CHAPTER 4

OSI Transport Layer

Objectives

Upon completion of this chapter, you will be able to answer the following questions:

- Why is there a need for the transport layer?
- What is the role of the transport layer as it provides the end-to-end transfer of data between applications?
- What is the role of two TCP/IP transport layer protocols: TCP and UDP?
- How do the key functions of the transport layer protocol, including reliability, port addressing, and segmentation, work?
- How do TCP and UDP handle the key functions?
- When is it appropriate to use TCP or UDP, and what are some examples of applications that use each protocol?

Key Terms

This chapter uses the following key terms. You can find the definitions in the Glossary.

flow control *page 104*

control data *page 106*

Internet Assigned Numbers Authority (IANA) *page 110*

well-known ports *page 110*

registered ports *page 111*

dynamic or private ports *page 111*

URG *page 117*

ACK *page 117*

PSH *page 117*

RST *page 117*

SYN *page 117*

FIN *page 117*

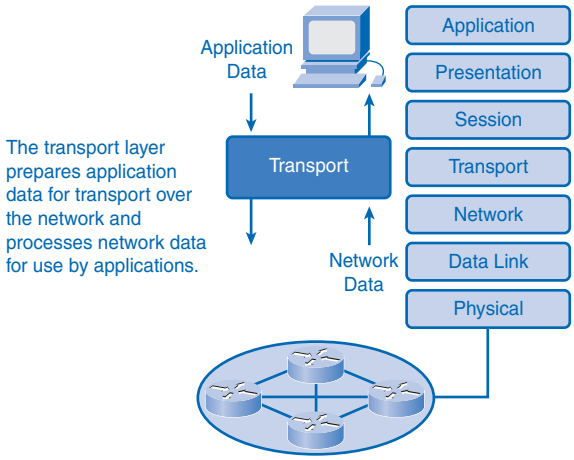
acknowledgment *page 118*

window size *page 121*

Data networks and the Internet support the human network by supplying seamless, reliable communication among people—both locally and around the globe. On a single device, people can use multiple services, such as e-mail, the web, and instant messaging, to send messages or retrieve information. Applications such as e-mail clients, web browsers, and instant messaging clients allow people to use computers and networks to send messages and find information.

Data from each of these applications is packaged, transported, and delivered to the appropriate server daemon or application on the destination device. The processes described in the OSI transport layer accept data from the application layer and prepare it for addressing at the network layer. The transport layer is responsible for the overall end-to-end transfer of application data, as shown in Figure 4-1.

Figure 4-1 OSI Transport Layer



This chapter examines the role of the transport layer in encapsulating application data for use by the network layer. The transport layer also encompasses these functions:

- Enables multiple applications to communicate over the network at the same time on a single device
- Ensures that, if required, all the data is received reliably and in order by the correct application
- Employs error-handling mechanisms

Roles of the Transport Layer

The transport layer provides transparent transfer of data between end users, providing reliable data transfer services to the upper layers. The transport layer controls the reliability of a given link through flow control, segmentation/desegmentation, and error control. Some protocols are state and connection oriented. This means that the transport layer can keep track of the segments and retransmit those that fail.

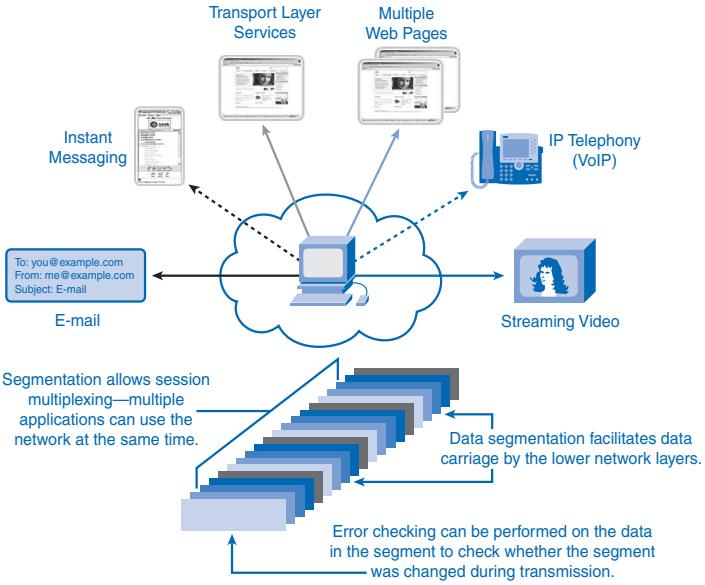
Purpose of the Transport Layer

The following are the primary responsibilities of the transport layer:

- Tracking the individual communications between applications on the source and destination hosts
- Segmenting data and managing each piece
- Reassembling the segments into streams of application data
- Identifying the different applications
- Performing flow control between end users
- Enabling error recovery
- Initiating a session

The transport layer enables applications on devices to communicate, as shown in Figure 4-2.

Figure 4-2 Enabling Applications on Devices to Communicate



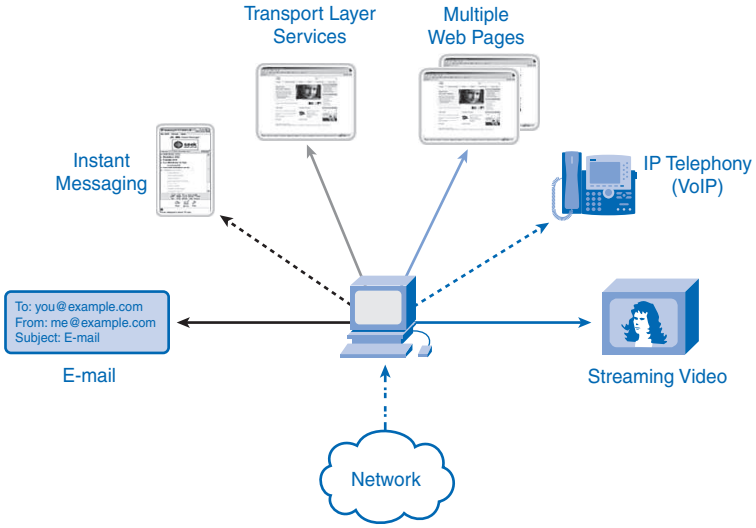
The next sections describe the different roles of the transport layer and data requirements for transport layer protocols.

Tracking Individual Conversations

Any host can have multiple applications that are communicating across the network. Each of these applications will be communicating with one or more applications on remote hosts. It is the responsibility of the transport layer to maintain the multiple communication streams between these applications.

Consider a computer connected to a network that is simultaneously receiving and sending e-mail and instant messages, viewing websites, and conducting a Voice over IP (VoIP) phone call, as shown in Figure 4-3. Each of these applications is sending and receiving data over the network at the same time. However, data from the phone call is not directed to the web browser, and text from an instant message does not appear in an e-mail.

Figure 4-3 Tracking the Conversations



The transport layer segments the data and manages the separation of data for different applications. Multiple applications running on a device receive the correct data.

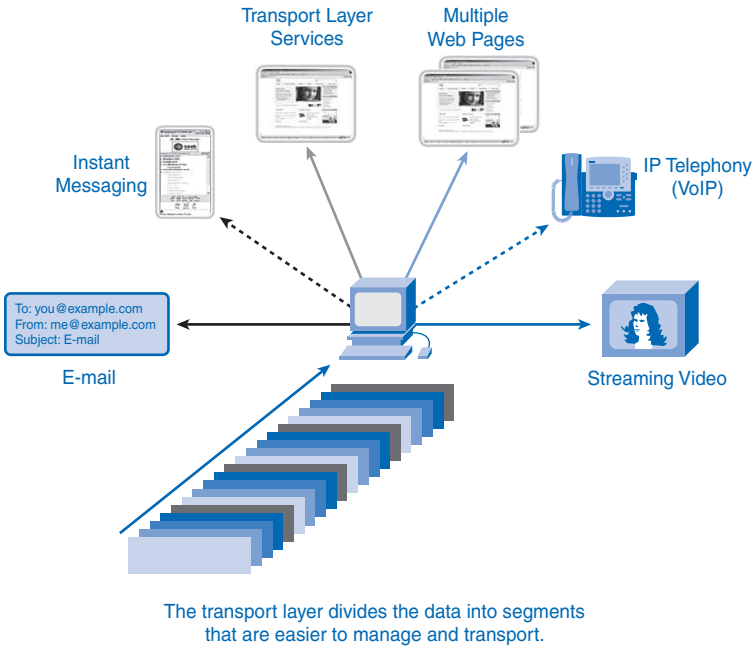
Segmenting Data

The application layer passes large amounts of data to the transport layer. The transport layer has to break the data into smaller pieces, better suited for transmission. These pieces are called segments.

This process includes the encapsulation required on each piece of data. Each piece of application data requires headers to be added at the transport layer to indicate to which communication it is associated.

Segmentation of the data, as shown in Figure 4-4, in accordance with transport layer protocols, provides the means to both send and receive data when running multiple applications concurrently on a computer. Without segmentation, only one application, the streaming video, for example, would be able to receive data. You could not receive e-mails, chat on instant messenger, or view web pages while also viewing the video.

Figure 4-4 Segmentation



Reassembling Segments

Because networks can provide multiple routes that can have different transmission times, data can arrive in the wrong order. By numbering and sequencing the segments, the transport layer can ensure that these segments are reassembled into the proper order.

At the receiving host, each segment of data must be reassembled in the correct order and then directed to the appropriate application.

The protocols at the transport layer describe how the transport layer header information is used to reassemble the data pieces into in-order data streams to be passed to the application layer.

Identifying the Applications

To pass data streams to the proper applications, the transport layer must identify the target application. To accomplish this, the transport layer assigns an identifier to an application.

The TCP/IP protocols call this identifier a *port number*. Each software process that needs to access the network is assigned a port number unique in that host. This port number is used in the transport layer header to indicate to which application that piece of data is associated.

At the transport layer, each particular set of pieces flowing between a source application and a destination application is known as a *conversation*. Dividing data into small parts, and sending these parts from the source to the destination, enables many different communications to be interleaved (multiplexed) on the same network.

The transport layer is the link between the application layer and the lower layers that are responsible for network transmission. This layer accepts data from different conversations and passes it down to the lower layers as manageable pieces that can be eventually multiplexed over the media.

Applications do not need to know the operational details of the network in use. The applications generate data that is sent from one application to another, without regard to the destination host type, the type of media over which the data must travel, the path taken by the data, the congestion on a link, or the size of the network.

Additionally, the lower layers are not aware that multiple applications are sending data on the network. Their responsibility is to deliver data to the appropriate device. The transport layer then sorts these pieces before delivering them to the appropriate application.

Flow Control

Network hosts have limited resources, such as memory or bandwidth. When the transport layer is aware that these resources are overtaxed, some protocols can request that the sending application reduce the rate of data flow. This is done at the transport layer by regulating the amount of data the source transmits as a group. ***Flow control*** can prevent the loss of segments on the network and avoid the need for retransmission.

As the protocols are discussed in this chapter, this service will be explained in more detail.

Error Recovery

For many reasons, it is possible for a piece of data to become corrupted, or lost, as it is transmitted over the network. The transport layer can ensure that all pieces reach their destination by having the source device retransmit any data that is lost.

Initiating a Session

The transport layer can provide connection orientation by creating a session between the applications. These connections prepare the applications to communicate with each other before any data is transmitted. Within these sessions, the data for a communication between the two applications can be closely managed.

Data Requirements Vary

Multiple transport layer protocols exist to meet the requirements of different applications. For example, users require that an e-mail or web page be completely received and presented for the information to be considered useful. Slight delays are considered acceptable to ensure that the complete information is received and presented.

In contrast, occasionally missing small parts of a telephone conversation might be considered acceptable. You can either infer the missing audio from the context of the conversation or ask the other person to repeat what he said. This is considered preferable to the delays that would result from asking the network to manage and resend missing segments. In this example, the user, not the network, manages the resending or replacement of missing information.

In today's converged networks, where the flow of voice, video, and data travels over the same network, applications with very different transport needs can be communicating on the same network. The different transport layer protocols have different rules allowing devices to handle these diverse data requirements.

Some protocols, such as UDP (User Datagram Protocol), provide just the basic functions for efficiently delivering the data pieces between the appropriate applications. These types of protocols are useful for applications whose data is sensitive to delays.

Other transport layer protocols, such as TCP (Transmission Control Protocol), describe processes that provide additional features, such as ensuring reliable delivery between the applications. While these additional functions provide more robust communication at the transport layer between applications, they have additional overhead and make larger demands on the network.

To identify each segment of data, the transport layer adds to the piece a header containing binary data. This header contains fields of bits. The values in these fields enable different transport layer protocols to perform different functions.

Supporting Reliable Communication

Recall that the primary function of the transport layer is to manage the application data for the conversations between hosts. However, different applications have different requirements for their data, and therefore different transport protocols have been developed to meet these requirements.

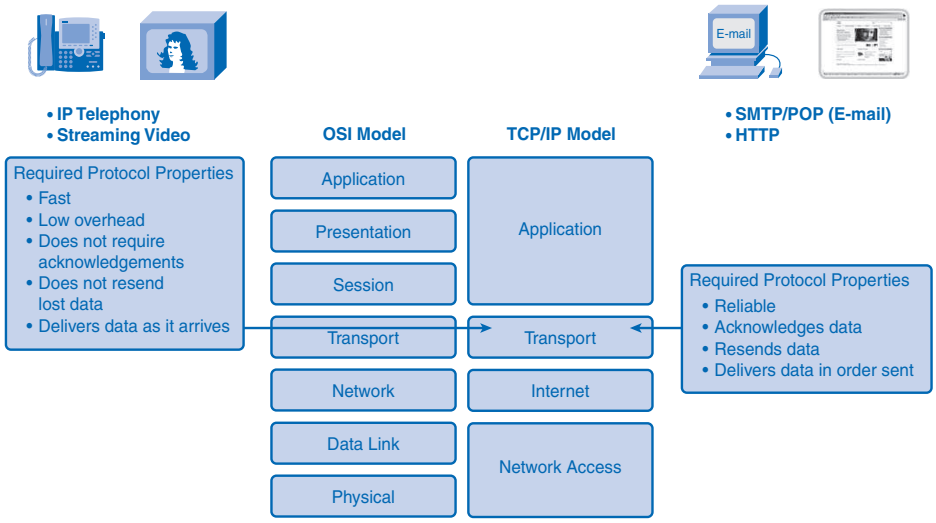
TCP is a transport layer protocol that can be implemented to ensure reliable delivery of the data. In networking terms, reliability means ensuring that each piece of data that the source sends arrives at the destination. At the transport layer, the three basic operations of reliability are

- Tracking transmitted data
- Acknowledging received data
- Retransmitting any unacknowledged data

The transport layer of the sending host tracks all the data pieces for each conversation and retransmits any data that the receiving host did not acknowledge. These reliability processes place additional overhead on the network resources because of the acknowledgment, tracking, and retransmission. To support these reliability operations, more ***control data*** is exchanged between the sending and receiving hosts. This control information is contained in the Layer 4 header.

This creates a trade-off between the value of reliability and the burden it places on the network. Application developers must choose which transport protocol type is appropriate based on the requirements of their applications, as shown in Figure 4-5. At the transport layer, protocols specify methods for either reliable, guaranteed delivery or best-effort delivery. In the context of networking, best-effort delivery is referred to as unreliable, because the destination does not acknowledge whether it received the data.

Figure 4-5 Transport Layer Protocols



Applications, such as databases, web pages, and e-mail, require that all the sent data arrive at the destination in its original condition for the data to be useful. Any missing data could cause a corrupt communication that is either incomplete or unreadable. Therefore, these applications are designed to use a transport layer protocol that implements reliability. The additional network overhead is considered to be required for these applications.

Other applications are more tolerant of the loss of small amounts of data. For example, if one or two segments of a video stream fail to arrive, it would only create a momentary disruption in the stream. This can appear as distortion in the image but might not even be noticeable to the user. Imposing overhead to ensure reliability for this application could reduce the usefulness of the application. The image in a streaming video would be greatly degraded if the destination device had to account for lost data and delay the stream while

waiting for its arrival. It is better to render the best image possible at the time with the segments that arrive and forego reliability. If reliability is required for some reason, these applications can provide error checking and retransmission requests.

TCP and UDP

The two most common transport layer protocols of the TCP/IP protocol suite are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Both protocols manage the communication of multiple applications. The differences between the two are the specific functions that each protocol implements.

User Datagram Protocol (UDP)

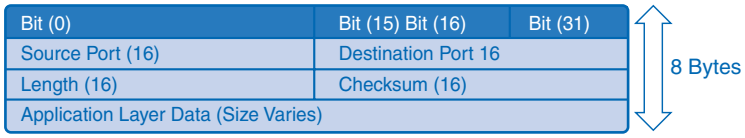
UDP is a simple, connectionless protocol, described in RFC 768. It has the advantage of providing low-overhead data delivery. The segments of communication in UDP are called *datagrams*. UDP sends datagrams as "best effort."

Applications that use UDP include

- Domain Name System (DNS)
- Video streaming
- Voice over IP (VoIP)

Figure 4-6 illustrates a UDP datagram.

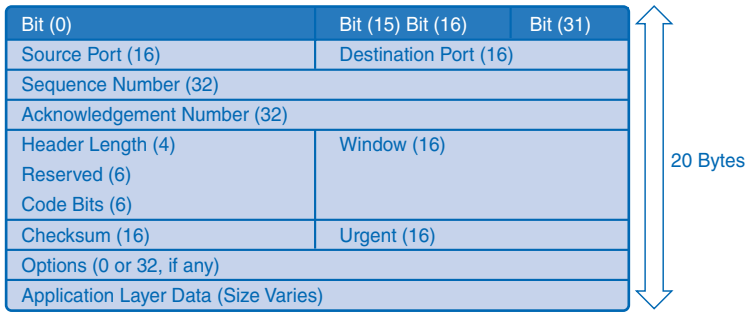
Figure 4-6 UDP Datagram



Transmission Control Protocol (TCP)

TCP is a connection-oriented protocol, described in RFC 793. TCP incurs additional overhead to gain functions. Additional functions specified by TCP are same-order delivery, reliable delivery, and flow control. Each TCP segment has 20 bytes of overhead in the header encapsulating the application layer data, whereas each UDP segment has only 8 bytes of overhead. Figure 4-7 shows the TCP datagram.

Figure 4-7 TCP Datagram



The following applications use TCP:

- Web browsers
- E-mail
- File transfers

Port Addressing

Consider the earlier example of a computer simultaneously receiving and sending e-mail, instant messages, web pages, and a VoIP phone call.

The TCP- and UDP-based services keep track of the various applications that are communicating. To differentiate the segments and datagrams for each application, both TCP and UDP have header fields that can uniquely identify these applications.

Identifying the Conversations

The header of each segment or datagram contains a source and destination port. The source port number is the number for this communication associated with the originating application on the local host. The destination port number is the number for this communication associated with the destination application on the remote host.

Port numbers are assigned in various ways, depending on whether the message is a request or a response. While server processes have static port numbers assigned to them, clients dynamically choose a port number for each conversation.

When a client application sends a request to a server application, the destination port contained in the header is the port number that is assigned to the service daemon running on the remote host. The client software must know what port number is associated with the server process on the remote host. This destination port number is configured, either by default or manually. For example, when a web browser application makes a request to a web server, the browser uses TCP and port number 80 unless otherwise specified. TCP port 80 is the default port assigned to web-serving applications. Many common applications have default port assignments.

The source port in a segment or datagram header of a client request is randomly generated. As long as it does not conflict with other ports in use on the system, the client can choose any port number. This port number acts like a return address for the requesting application. The transport layer keeps track of this port and the application that initiated the request so that when a response is returned, it can be forwarded to the correct application. The requesting application port number is used as the destination port number in the response coming back from the server.

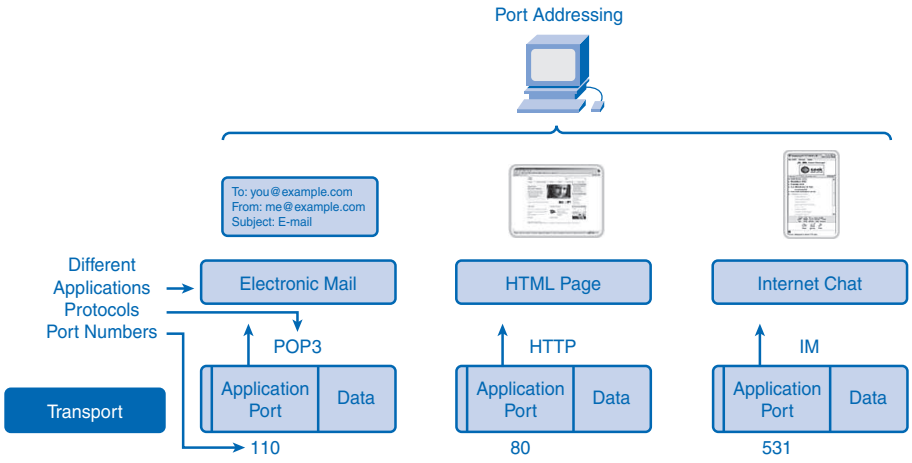
The combination of the transport layer port number and the network layer IP address assigned to the host uniquely identifies a particular process running on a specific host device. This combination is called a *socket*. Occasionally, you can find the terms *port number* and *socket* used interchangeably. In the context of this book, the term *socket* refers only to the unique combination of IP address and port number. A socket pair, consisting of the source and destination IP addresses and port numbers, is also unique and identifies the conversation between the two hosts.

For example, an HTTP web page request being sent to a web server (port 80) running on a host with a Layer 3 IPv4 address of 192.168.1.20 would be destined to socket 192.168.1.20:80.

If the web browser requesting the web page is running on host 192.168.100.48 and the dynamic port number assigned to the web page would be 192.168.100.48:49152.

These unique identifiers are the port numbers, and the process of identifying the different conversations through the use of port numbers is shown in Figure 4-8.

Figure 4-8 Identifying Conversations



Port Addressing Types and Tools

The ***Internet Assigned Numbers Authority (IANA)*** assigns port numbers. IANA is a standards body that is responsible for assigning various addressing standards.

The different types of port numbers are

- Well-known ports (numbers 0 to 1023)
- Registered ports (numbers 1024 to 49151)
- Dynamic or private ports (numbers 49152 to 65535)

The following sections describe the three types of port numbers and examples of when both TCP and UDP might use the same port number. You also learn about the netstat network utility.

Well-Known Ports

Well-known ports (numbers 0 to 1023) are reserved for services and applications. They are commonly used for applications such as HTTP (web server), POP3/SMTP (e-mail server), and Telnet. By defining these well-known ports for server applications, client applications can be programmed to request a connection to that specific port and its associated service. Table 4-1 lists some well-known ports for TCP and UDP.

Table 4-1 Well-Known Ports

Well-Known Port	Application	Protocol
20	File Transfer Protocol (FTP) Data	TCP
21	File Transfer Protocol (FTP) Control	TCP
23	Telnet	TCP
25	Simple Mail Transfer Protocol (SMTP)	TCP
69	Trivial File Transfer Protocol (TFTP)	UDP
80	Hypertext Transfer Protocol (HTTP)	TCP
110	Post Office Protocol 3 (POP3)	TCP
194	Internet Relay Chat (IRC)	TCP
443	Secure HTTP (HTTPS)	TCP
520	Routing Information Protocol (RIP)	UDP

Registered Ports

Registered ports (numbers 1024 to 49151) are assigned to user processes or applications. These processes are primarily individual applications that a user has chosen to install rather than common applications that would receive a well-known port. When not used for a server resource, a client can dynamically select a registered port as its source port. Table 4-2 lists registered ports for TCP and UDP.

Table 4-2 Registered Ports

Registered Port	Application	Protocol
1812	RADIUS Authentication Protocol	UDP
1863	MSN Messenger	TCP
2000	Cisco Skinny Client Control Protocol (SCCP, used in VoIP applications)	UDP
5004	Real-Time Transport Protocol (RTP, a voice and video transport protocol)	UDP
5060	Session Initiation Protocol (SIP, used in VoIP applications)	UDP
8008	Alternate HTTP	TCP
8080	Alternate HTTP	TCP

Dynamic or Private Ports

Dynamic or private ports (numbers 49152 to 65535), also known as ephemeral ports, are usually assigned dynamically to client applications when initiating a connection. It is not common for a client to connect to a service using dynamic or private ports (although some peer-to-peer file-sharing programs do).

Using Both TCP and UDP

Some applications can use both TCP and UDP. For example, the low overhead of UDP enables DNS to serve many client requests very quickly. Sometimes, however, sending the requested information can require the reliability of TCP. In this case, both protocols use the well-known port number of 53 with this service. Table 4-3 lists examples of registered and well-known TCP and UDP common ports.

Table 4-3 TCP/UDP Common Ports

Common Port	Application	Port Type
53	DNS	Well-known TCP/UDP common port
161	SNMP	Well-known TCP/UDP common port
531	AOL Instant Messenger, IRC	Well-known TCP/UDP common port
1433	MS SQL	Registered TCP/UDP common port
2948	WAP (MMS)	Registered TCP/UDP common port

netstat Command

Sometimes it is necessary to know which active TCP connections are open and running on a networked host. The **netstat** command is an important network utility that you can use to verify those connections. **netstat** lists the protocol in use, the local address and port number, the destination address and port number, and the state of the connection.

Unexplained TCP connections can indicate that something or someone is connected to the local host, which is a major security threat. Additionally, unnecessary TCP connections can consume valuable system resources, thus slowing the host's performance. Use **netstat** to examine the open connections on a host when performance appears to be compromised.

Many useful options are available for the **netstat** command. Example 4-1 shows **netstat** output.

Example 4-1 netstat Command

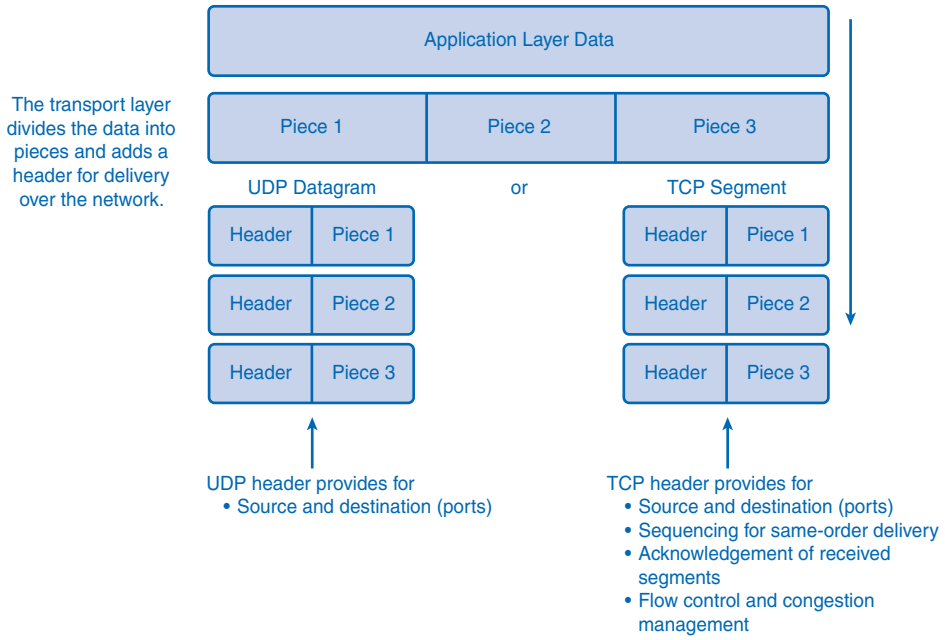
```
C:\> netstat

Active Connections
Proto      Local Address    Foreign Address           State
TCP        kenpc:3126       192.168.0.2:netbios-ssn   ESTABLISHED
TCP        kenpc:3158       207.138.126.152:http      ESTABLISHED
TCP        kenpc:3159       207.138.126.169:http      ESTABLISHED
TCP        kenpc:3160       207.138.126.169:http      ESTABLISHED
TCP        kenpc:3161       sc.msn.com:http           ESTABLISHED
TCP        kenpc:3166       www.cisco.com:http        ESTABLISHED
C:\>
```

Segmentation and Reassembly: Divide and Conquer

Chapter 2, "Communicating over the Network," explained how an application passes data down through the various protocols to create a protocol data unit (PDU) that is then transmitted on the medium. At the application layer, the data is passed down and is segmented

into pieces. A UDP segment (piece) is called a *datagram*. A TCP segment (piece) is called a *segment*. A UDP header provides source and destination (ports). A TCP header provides source and destination (ports), sequencing, acknowledgments, and flow control. At the destination host, this process is reversed until the data can be passed up to the application. Figure 4-9 provides an example.

Figure 4-9 Transport Layer Functions



Some applications transmit large amounts of data—in some cases, many gigabytes. Sending all this data in one large piece would be impractical. A large piece of data could take minutes or even hours to send, and no other network traffic could be transmitted at the same time. In addition, if errors occurred during the transmission, the entire data file would be lost or would have to be re-sent. Network devices would not have memory buffers large enough to store this much data while it is being transmitted or received. The size of the segment varies depending on the networking technology and specific physical medium in use.

Dividing application data into segments both ensures that data is transmitted within the limits of the media and that data from different applications can be multiplexed onto the media. TCP and UDP handle segmentation differently.

In TCP, each segment header contains a sequence number. This sequence number allows the transport layer functions on the destination host to reassemble segments in the order in which they were transmitted. This ensures that the destination application has the data in the exact form the sender intended.

Although services using UDP also track the conversations between applications, they are not concerned with the order in which the information was transmitted or in maintaining a connection. The UDP header does not include a sequence number. UDP is a simpler design and generates less overhead than TCP, resulting in a faster transfer of data.

Information can arrive in a different order than it was transmitted because different packets can take different paths through the network. An application that uses UDP must tolerate the fact that data might not arrive in the order in which it was sent.

Packet Tracer Activity

UDP and TCP Port Numbers (4.1.6.2)

In this activity, you will "look inside" packets to see how DNS and HTTP use port numbers. Use file e1-4162.pka on the CD-ROM that accompanies this book to perform this activity using Packet Tracer.

TCP: Communicating with Reliability

TCP is often referred to as a connection-oriented protocol, a protocol that guarantees reliable and in-order delivery of data from sender to receiver.

In the following sections, you explore how this is managed. Connection establishment and termination are discussed, along with the use of three-way handshake. Flow control, the use of windowing as congestion control, and retransmission of data are presented.

Making Conversations Reliable

The key distinction between TCP and UDP is reliability. The reliability of TCP communication is performed using connection-oriented sessions. Before a host using TCP sends data to another host, the transport layer initiates a process to create a connection with the destination. This connection enables the tracking of a session, or communication stream, between the hosts. This process ensures that each host is aware of and prepared for the communication. A complete TCP conversation requires the establishment of a session between the hosts in both directions.

After a session has been established, the destination sends acknowledgments to the source for the segments that it receives. These acknowledgments form the basis of reliability within the TCP session. As the source receives an acknowledgment, it knows that the data has been successfully delivered and can quit tracking that data. If the source does not receive an acknowledgment within a predetermined amount of time, it retransmits that data to the destination.

Part of the additional overhead of using TCP is the network traffic generated by acknowledgments and retransmissions. The establishment of the sessions creates overhead in the

form of additional segments being exchanged. Additional overhead is the result of keeping track of acknowledgments and the retransmission process the host must undertake if no acknowledgment is received.

Reliability is achieved by having fields in the TCP segment, each with a specific function. These fields will be discussed in later sections.

TCP Server Processes

As discussed in Chapter 3, "Application Layer Functionality and Protocols," application processes run on servers. These processes wait until a client initiates communication with a request for information or other services.

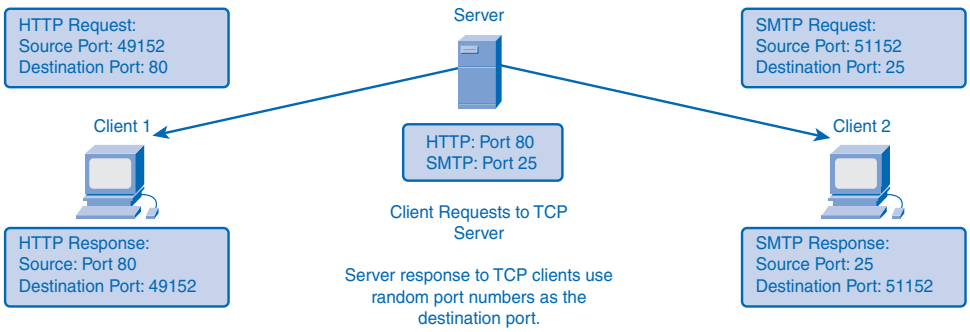
Each application process running on the server is configured to use a port number, either by default or manually by a system administrator. An individual server cannot have two services assigned to the same port number within the same transport layer services. A host running a web server application and a file transfer application cannot have both configured to use the same port (for example, TCP port 8080).

When an active server application is assigned to a specific port, that port is considered to be "open" on the server. This means that the transport layer accepts and processes segments addressed to that port. Any incoming client request addressed to the correct socket is accepted, and the data is passed to the server application. There can be many simultaneous ports open on a server, one for each active server application. It is common for a server to provide more than one service, such as a web server and an FTP server, at the same time.

One way to improve security on a server is to restrict server access to only those ports associated with the services and applications that should be accessible to authorized requestors.

Figure 4-10 shows the typical allocation of source and destination ports in TCP client/server operations.

Figure 4-10 Clients Sending TCP Requests



TCP Connection Establishment and Termination

When two hosts communicate using TCP, a connection is established before data can be exchanged. After the communication is completed, the sessions are closed and the connection is terminated. The connection and session mechanisms enable TCP's reliability function.

TCP Three-Way Handshake

The host tracks each data segment within a session and exchanges information about what data is received by each host using the information in the TCP header.

Each connection represents two one-way communication streams, or sessions. To establish the connection, the hosts perform a three-way handshake. Control bits in the TCP header indicate the progress and status of the connection. The three-way handshake performs the following functions:

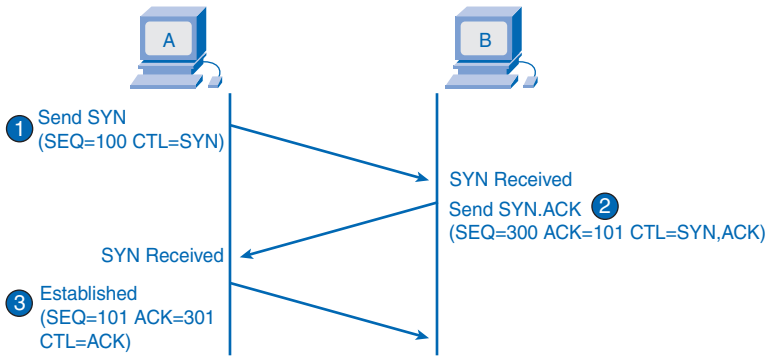
- Establishes that the destination device is present on the network
- Verifies that the destination device has an active service and is accepting requests on the destination port number that the initiating client intends to use for the session
- Informs the destination device that the source client intends to establish a communication session on that port number

In TCP connections, the host serving as a client initiates the session to the server. The three steps in TCP connection establishment are as follows:

1. The initiating client sends a segment containing an initial sequence value, which serves as a request to the server to begin a communications session.
2. The server responds with a segment containing an acknowledgment value equal to the received sequence value plus 1, plus its own synchronizing sequence value. The acknowledgment value is 1 greater than the sequence number because there is no data contained to be acknowledged. This acknowledgment value enables the client to tie the response back to the original segment that it sent to the server.
3. The initiating client responds with an acknowledgment value equal to the sequence value it received plus 1. This completes the process of establishing the connection.

Figure 4-11 shows the steps to establish a TCP connection.

Figure 4-11 TCP Connection Establishment: SYN ACK



To understand the three-way handshake process, it is important to look at the various values that the two hosts exchange. Within the TCP segment header, the following six 1-bit fields contain control information used to manage the TCP processes:

- ***URG*:** Urgent pointer field significant
- ***ACK*:** Acknowledgment field significant
- ***PSH*:** Push function
- ***RST*:** Reset the connection
- ***SYN*:** Synchronize sequence numbers
- ***FIN*:** No more data from sender

These fields are referred to as *flags*, because the value of one of these fields is only 1 bit and, therefore, has only two values: 1 or 0. When a bit value is set to 1, it indicates what control information is contained in the segment.

The next sections describe each step of the three-way handshake in more detail.

Step 1: SYN

A TCP client begins the three-way handshake by sending a segment with the SYN control flag set, indicating an initial value in the sequence number field in the header. This initial value for the sequence number, known as the initial sequence number (ISN), is randomly chosen and is used to begin tracking the flow of data from the client to the server for this session. The ISN in the header of each segment is increased by 1 for each byte of data sent from the client to the server as the data conversation continues. The SYN control flag is set and the relative sequence number is at 0.

Step 2: SYN and ACK

The TCP server needs to acknowledge the receipt of the SYN segment from the client to establish the session from the client to the server. To do so, the server sends a segment back to the client with the ACK flag set, indicating that the acknowledgment number is significant. With this flag set in the segment, the client recognizes this as an acknowledgment that the server received the SYN from the TCP client.

The value of the ***acknowledgment*** number field is equal to the client ISN plus 1. This establishes a session from the client to the server. The ACK flag will remain set for the balance of the session. The conversation between the client and the server is two one-way sessions: one from the client to the server and the other from the server to the client. In this second step of the three-way handshake, the server must initiate the response from the server to the client. To start this session, the server uses the SYN flag in the same way that the client did. It sets the SYN control flag in the header to establish a session from the server to the client. The SYN flag indicates that the initial value of the sequence number field is in the header. This value will be used to track the flow of data in this session from the server back to the client.

Step 3: ACK

Finally, the TCP client responds with a segment containing an ACK that is the response to the TCP SYN sent by the server. This segment does not include user data. The value in the acknowledgment number field contains one more than the ISN received from the server. After both sessions are established between client and server, all additional segments exchanged in this communication will have the ACK flag set.

You can add security to the data network by doing the following:

- Denying the establishment of TCP sessions
- Allowing sessions to be established for specific services only
- Allowing traffic only as a part of already established sessions

You can implement this security for all TCP sessions or only for selected sessions.

TCP Session Termination

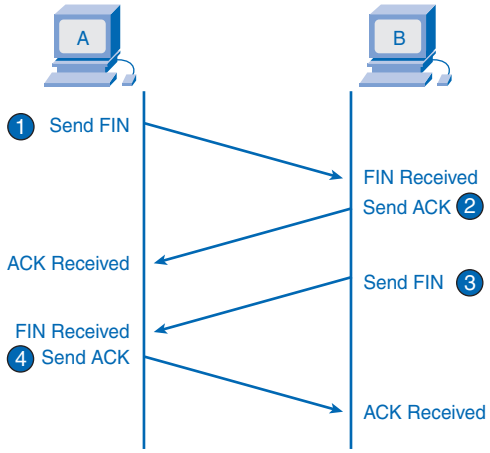
To close a connection, the FIN control flag in the segment header must be set. To end each one-way TCP session, a two-way handshake is used, consisting of a FIN segment and an ACK segment. Therefore, to terminate a single conversation supported by TCP, four exchanges are needed to end both sessions:

1. When the client has no more data to send in the stream, it sends a segment with the FIN flag set.

2. The server sends an ACK to acknowledge the receipt of the FIN to terminate the session from client to server.
3. The server sends a FIN to the client, to terminate the server-to-client session.
4. The client responds with an ACK to acknowledge the FIN from the server.

Figure 4-12 shows the steps used to terminate a TCP connection.

Figure 4-12 TCP Connection Termination: FIN ACK



Note

In this explanation, the terms *client* and *server* are used in this description as a reference for simplicity, but the termination process can be initiated by any two hosts that complete the session.

When the client end of the session has no more data to transfer, it sets the FIN flag in the header of a segment. Next, the server end of the connection will send a normal segment containing data with the ACK flag set using the acknowledgment number, confirming that all the bytes of data have been received. When all segments have been acknowledged, the session is closed.

The session in the other direction is closed using the same process. The receiver indicates that there is no more data to send by setting the FIN flag in the header of a segment sent to the source. A return acknowledgment confirms that all bytes of data have been received and that the session is, in turn, closed.

It is also possible to terminate the connection by a three-way handshake. When the client has no more data to send, it sends a FIN to the server. If the server also has no more data to send, it can reply with both the FIN and ACK flags set, combining two steps into one. The client replies with an ACK.

Packet Tracer Activity

TCP Session Establishment and Termination (4.2.5.2)

In this activity, you will study the TCP three-way handshake for session establishment and the TCP process for session termination. Many application protocols use TCP, and visualizing the session establishment and termination processes with Packet Tracer will deepen your understanding. Use file e1-4252.pka on the CD-ROM that accompanies this book to perform this activity using Packet Tracer.

TCP Acknowledgment with Windowing

One of TCP's functions is to make sure that each segment reaches its destination. The TCP services on the destination host acknowledge the data that they have received to the source application.

The segment header sequence number and acknowledgment number are used together to confirm receipt of the bytes of data contained in the segments. The sequence number indicates the relative number of bytes that have been transmitted in this session, including the bytes in the current segment. TCP uses the acknowledgment number in segments sent back to the source to indicate the next byte in this session that the receiver expects to receive. This is called *expectational acknowledgment*.

The source is informed that the destination has received all bytes in this data stream up to, but not including, the byte indicated by the acknowledgment number. The sending host is expected to send a segment that uses a sequence number that is equal to the acknowledgment number.

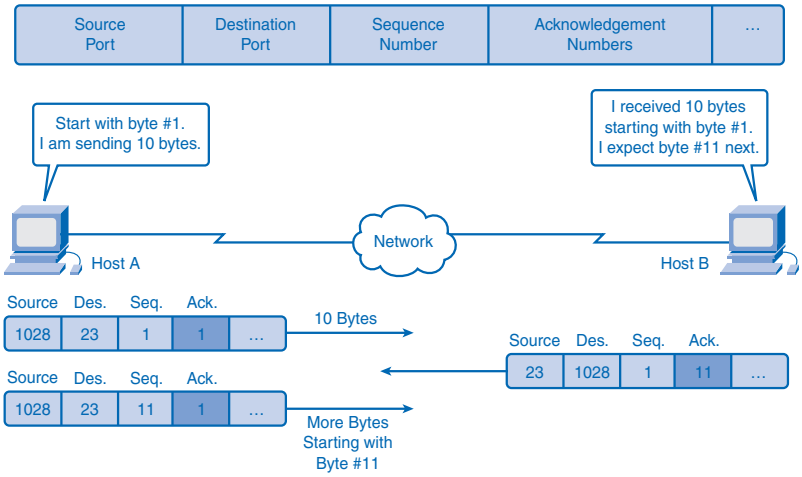
Remember, each connection is actually two one-way sessions. Sequence numbers and acknowledgment numbers are being exchanged in both directions.

In Figure 4-13, the host on the left is sending data to the host on the right. It sends a segment containing 10 bytes of data for this session and a sequence number equal to 1 in the header.

Host B receives the segment at Layer 4 and determines that the sequence number is 1 and that it has 10 bytes of data. Host B then sends a segment back to host A to acknowledge the receipt of this data. In this segment, the host sets the acknowledgment number to 11 to indicate that the next byte of data it expects to receive in this session is byte number 11.

When host A receives this acknowledgment, it can now send the next segment containing data for this session starting with byte number 11.

Figure 4-13 Acknowledgment of TCP Segments



Looking at this example, if host A had to wait for acknowledgment of the receipt of each 10 bytes, the network would have a lot of overhead. To reduce the overhead of these acknowledgments, multiple segments of data can be sent and acknowledged with a single TCP message in the opposite direction. This acknowledgment contains an acknowledgment number based on the total number of segments received in the session.

For example, starting with a sequence number of 2000, if 10 segments of 1000 bytes each were received, an acknowledgment number of 12001 would be returned to the source.

The amount of data that a source can transmit before an acknowledgment must be received is called the ***window size***. Window size is a field in the TCP header that enables the management of lost data and flow control.

TCP Retransmission

No matter how well designed a network is, data loss will occasionally occur. Therefore, TCP provides methods of managing these segment losses, including a mechanism to retransmit segments with unacknowledged data.

A destination host service using TCP usually only acknowledges data for contiguous sequence bytes. If one or more segments are missing, only the data in the segments that complete the stream is acknowledged. For example, if segments with sequence numbers 1500 to 3000 and 3400 to 3500 were received, the acknowledgment number would be 3001, because segments with the sequence numbers 3001 to 3399 have not been received.

When TCP at the source host has not received an acknowledgment after a predetermined amount of time, it will go back to the last acknowledgment number that it received and

retransmit data from that point forward. The retransmission process is not specified by RFC 793 but is left up to the particular implementation of TCP.

For a typical TCP implementation, a host can transmit a segment, put a copy of the segment in a retransmission queue, and start a timer. When the data acknowledgment is received, the segment is deleted from the queue. If the acknowledgment is not received before the timer expires, the segment is retransmitted.

Hosts today can also employ an optional feature called *selective acknowledgments*. If both hosts support selective acknowledgments, it is possible for the destination to acknowledge bytes in noncontiguous segments, and the host would only need to retransmit the missing data.

TCP Congestion Control: Minimizing Segment Loss

TCP provides congestion control through the use of flow control and dynamic window sizes. The following sections discuss how these techniques minimize segment loss that minimizes network overhead caused by retransmission of lost segments.

Flow Control

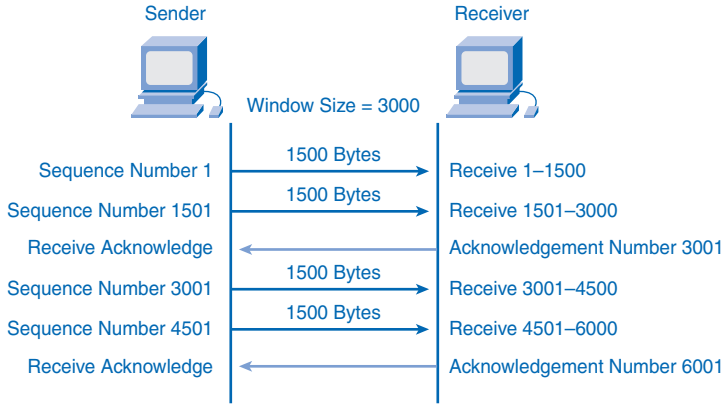
Flow control assists the reliability of TCP transmission by adjusting the effective rate of data flow between the two services in the session. When the source is informed that the specified amount of data in the segments is received, it can continue sending more data for this session.

The window size field in the TCP header specifies the amount of data that can be transmitted before an acknowledgment must be received. The initial window size is determined during the session startup through the three-way handshake.

The TCP feedback mechanism adjusts the effective rate of data transmission to the maximum flow that the network and destination device can support without loss. TCP attempts to manage the rate of transmission so that all data will be received and retransmissions will be minimized.

Figure 4-14 shows a simplified representation of window size and acknowledgments. In this example, the initial window size for a TCP session represented is set to 3000 bytes. When the sender has transmitted 3000 bytes, it waits for an acknowledgment of these bytes before transmitting more segments in this session. After the sender has received this acknowledgment from the receiver, the sender can transmit an additional 3000 bytes.

Figure 4-14 TCP Segment Acknowledgment and Window Size



During the delay in receiving the acknowledgment, the sender will not send additional segments for this session. In periods when the network is congested or the resources of the receiving host are strained, the delay can increase. As this delay grows longer, the effective transmission rate of the data for this session decreases. The slowdown in data rate helps reduce the resource contention.

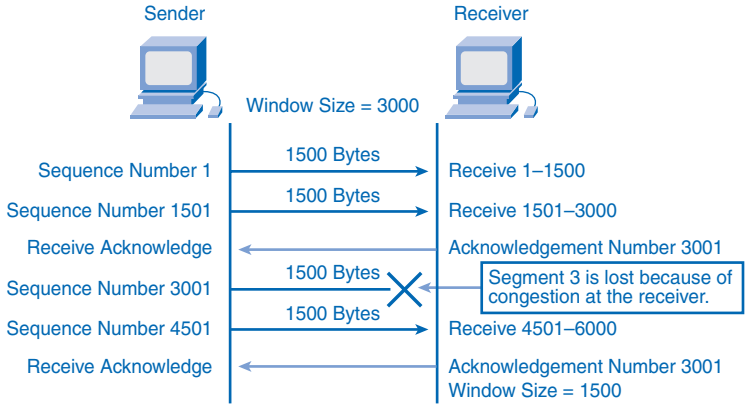
Dynamic Window Sizes

Another way to control the data flow is to use dynamic window sizes. When network resources are constrained, TCP can reduce the window size to require that received segments be acknowledged more frequently. This effectively slows the rate of transmission because the source must wait for data to be acknowledged.

The TCP receiving host sends the window size value to the sending TCP to indicate the number of bytes that it is prepared to receive as a part of this session. If the destination needs to slow the rate of communication because of limited buffer memory, it can send a smaller window size value to the source as part of an acknowledgment.

As shown in Figure 4-15, if a receiving host has congestion, it can respond to the sending host with a segment with a reduced window size. Figure 4-15 shows a loss of one of the segments. The receiver changed the window size field in the TCP header of the returning segments in this conversation from 3000 to 1500. This caused the sender to reduce the window size to 1500.

Figure 4-15 TCP Congestion and Flow Control



After periods of transmission with no data losses or constrained resources, the receiver will begin to increase the window size field. This reduces the overhead on the network because fewer acknowledgments need to be sent. Window size will continue to increase until data loss occurs, which will cause the window size to be decreased.

This dynamic increasing and decreasing of window size is a continuous process in TCP, which determines the optimum window size for each TCP session. In highly efficient networks, window sizes can become very large because data is not being lost. In networks where the underlying infrastructure is being stressed, the window size will likely remain small.

UDP: Communicating with Low Overhead

UDP is a simple protocol that provides the basic transport layer functions. It has much lower overhead than TCP, because it is not connection oriented and does not provide the sophisticated retransmission, sequencing, and flow control mechanisms. The following sections compare how UDP handles low overhead and reliability, data reassembly, server processes and requests, and client processes.

UDP: Low Overhead Versus Reliability

Because UDP has low overhead and does not provide the functionality that TCP provides for reliability, care must be taken when you choose to use UDP. Applications that use UDP are not always unreliable. Using UDP simply means that reliability is not provided by the transport layer protocol and must be implemented elsewhere if required.

Some applications, such as online games or VoIP, can tolerate loss of some data. If these applications used TCP, they might experience large delays while TCP detects data loss and retransmits data. These delays would be more detrimental to the application than small data losses. Some applications, such as DNS, will simply retry the request if they do not receive a response, and therefore they do not need TCP to guarantee the message delivery. The low overhead of UDP makes it desirable for such applications.

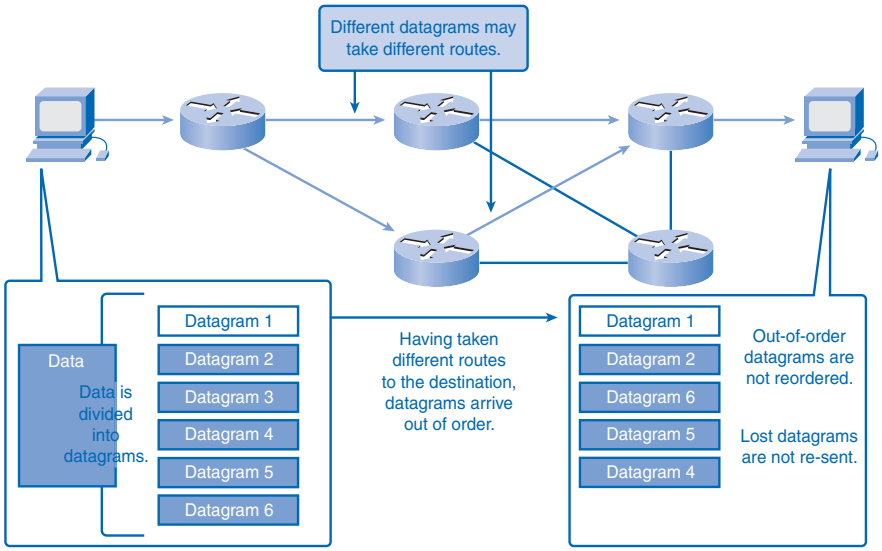
UDP Datagram Reassembly

Because UDP is connectionless, sessions are not established before communication takes place as they are with TCP. UDP is transaction based. In other words, when an application has data to send, it simply sends the data.

Many applications that use UDP send small amounts of data that can fit in one segment. However, some applications will send larger amounts of data that must be split into multiple segments The UDP PDU is referred to as a datagram, although the terms *segment* and *datagram* are sometimes used interchangeably to describe a transport layer PDU.

When multiple datagrams are sent to a destination, they can take different paths and arrive in the wrong order, as shown in Figure 4-16. UDP does not keep track of sequence numbers the way TCP does. UDP has no way to reorder the datagrams into their transmission order. Therefore, UDP simply reassembles the data in the order that it was received and forwards it to the application. If the sequence of the data is important to the application, the application will have to identify the proper sequence of the data and determine how it should be processed.

Figure 4-16 UDP Data Reassembly



UDP Server Processes and Requests

Like TCP-based applications, UDP-based server applications are assigned well-known or registered port numbers. When these applications or processes are running, they will accept the data matched with the assigned port number. When UDP receives a datagram destined for one of these ports, it forwards the application data to the appropriate application based on its port number.

UDP Client Processes

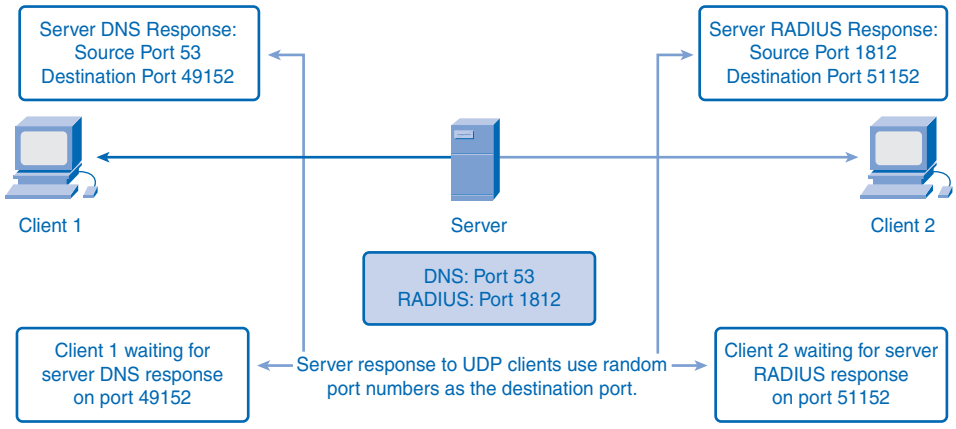
As with TCP, client/server communication is initiated by a client application that is requesting data from a server process. The UDP client process randomly selects a port number from the dynamic range of port numbers and uses this as the source port for the conversation. The destination port will usually be the well-known or registered port number assigned to the server process.

Randomized source port numbers also help with security. If there is a predictable pattern for destination port selection, an intruder can more easily simulate access to a client by attempting to connect to the port number most likely to be open.

Because UDP does not create a session, as soon as the data is ready to be sent and the ports are identified, UDP can form the datagram and pass it to the network layer to be addressed and sent on the network.

Remember, after a client has chosen the source and destination ports, the same pair of ports is used in the header of all datagrams used in the transaction. For the data returning to the client from the server, the source and destination port numbers in the datagram header are reversed. Figure 4-17 shows the clients sending UDP requests.

Figure 4-17 Clients Sending UDP Requests



UDP Operation (4.4.4.2)

In this activity, you examine how DNS uses UDP. Use file e1-4442.pka on the CD-ROM that accompanies this book to perform this activity using Packet Tracer.

Summary

The transport layer provides data network needs by

- Tracking the individual communications between applications on the source and destination hosts
- Segmenting data and managing each piece
- Reassembling the segments into streams of application data
- Identifying the different applications
- Performing flow control between end users
- Enabling error recovery
- Initiating a session

UDP and TCP are common transport layer protocols. UDP datagrams and TCP segments have headers prefixed to the data that include a source port number and destination port number. These port numbers enable data to be directed to the correct application running on the destination computer.

TCP does not pass data to the network until it knows that the destination is ready to receive it. TCP then manages the flow of the data and resends any data segments that are not acknowledged as being received at the destination. TCP uses mechanisms of the three-way handshake, timers, acknowledgments, and dynamic windowing to achieve these reliable features. This reliability does, however, impose overhead on the network in terms of much larger segment headers and more network traffic between the source and destination managing the data transport.

If the application data needs to be delivered across the network quickly, or if network bandwidth cannot support the overhead of control messages being exchanged between the source and the destination systems, UDP is the developer's preferred transport layer protocol. UDP does not track or acknowledge the receipt of datagrams at the destination; it just passes received datagrams to the application layer as they arrive. UDP does not resend lost datagrams; however, this does not necessarily mean that the communication itself is unreliable. The application layer protocols and services can process lost or delayed datagrams if the application has these requirements.

The choice of transport layer protocol is made by the developer of the application to best meet the user requirements. The developer bears in mind, though, that all the other layers play a part in data network communications and will influence its performance.

Labs

The labs available in the companion *Network Fundamentals, CCNA Exploration Labs and Study Guide* (ISBN 1-58713-203-6) provide hands-on practice with the following topics introduced in this chapter:

Lab 4-1: Observing TCP and UDP Using netstat (4.5.1.1)

In this lab, you will examine the **netstat** (network statistics utility) command on a host computer and adjust **netstat** output options to analyze and understand TCP/IP transport layer protocol status.

Lab 4-2: TCP/IP Transport Layer Protocols, TCP and UDP (4.5.2.1)

In this lab, you will use Wireshark to capture and identify TCP header fields and operation during an FTP session and UDP header fields and operation during a TFTP session.

Lab 4-3: Application and Transport Layer Protocols (4.5.3.1)

In this lab, you will use Wireshark to monitor and analyze client application (FTP and HTTP) communications between a server and clients.

Many of the hands-on labs include Packet Tracer companion activities, where you can use Packet Tracer to complete a simulation of the lab. Look for this icon in *Network Fundamentals, CCNA Exploration Labs and Study Guide* (ISBN 1-58713-203-6) for hands-on labs that have Packet Tracer companion activities.

Check Your Understanding

Complete all the review questions listed here to test your understanding of the topics and concepts in this chapter. Appendix A, "Check Your Understanding and Challenge Questions Answer Key," lists the answers.

1. Which port number is used by HTTP?

   A. 23
   B. 80
   C. 53
   D. 110

2. Which port number is used with SMTP?

   A. 20
   B. 23
   C. 25
   D. 143

3. Which characteristics are part of TCP? (Choose two.)

   A. Reliable
   B. Connectionless
   C. No flow control
   D. Resends anything not received

4. At the transport layer, which of the following controls is used to keep a transmitting host from overflowing the buffers of a receiving host?

   A. Best effort
   B. Encryption
   C. Flow control
   D. Congestion avoidance

5. End systems use port numbers to select the proper application. What is the lowest port number that can be dynamically assigned by the host system?

   A. 1
   B. 128
   C. 256
   D. 1024

6. During data transfer, what is the receiving host responsible for? (Choose the best two answers.)

   A. Encapsulation
   B. Bandwidth
   C. Segmentation
   D. Acknowledgment
   E. Reassembly

7. What are the transport layer's responsibilities?

8. Why does TCP use a sequence number in the header?

   A. To reassemble the segments into data
   B. To identify the application layer protocol
   C. To indicate the number of the next expected byte
   D. To show the maximum number of bytes allowed during a session

9. Which of the following determines how much data a sending host running TCP/IP can transmit before it must receive an acknowledgment?

   A. Segment size
   B. Transmission rate
   C. Bandwidth
   D. Window size

10. What is the purpose of TCP/UDP port numbers?

    A. To indicate the beginning of a three-way handshake
    B. To reassemble the segments into the correct order
    C. To identify the number of data packets that can be sent without acknowledgment
    D. To track the different conversations crossing the network at the same time

11. What does segmentation provide to communications?

12. In networking terms, what is reliability?

13. List three network applications that use TCP.

14. List three network applications that use UDP.

15. What is contained in the header of each segment or datagram?

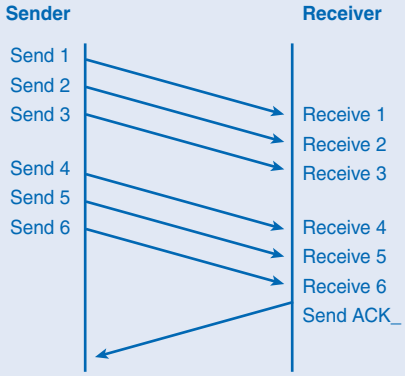
16. What is the purpose of sequence numbers?

Challenge Questions and Activities

These questions require a deeper application of the concepts covered in this chapter. You can find the answers in Appendix A.

1. Which acknowledgment number should be sent by the receiver shown in Figure 4-18?

Figure 4-18 Challenge Question 1



2. What is the protocol number for UDP?

   A. 1
   B. 3
   C. 6
   D. 17

3. What is the default DNS port number?

   A. 1025
   B. 53
   C. 110
   D. 143

4. What is the netstat utility used for on a host?

5. Explain an expectational acknowledgment.

Look for this icon in *Network Fundamentals, CCNA Exploration Labs and Study Guide* (ISBN 1-58713-203-6) for instructions on how to perform the Packet Tracer Skills Integration Challenge for this chapter.