

Week 10

Transport layer The data link layer is responsible for delivery of frames between two neighboring nodes over a link. This is called *node-to-node delivery*. The network layer is responsible for delivery of datagrams between two hosts. This is called *host-to-host delivery*. Communication on the Internet is not defined as the exchange of data between two nodes or between two hosts. Real communication takes place between two processes (application programs). We need process-to-process delivery. However, at any moment, several processes may be running on the source host and several on the destination host. To complete the delivery, we need a mechanism to deliver data from one of these processes running on the source host to the corresponding process running on the destination host. The transport layer is responsible for process-to-process delivery-the delivery of a packet, part of a message, from one process to another. Two processes communicate in a client/server relationship.

Client/Server Paradigm

Although there are several ways to achieve process-to-process communication, the most common one is through the client/server paradigm. A process on the local host, called a client, needs services from a process usually on the remote host, called a server. Both processes (client and server) have the same name. For example, to get the day and time from a remote machine, we need a Daytime client process running on the local host and a Daytime server process running on a remote machine. Operating systems today support both multiuser and multiprogramming environments. A remote computer can run several server programs at the same time, just as local computers can run one or more client programs at the same time. For communication, we must define the following:

1. Local host
2. Local process
3. Remote host
4. Remote process

Addressing

Whenever we need to deliver something to one specific destination among many, we need an address. At the data link layer, we need a MAC address to choose one node among several nodes if the connection is not point-to-point. A frame in the data link layer needs a destination MAC address for delivery and a source address for the next node's reply. At the network layer, we need an IP address to choose one host among millions. A datagram in the network layer needs a destination IP address for delivery and a source IP address for the destination's reply. At the transport layer, we need a transport layer address, called a port number, to choose among multiple processes running on the destination host. The destination port number is needed for delivery; the source port number is needed for the reply. In the Internet model, the port numbers are 16-bit integers between 0 and 65,535. The client program defines itself with a port number, chosen

randomly by the transport layer software running on the client host. This is the ephemeral port number. The server process must also define itself with a port number. This port number, however, cannot be chosen randomly. If the computer at the server site runs a server process and assigns a random number as the port number, the process at the client site that wants to access that server and use its services will not know the port number. Of course, one solution would be to send a special packet and request the port number of a specific server, but this requires more overhead. The Internet has decided to use universal port numbers for servers; these are called well-known port numbers. There are some exceptions to this rule; for example, there are clients that are assigned well-known port numbers. Every client process knows the well-known port number of the corresponding server process. For example, while the Daytime client process, discussed above, can use an ephemeral (temporary) port number 52,000 to identify itself, the Daytime server process must use the well-known (permanent) port number 13.

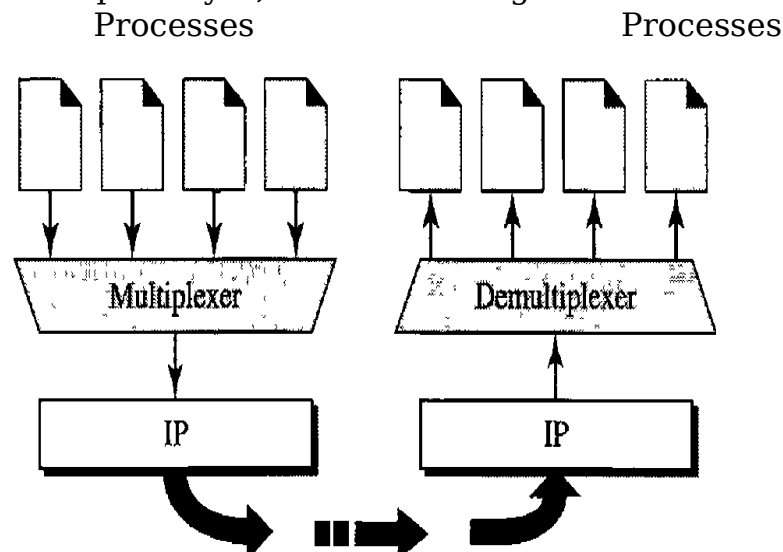
IANA Ranges

The IANA (Internet Assigned Number Authority) has divided the port numbers into three ranges: well known, registered, and dynamic (or private).

- Well-known ports. The ports ranging from 0 to 1023 are assigned and controlled by IANA. These are the well-known ports.
- Registered ports. The ports ranging from 1024 to 49,151 are not assigned or controlled by IANA. They can only be registered with IANA to prevent duplication.
- Dynamic ports. The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used by any process. These are the ephemeral ports.

Multiplexing and Demultiplexing

The addressing mechanism allows multiplexing and demultiplexing by the transport layer, as shown in Figure



Multiplexing

At the sender site, there may be several processes that need to send packets. However, there is only one transport layer protocol at any time. This is a many-to-one relationship and requires multiplexing. The protocol accepts messages from different processes, differentiated by their assigned port numbers. After adding the header, the transport layer passes the packet to the network layer.

Demultiplexing

At the receiver site, the relationship is one-to-many and requires demultiplexing. The transport layer receives datagrams from the network layer. After error checking and dropping of the header, the transport layer delivers each message to the appropriate process based on the port number.

USER DATAGRAM PROTOCOL (UDP) The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to process communication instead of host-to-host communication. Also, it performs very limited error checking. If UDP is so powerless, why would a process want to use it? With the disadvantages come some advantages. UDP is a very simple protocol using a minimum of overhead. If a process wants to send a small message and does not care much about reliability, it can use UDP. Sending a small message by using UDP takes much less interaction between the sender and receiver than using TCP or SCTP.

User Datagram

UDP packets, called user datagrams, have a fixed-size header of 8 bytes. Figure shows the format of a user datagram.

Header 8 byte	Data
---------------	------

Source port number 16 bits	Destination Port number 16 bits
Total Length 16 bits	Checksum 16 bits

The fields are as follows:

- **Source port number.** This is the port number used by the process running on the source host. It is 16 bits long, which means that the port number can range from 0 to 65,535. If the source host is the client (a client sending a request), the port number, in most cases, is an ephemeral port number requested by the process and chosen by the UDP software running on the source host. If the source host is the server (a server sending a response), the port number, in most cases, is a well-known port number.
- **Destination port number.** This is the port number used by the process running on the destination host. It is also 16 bits long. If the destination host is the server (a client sending a request), the port number, in most cases, is a well-known port number. If the destination

host is the client (a server sending a response), the port number, in most cases, is an ephemeral port number. In this case, the server copies the ephemeral port number it has received in the request packet.

- **Length.** This is a 16-bit field that defines the total length of the user datagram, header plus data. The 16 bits can define a total length of 0 to 65,535 bytes. However, the total length needs to be much less because a UDP user datagram is stored in an IP datagram with a total length of 65,535 bytes. The length field in a UDP user datagram is actually not necessary. A user datagram is encapsulated in an IP datagram. There is a field in the IP datagram that defines the total length. There is another field in the IP datagram that defines the length of the header. So if we subtract the value of the second field from the first, we can deduce the length of a UDP datagram that is encapsulated in an IP datagram. However, the designers of the UDP protocol felt that it was more efficient for the destination UDP to calculate the length of the data from the information provided in the UDP user datagram rather than ask the IP software to supply this information. We should remember that when the IP software delivers the UDP user datagram to the UDP layer, it has already dropped the IP header.
- **Checksum.** This field is used to detect errors over the entire user datagram (header plus data).

Use of UDP

The following lists some uses of the UDP protocol:

- UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control. It is not usually used for a process such as FrP that needs to send bulk data.
- UDP is suitable for a process with internal flow and error control mechanisms. For Example, the Trivial File Transfer Protocol (TFTP) process includes flow and error control. It can easily use UDP.
- UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.
- UDP is used for management processes such as SNMP.
- UDP is used for some route updating protocols such as Routing Information Protocol (RIP).

TCP

The second transport layer protocol we discuss in this chapter is called Transmission Control Protocol (TCP). TCP, like UDP, is a process-to-process (program-to-program) protocol. TCP, therefore, like UDP, uses port numbers. Unlike UDP, TCP is a connection oriented protocol; it creates a virtual connection between two TCPs to send data. In addition, TCP uses flow and error control mechanisms at the transport level. In brief, TCP is called a *connection-oriented, reliable* transport protocol. It adds connection-oriented and reliability features to the services of IP.

TCP Segment

Header	Data
--------	------

Source Port Address 16 bits							Destination Port Address 16 bits						
Sequence number 32 bits													
Acknowledgement number 32 bits													
HLEN 4 bits		Reserved 6 bits		U R G	A C K	P S H	S Y N	F I N	Window Size 16 bits				
Checksum 16 bits							Urgent Pointer 16 bits						
Option and Padding													

The segment consists of a 20- to 60-byte header, followed by data from the application program. The header is 20 bytes if there are no options and up to 60 bytes if it contains options. We will discuss some of the header fields in this section.

- ❖ **Source port address.** This is a 16-bit field that defines the port number of the application program in the host that is sending the segment. This serves the same purpose as the source port address in the UDP header.
- ❖ **Destination port address.** This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment. This serves the same purpose as the destination port address in the UDP header.
- ❖ **Sequence number.** This 32-bit field defines the number assigned to the first byte of data contained in this segment. As we said before, TCP is a stream transport protocol. To ensure connectivity, each byte to be transmitted is numbered. The sequence number tells the destination which byte in this sequence comprises the first byte in the segment. During connection establishment, each party uses a random number generator to create an initial sequence number (ISN), which is usually different in each direction.
- ❖ **Acknowledgment number.** This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party. If the receiver of the segment has successfully received byte number x from the other party, it defines $x + 1$ as the acknowledgment number. Acknowledgment and data can be piggybacked together.
- ❖ **Header length.** This 4-bit field indicates the number of 4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field can be between 5 ($5 \times 4 = 20$) and 15 ($15 \times 4 = 60$).
- ❖ **Reserved.** This is a 6-bit field reserved for future use.
- ❖ **Control.** This field defines 6 different control bits or flags as shown in Figure One or more of these bits can be set at a time these bits enable flow control, connection establishment and termination, connection

abortion, and the mode of data transfer in TCP. A brief description of each bit is shown below

Description of flags in the control field

Flag Description

URG The value of the urgent pointer field is valid.

ACK The value of the acknowledgment field is valid.

PSH Push the data.

RST Reset the connection.

SYN Synchronize sequence numbers during connection.

FIN Terminate the connection

- ❖ **Window size.** This field defines the size of the window, in bytes, that the other party must maintain. Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes. This value is normally referred to as the receiving window (rwnd) and is determined by the receiver. The sender must obey the dictation of the receiver in this case.
- ❖ **Checksum.** This 16-bit field contains the checksum. The calculation of the checksum for TCP follows the same procedure as the one described for UDP. However, the inclusion of the checksum in the UDP datagram is optional, whereas the inclusion of the checksum for TCP is mandatory. The same pseudoheader, serving the same purpose, is added to the segment. For the TCP pseudoheader, the value for the protocol field is 6.
- ❖ **Urgent pointer.** This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data. It defines the number that must be added to the sequence number to obtain the number of the last urgent byte in the data section of the segment. This will be discussed later in this chapter.
- ❖ **Options.** There can be up to 40 bytes of optional information in the TCP header.

A TCP Connection

TCP is connection-oriented. A connection-oriented transport protocol establishes a virtual path between the source and destination. All the segments belonging to a message are then sent over this virtual path. Using a single virtual pathway for the entire message facilitates the acknowledgment process as well as retransmission of damaged or lost frames. You may wonder how TCP, which uses the services of IP, a connectionless protocol, can be connection-oriented. The point is that a TCP connection is virtual, not physical. TCP operates at a higher level. TCP uses the services of IP to deliver individual segments to the receiver, but it controls the connection itself. If a segment is lost or corrupted, it is retransmitted. Unlike TCP, IP is unaware of this retransmission. If a segment arrives out of order, TCP holds it until the missing segments arrive; IP is unaware of this reordering. In TCP, connection-oriented transmission requires three phases: connection establishment, data transfer, and connection termination.

Connection Establishment

TCP transmits data in full-duplex mode. When two TCPs in two machines are connected,

they are able to send segments to each other simultaneously. This implies that

each party must initialize communication and get approval from the other party before

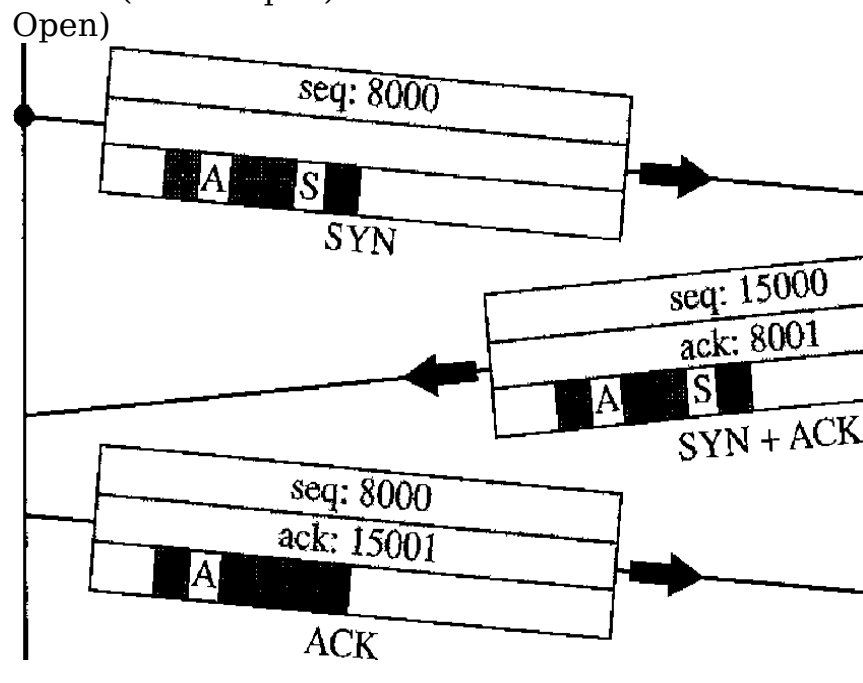
any data are transferred.

Three-Way Handshaking The connection establishment in TCP is called three way handshaking. In our example, an application program, called the client, wants to make a connection with another application program, called the server, using TCP as the transport layer protocol. The process starts with the server. The server program tells its TCP that it is ready to accept a connection. This is called a request for a *passive open*. Although the server TCP is ready to accept any connection from any machine in the world, it cannot make the connection itself.

The client program issues a request for an *active open*. A client that wishes to connect to an open server tells its TCP that it needs to be connected to that particular server. TCP can now start the three-way handshaking process as shown in Figure. To show the process, we use two time lines: one at each site. Each segment has values for all its header fields and perhaps for some of its option fields, too. However, we show only the few fields necessary to understand each phase. We show the sequence number

Client (Active open)

Server (Passive



the acknowledgment number, the control flags (only those that are set), and the window size, if not empty. The three steps in this phase are as follows.

1. The client sends the first segment, a SYN segment, in which only the SYN flag is set. This segment is for synchronization of sequence numbers. It consumes one sequence number. When the data transfer starts, the sequence number is incremented by 1. We can say that the SYN segment carries no real data, but we can think of it as containing 1 imaginary byte.
2. The server sends the second segment, a SYN +ACK segment, with 2 flag bits set: SYN and ACK. This segment has a dual purpose. It is a SYN segment for communication in the other direction and serves as the acknowledgment for the SYN segment. It consumes one sequence number.
3. The client sends the third segment. This is just an ACK segment. It acknowledges the receipt of the second segment with the ACK flag and acknowledgment number field. Note that the sequence number in this segment is the same as the one in the SYN segment; the ACK segment does not consume any sequence numbers.

Simultaneous Open A rare situation, called a simultaneous open, may occur when both processes issue an active open. In this case, both TCPs transmit a SYN + ACK segment to each other, and one single connection is established between them.

SYN Flooding Attack The connection establishment procedure in TCP is susceptible to a serious security problem called the SYN flooding attack. This happens when a malicious attacker sends a large number of SYN segments to a server, pretending that each of them is coming from a different client by faking the source IP addresses in the datagrams. The server, assuming that the clients are issuing an active open, allocates the necessary resources, such as creating communication tables and setting timers. The

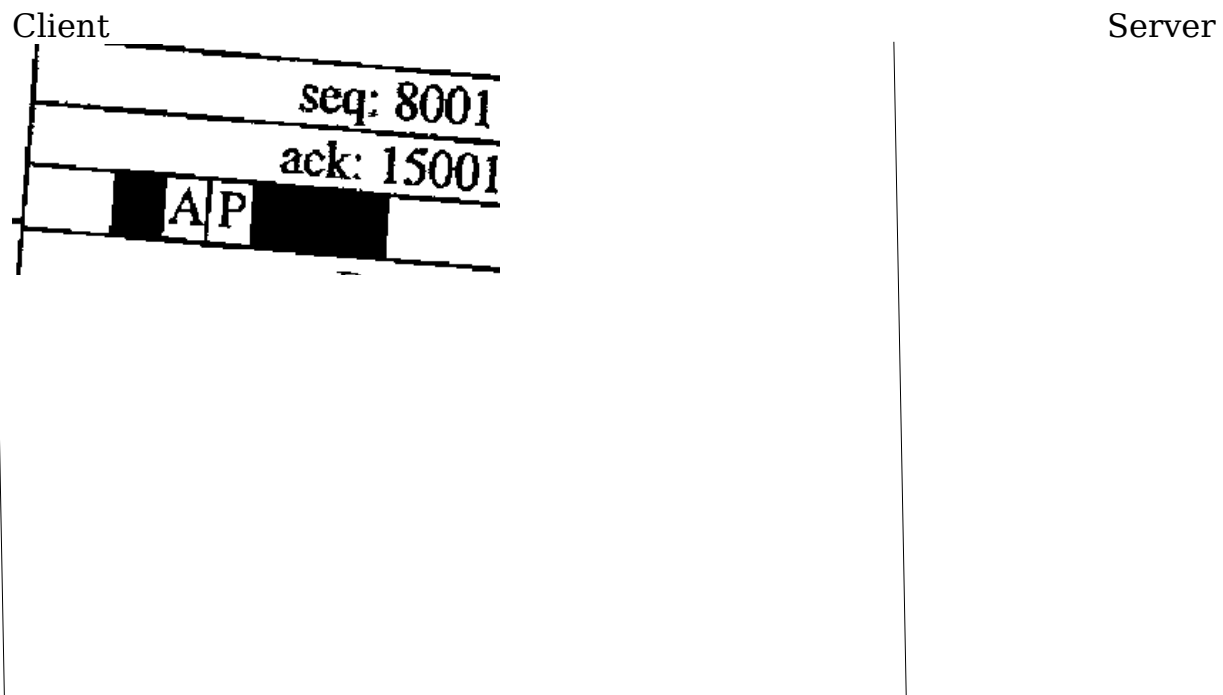
TCP server then sends the SYN +ACK segments to the fake clients, which are lost. During this time, however, a lot of resources are occupied without being used. If, during this short time, the number of SYN segments is large, the server eventually runs out of resources and may crash. This SYN flooding attack belongs to a type of security attack known as a denial-of-service attack, in which an attacker monopolizes a system with so many service requests that the system collapses and denies service to every request.

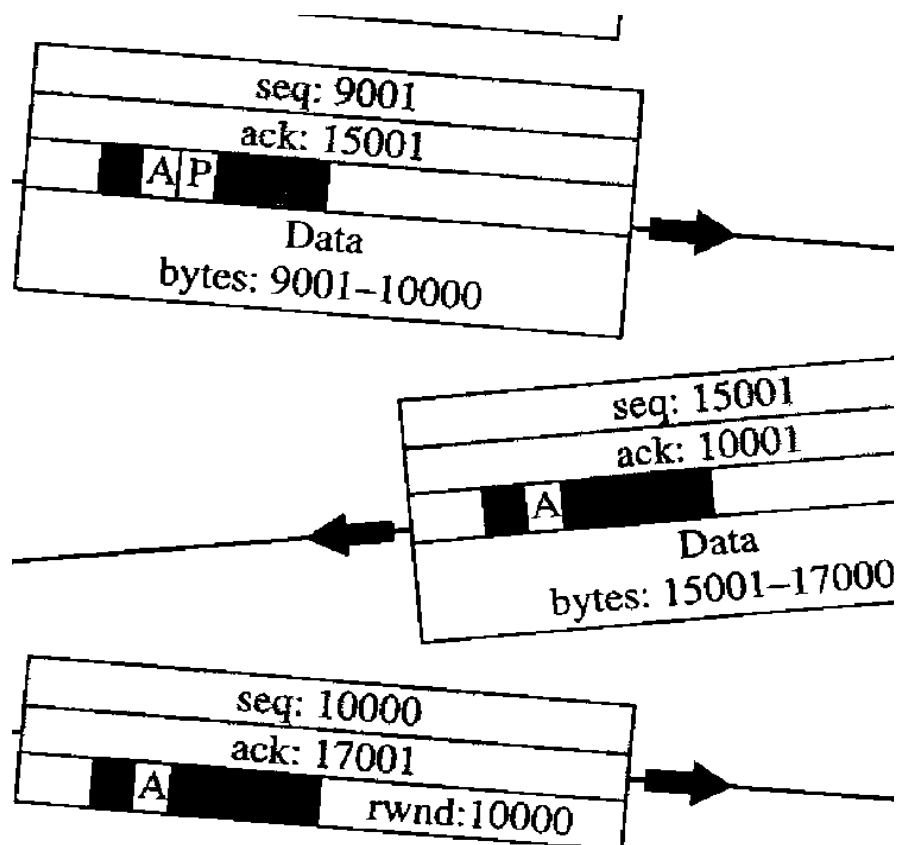
Data Transfer

After connection is established, bidirectional data transfer can take place. The client and server can both send data and acknowledgments. We will study the rules of acknowledgment later in the chapter; for the moment, it is enough to know that data traveling in the same direction as an acknowledgment are carried on the same segment. The acknowledgment is piggybacked with the data. Figure shows an example. In this example, after

connection is established (not shown in the figure), the client sends 2000 bytes of data in two segments. The server then sends 2000 bytes in one segment. The client sends one more segment. The first three segments carry both data and acknowledgment, but the last segment carries only an acknowledgment because there are no more data to be sent. Note the values of the sequence and acknowledgment numbers. The data segments sent by the client have the PSH (push) flag set so that the server TCP knows to deliver data to the server process as soon as they are received. We discuss the use of this flag in greater detail later. The segment from the server, on the other hand, does not set the push flag. Most TCP implementations have the option to set or not set this flag. Pushing Data We saw that the sending TCP uses a buffer to store the stream of data coming from the sending application program. The sending TCP can select the segment size. The receiving TCP also buffers the data when they arrive and delivers them to the application program when the application program is ready or when it is convenient for the receiving TCP. This type of flexibility increases the efficiency of TCP. However, on occasion the application program has no need for this flexibility_ For example; consider an application program that communicates interactively with another application program on the other end. The application program on one site wants to send a keystroke to the application at the other site and receive an immediate response. Delayed transmission and delayed delivery of data may not be acceptable by the application program.

TCP can handle such a situation. The application program at the sending site can request a *push* operation. This means that the sending TCP must not wait for the window to be filled. It must create a segment and send it immediately. The sending TCP must also set the push bit (PSH) to let the receiving TCP know that the segment includes data that must be delivered to the receiving application program as soon as possible and not to wait for more data to come. Although the push operation can be requested by the application program, most current implementations ignore such requests. TCP can choose whether or not to use this feature.





Urgent Data TCP is a stream-oriented protocol. This means that the data are presented from the application program to TCP as a stream of bytes. Each byte of data has a position in the stream. However, on occasion an application program needs to send *urgent* bytes. This means that the sending application program wants a piece of data to be read out of order by the receiving application program. As an example, suppose that the sending application program is sending data to be processed by the receiving application program. When the result of processing comes back, the sending application program finds that everything is wrong. It wants to abort the process, but it has already sent a huge amount of data. If issues of an abort command (control +C), these two characters will be stored at the end of the receiving TCP buffer. It will be delivered to the receiving application program after all the data have been processed. The solution is to send a segment with the URG bit set. The sending application program tells the sending TCP that the piece of data is urgent. The sending TCP creates a segment and inserts the urgent data at the beginning of the segment. The rest of the segment can contain normal data from the buffer. The urgent pointer field in the header defines the end of the urgent data and the start of normal data. When the receiving TCP receives a segment with the URG bit set, it extracts the urgent data from the segment, using

the value of the urgent pointer, and delivers them, out of order, to the receiving application program.

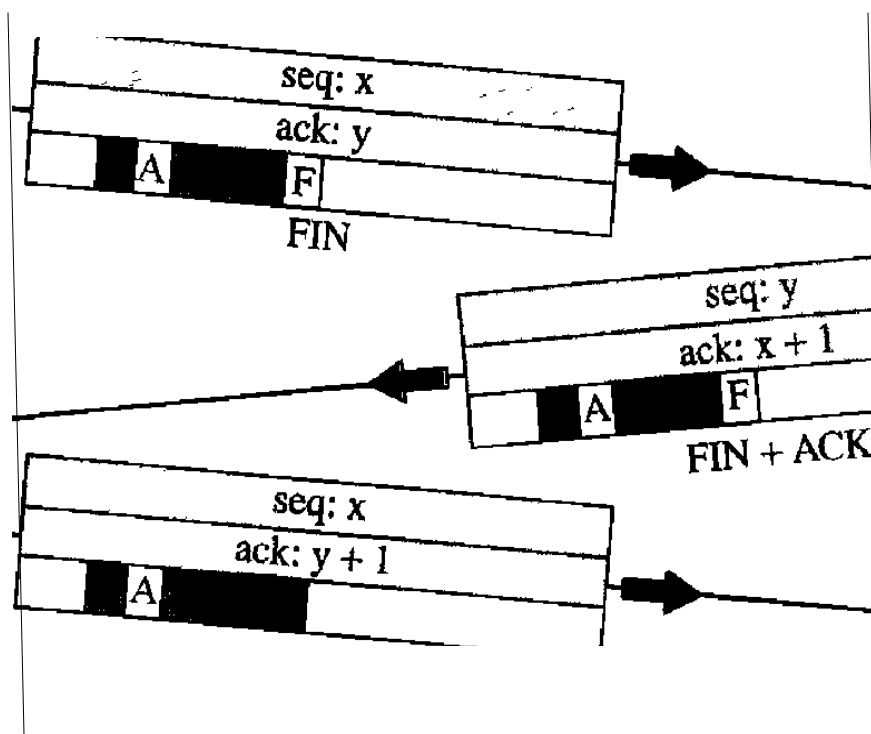
Connection Termination

Any of the two parties involved in exchanging data (client or server) can close the connection, although it is usually initiated by the client. Most implementations today allow two options for connection termination: three-way handshaking and four-way handshaking with a half-close option. Three-Way Handshaking Most implementations today allow *three-way handshaking* for connection termination as shown in Figure.

1. In a normal situation, the client TCP, after receiving a close command from the client process, sends the first segment, a FIN segment in which the FIN flag is set. Note that a FIN segment can include the last chunk of data sent by the client, or it

Client (Active Close)
Close)

Server (Passive
Close)



can be just a control segment as shown in Figure. If it is only a control segment, it consumes only one sequence number

2. The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN +ACK segment, to confirm the receipt of the FIN segment from the client and at the same time to announce the closing of the connection in the other direction. This segment can also contain the last chunk of data from

the server. If it does not carry data, it consumes only one sequence number.

3. The client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server. This segment contains the acknowledgment number, which is 1 plus the sequence number received in the FIN segment from the server. This segment cannot carry data and consumes no sequence numbers.

CONGESTION

An important issue in a packet-switched network is **congestion**. Congestion in a network may occur if the **load** on the network-the number of packets sent to the network-is greater than the *capacity* of the network-the number of packets a network can handle. **Congestion control** refers to the mechanisms and techniques to control the congestion and keep the load below the capacity.

CONGESTION CONTROL

Congestion control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened. In general, we can divide congestion control mechanisms into two broad categories: open-loop congestion control (prevention) and closed-loop congestion control (removal).

Open loop Congestion Control: In open-loop congestion control, policies are applied to prevent congestion before it happens. In these mechanisms, congestion control is handled by either the source or the destination. We give a brief list of policies that can prevent congestion.

- Retransmission policy
- Window policy
- Acknowledgment policy
- Discarding policy
- Admission policy

Closed Loop Congestion Control Closed-loop congestion control mechanisms try to alleviate congestion after it happens. Several mechanisms have been used by different protocols. We describe a few of them here.

- Back pressure
- Choke packet
- Implicit signaling
- Explicit signaling