. In reality, this would quickly lead to this server being overloaded by a huge amount of requests and eventually fail. Moreover, having one single server delivering DNS service poses significant reliability problems risking to bring the whole Internet activity to a halt.

As a consequence, the DNS space is organized in a tree-like structure as illustrated in Fig. 3.10.

The top-level contains generic domains such as .com, .edu, .net, etc., and country domains such as .us, .fr, .ie, .de, etc. Each of the top-level domains is the root of a tree of sub-domains. A leaf domain is a domain that does not have any sub-domains and may represent a host or a organization with hundreds of hosts.

The domain name tree is organized in zones and each of these zones is served by a primary name server and several secondary ones.

There are two types of queries supported by named servers. To exemplify, we consider the host in Fig. 3.10, running a client application willing to initiate a TCP connection with a server running on the host represented by the domain name

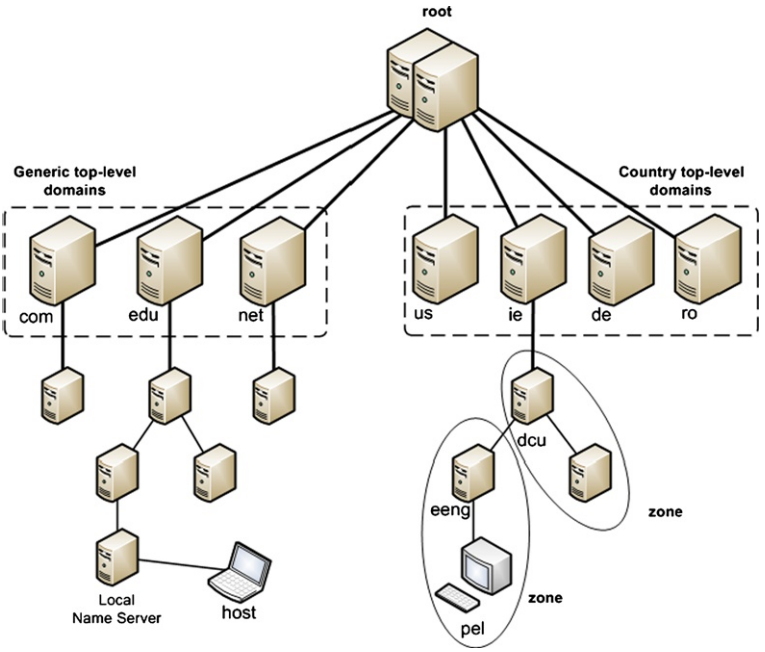


Fig. 3.10 DNS space

pel.eeng.dcu.ie. The client application will call a resolver procedure passing the domain name as a parameter. The resolver will use UDP to send local name server a request for the IP corresponding to *pel.eeng.dcu.ie*. Assuming this domain has never been accessed from the host machine before, the local name server will not have any records of it.

Depending on the type of query (recursive or non-recursive) the local name server will forward the query to the top-level name server (i.e., ie) or will reply with the address of the top-level name server.

For the rest of the example will assume the query is recursive. The local name server forwards the query to the *.ie* name server. The top-level server does not have records of the leaf domains but has records of the next level sub-domains. Consequently, it forwards the request to *dcu* domain server which further forwards the request to *eeng* name server. The *eeng* name server retrieves the authoritative record from its database and forwards it to the originator of the query which further returns the record towards the local name server of the client host. The local name server sends the record to the resolver of client application. The authoritative record comes from the server that manages the domain (i.e., *eeng* name server) is always up to date. In the context of the presented example, the client host's local name server caches the record for quick future name resolution (the cached record will be kept for as long as the Time to Live parameter specifies, in order to avoid stalled data).

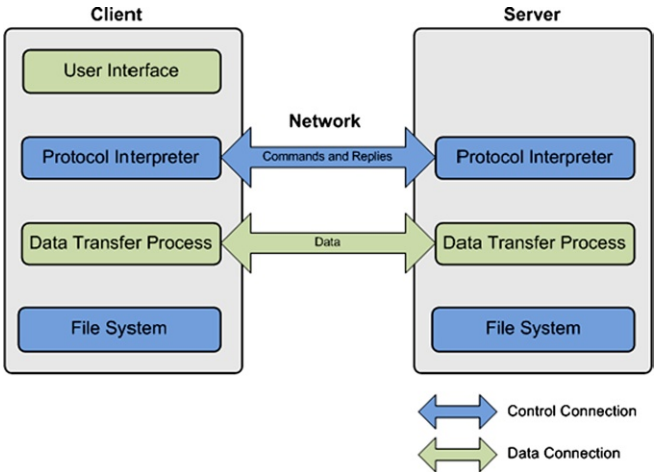


Fig. 3.11 FTP session

3.2.4.2 File Transfer Protocol

File transfers account for large amounts of data exchange over the Internet. File transfers involve clients transferring file content in a reliable manner and efficiently to and from servers, but also data exchange between peers in peer-to-peer settings. In general, file transfers are performed using the File Transfer Protocol (FTP).

FTP was developed in 1985 and is still widely used today. The protocol has been first defined in RFC 959, but then several extensions have been proposed to enhance flexibility and security (RFC 1579, RFC 2228) [19–21]. FTP works on top of TCP and in general uses port 21; however, the administrator may choose to use different ports.

The client connects to the server and sends commands, and the server responds with command status messages. In general, each session involves at least one file transfer. The basic principle of file transfers using FTP is outlined in Fig. 3.11.

FTP involves two connections: control and data connections. FTP commands and replies are exchanged via the control connection, while data is exchanged over the data connection. Control connection must be working when the data is transferred. In practice, a single connection is used for both data and control.

Commands can be grouped in three categories.

- Access control commands include:
  - USER—indicates the user;
  - PASS—indicates the password;
  - CWD—change directory;
  - CDUP—change directory to parent;
  - QUIT—logout.

- Transfer parameter commands include:
  - PORT—publish local data port;
  - PASV—server passive (listen);
  - TYPE—indicate data representation (A-ASCII, E-EBCDIC, I-Image, L-Local);
  - MODE—indicate transfer mode (S-Stream, B-Block, C-Compressed);
  - STRU—establish file structure (F-FILE, R-RECORD, P-PAGE).
- Service commands include:
  - RETR—retrieve file;
  - STOR—send and store the file remotely;
  - APPE—send file and append;
  - DELE—delete the file;
  - MKD—make a new directory;
  - RMD—remove a directory;
  - PWD—print working directory;
  - LIST—list files.

Every command must generate at least one reply from the server. This enables the synchronization of requests sent by clients and actions performed by the server and also allows the clients to know the server status. In general, the reply is a single line; however, multiple lines are also accepted. The reply must contain a three digit status code which enables machines to assess server status and a text message which describes the server status in human language.

There are several issues when using FTP for file transfers. Security is an important issue for many companies that have installed firewalls. Firewalls prevent unauthorized users from getting access to the networks. However, firewalls may also inadvertently prevent valid users from accessing some resources. When FTP is involved the network administrators must design rules for classes of FTP connections which may be a costly and error prone process.

Another issue is standardization. There are many FTP client applications with different interpretations of the FTP protocol. Consequently, FTP server administrators must know how to support all of these different client classes.

An alternative to FTP is Web-based file transfer. A Web-based file transfer client runs within the Web browser. There is no need for any software to install, license to purchase or software to maintain. Additionally, there is no need to set-up firewall rules for each user class.

3.3 Services

3.3.1 Electronic Mail

Nowadays, the e-mail service is one of the most used means of electronic communications. It involves users sending messages to other users via the network. In

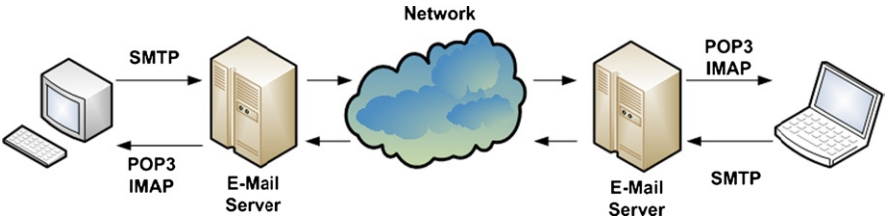
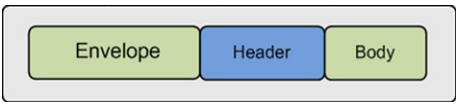


Fig. 3.12 E-mail message exchange

Fig. 3.13 E-mail message format



practice, client machines enable messages to be written and sent to local e-mail hosts (e-mail servers), which communicate with remote clients' e-mail hosts. Remote clients enable the contact with the remote e-mail servers for the messages to be retrieved and then read by the remote users.

The e-mail system is composed of a User Agent (UA) and a Message Transfer Agent (MTA). The UA allows users to send and retrieve messages and may also provide a graphical user interface. The MTA transfers the messages from the source to the destination.

The e-mail system requires several services to be provided by the two agent components: UA and MTA. Composition is provided by UA and refers to the creation of messages and reply messages. Transfer is ensured by MTA and refers to delivering the messages from source to destination. Reporting, also provided by MTA, involves informing the sender about the status of the messages sent. Displaying is provided by UA through the user interface and involves displaying the message so that it can be accessed by the user. Depending on the type of content, sometimes the messages need to be converted before displaying. Often another program is invoked, such as plug-in (embedded in the mail client application) or application (independent from the mail client application). Disposition, managed by the UA, refers to what the remote user does with the message (e.g., save, delete, etc.).

Figure 3.12 shows the basic principle of the e-mail service.

The e-mail message structure involves an envelope, a header, and a body, as outlined in Fig. 3.13 and is formalized in RFC 822 and RFC 2822. The envelope encapsulates the message and contains all info required to transport the message such as destination address, priority, and security level. The header contains the control information required to display the message (e.g., date, subject). The body represents the content useful to the human user.

Multipurpose Internet Mail Extensions (MIME) is a standard (RFC 1341, RFC 2045-2049) that extends the format (RFC 822) of the e-mail messages to support extra features and encoding rules for non-ASCII messages. These features include characters with accents (e.g., in French, German, etc.), text in non-Latin al-

phabets (e.g., Cyrilic, Hebrew, etc.), text in non-alphabetic languages (e.g., Chinese, Japanese), non-text data (e.g., multimedia, images, audio).

MIME defines five new message headers. The new headers include:

- MIME-Version—Indicates MIME version;
- Content-Description—String describing the content;
- Content-Id—Unique identifier;
- Content-Transfer-Encoding—How body is wrapped for transmission (e.g., 7-bit ASCII, 8-bit codes, base64 binary, etc.);
- Content-Type—Type and format of content, RFC 2045 defines 7 types, including Text (Plain, Enriched), Image(Gif, Jpeg), Audio (Basic), Video (Mpeg) Application, Message (RFC 822, Partial, External-body), Multipart.

Messages not including MIME-Version header are assumed to be in English plain text.

Simple Mail Transfer Protocol (SMTP), standardized in RFC 821, allows messages to be sent from UA to MTA. SMTP works on top of TCP and in general uses port 25.

The client initiates the TCP connection with the server and waits for the server to state it is ready. After the server confirms it is ready, the communication sequence commences. The client sends commands and the server responds with command status messages. Status messages include ASCII encoded numeric codes and details in text. The order of the commands is very important for the success of the message sending operation.

The SMTP commands include:

- HELO—identifies client;
- MAIL FROM:—starts a mail transfer session and identifies the mail sender;
- RCPT TO:—identifies one recipient; there may be multiple RCPT TO: commands;
- DATA—sender ready to transmit a series of lines of text, each ending with CR&LF. A line containing only a period "." indicates the end of the data;
- QUIT—request to finish the session and close the connection.

Extended SMTP (ESMTP) was defined in RFC 2821. EHLO is the new command for identifying the client. Only ESMTP servers accept extended hellos. An SMTP server rejects this command, thus the client will know what type of server it communicates with. Other set of commands and parameters are defined, too.

Post-Office Protocol version 3 (POP3), standardized in RFC 1939, allows messages to be accessed by the client software (UA) on the e-mail server (MTA).

POP3 works on top of TCP and in general uses port 110. The protocol message sequence includes the following stages. After the client connects to the server, it waits for the server to state it is ready. Once the server confirms its availability, the client starts sending commands, which determine the server to perform actions and respond with status messages.

POP3 requires sequential passing through three states: authorization, transaction, and update.

Table 3.1 IMAP vs. POP3

Protocol	RFC	TCP Port	Email store	Email read	Mailboxes	Partial message
POP3	1939	110	Client	Offline	Simple	No
IMAP	2060	143	Server	Online	Multiple	Yes

During the *authorization* phase the client sends username and password details to the server. The following commands are involved:

- USER username—identifies the username;
- PASS password—indicates the password.

During the *transaction* phase the client retrieves the list of messages or a particular one. The client may mark for deletion some of the messages. The following commands are involved:

- LIST—lists e-mails received in order;
- RETR no—retrieves message number no;
- DELE no—marks for deletion message number no.

During the *update* phase, the QUIT command is sent (QUIT starts the update), and the server finishes deleting all the messages marked for deletion, then it sends a disconnect message and disconnects the client.

POP3 allows the client to download the messages locally (on the client machine) and manipulate them offline.

Internet Message Access Protocol (IMAP), standardized in RFC 2060, is also used to access the e-mail messages on the server.

IMAP works on top of TCP and in general listens at port 143. The protocol message sequence includes the following stages. After the client connects to the server, it waits for the server to state it is ready. After the server confirms its status, the client sends commands and the server responds with status messages, after performing the required actions.

IMAP assumes that the server keeps all messages and the client accesses them online. IMAP enables the user to use multiple mailboxes and permits e-mail access from multiple locations.

Specific commands are defined by IMAP for searching messages, reading messages or part of them, creating, manipulating multiple mailboxes, addressing an e-mail by attributes (e.g., from source), etc.

Table 3.1 summarizes the main differences between IMAP and POP3.

3.3.2 The World Wide Web

The World Wide Web or the Web, as it is widely known, represents a framework allowing client machines to access linked documents spread over millions of servers all over the Internet.

Figure 3.14 show the principle of accessing web documents over the Internet.

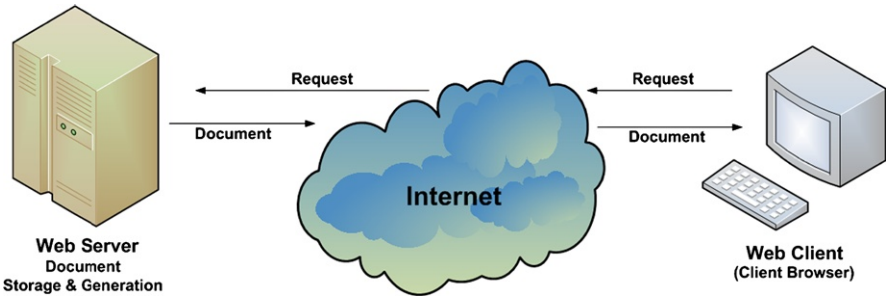


Fig. 3.14 World Wide Web document delivery process

The web documents or web pages, or just pages, consist of a collection of text, images, and lately video content. A web page may also contain links to other web pages. These links are called hyperlinks and can be attached to most of the elements of a web page but mostly to text and images.

The web pages are displayed on the client machine by an application called a browser. Internet Explorer, Firefox, and Chrome are among the most popular.

Web content transfer accounts for most data transfers over the Internet. It involves communications between Web clients (browsers) and Web servers where clients request a piece of Web content from servers and the servers respond delivering it (Fig. 3.14). In general, a series of response requests are part of a web communication session.

The web pages are stored or generated by a web server and are delivered to the client on request. The protocol used by the web client (browser) to interact with a web-server is Hypertext Transfer Protocol (HTTP). HTTP is standardized in RFC 1945 and RFC 2616, and works in general on top of TCP. HTTP uses in general port 80, but other ports can be used as well. There may be one (HTTP v.1.0) or multiple simultaneous connections (HTTP v. 1.1) initiated by the client to the server. Client sends commands and server responds with command status messages. In general, each session involves at least one request response.

Web pages or documents can be classified into three categories: Static, Active, and Dynamic.

Static documents are identically delivered at every request and to any user. These documents are modified by replacing the original file on the server. These documents are created using languages such as HTML, XML, XHTML, CSS, XSL, and are easy to create. Fast to retrieve, these documents do not require much processing on the server or client. Being static, these types of documents can be cached on the client's machine or in nearby servers for faster delivery.

Despite the performance advantages, static documents are difficult to maintain consistent and up to date, offer little user personalization and are not suitable to create large sites.

Active documents are static documents containing executable code which is executed at the client, basically by the browser. Most common executable code executed at the client side is Javascript and Java Applets.

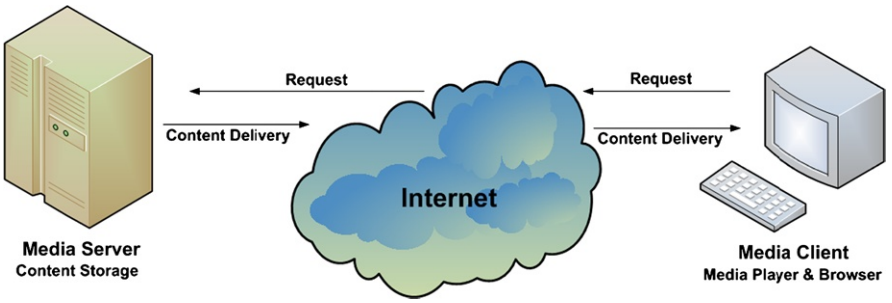


Fig. 3.15 Multimedia content delivery process

Among the advantages are user interactivity, limited user personalization, data display customization, cache friendly, and distributed resource requirements (the code runs on the client machine hence no server resource is required).

Among the disadvantages is the fact that the user runs unknown code which may pose security issues. Also, running code on the client machine may lead to increased delays depending on the performance of the machine used.

Dynamic documents are generated on the fly by the server, at client request. This type of document enables user personalization, supports database access, data display customization and can use time and date sensitive code.

Among the disadvantages of using dynamic documents are their complexity, high resource requirements on the server side and the fact that they are not cache friendly.

Among the most popular technologies for dynamic documents processing are Hypertext Processor (PHP), Java Servlets, and Java Server Pages (JSP).

3.3.3 Multimedia-Based Services

Multimedia represents content of different forms including text, images, audio, video, and animations. Multimedia content is increasingly popular nowadays and accounts for a high share of the data traffic transported over the Internet.

Even the web documents discussed in the previous section contain at least some images and text, which turns them into multimedia content.

However, multimedia applications include a wide range of scenarios from IP Television-to-media streaming to hand-held devices to delivering web documents including images and embedded video or animations.

At its basics, a multimedia application involves various types of content transferred over the network between a media server (can be a web server) and a media client, as presented in Fig. 3.15.

As mentioned before, multimedia includes various forms of content, each type have different requirements in terms of network bandwidth and timely delivery.

Text, usually requires low bandwidth and no real-time constraints. Text is usually required for subtitles, annotations, and meta-data as well as standard content

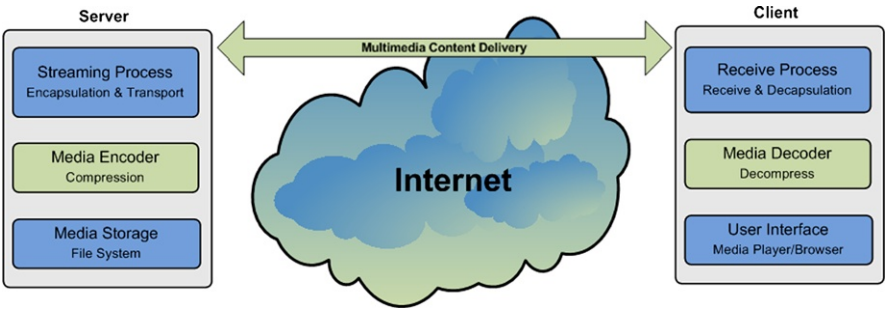


Fig. 3.16 Steps involved by multimedia content delivery process

in web document. Audio requires relatively low bandwidth and has real-time constraints. Still images require higher bandwidths (depending on the size and encoding of the image) and no real-time constraints. Animation consists of a set of still images displayed successively and require higher bandwidth (depending on the size and encoding of the image) and no real-time constraints.

One of the most prevailing form of content involved by multimedia applications is video content. Video content consists of a sequence of still images named frames displayed in a predefined order and at precise timing to create the illusion of motion. It requires high bandwidth and has real-time constraints. The raw frames of a video clip would require a huge amount of data storage, even for today standards, and would be impractical to transfer over the network. To exemplify, a 120 minute video in standard VGA (640×480) resolution at 25 frames per second requires about 154 gigabytes of storage.

In order to reduce the amount of data and make video storage and streaming effective, compression techniques have to be used. Consequently, during video encoding, compression algorithms are employed to reduce the amount of data required to store and transport the video data.

As a consequence, streaming video content from a server to a client involves three main steps: compression, encapsulation, and transport. The process is illustrated in Fig. 3.16.

Video compression relies on a good understanding of the human psycho-visual perception system which allows for the exploitation of redundancies in the video signals. Compression can be lossless or lossy, depending of the possibility to recover the original image identically or not.

Video compression standards have been developed by the Moving Picture Experts Group (MPEG) and International Telecommunication Union (ITU).

MPEG compression standards include:

- MPEG-1—Combined audio–video signal, average bit-rate of 1 Mbps in Standard input format (SIF), 352×288 pixels at 25 frames/s or 352×240 pixels at 29.97 frames/s;
- MPEG-2—Compression of standard definition (SD) and high definition (HD) interlaced video signals. Very high bitrates (up to 20 Mbps) and high picture quality;

- MPEG-3—Addressed HDTV compression, was discontinued;
- MPEG-4—Consists of two distinct compression algorithms: MPEG-4 Part 2 (Visual) and MPEG-4 Part 10 (AVC).

ITU compression standards include the H.26x family:

- ITU-T R. H.261—Teleconferencing and videophone applications, ISDN lines as the transport network infrastructure. Bitrates range from 40 Kbps to 2 Mbps in multiplies of 64 Kbps;
- ITU-T R. H.262—It is identical with the MPEG-2 standard;
- ITU-T R. H.263—Similar to H.261, provides better performance and flexibility. Low bit-rate (below 64 Kbps), however, this target has been relaxed;
- ITU-T R. H.264—Identical with MPEG-4 (AVC). Has become a key technology for multimedia applications. H.264 provides good video quality while substantially reducing the bit rates and latency.

Proprietary compression solutions include:

- VC-1 SMPTE 421M—Standardized by the Society of Motion Picture and Television Engineers (SMPTE). Video codec specification in the next generation optical media formats, such as HD-DVD and Blu-ray. It is developed by Microsoft and was originally known as the Microsoft Windows Media 9;
- Audio Video Coding Standard (AVS)—Audio Video coding Standard Workgroup of China. AVS Part 2 designed for HDTV. AVS Part7 (AVS-P7) for low complexity, low picture resolution applications for the mobile environment;
- Apple QuickTime—Developed by Apple;
- Real Media—Developed by Progressive Networks.

Compression converts data to be stored efficiently. Encapsulation wraps the compressed data in a container which specifies how the data should be stored, transported, and displayed. Encapsulated multimedia content includes video and audio streams, meta-data, subtitles, and synchronization information. Multimedia container formats include audio container formats and flexible container formats.

Audio container formats include:

- Audio Interchange File Format (AIFF) (Mac OS);
- Waveform Audio File Format (WAV) (Windows);
- MPEG-1 or MPEG-2 Audio Layer III (MP3).

Flexible containers include audio, video and other types of data:

- 3GP—3G Mobile phones (Third Generation Partnership Project);
- AVI—Audio Video Interleave (Microsoft Windows container);
- FLV—Internet video delivery with Flash Player (Adobe Systems);
- MOV—QuickTime File Format (Apple Inc.);
- MPEG-TS—MPEG-2 transport stream for digital broadcasting and for transportation over unreliable media;
- MP4—MPEG-4 Part 14, audio and video container for MPEG-4.

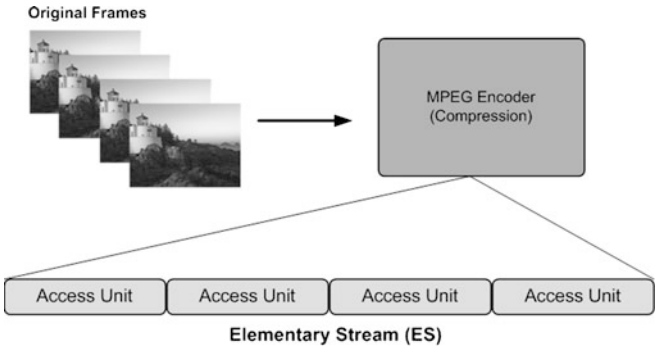


Fig. 3.17 MPEG-2 elementary stream

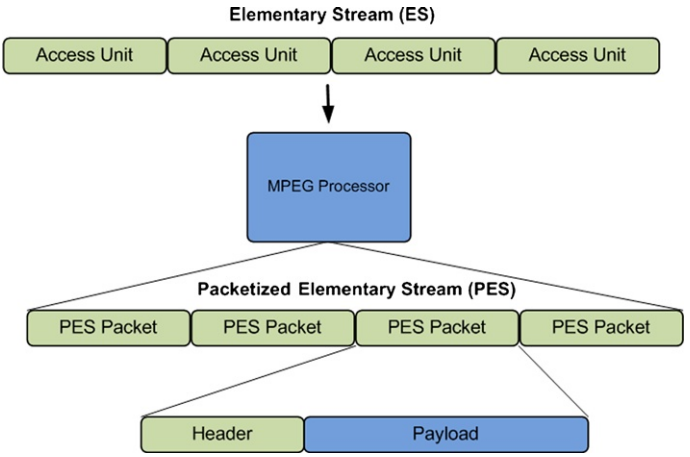


Fig. 3.18 MPEG-2 packetized elementary stream

To exemplify we present the MPEG-2 encapsulation process.

The original video frames are first compressed by the MPEG-2 video compressor. The result is split in access units. An access unit represents the fundamental unit of encoding; for video this is usually an encoded frame. The process is shown in Fig. 3.17.

The compressed content stored in access units is further split into a Packetized Elementary Stream.

The format of the packets is presented in Fig. 3.18.

The payload contains an integral number of access units. The header consists of a packet start code prefix (3 bytes), stream ID (1 byte), PES packet length (2 bytes), optional PES header (variable length), and stuffing bytes.

The optional PES header includes:

- Data Alignment Indicator—The payload starts with video or audio;
- Copyright Information—Copyright protected;

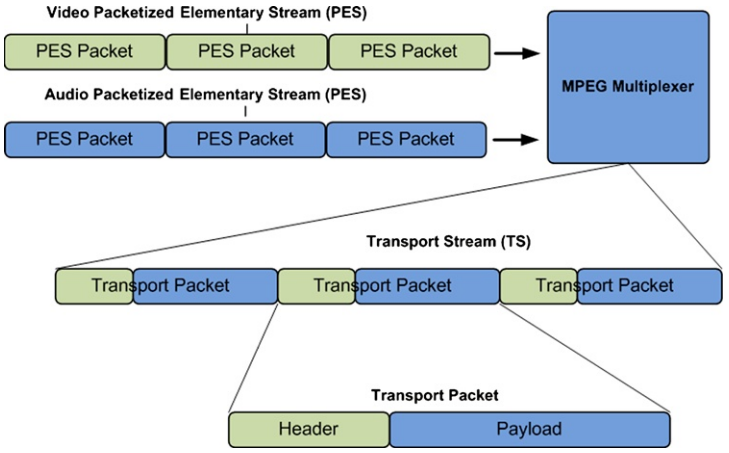


Fig. 3.19 MPEG-2 streaming process

- Presentation Time Stamp (PTS)—synchronizes a set of elementary streams and controls the playback rate;
- Elementary Stream Clock Reference (ESCR);
- Elementary Stream Rate—the ES encoding rate;
- CRC—Monitors errors in the previous PES packet.

Multimedia content delivery or transport may use one of three major techniques: broadcast, unicast, or multicast content delivery.

Broadcasting is cost-effective in terms of bandwidth and resource utilization. However, video-on-demand-like services decrease broadcasting popularity.

Unicast supports video-on-demand services as well as broadcast services. Network resources are used only when necessary and support content adaptation for each user separately.

Multicast is beneficial for group content delivery in applications such as video conferencing. However, the management of multicast groups is difficult and complex.

The process of streaming the packetized elementary streams is presented in Fig. 3.19. The packetized elementary stream is further split into transport packets, each consisting of a header and a payload.

The transport packets have fixed sizes of 188 bytes and a payload of 184 bytes. The header of 4 bytes consists of:

- Synchronization byte—0x47 (0100 0111);
- Three flag bits—Transport error, Payload Start Indicator, Priority;
- Packet Identifier (PID)—13 bits;
- Scrambling control—2 bits, used for payload encryption.
- Adaption Control (2 bits):
  - 01—No adaptation field, payload only;
  - 10—Adaptation field only, no payload;

  - 11—Adaptation field followed by payload;
  - 00—RESERVED.
- Continuity Counter (4 bits).

Multimedia content can be distributed over any transport protocol such as TCP and UDP. For real-time multimedia streaming, where a certain level of packet loss is acceptable, UDP is preferred over TCP. TCP's congestion control and reliability may affect the real-time delivery of transport stream packets. In the context of real-time streaming of video or audio data, a packet arriving late is as good as a lost packet.

Congestion represents a major issue especially in the context of bandwidth-hungry multimedia applications. Adaptive multimedia streaming applications alter both the encoding and transmission process parameters in order to reduce the amount of data required to describe the content and consequently to be delivered in order to match the available network capacity.

Protocols specific to real-time data delivery were also developed.

Real-time Transport Protocol (RTP) is used for delivering multimedia data over the IP networks. RTP uses transport layer protocols such as UDP and is considered an application layer protocol which delivers multimedia data itself.

Real-time Transport Control Protocol (RTCP) controls data delivery over RTP. RTP and RTCP use different port numbers (even and odd). RTCP delivers control packets carrying information such as throughput, loss, and jitter, information which is not used by RTP, but it is usable by the application directly (bit-rate adaptation). In this context, RTP cannot guarantee the Quality of Service (QoS) at all.

Real Time Streaming Protocol (RTSP) enables a client to have the features of a VCR, such as play, stop, and pause. It is used in conjunction with RTP for delivering multimedia data.

3.4 Conclusions

This chapter has presented the network protocol stacks for the theoretical ISO OSI reference model and the practical TCP/IP model, and has described the layer-based hierarchical data delivery paradigm. As network programming mostly concerns application and transport layers, most relevant protocols at these layers were presented.

The chapter has also described in details services as highly important components of the network framework and has discussed the e-mail, World Wide Web, and multimedia-based services as the most relevant.

The next chapter introduces basic network programming aspects in the context of supporting these services.

References

1. Postel J (1981) Transmission control protocol. Edited by Jon Postel. Available at http://rfc.sunsite.dk/rfc/rfc793.html

2. Postel J (August 1980) User datagram protocol. RFC 768, Internet engineering task force
3. Ramakrishnan K, Floyd S, Black D (2001) The addition of explicit congestion notification (ECN) to IP
4. Allman M, Paxson V, Stevens W (1999) TCP congestion control
5. Krasic C, Li K, Walpole J (2001) The case for streaming multimedia with TCP. In: 8th international workshop on interactive distributed multimedia systems (iDMS 2001), pp 213–218
6. Pantos R (October 2012) E.W.M.: Apple's http live streaming. Technical report, International internet draft
7. Stockhammer T (2011) Dynamic adaptive streaming over http –: standards and design principles. In: Proceedings of the second annual ACM conference on multimedia systems. MMSys '11. ACM, New York, NY, pp 133–144
8. Sandeep (2001) An experimental study of TCP's energy consumption over a wireless link. In: European personal mobile communications conference, IEEE
9. Zorzi M, Rao RR (2001) Energy efficiency of TCP in a local wireless environment. Mob Netw Appl 6:265–278
10. Giannoulis S, Antonopoulos C, Topalis E, Athanasopoulos A, Prayati A, Koubias S TCP vs. UDP performance evaluation for CBR traffic on wireless multihop networks
11. Jacobson V (1995) Congestion avoidance and control. SIGCOMM Comput Commun Rev 25:157–187
12. Jacobson V (April 1990) Modified TCP congestion avoidance algorithm. end2end-interest mailing list
13. Hoe JC (June 1995) Start-up dynamics of TCP's congestion control and avoidance schemes. Master's thesis, Massachusetts Institute of Technology
14. Mathis M, Mahdavi J, Floyd S, Romanow A (October 1996) TCP selective acknowledgment options. RFC 2018 (proposed standard)
15. Fall K, Floyd S (1996) Simulation-based comparisons of Tahoe, Reno and SACK TCP. SIGCOMM Comput Commun Rev 26:5–21
16. Seddik-Ghaleb A, Ghamri-Doudane Y, Senouci SM (2006) A performance study of tcp variants in terms of energy consumption and average goodput within a static ad hoc environment. In: Proceedings of the 2006 international conference on wireless communications and mobile computing. IWCMC '06. ACM, New York, NY, pp 503–508
17. Brakmo LS, O'Malley SW, Peterson LL (1994) In: TCP Vegas: new techniques for congestion detection and avoidance, pp 24–35
18. Mascolo S, Casetti C, Gerla M, Sanadidi MY, Wang R (2001) TCP westwood: bandwidth estimation for enhanced transport over wireless links. In: Proceedings of the 7th annual international conference on mobile computing and networking. MobiCom '01. ACM, New York, NY, pp 287–297
19. Postel J, Reynolds J (October 1985) File transfer protocol. RFC 959 (standard) Updated by RFCs 2228, 2640, 2773, 3659, 5797
20. Bellovin S (1994) Firewall-friendly FTP
21. Horowitz M, Lunt S (October 1997) FTP security extensions. RFC 2228 (proposed standard)

Chapter 4
Basic Network Programming

Abstract This chapter introduces some of the basic principles used for developing network-based applications. Multi-programming and multi-tasking paradigms are introduced as two of the basic concepts of programming. Threads and processes are discussed, emphasizing multi-threaded application development in Java. Inter-thread and inter-process communication techniques and paradigms are also presented, as some of the basic mechanisms for network applications communication.

4.1 Introduction

Network application programming uses high level programming languages and involves a set of principles and techniques. Implementation requires access to various APIs and support from different application development environments. This chapter introduces some of the basic principles used for developing network-based applications using Java programming language. However, the basic concepts and techniques remain the same regardless of the programming language employed and represent the basis for building data communication-based applications.

This chapter discusses the concept of multi-programming, which involves multi-tasking, and presents how it is implemented in standard operating systems. Multi-programming is a very important technique to both achieve computation parallelism and exploit the multi-core architectures of most of the current processors. The chapter also describes some basic aspects of the multi-programming paradigm including processes, threads, inter-thread communication and synchronization, and inter-process communications.

4.2 Multi-programming and Multi-tasking

Originally, uni-programming was the solution of choice for technical reasons. It involves only one user program running on any computer at a time. This was a feasible solution for the early computers which were, in fact, processing machines dedicated to performing a single critical task such as bulk data processing, statistical data analysis, or enterprise resource planning.

B. Ciubotaru, G.-M. Muntean, *Advanced Network Programming – Principles and Techniques*, Computer Communications and Networks,
DOI 10.1007/978-1-4471-5292-7_4, © Springer-Verlag London 2013

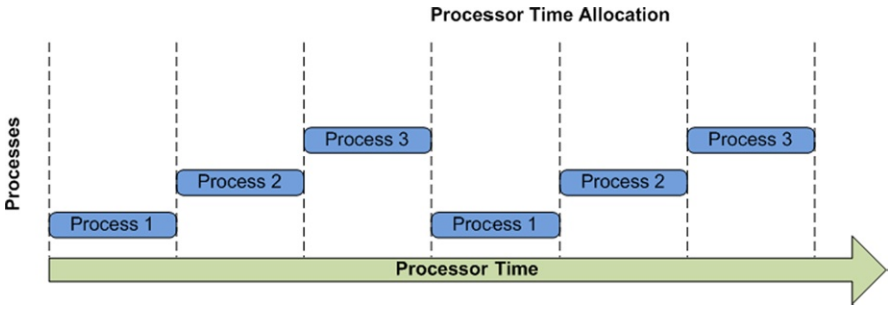


Fig. 4.1 Multi-tasking paradigm

The development of personal computers, and the related diversification in application types determined a definite trend towards widening of user processing requirements. This has lead to uni-programming becoming deprecated. Multi-programming has emerged as a processor allocation paradigm where multiple user programs run on the same computer at the same time.

The main problem with multi-programming is the limited number of CPUs (often a single one) the host machine has. Current CPU architectures involve multiple processing units (cores), which increase the parallel processing capacity of the machine. However, the number of independent processing cores still does not match the number of user programs running at the same time. This is an obvious situation which requires an allocation solution such as all the programs to be able to access the CPU. However, the more user programs run simultaneously on the same machine, the higher the pressure put on the allocation of CPU time.

In this context, the solution to the multi-programming problem is represented by the multi-tasking paradigm. Multi-tasking creates the illusion of concurrency (parallel execution of user programs) by allocating chunks of processor time to each of the running applications sequentially. Originally, each application has been associated with sequential tasks and a single process to run them. However, application development and deployment has moved forward and further benefits from the already described processing parallelism by assigning the same application's tasks to multiple processes which can individually request CPU time.

This software approach to achieve parallel processing requires that the operating systems perform fast switching of CPU between different processes. As a consequence, at any given time a single process only runs on any one processing unit (core). Consequently, the number of processes which run in parallel on a machine is equal to the existing number of processing units on that machine. However, the fast switching between processes creates the illusion (from user perspective) of all processes running in parallel.

Figure 4.1 graphically presents an illustration of the multi-tasking concept. In this figure, three distinct processes are allocated processor time by a single core CPU in a round-robin manner.

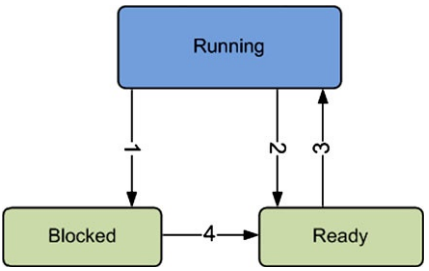


Fig. 4.2 Process state transition

4.3 Processes

A process is a running program sequence along with all the resources that its code can affect (also known as process context). The process context includes the process state, an image of the executable machine code corresponding to the program, allocated memory, descriptors of resources used by the process such as file descriptors or handlers, security attributes such as process owner and process permissions, and last, but not least, processor state like content of registers and physical memory addressing.

The process state represents the status of the process with respect to processor time usage. A process may be in one of three states:

- Running—The process is using the CPU (it has been allocated processor time and the processes machine code is physically executed by the processor).
- Blocked—The process is unable to run until some external event occurs (e.g., data is received from the network). CPU could be free during this period if none of the existing processes is in position to run.
- Ready (Runnable)—The process is ready to run (does not have to wait for any event to occur), but it is temporarily stopped by the operating system to let other processes run on the same CPU.

Figure 4.2 graphically presents the state transitions occurring during the lifetime of a process. Depending on the operating system's scheduling algorithm and the particularities of the process (user application), the status of the process will periodically oscillate between the three states described in the figure.

Figure 4.2 illustrates all possible state transitions which may occur in the following situations:

- Transition 1—Occurs when a process cannot continue, as it is waiting for some external event. For example, when a process initiates a connection to the server, the process will be blocked until the server replies.
- Transition 2—Caused by the process scheduler when it decides to temporarily stop the execution of the current process and give another process the chance to run. The process is interrupted and its state is saved in order to resume operation from the same point it was interrupted without any disruption.
- Transition 3—Caused by the process scheduler when it decides to give a ready process the chance to run. Transitions 2 and 3 are basically creating the illusion of processing parallelism.

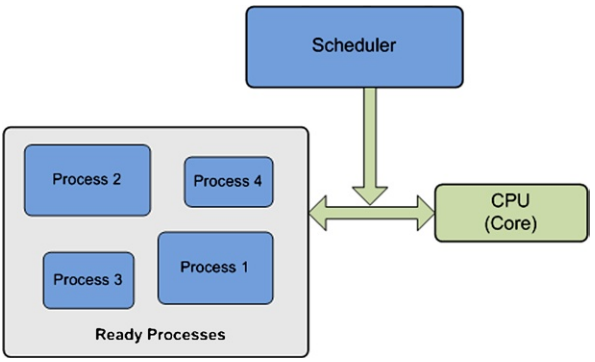


Fig. 4.3 Process scheduling

- Transition 4—Occurs when the external event that the blocked process was waiting for (such as the arrival of some input messages) occurs. This transition is usually triggered by a processor interruption signal generated by the corresponding I/O peripheral or a network interface.

When multiple processes are in the READY state, the operating system must decide which one of them to run first. It employs a scheduling algorithm to determine the processes which will be allocated processor time and in what order. Scheduling algorithms, as graphically depicted in Fig. 4.3, also determine when to stop one process and give CPU time to another process. Scheduling algorithms may do this voluntary ("non-preemptive scheduling") or forced ("preemptive scheduling").

Non-preemptive scheduling involves the processes giving up processor time willingly to allow other processes to run. This usually happens when the currently running process has to switch to the BLOCKED state, while waiting for external event. When preemptive scheduling is used, the currently running process is forced into the READY state to allow other processes to run. Preemptive scheduling can be performed according to a scheduling policy.

There are various scheduling policies including:

- Priority scheduling—processes with higher priority will be allocated processor time more often. Process priority is an important feature when critical applications are running on the host machine. Moreover, when less important tasks or less time-critical applications run (e.g., operating system updates), they may be allocated lower priority in order to minimize the impact on other running applications.
- First come–first served—the first process in the queue of READY process will be allocated processor time. This treats all the process equally, and there is no method to prioritize critical processes.
- Shortest job first—the process requiring less processor time will be given priority. This leaves longer processes with lower priority and may jeopardize to some extent their operation if many light processes are running at the same time.
- Shortest remaining job first—the processes requiring the shortest time to complete will be given the highest priority. This approach involves giving priority

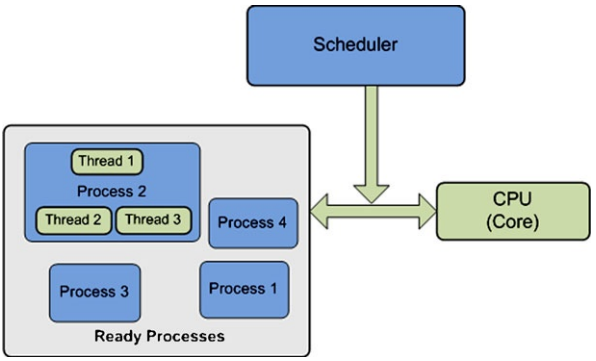


Fig. 4.4 Thread scheduling

to processes which are about to finish their operation, leading to a faster de-congestion of the process queue.

- Round-robin—equal processor time slices are assigned to all processes.

4.4 Threads

A thread is a sequence of a program that performs certain tasks and executes within a process. Often threads are seen as lightweight processes, as they have their own stack, but share memory and data as well as descriptors of resources with other threads within the same process.

Similar to processes, threads may be allocated different priorities depending on their role within the application process. Thread priority can be associated to the thread during its creation and can be changed during the operation of the thread.

Threads can run in preemptive mode (operating system interrupts thread execution at regular intervals to give processing time to other threads) or in cooperative mode (a thread can access the CPU for as long as it needs).

Threads and their scheduling to access the processor time is illustrated in Fig. 4.4.

4.5 Multi-threading

Multi-threading provides another level of parallelism for task execution, with less overhead. When multiple threads exist, different tasks can be performed in parallel using common data and resources. Thread context switching is less complex and faster than process context switching, making multi-treading an efficient way to emulate parallelism. Thread switching efficiency is mainly determined by the fact that threads own less resources than processes which need to be saved prior to switching the context (they share resources such as memory and descriptors with other threads).

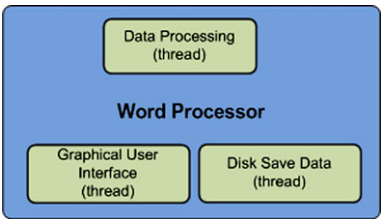


Fig. 4.5 Multi-threading

In general, a process consists of many threads, each running at the same time within the process context and performing a unique task. An example of multi-threading is graphically presented in Fig. 4.5. In this example, a word processor application (process) may use multiple threads, each performing a particular task. The graphical user interface (GUI) is running a separate thread, the data processing module has its own thread as well as the module dealing with saving data on the disk.

Despite the evident benefits multi-threading brings, in terms of application design, the number of threads should be kept to the minimum, in order not to overload the system with non-necessary context switches. Additionally, threads should be employed when there are clear benefits from using parallel processing only, as involving multi-threading in a highly sequential series of tasks only adds complexity to otherwise a simple solution.

Thread priorities have to be managed according to the application purpose and requirements. A thread which manages time-critical tasks should be given higher priority than the other threads. Such an example is the graphical user interface (GUI) which should be allocated a dedicated thread with a higher priority than other threads in order to keep the interaction with the user within the corresponding real-time requirements.

4.6 Multi-threading in Java

Java provides built-in support for multi-threaded programming. It offers two methods for using threads within an application. A thread can be created by extending the *Thread* class, or by implementing the *Runnable* interface. Both are equally efficient in terms of using the threads, but the latter is sometimes preferred as it both provides a clearer separation between the behavior of the thread and the thread itself and enables better reuse.

Next these two approaches are shown in a hands-on manner by providing the step-by-step solution design and implementation when trying to perform parallel activities within the same application.

4.6.1 *Extending* Thread *Class*

This example will both count and repeatedly display Hello! *messages, while availing from processing parallelism.*

- Step 1—Divide application work in tasks and allocate each of them to a thread:

```
/*CountThread does the counting*/
/*HelloThread does the printing*/
```

- Step 2—CountThread extends the Thread class:

```
/*CountThread increments a value and prints it.*/
class CountThread extends Thread
{
  /*time the thread is paused for ( in ms).*/
  int  pause;
  /*number of times the message is printed.*/
  private static final int TIMES = 10;

  /*run() is the method doing the actual thread task.*/
  public void run()
  {
    /*i is incremented and printed within
    the for loop.*/
    for (int i=0; i<TIMES; i++)
    {
      try
      {
        /*print the value of i.*/
        System.out.println(i);

        /*generate a random sleep interval.*/
        pause = (int)(Math.random() * 3000);

        /*put the thread to sleep.*/
        sleep(pause);
      }
      catch (InterruptedException e)
      {
        /*print the exception message when necessary.*/
        System.out.println(e.toString());
      }
    }
  }
}
```

- Step 3—HelloThread extends the Thread class:

```
/*HelloThread prints "Hello!" at random intervals.*/
class HelloThread extends Thread
{
  /*time the thread is paused for (expressed in ms).*/
  int  pause;
  /*number of times the message is printed.*/
  private static final int TIMES = 10;

  /*run() is the method doing the actual thread task.*/
  public void run()
  {

    /*"Hello" is printed TIMES times.*/
    for (int i=0; i<TIMES; i++)
    {
      try
      {
        /*print the message.*/
        System.out.println("Hello!");

        /*generate a random sleep interval.*/
        pause = (int)(Math.random() * 3000);

        /*put the thread to sleep.*/
        sleep(pause);
      }
      catch (InterruptedException e)
      {
        /*print the exception message.*/
        System.out.println(e.toString());
      }
    }
  }
}
```

- Step 4—Instantiate the two thread classes and start their execution:

```
/*main thread application.*/
public class ThreadHelloCount
{
  public static void main(String[] args)
  {
    /*create the CountThread thread.*/
    CountThread count = new CountThread();
```

```
    /*create the HelloThread thread.*/
    HelloThread hello = new HelloThread();

    /*start the CountThread instance.*/
    count.start();

    /*start the HelloThread instance.*/
    hello.start();
  }
}
```

Note that when dividing application work in tasks with the aim to have them associated with threads for parallel execution, each task has to be able to execute independently from the other tasks because otherwise the execution concurrency provides no benefit to the overall application. The run() method of the Thread class is overridden here to perform the core activity of the tasks. The run() method is invoked by the start() method, when the thread is started. The two threads execute loops in which counting and/or printing occurs. After each iteration, a call to the sleep() method determines the threads to suspend their execution. As a result the threads will be in the BLOCKED state until the sleep duration of time indicated when sleep() was called elapses and the threads return to the READY state. This allows the scheduler to perform its scheduling activity and have the threads interleave their execution. However, no activity is performed before the two thread classes are instantiated and their start() methods are called. This is done in the main() method of the main thread application class ThreadHelloCount.

4.6.2 Implementing Runnable *Interface*

Next is an example of multi-threading using implementations of the *Runnable* interface to create threads. The example will perform in parallel repeat printing of the current date and repeat display of a user message.

- Step 1—Divide application work in tasks and allocate each of them to a class:

```
/*DateRunnable does the current date and time printing*/
/*MsgRunnable does the user message printing*/
```

- Step 2—DateRunnable implements the Runnable interface:

```
import java.util.Date;

/*Prints date and time at random intervals.*/
class DateRunnable implements Runnable
{
```

```
  /*current date.*/
  private Date date;
  /*number of times the message is printed.*/
  private static final int TIMES = 10;

  /*constructor for the DateRunnable class.*/
  public DateRunnable(Date aDate)
  {
    date = aDate;
  }

  /*run() is the method that does the thread task.*/
  public void run()
  {
    /*the for loop prints the message TIMES times.*/
    for (int i=0; i<TIMES; i++)
    {
      try
      {
        /*create a new Date object */
        /*containing the current date.*/
        Date nowDate = new Date();

        /*prints the date provided (date)*/
        /*and the current date.*/
        System.out.println("started:"
                + date + " now:" + nowDate);

        /*generate a random wait interval.*/
        int pause = (int)(Math.random() * 3000);

        /*the thread will sleep.*/
        Thread.sleep(pause);
      }
      catch (InterruptedException e)
      {
        /*print the exception message.*/
        System.out.println(e.toString());
      }
    }
  }
}
```

- Step 3—MsgRunnable implements the Runnable interface:

```
/*MsgRunnable prints a user message at random
```

```
 *intervals.*/
class MsgRunnable implements Runnable
{
  /*message to be printed.*/
  private String message;
  /*number of times the message is printed.*/
  private static final int TIMES = 10;
  /*constructor for the MsgRunnable class.*/
  public MsgRunnable(String aMessage)
  {
    message = aMessage;
  }

  /*run() is the method that does the thread task.*/
  public void run()
  {
    /*the for loop will iterate TIMES times to print
     *the message.*/
    for (int i=0; i<TIMES; i++)
    {
      try
      {
        /*print the message.*/
        System.out.println(message);

        /*generate a random wait interval.*/
        int pause = (int)(Math.random() * 3000);

        /*the thread will sleep.*/
        Thread.sleep(pause);
      }
      catch (InterruptedException e)
      {
        /*print the exception message.*/
        System.out.println(e.toString());
      }
    }
  }
}
```

- Step 4—Instantiate the two runnable classes, create threads, and start their execution:

```
import java.util.Date;

/*main thread application.*/
```

```
public class RunnableMsgDate
{
  public static void main(String[] args)
  {
    /*create runnable objects*/
    MsgRunnable mr = new MsgRunnable("Hello!");
    DateRunnable dr = new DateRunnable (new Date());

    /*create thread objects*/
    Thread mt = new Thread(mr);
    Thread dt = new Thread(dr);

    /*start threads*/
    mt.start();
    dt.start();
  }
}
```

The same comments stand regarding the division of application work in independent tasks for thread-based parallel execution in order to avail from any performance benefits. In this example, the *run()* method of the *Runnable* interface is implemented in a runnable class in order to execute the tasks when invoked. The runnable class needs to be instantiated, then associated with a thread before the thread execution start will determine the *run()* method to be invoked (by the Thread's *start()* method). The two runnable classes execute loops in which the current date or a user message is printed. After each iteration, a call to the Thread's *sleep()* method determines execution suspension for the period of time indicated as a parameter in the *sleep()* method call. When this period elapses, the associated threads return to the READY state and are eligible for execution scheduling by the scheduler. However, thread executions start only after the two runnable classes get instantiated, two thread classes are created and associated with the corresponding runnable instances and the two thread *start()* methods are called. This is done in the *main()* method of the main thread application class *RunnableMsgDate*.

There are several important methods of the Thread class which are used for dealing with threads:

- *void start()*—Causes the thread to start its execution (JVM calls thread's run() method);
- *void run()*—Executes thread's task. If the thread was constructed from another Runnable object, it calls automatically that object's run() method;
- *void setName(String name)* and *String getName()*—Change and retrieve the name of the Thread when called;
- *int getPriority()* and *void setPriority(int)*—Get and set thread's priority. The possible values are between 1 and 10;

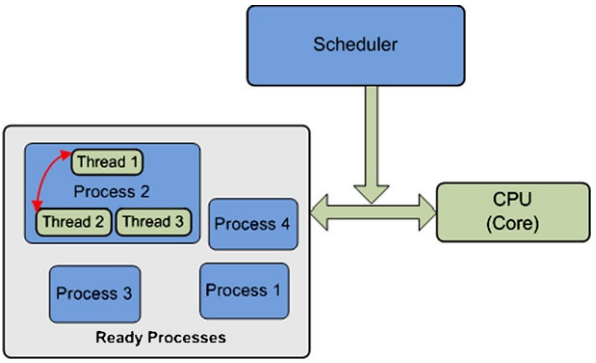


Fig. 4.6 Inter-thread communication

- *static void sleep(long)* and *static void sleep(long, long)*—Cause the thread to cease execution for the specified number of milliseconds and milliseconds + nanoseconds, respectively.
- *static void yield()*—Causes the thread to temporarily pause and allows other threads to execute;
- *void join(long millisec)*—It is usually invoked by the parent thread causing it to block until the child thread terminates or the specified number of milliseconds pass.
- *boolean isAlive()*—Return true if the thread is alive.

The only method of the Runnable interface is *void run()* and it requires implementation as can be seen in the example already presented.

4.7 Inter-thread and Inter-process Communication

4.7.1 Inter-thread Communication

The inter-thread communication focuses on exchanging data between different threads. As threads execute on the same machine and they share the process data space, inter-thread communication is mostly performed using common data variables. The principle of inter-thread communication is graphically presented in Fig. 4.6.

As multiple threads executing in parallel may access and modify the same data variables, the results may not be predictable. In order to solve this issue, only one thread is allowed to modify the data at a time (inter-thread synchronization is required). There are various mechanisms available to enable inter-thread communication and they are presented in details in [1], including:

- Shared memory;
- Semaphores;
- Message passing;

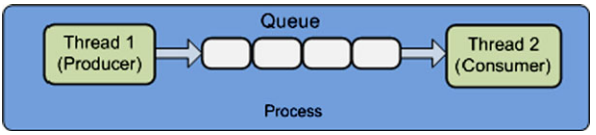


Fig. 4.7 Consumer–producer problem

- Signals;
- Named pipes.

4.7.2 Producer–Consumer Problem

A basic thread synchronization scenario is the producer–consumer problem, as presented in Fig. 4.7. Some threads store items in a queue (producers), while other threads collect items from the queue (consumers) and use them. The queue is a shared memory space and needs to be protected from access by the multiple producer and consumer threads. Next this mechanism will be described in the context of the solution.

An example of inter-thread communication synchronization based on the producer–consumer problem.

- Step 1—Divide the problem and solution in producer, consumer, and shared queue classes:

```
/*class implementing the shared queue*/
class SynchQueue;
/*class extending Thread dealing with the producer*/
class Producer extends Thread;
/*class extending Thread dealing with the consumer*/
class Consumer extends Thread;
```

- Step 2—Design the shared queue class:

```
/*class implementing the shared queue*/
class SynchQueue {
  /*indicate the location of queue's front and back.*/
  /*the consumer reads from the front*/
  /*the producer writes to the back.*/
  private int front = 0, back = 0;
  /*indicates the number of items in the queue.*/
  private int noItems = 0;
  /*queue buffer*/
  private int[] tabItems;
  /*maximum number of items in the queue.*/
  private int maxnoItems;
```

```
/*SynchQueue constructor.*/
public SynchQueue (int maxsize)
{
  maxnoItems = maxsize;
  tabItems = new int[maxnoItems];
}

/*returns the number of items in the queue.*/
public int queueSize() { return noItems; }

/*method used to insert elements in the queue.*/
public synchronized void insert (int item)
{
  /*check for space availability.*/
  while (noItems == maxnoItems)
  {
    try
    {
      /*waits for consumers to free space.*/
      wait();
    }
    catch(InterruptedException ex) {};
  }
  /*insert the item at the back.*/
  tabItems[back] = item;
  /*move the back index one step.*/
  back = (back + 1) \% maxnoItems;
  /*increment the number of items.*/
  noItems += 1;
  /*notify all threads waiting*/
  /*for the object that it is free.*/
  notifyAll();
}

/*method used to remove objects from the queue.*/
public synchronized int remove()
{
  int item;
  /*wait if the queue is empty.*/
  while (noItems == 0)
  {
    try
    {
      wait();
```

```
      }
      catch(InterruptedException ex)
      {
      };
    }
    /*retrieve the item at the front.*/
    item = tabItems[front];
    /*move the from index one step.*/
    front = (front + 1) \% maxnoItems;
    /*decrease the number of items.*/
    noItems -= 1;
    /*notify all threads waiting*/
    /*as space has been freed.*/
    notifyAll();

    return item;
  }
}
```

- Step 3—Design the consumer class extending Thread:

```
/*the consumer thread class.*/
class Consumer extends Thread
{
  /*the queue member.*/
  private SynchQueue synQue;

  /*constructor takes the queue as a parameter.*/
  public Consumer(SynchQueue que)
  {
     synQue = que;
  }
  /*run() method performs the consumer task.*/
  public void run()
  {
    int item = 0;
    do
    {
      /*retrieve an item from the queue.*/
      item = synQue.remove();
      /*print the thread name and the item.*/
      System.out.println
        ("Consumer:" + this + " value:" + item);
    }
    while (item != -1);
    /*iterate until the value of*/
```

```
    /*the item retrieved is -1.*/
  }
}
```

- Step 4—Design the producer class extending Thread:

```
/*the producer thread class.*/
class Producer extends Thread
{
  /*the queue member.*/
  private SynchQueue synQue;
  /*min and max values for the items.*/
  private int minItem, maxItem;

  /*constructor taking the queue and */
  /*min and max no item values as parameters.*/
  public Producer(SynchQueue que, int min, int no)
  {
    synQue = que;
    minItem = min; maxItem = min + no;
  }

  /*run() method performs the producer task.*/
  public void run()
  {
    /*loop used to generate items.*/
    for (int item = minItem; item <= maxItem; item ++)
    {
      /*print the item and thread name.*/
      System.out.println
        ("Producer:" + this + " value:" + item);
      /*insert the item in the queue.*/
      synQue.insert(item);
    }
  }
}
```

- Step 5—Design the producer-consumer class:

```
/*Creates a number of producers (noProd)*/
/*and consumers (noCons) and one synchronized queue.*/
/*Starts them and eventually terminates the process*/
/*by inserting  1 items in the queue noCons times.*/
class MultiProdCons
{
  public static void main(String[] args)
  {
```

```
    /*number of consumers and producers.*/
    int noCons = 3, noProds = 4;

    /*create the 5 element queue.*/
    SynchQueue sque = new SynchQueue(5);

    /*create the consumers and producers.*/
    Consumer[] cons  = new Consumer[noCons];
    Producer[] prods = new Producer[noProds];

    /*start the consumers.*/
    for (int i = 0; i < noCons; i += 1 )
    {    cons[i] = new Consumer(sque);
      cons[i].start();
    }

    /*start the producers.*/
    for (int i = 0; i < noProds; i += 1 )
    {    prods[i] = new Producer(sque, i*100, 50);
      prods[i].start();
    }

    /*wait for the producers to finish.*/
    for (int i = 0; i < noProds; i += 1 )
    {    try { prods[i].join(); }
      catch(InterruptedException ex) {};
    }

    /*insert -1 in the queue for*/
    /*each consumer to terminate.*/
    for (int i = 0; i < noCons; i += 1 )
    {
      sque.insert( -1 );
    }

    /*wait for the consumers to terminate.*/
    for (int i = 0; i < noCons; i += 1 )
    {    try { cons[i].join(); }
      catch(InterruptedException ex) {};
    }
    System.out.println( "successful completion" );
  }
}
```

The key aspect in the solution to the producer–consumer problem is the synchronized use of the queue. Java has a very strong mechanism which labels a code sequence with the keyword *synchronized* and prevents multiple threads from executing the code in parallel, protecting the integrity of the shared variables and memory space. When any thread is executing the synchronized method, all other threads that invoke any of the synchronized methods for the same object suspend their execution (enter the BLOCKED state) until the first thread finishes the execution of the synchronized method. Once this happens, another thread is allowed to execute the synchronized code in an exclusive manner, and so on.

The second aspect worth mentioning is the mechanism introduced to prevent consumer threads from using processing resources while there are no items to be retrieved from the queue (the queue is empty) and producers from executing when there is no space in the queue to place their products (the queue is full). When the queue is empty, calls to the *wait()* method send the consumer threads attempting to fetch items to the BLOCKED state. Similarly, calls to the *wait()* method send the producers attempting to generate items to the BLOCKED state when the queue is full. However, an item produced and placed in the queue or an item retrieved from the queue by a consumer determines calls to *notifyall()* enabling threads to exit from their BLOCKED state and enter READY state, waiting for the scheduler to give them processor time and resume execution. In this way, both processing concurrency and efficiency is achieved.

4.7.3 Inter-process Communication

Inter-process communication focuses on exchanging data between different processes. Communicating processes can run on the same machine or on different machines. Communication between processes that run on the same machine is similar to inter-thread communication. However, as the processes do not share the same data space, there is a need for the processes to share some memory with each other first. Communication between processes that run on different machines involves communication via networks (networking).

These inter-communicating processes must run on machines that are interconnected via a network (wired or wireless), and the data is exchanged using protocols, organized hierarchically in a protocol stack.

The principle of inter-process communication when the processes are running on the same machine or on separate machines is presented in Fig. 4.8.

4.8 Conclusions

This chapter has discussed multi-programming and multi-tasking as very important aspects of the current computational complexityand diversity of applications by making use of computation parallelism and exploiting the multi-core architectures of

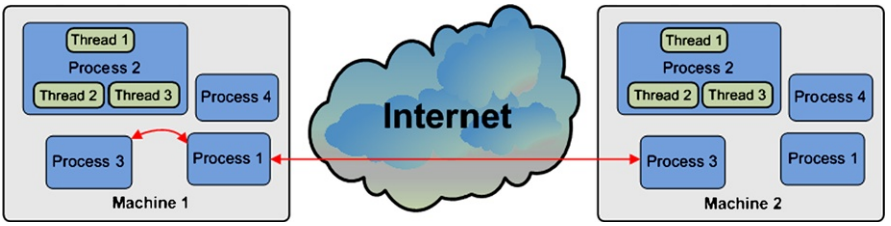


Fig. 4.8 Inter-process communication: (**a**) via shared memory, (**b**) involving networking

many existing processors. The chapter has also discussed processes, threads, inter-thread communication and synchronization, and inter-process communication, and has provided examples described step-by-step which help the readers understand the major issues encountered.

References

1. Tanenbaum AS (2007) Modern operating systems, 3rd edn. Prentice Hall, Upper Saddle River, NJ

Chapter 5
Sockets

Abstract In order to support the inter-process communication, specific support has to be provided by both the operating system and the programming language used. This chapter presents and discusses sockets, as one of the major solutions employed by network programming for the inter-process communications. Sockets provide the application developer with direct basic access to transport protocols, offering data packet transport services between a sender and a receiver host over the network, while hiding the complexity and implementation details of the protocol stack below. Sockets' examples are presented in details when two of the most popular transport protocols are employed in turn: the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

5.1 Introduction

This chapter presents and discusses sockets, as one of the major solutions employed by network programming for inter-process communications. Sockets provide the application developer the direct basic access to transport protocols, offering data packet transport services between a sender and a receiver host over the network, while hiding the complexity and implementation details of the protocol stack below. Sockets examples are presented in details when the two most popular transport protocols used in the Internet are employed in turn: the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

5.2 Socket Definition and Types

Sockets are network communication link end-points between two applications (i.e., server and client). They offer basic transport data communication support and hide lower layer implementation details. They provide a higher level of abstraction for the communication infrastructure beneath and enable support for fast and easy network-based applications development.

The basic principle of socket-based data communication is graphically presented in Fig. 5.1.

B. Ciubotaru, G.-M. Muntean, *Advanced Network Programming – Principles and Techniques*, Computer Communications and Networks,
DOI 10.1007/978-1-4471-5292-7_5, © Springer-Verlag London 2013

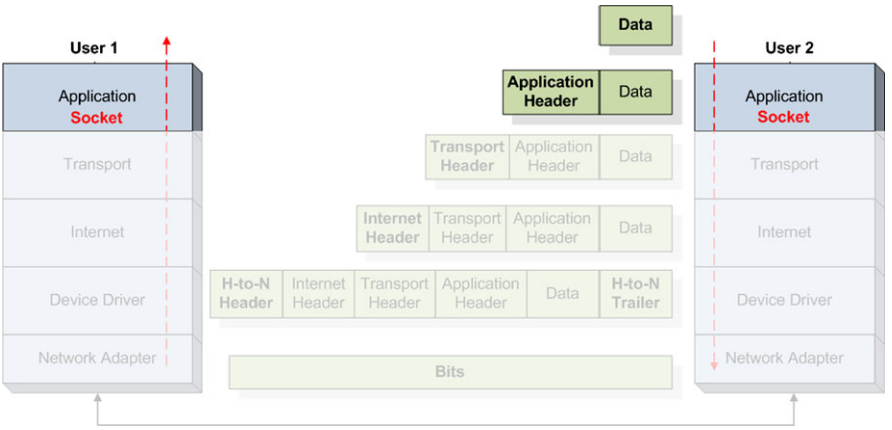


Fig. 5.1 Socket-based network data communication

There are two type of sockets that can be used for application development: transport layer sockets and application layer sockets.

Transport layer sockets make use of transport-layer protocols such as the Transmission Control Protocol (TCP) [2] and User Datagram Protocol (UDP) [1]. UDP is a connectionless non-reliable transmission of datagrams protocol, similar to the postal service. TCP is a connection-based reliable, orderly transmission of data packets, similar to the telephone service.

Application layer sockets make use of application-layer protocols such as the Hypertext Transfer Protocol (HTTP) [3] and Simple Mail Transfer Protocol (SMTP) [4]. HTTP is a TCP-based web page delivery service, while SMTP is a TCP-based e-mail delivery service.

5.3 Socket-Based Network Communications

The network communication using sockets involves several basic steps which have to be followed by the application developer in order to have a functional transport mechanism between the two communicating parties.

The first step involves creating and opening sockets. Each communicating party requires a separate socket. When creating/opening a socket, the important parameters to be provided include the IP address, port number, and communication protocol (TCP or UDP).

The second step involves establishing contact or associating the socket with another socket. In order to be able to communicate, the sockets have to use the same protocol. In general, the client knows the server's IP address, port number, and the protocol used and contacts it using these details. Depending on the protocol, it can either request a service or establish a connection.

The third step consists of exchanging data between the two parties and represents the main stage of network application's communication tasks. This step usually happens recursively. Information is sent and received by the two communicating sockets.

The last step involves closing and destroying the sockets which closes the communication end-point. After this step the socket cannot be used for communication any more.

5.3.1 UDP Sockets

When using UDP sockets the connection between the client and the server is not maintained throughout the communication session. Each datagram packet is sent as an isolated transmission when necessary. There are no guarantees that the packets arrive in order at the destination or that the packets arrive at the destination at all.

5.3.1.1 UDP Sockets—Server Side

Java UDP server communication steps include the following:

- Step 1—Create a datagram socket object.

```
DatagramSocket dgramSocket =
          new DatagramSocket(portno);
 /*1024 <= portno <= 65535*/
```

- Step 2—Create a buffer to store the incoming datagrams:

```
byte[] buffer = new byte[256];
/*-128 <= byte value <= 127*/
```

- Step 3—Create a datagram packet object for incoming datagrams:

```
DatagramPacket inPkt =
          new DatagramPacket(buffer, buffer.length);
```

- Step 4—Accept an incoming datagram:

```
dgramSocket.receive(inPkt);
```

- Step 5—Get sender's address and port number from the datagram:

```
InetAddress cliAddress = inPkt.getAddress();
int cliPort = inPkt.getPort();
```

- Step 6—Retrieve the data from the buffer:

```
String msgIn =
  new String(inPkt.getData(), 0, inPkt.getLength());
```

- Step 7—Create the response datagram:

```
msgOut = ("Message " + numMessages + ":" + messageIn);
DatagramPacket outPkt =
     new DatagramPacket(msgOut.getBytes(),
                msgOut.length(),
                cliAddress,
                cliPort);
```

- Step 8—Send the response datagram:

```
dgramSocket.send(outPkt);
```

- Step 9—Repeat communication if necessary:

```
while(condition);
```

- Step 10—Close the datagram socket:

```
dgramSocket.close();
```

Java UDP socket communication may throw exceptions that need to be caught and treated. The following example shows how to catch exceptions thrown by the UDP sockets.

```
try{
      /*attempt to create the socket*/
      dgramSocket = new DatagramSocket(PORT);
    }

    catch (SocketException e) {
      /*this exception may be triggered when*/
      /*the port is already in use.*/
      System.out.println("Unable to attach to port!");
      System.exit(1);
    }
```

The following example illustrates the use of sockets to create a server application using the UDP protocols for data transport.

Server UDP Socket Communication Example:

```
import java.io.*;
import java.net.*;

/*UDP server class.*/
public class UDPEchoServer
```

Code Listing 5.1 UDPEchoServer.java

```
{
  /*port used by the server.*/
  private static final int PORT = 1234;

  /*the datagram socket specific to UDP.*/
  private static DatagramSocket dgramSocket;
  /*incoming and outgoing packets objects.*/
  private static DatagramPacket inPkt, outPkt;
  /*packet data buffer.*/
  private static byte[] buffer;

  public static void main(String[] args)
  {
    System.out.println("Opening port...\n");
    try
    {
    /*create the datagram socket*/
    dgramSocket = new DatagramSocket(PORT);
    }
    catch(SocketException e)
    {
    /*handle potential exceptions.*/
    System.out.println("Error attach port!");
    System.exit(1);
    }
    run();
  }

/*run() performs the main task of the server.*/
private static void run() {
try  {
  /*buffers for the messages to be sent and received.*/
  String msgIn,msgOut;
  /*number of messages.*/
  int numMsgs = 0;
  do {
    /*create the packet buffer.*/
    buffer = new byte[256];

    /*create the incoming packet.*/
    inPkt = new DatagramPacket(buffer,buffer.length);

    /*receive the packet from the client.*/
    dgramSocket.receive(inPkt);
```

Code Listing 5.1 (Continued)

```
    /*retrieve the sender's IP address.*/
    InetAddress cliAddress = inPkt.getAddress();

    /*retrieve the sender's port number.*/
    int cliPort = inPkt.getPort();

    /*store the content of the message.*/
    msgIn =
      new String(inPkt.getData(),0,inPkt.getLength());
    System.out.println("Message received.");
    /*increment the message number.*/
    numMsgs++;

    /*generate the outgoing message.*/
    msgOut = ("Msg "+numMsgs+ ": "+msgIn);
    /*create the outgoing packet.*/
    outPkt = new DatagramPacket(msgOut.getBytes(),
          msgOut.length(),cliAddress,cliPort);

    /*send the outgoing packet to the client.*/
    dgramSocket.send(outPkt);
  } while(true);
  }
  catch(IOException e) {
    e.printStackTrace();
  }
  finally{
    /*close the socket and release its resources.*/
    dgramSocket.close();
  }
  }
}
```

Code Listing 5.1 (Continued)

5.3.1.2 UDP Sockets—Client Side

Java UDP client communication steps include:

- Step 1—Create a datagram socket object:

```
DatagramSocket dgramSocket = new DatagramSocket;
/*a default port no will be selected*/
```

- Step 2—Create the outgoing datagram:

```
BufferedReader userEntry =
new BufferedReader(new InputStreamReader(System.in));

System.out.print("Enter message: ");

String msg = userEntry.readLine();

DatagramPacket outPkt =
new DatagramPacket(msg.getBytes(),
                   msg.length(), host, portno);
```

- Step 3—Send the response datagram:

```
dgramSocket.send(outPkt);
```

- Step 4: Create a buffer to store the incoming datagrams:

```
byte[] buffer = new byte[256];
```

- Step 5—Create a datagram packet object for incoming datagrams:

```
DatagramPacket inPkt =
          new DatagramPacket(buffer, buffer.length);
```

- Step 6—Accept an incoming datagram:

```
dgramSocket.receive(inPkt);
```

- Step 7—Retrieve the data from the buffer:

```
String msgIn =
new String(inPkt.getData(), 0, inPkt.getLength());
```

- Step 8—Close the datagram socket:

```
dgramSocket.close();
```

Next a client UDP socket communication example is presented.

```
import java.io.*;
import java.net.*;

/*UDP client class*/
public class UDPEchoClient

{
  /*server IP*/
  private static InetAddress host;
```

Code Listing 5.2 UDPEchoClient.java

```
/*server port*/
private static final int PORT = 1234;


/*datagram socket*/
private static DatagramSocket dgramSocket;

/*incoming and outgoing packets.*/
private static DatagramPacket inPkt, outPkt;

/*packet buffer*/
private static byte[] buff;

/*messages content storage*/
private static String msg = "", msgIn = "";


public static void main(String[] args)
{
  try
  {
  host = InetAddress.getLocalHost();
  /*or get InetAddress of server*/
  }
  catch(UnknownHostException e)
  {
  /*handle exception*/
  System.out.println("Host not found!");
  System.exit(1);
  }
  run();
}

private static void run() {
try  {
  /*create datagram socket*/
  dgramSocket = new DatagramSocket();
  /*create the buffer reader to read from the
  console*/
  BufferedReader userEntry = new BufferedReader(
      new InputStreamReader(System.in));
  do {
    System.out.print("Enter message: ");
```

Code Listing 5.2 (Continued)

```
      /*read user entry*/
      msg = userEntry.readLine();
      /*send messages until BYE is sent*/
      if (!msg.equals("BYE")) {
        /*create the packet*/
        outPkt = new DatagramPacket(msg.getBytes(),
          msg.length(), host, PORT);
        /*send the packet*/
        dgramSocket.send(outPkt);

        /*allocate packet buffer*/
        buff = new byte[256];
        /*create incoming packet*/
        inPkt = new DatagramPacket(buff, buff.length);
        /*receive incoming packet*/
        dgramSocket.receive(inPkt);
        /*store packet content*/
        msgIn = new String(inPkt.getData(),
          0, inPkt.getLength());
        System.out.println("SERVER: " + msgIn);
      }
    } while (!msg.equals("BYE"));
  }

  catch(IOException e){
    e.printStackTrace();
  }
  finally{
    /*close the socket and release its resources*/
    dgramSocket.close();
    }
  }
}
```

Code Listing 5.2 (Continued)

5.3.2 TCP Sockets

As TCP is a connection-oriented protocol, when TCP sockets are used, connections are established between client and server hosts. Client and server TCP sockets are created first and are bound for the duration of the data communication session. Following connection establishment, TCP packets are sent to the partner's socket.

These packets are guaranteed to arrive (if lost, retransmission occurs) and are received in order at the destination.

Exchanging messages using TCP sockets involves a set of steps that must be followed by the application developer. Next these steps are presented, with the focus on server and client side, respectively.

5.3.2.1 TCP Sockets—Server Side

Java TCP server communication steps:

- Step 1—Create a TCP server socket object.

```
ServerSocket servSock = new ServerSocket(portno);
/*1024 <= portno <= 65535*/
```

- Step 2—Set the server to wait (block) for clients to connect.

```
Socket sock = servSock.accept();
/*sock is a socket object.*/
```

- Step 3—Set input and output streams.

```
BufferedReader in =
new BufferedReader(
  new InputStreamReader(sock.getInputStream()));

PrintWriter out =
new PrintWriter(sock.getOutputStream(), true);
```

- Step 4—Send and receive data.

```
out.println("Waiting...");
String msg = in.readLine();
```

- Step 5—Close the connection.

```
sock.close();
```

Server TCP Socket Communication Example:

```
import java.io.*;
import java.net.*;

/*TCP-based echo server*/
public class TCPEchoServer
{
```

Code Listing 5.3 TCPEchoServer.java

```
/*server socket*/
private static ServerSocket servSock;
/*server port*/
private static final int PORT = 1234;

public static void main(String[] args)
{
  System.out.println("Opening port\n");
  try
  {
    /*Create the server socket*/
    servSock = new ServerSocket(PORT);
  }
  catch(SocketException e)
  {

    /*handle potential exceptions*/
    System.out.println("Error attach port!");
    System.exit(1);
  }
  catch (IOException e)
  {
    /*handle potential exceptions*/
    System.out.println("Error create socket!");
    System.exit(1);
  }

  /*perform the echo service indefinitely*/
  do {
    run();
  }
  while (true);
}

private static void run() {
  /*data socket*/
  Socket sock = null;
  try  {
    /*listen for incoming connections*/
    link = servSock.accept();
    /*create a socket buffer reader*/
    BufferedReader in = new BufferedReader(
      new InputStreamReader(sock.getInputStream()));
    /*create the socket writer*/
```

Code Listing 5.3 (Continued)

```
      PrintWriter out = new PrintWriter(
        sock.getOutputStream(),true);

      int numMsgs = 0;
      /*read from the data socket*/
      String msg = in.readLine();
      while (!msg.equals("BYE"))
      {
        System.out.println("Message received.");
        numMsgs++;
        out.println("Message " + numMsgs + ": " + msg);

        msg = in.readLine();
      }
      out.println(numMsgs + " messages received.");
    }
    catch(IOException e) {
      e.printStackTrace();   }
    finally{
      /*close the socket*/
      try {
        sock.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
  }
}
```

Code Listing 5.3 (Continued)

5.3.2.2 TCP Sockets—Client Side

Java TCP client communication steps:

- Step 1—Create a TCP client socket and establish a connection to the server.

```
InetAddress srvIPAddr;
int srvPortNo = 1234;
Socket sock = new Socket(srvIPAddr.getLocalHost(),
                         srvPortNo);
/*sock is a socket object*/
```

- Step 2—Set input and output streams.

```
BufferedReader in =
new BufferedReader(
  new InputStreamReader(sock.getInputStream()));

PrintWriter out =
new PrintWriter(sock.getOutputStream(), bAutoFlush);
```

- Step 3—Send and receive data.

```
out.println("Waiting for data");
String msgIn = in.readLine();
```

- Step 4—Close the connection.

```
sock.close();
```

Client TCP socket communication example:

```
import java.io.*;
import java.net.*;

/*TCP client class*/
public class TCPEchoClient
{
  /*server IP*/
  private static InetAddress host;
  /*server port*/
  private static final int PORT = 1234;

  public static void main(String[] args)
  {
    System.out.println("Opening port\n");
    try
    {
      /*create server IP address  object*/
      host = InetAddress.getLocalHost();
    }
    catch(UnknownHostException e)
    {
      System.out.println("Host not found!");
      System.exit(1);
    }
    run();
  }
```

Code Listing 5.4 TCPEchoClient.java

```
  private static void run()
  {
    Socket link = null;
    try
    {
      /*create data socket*/
      sock = new Socket(host, PORT);
      /*create socket reader and writer*/
      BufferedReader in = new BufferedReader(new
        InputStreamReader (sock.getInputStream()));
      PrintWriter out = new PrintWriter(
        sock.getOutputStream(), true);

      /*Set up stream for user entry*/
      BufferedReader reader =
        new BufferedReader(new
          InputStreamReader(System.in));

      /*storage for message and response message*/
      String msgOut, msgIn;
      do
      {
        System.out.print("Enter message: ");
        /*read user message*/
        msgOut =  reader.readLine();

        /*send the message*/
        out.println(msgOut);

        /*read the response*/
        msgIn = in.readLine();
        System.out.println("SERVER> " + msgIn);
      } while (!message.equals("BYE"));
    }

    catch(IOException e)
    {
      e.printStackTrace();
    }

    finally
    {
      /*close the data socket*/
```

Code Listing 5.4 (Continued)

```
      try {
        sock.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
  }
}
```

Code Listing 5.4 (Continued)

5.4 Conclusions

This chapter has introduced sockets as the most popular solution for inter-process communications. Sockets have been introduced at the application and transport layer, respectively, allowing for network programming to focus on higher layer aspects, while the implementation details of the lower layer protocol stack are hidden. The chapter has described the socket-based network programming stages step-by-step with the help of examples. The sockets examples presented in details employ in turn TCP and UDP, respectively.

References

1. Postel J (August 1980) User datagram protocol. RFC 768, Internet engineering task force
2. Postel J (1981) RFC 793—Transmission control protocol (TCP). RFC 793
3. Fielding R, Gettys J, Mogul J, Frystyk H, Masinter L, Leach P, Berners-Lee T (1999) RFC 2616—Hypertext transfer protocol (HTTP/1.1). RFC 2616
4. Klensin J (2008) RFC 5321—Simple mail transfer protocol (SMTP). RFC 5321

Chapter 6
Socket-Based Client–Server Communication

Abstract Sockets offer the basic mechanisms for data communication between two processes, each running on a distinct machine. This chapter describes the socket-based client–server communication mechanism and details the basics of client–server applications programming, including multi-threaded servers. Unicast, multicast, and broadcast communication paradigms are also introduced in this chapter.

6.1 Introduction

Client–server is a request–response remote communication model that involves processes requesting services from other processes which offer these services via the network.

The processes offering services by executing certain tasks following remote process requests are known as servers. In general, the servers receive requests from remote processes, execute the tasks associated with these services, and dispatch responses back to the requesting entities. Examples of services include database information retrieval and updates, file system access services, and dedicated user-application tasks.

The processes that contact the servers and request them to perform services are known as clients. In general, client processes manage user-interfaces, validate data entered by users, dispatch requests to servers, collect servers' responses, and process and/or display the information received.

The client and server definitions were introduced in relation to service requesting and service providing processes, but they can equally be used for applications and machines. Server applications run in general on powerful computers which are often located in dedicated places such as data centers. Client applications usually run on user devices which may be desktop-PCs, laptop-PCs, netbooks, gaming consoles, or mobile hand-held devices. The machines which host client applications are often referred to as clients, and the computers which run server applications are known as servers. Although theoretically both the server and the client applications may run on the same physical machine, this is almost never the case, mostly due to the different requirements in terms of processing power of these machines.

Both client-side and server-side machines may run in parallel several client and server applications, respectively.

B. Ciubotaru, G.-M. Muntean, *Advanced Network Programming – Principles and Techniques*, Computer Communications and Networks,
DOI 10.1007/978-1-4471-5292-7_6, © Springer-Verlag London 2013

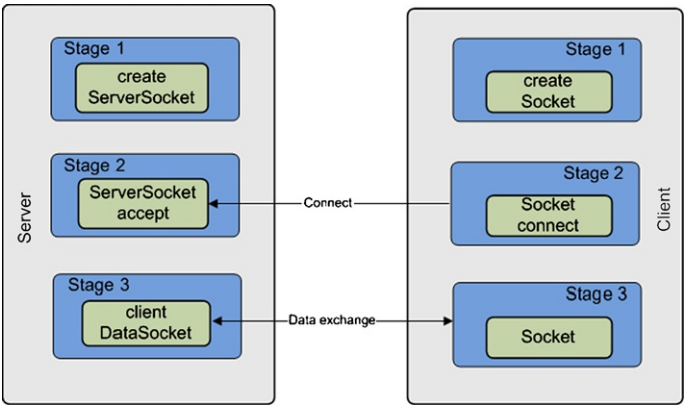


Fig. 6.1 Client–server communication steps

For example, a standard client machine would normally run several applications such as a web browser, an e-mail client, and maybe a media player receiving a video stream over the Internet. All these applications act as clients accessing services and content (e.g., web documents, e-mail messages, and multimedia content) from remote server applications.

Similarly, server machines may run several applications in parallel. A good example would be a data center server machine running a web server (e.g., Apache) and a database server (e.g. MySQL) in parallel. Although the scope is common for the two services (web content delivery and database services), they represent distinct applications running on the same physical hardware platform. However, most of the time there is a dedicated server machine (or servers) which runs a server application, offering a single service of which multiple client applications can avail. The next sections will present step-by-step socket-based network programming solutions to implement such a client–server paradigm. The chapter discusses unicast, multicast, and broadcast communications and presents in this context relevant data transfer application examples.

6.2 Basic Client–Server Application Programming

Server and client applications often use sockets to initiate communication sessions and perform data and/or message exchange in a request–response manner. Basic socket-based solutions enable one connection to be established between a client and a server at a time only. The idea behind this simple solution is graphically illustrated in Fig. 6.1 and is described step-by-step in the context of both UDP and TCP socket-based connectivity.

However, in most cases servers need to deal with multiple clients at a time, and consequently more advanced solutions are required. A very good option which pro-

vides good scalability is to make use of a thread-based approach for building the server application.

6.3 Multi-threaded Server Applications

A server application capable of handling multiple simultaneous clients can be built by making use of both multi-threading support and socket-based communications. Although theoretically infinite, the number of clients that a server can handle simultaneously is, in fact, limited. Indeed, the processing resources of the operating system and the physical machine on which the server application runs, as well as transport capacity of the network over which data is exchanged, limit the number of independent communication sessions established and managed simultaneously. We will present this approach in the context of a multi-threaded TCP server application example, which simply echoes the message received from the client application.

This multi-threaded TCP server application makes use of a specially built socket communicating thread class which, given the communication socket-based link has been established, exchanges messages with the client application. The major steps for building this server application component are as follows:

- Step 1—Create a server class which extends Thread.

```
public class SingleTCPEchoServer extends Thread
```

- Step 2—Get a handle to an already established communicating socket.

```
/*private socket variable*/
private static Socket sock;

/*set the data socket*/
sock = s;
```

- Step 3—Create reader and writer objects for socket communication.

```
/*BufferReader used to read data from data socket*/
private BufferedReader in;
/*PrinterWriter used to write to the data socket*/
private PrintWriter out;

/*create the BufferReader for reading from socket*/
in = new BufferedReader(new InputStreamReader(
    sock.getInputStream()));
/*create the PrintWriter for socket writing*/
out = new PrintWriter(sock.getOutputStream(),true);
```

- Step 4—Override run() to receive and send data.

```
/*read message from the data socket (client)*/
```

```
    String msg = in.readLine();

    /*send the reply message to the client*/
    out.println("Message " + numMessages + ": " + msg);
```

- Step 5—Start the thread.

```
    /*call the run() method*/
    start();
```

- Step 6—Deal with potential exceptions.

```
    try{ [...] }
    catch (IOException e) { [...] }
```

The full thread-based TCP socket server application example is presented next.

```
import java.io.*;
import java.net.Socket;

/*single-threaded server class*/
/*handles client communication*/
public class SingleTCPEchoServer extends Thread {

  /*client data socket*/
  private static Socket sock;

  /*server port*/
  private static final int PORT = 1234;

  /*BufferReader used to read data from data socket*/
  private BufferedReader in;

  /*PrinterWriter used to write to the data socket*/
  private PrintWriter out;

  /*Constructor for the single threaded server*/
  public SingleTCPEchoServer(Socket s)
    throws IOException {

    /*set the data socket*/
    sock = s;

    /*create the BufferReader from data socket*/
```

Code Listing 6.1 SingleTCPEchoServer.java

```
    in = new BufferedReader(new InputStreamReader(
      sock.getInputStream()));

    /*create the PrintWriter for data socket*/
    out = new PrintWriter(
        sock.getOutputStream(),true);

    /*If any of the above calls throws an exception,
    the caller will close the socket.
    Otherwise the thread will close it.*/

    /*call the run() method*/
    start();
  }

/*run() method performs the actual task*/
public void run()
{
  try {
    int numMessages = 0;

    /*read message from the data socket (client)*/
    String msg = in.readLine();
    /*verify if the message is BYE*/
    while (!msg.equals("BYE"))
    {
      System.out.println("Message received.");

      /*count the number of messages received*/
      numMessages++;

      /*send the reply message to the client*/
      out.println("Message " + numMessages +
                  ": " + msg);

      /*read the next message*/
      msg = in.readLine();
    }

    /*at this point BYE has been received*/
    /*the server reports the number of received
    messages*/
    out.println(numMessages + " messages received.");
    }
```

Code Listing 6.1 (Continued)

```
    catch (IOException e)
    {
      e.printStackTrace();
    }
    finally
    {
      try

      {
        System.out.println("\n Closing connection");
        /*close the data socket*/
        sock.close();
      }
      catch(IOException e)
      {
        System.out.println("Unable to disconnect!");

        System.exit(1);
      }
    }
  } /*run()*/
} /*SingleTCPEchoServer*/
```

Code Listing 6.1 (Continued)

The multi-threaded TCP server application is, in fact, a dispatcher which creates a ServerSocket at the server which waits for incoming connection requests from client applications. This wait is performed through a blocking call to accept(). As soon as a client request is received, the call to accept() completes, and a new single TCP server application thread, built as already described, will be created. If successful in establishing the connection with the client, accept() returns a communicating Socket which is passed as a parameter to the newly created single TCP thread, enabling direct communication between this newly created server thread and the client. After the new thread is started, the multi-threaded server application call again accept() waiting for another client connection request. This sequence of operations continues over and over again and is limited just by hardware and software resources of the server machine. This process is illustrated in Fig. 6.2 and described step-by-step next.

- Step 1—Create a server socket at a given port number.

```
/*The server socket is defined as a class member.*/
private static ServerSocket servSock;
/*Create the server socket to listen on PORT*/
servSock = new ServerSocket(PORT);
```

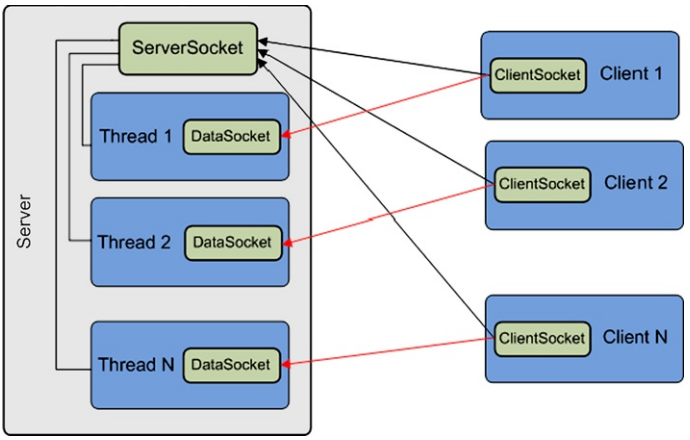


Figure 6.2 Multi-threaded server paradigm

- Step 2—In a loop wait for client connection requests.

```
do
{
  /*Blocks until a connection occurs.*/
  Socket socket = servSock.accept();
  [...]
} while (true);
```

- Step 3—For each client create a thread server to communicate via socket.

```
do {
  [...]
  /*Create a thread server to handle the client.*/
  new SingleTCPEchoServer(socket);

} while (true);
```

- Step 4—Deal with potential exceptions.

```
try{ [...] }
catch (IOException e) { [...] }
```

The complete multi-threaded TCP socket-based server application example is provided below.

```
import java.io.*;
import java.net.*;
```

Code Listing 6.2 MultiTCPEchoServer.java

```
/*Class implementing the multi-threaded echo server.*/
/*This server receives a message from the clients
 *and replies with the same message back.
 */
public class MultiTCPEchoServer {

  /*The server socket is defined as a class member.*/
  private static ServerSocket servSock;
  /*The port number is defined as a member.*/
  /*The server will listen on this port.*/
  private static final int PORT = 1234;

  /*Data socket is defined as a member.*/
  /*Socket to be used for */
  /*communication with the client.*/
  Socket sock = null;

  /*Constructor.*/
  public MultiTCPEchoServer() { }

  /*The main function to be run
   *when the server application stars.*/
  public static void main (String[] args)
      throws IOException
  {
  System.out.println("Opening port\n");

  try
  {
    /*Create the server socket to listen on PORT*/
    servSock = new ServerSocket(PORT);

  }
  catch (IOException e)
  {
    /*Handles potential exceptions
     *thrown while the server socket is created.*/
    /*Most common exception is triggered
     *when the chosen port is already used*/
    System.out.println("Port error!");
    System.exit(1);
  }
```

Code Listing 6.2 (Continued)

```
  /*At this point the server socket
   *was successfully created.*/
  try
  {
    /*main server loop.*/
    do
    {
      /*Server accepts connections from client.*/
      /*The Accept method blocks
       *until a connection occurs.*/
      Socket socket = servSock.accept();
      try
      {
        /*Create a single-threaded server.*/
        /*This will handle the client.*/
        new SingleTCPEchoServer(socket);
      }

      catch(IOException e)
      {
        /*Handle potential exceptions.*/
        /*As the creation of the
         *single-threaded server failed,
         *communication with the client can not start,
         *so the data socket is closed.
         */
        socket.close();
      }
    } while (true);
  }
  finally
  {
    /*When the server end its operation,
     *the server socket is closed.
     */
    servSock.close();
  }
 }
}
```

Code Listing 6.2 (Continued)

This multi-threaded TCP socket-based server application example works with a TCP socket-based client application similar to the one presented in the chapter on Sockets when discussing TCP connectivity.

6.4 Unicast, Multicast, and Broadcast Communications

Unicast refers to one-to-one communication and in general is performed by the sender following a receiver request.

Broadcast communication is when a single sender transmits data to all the devices connected to the network having an IP address in a certain range. This transmission can target all local subnet devices, all nodes in the local network, etc.

Multi-casting involves one-to-many communications between a sender and a set of receivers. These receivers must belong to a multicast group, which has to be established prior to any data communication. This multicast group (defined using a class D IP address) has to be established with network support. Unfortunately, not all networks enable multicast transmissions. All class D IP addresses are multicast addresses and range from 224.0.0.0 to 235.255.255.255. Any such IP address can be allocated to the newly formed multicast group, provided that it has not been already used in the same network domain.

The sender has to send data to the multicast group IP address. The receivers need to join the multicast group in order to receive data from the transmitting sender. However, the sender is not required to join the multicast group in order to transmit data.

All members of the multicast group will receive a copy of the data transmitted by the server. This is enabled by the multicast-enabled routers which support multicast tree-like routing of packets by multiplicating data packets and sending them towards the receivers which belong to the multicast group. In order to stop receiving data, the clients have to leave the multicast group.

As TCP is a point-to-point protocol and is most suitable for unicast transmissions, it cannot be used for multi-cast data delivery. Instead, UDP is the transport protocol used for multicast datagram packet distribution.

In terms of implementation, broadcast data transmission can be seen as an extension of multi-cast. In order to achieve broadcasting, all connected devices need to join the same multicast group, and consequently they will all receive a copy of the packets sent by the sender. In this context, broadcast data communications represent a special case of multicast and will not be dealt with separately.

In order to receive data from the server in a multicast manner, several steps have to be performed by the application. In the following paragraphs, these steps are detailed in the context of multicast receiver and sender applications, respectively.

Java multicast receiving data application:

- Step 1—Define port and address of multicast group to join:

```
/*multicast group port number*/
/*the server will send data to this port*/
int PORT = 5000;
/*multicast group IP address*/
String GROUP = "225.4.5.6";
```

- Step 2—Create MulticastSocket object and bind it to port PORT

```
/*MulticastSocket is a specific class
 *implementing multicast communication endpoints*/
MulticastSocket ms = new MulticastSocket(PORT);
```

- Step 3—Join the multicast group with address GROUP

```
/*MulticastGroup class provides joinGroup method
 *used to join the group using its IP*/
ms.joinGroup(InetAddress.getByName(GROUP));
```

- Step 4—Create a DatagramPacket and receive a packet.

```
/*allocate the packet buffer*/
byte buf[] = byte[1024];
/*create the packet*/
DatagramPacket pack =
  new DatagramPacket(buf, buf.length);
/*receive a packet from the multicast socket*/
ms.receive(pack);
```

- Step 5—Use the data received (print it on screen).

```
/*print the details of the sender (server)*/
System.out.println("Received data from: "
      + pack.getAddress().toString() + ":"
      + pack.getPort() + " with length: "
      +  pack.getLength());
/*print the message*/
System.out.write(pack.getData(), 0,
                 pack.getLength());
/*print a new line to separate the messages.*/
System.out.println();
```

- Step 6—Leave the multicast group and close the socket.

```
/*use leaveGroup to leave the multicast group*/
ms.leaveGroup(InetAddress.getByName(group));
/*close the multicast socket*/
ms.close();
```

On the sender side, the application has to follow another set of steps in order to be able to send data to a multicast group. In the following paragraphs, these steps are detailed.

Java multicast sending data application:

- Step 1—Define port, address of multicast group to send to and time-to-live.

```
/*port number to which the packets will be send*/
int PORT = 5000;
```

```
/*the multicast group IP address*/
String GROUP = "225.4.5.6";
/*time-to-live for packets sent
to the multicast group*/
byte TTL = 3;

/*Note: The TTL sets the IP time-to-live
specifying over how many "hops"
the packets will be forwarded in the
MulticastGroup before they expire.*/
```

- Step 2—Create MulticastSocket object.

```
/*create the multicast socket*/
MulticastSocket ms = new MulticastSocket();
```

- Step 3—Create a DatagramPacket and copy some data in it.

```
/*create the packet buffer*/
byte buf[] = byte[1024];
/*generate a message into the buffer*/
for (int i = 0; i < buf.length; i++)
  buf[i] = (byte)i;
/*create the datagram packet*/
/*the multicast IP address is used
 *for the creation of this packet*/
DatagramPacket pack =
  new DatagramPacket(buf, buf.length,
    InetAddress.getByName(GROUP), PORT);
```

- Step 4—Send the DatagramPacket to the multicast group.

```
/*send the packet specifying
the TTL as a parameter*/
ms.send(pack, TTL);
```

- Step 5—Close the socket.

```
/*close the multicast socket*/
ms.close();
```

6.5 Conclusion

This chapter has introduced the client–server communication paradigm and in its context has presented a step-by-step socket-based network programming example to implement a relevant application. The chapter then discussed unicast, multicast, and broadcast communications and presented a multi-cast data transfer application example which can also be used for data broadcast.