TRANSPORT LAYER

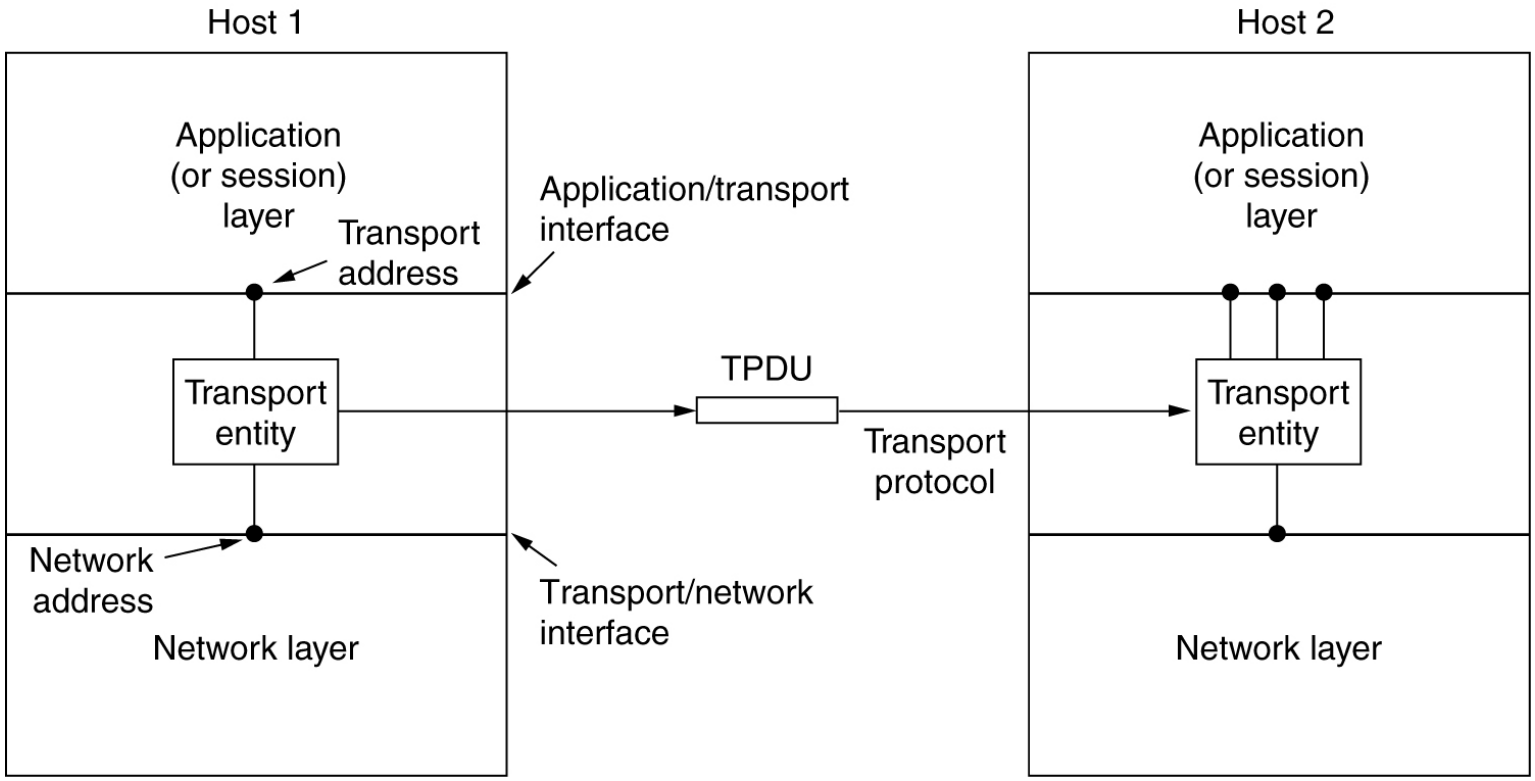
The network layer provides end-to-end packet delivery using datagrams or virtual circuits.

The transport layer builds on the network layer to provide **data transport from a process on a source machine to a process on a destination machine** with a desired level of reliability that is independent of the physical networks currently in use.

Services Provided to the Upper Layers

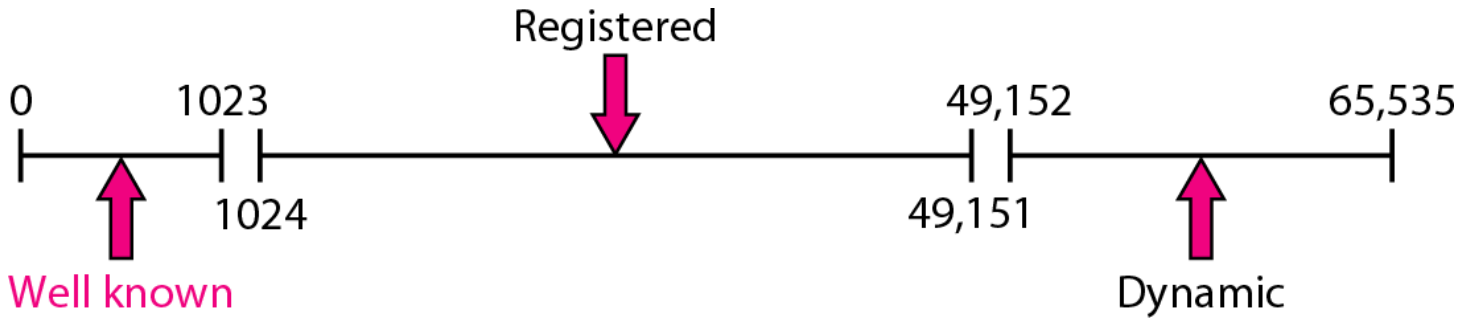
The ultimate goal of the transport layer is to provide efficient, reliable, and cost-effective data transmission service to its users, normally processes in the application layer.

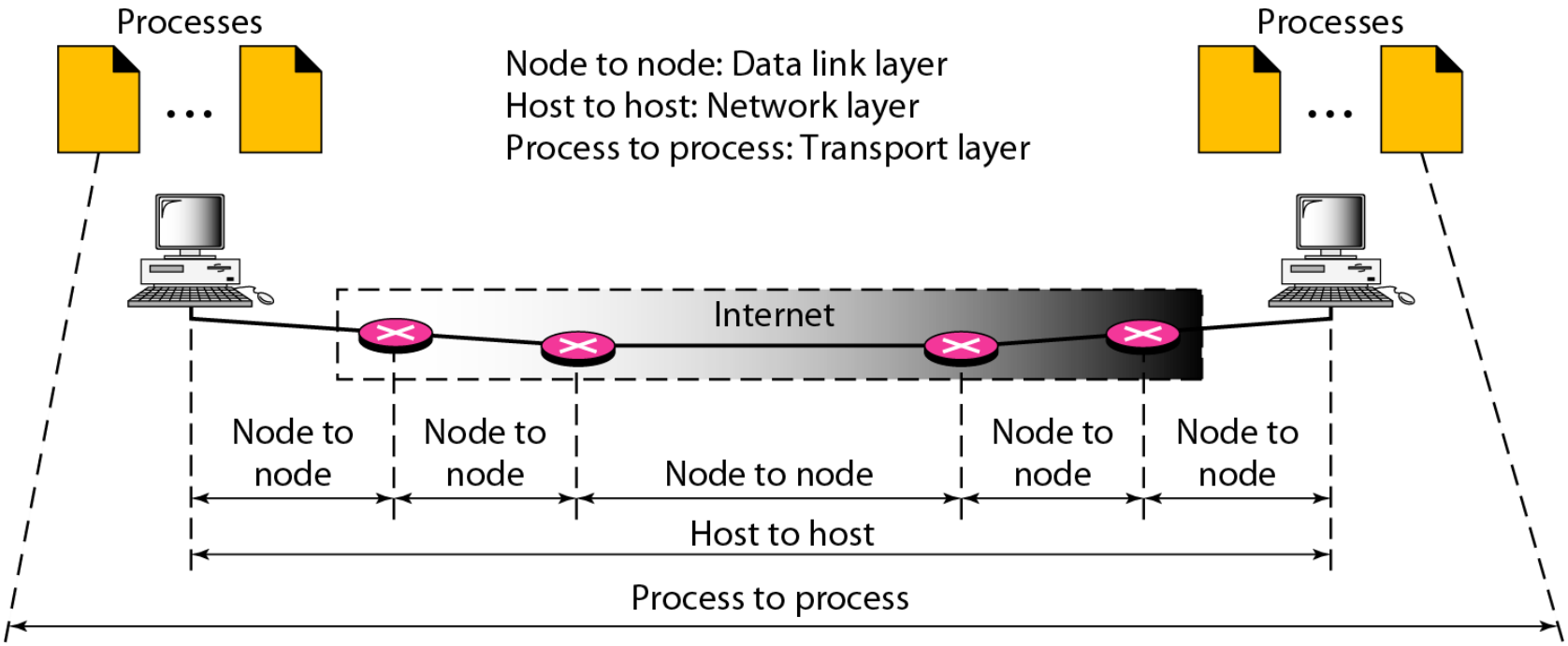
To achieve this, the transport layer makes use of the services provided by the network layer. The software and/or hardware within the transport layer that does the work is called the **transport entity**.



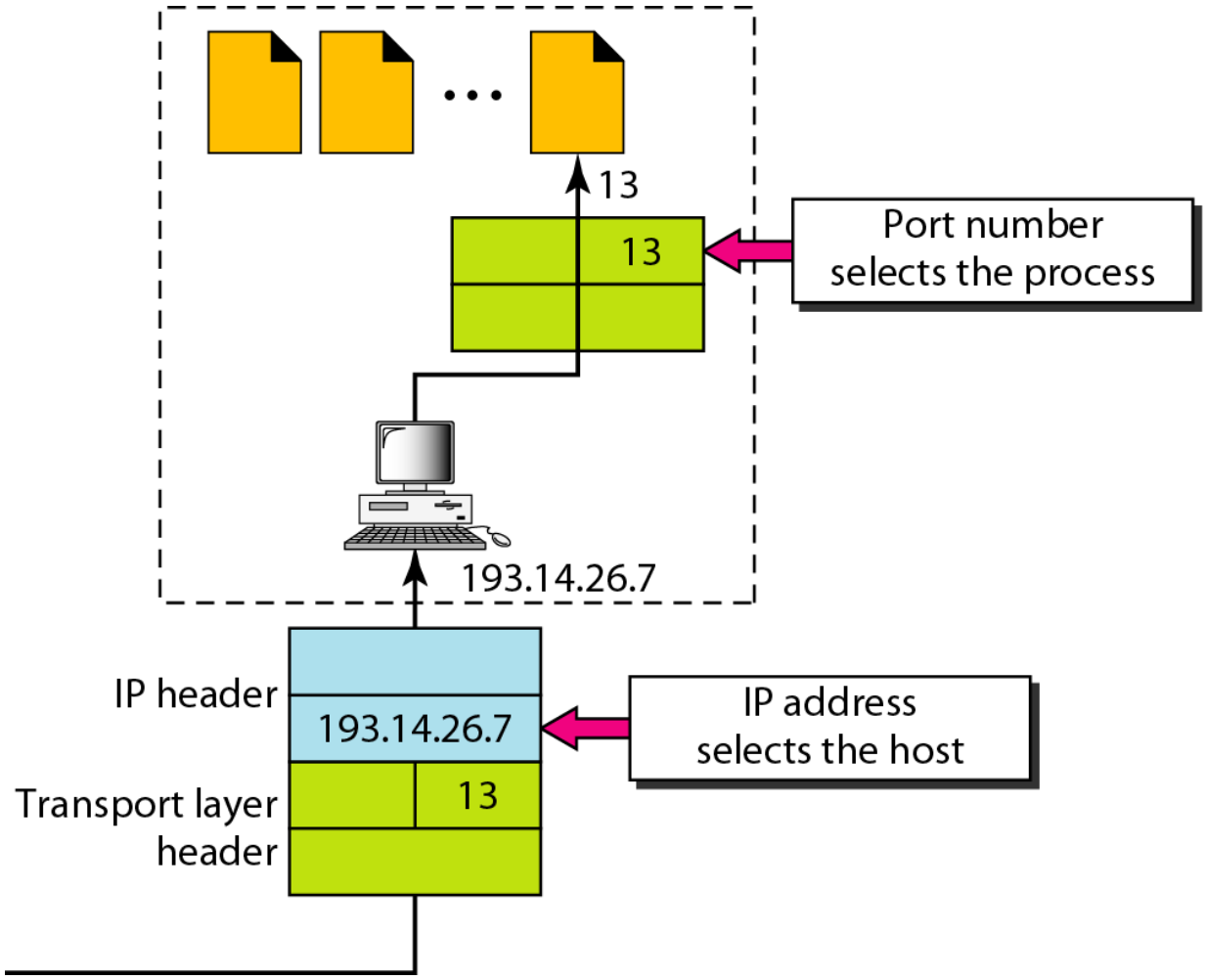
The network, transport, and application layers.

The connection-oriented transport service. connections have three phases: establishment, data transfer, and release.
Addressing and flow control
The connectionless transport service .




Processes
Processes
Node to node: Data link layer
Host to host: Network layer
Process to process: Transport layer
Internet
Node to node
Node to node
Node to node
Node to node
Node to node
Host to host
Process to process

IP addresses versus port numbers



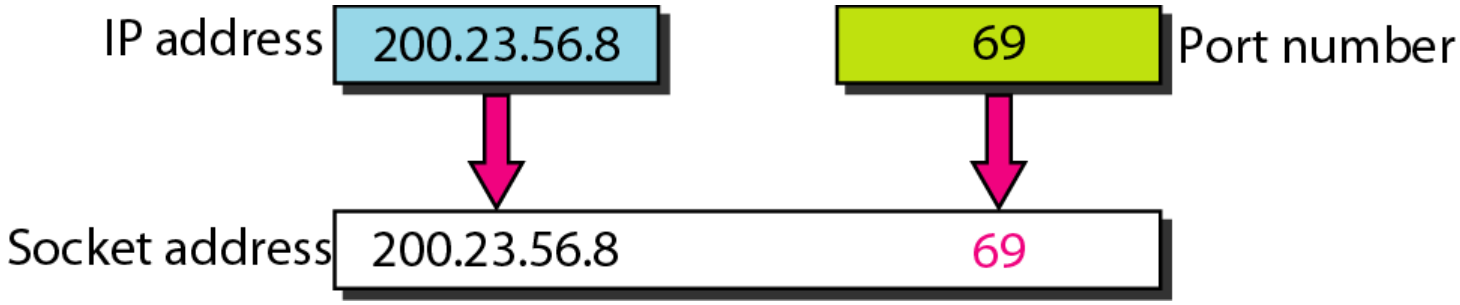

IP address
200.23.56.8
69
Port number
Socket address
200.23.56.8
69

Figure 23.6 *Multiplexing and demultiplexing*

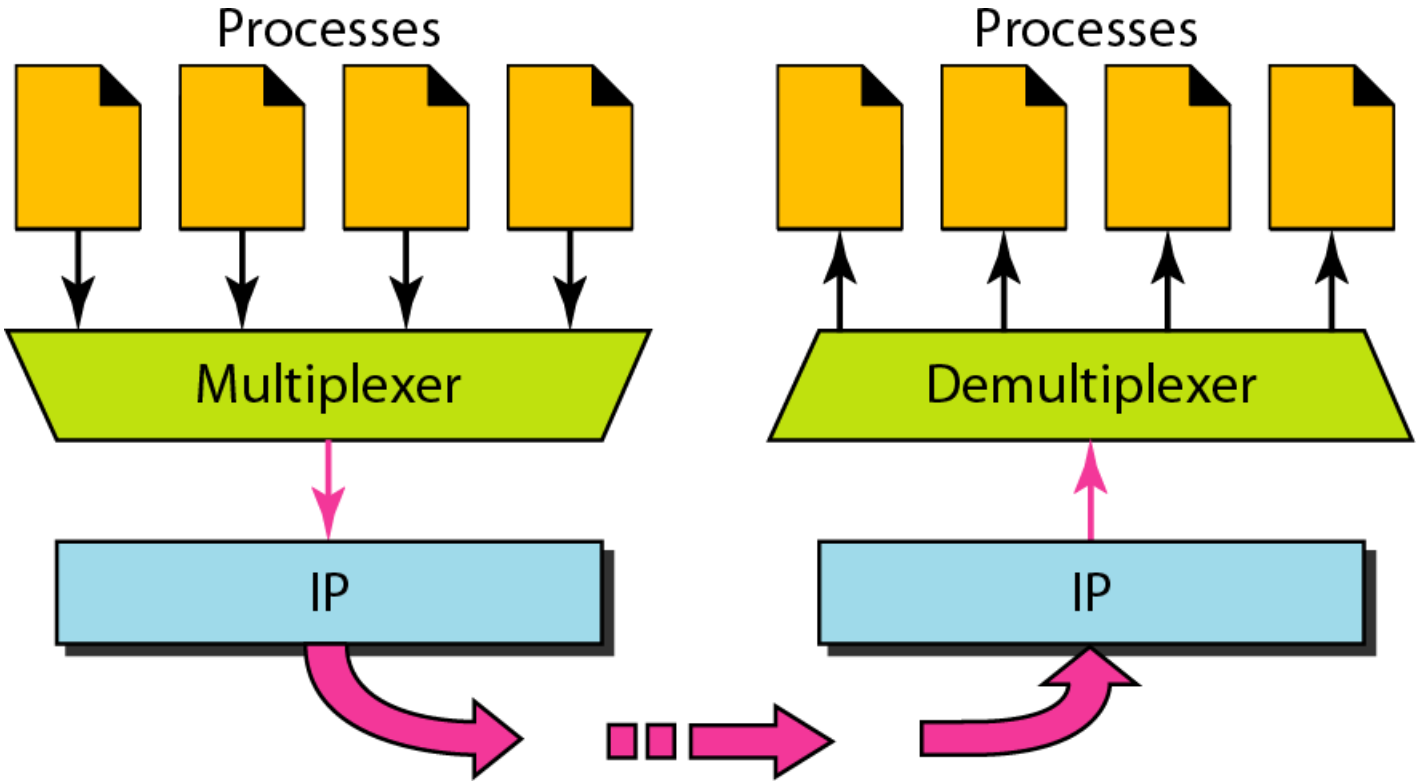


Figure 23.7 *Error control*

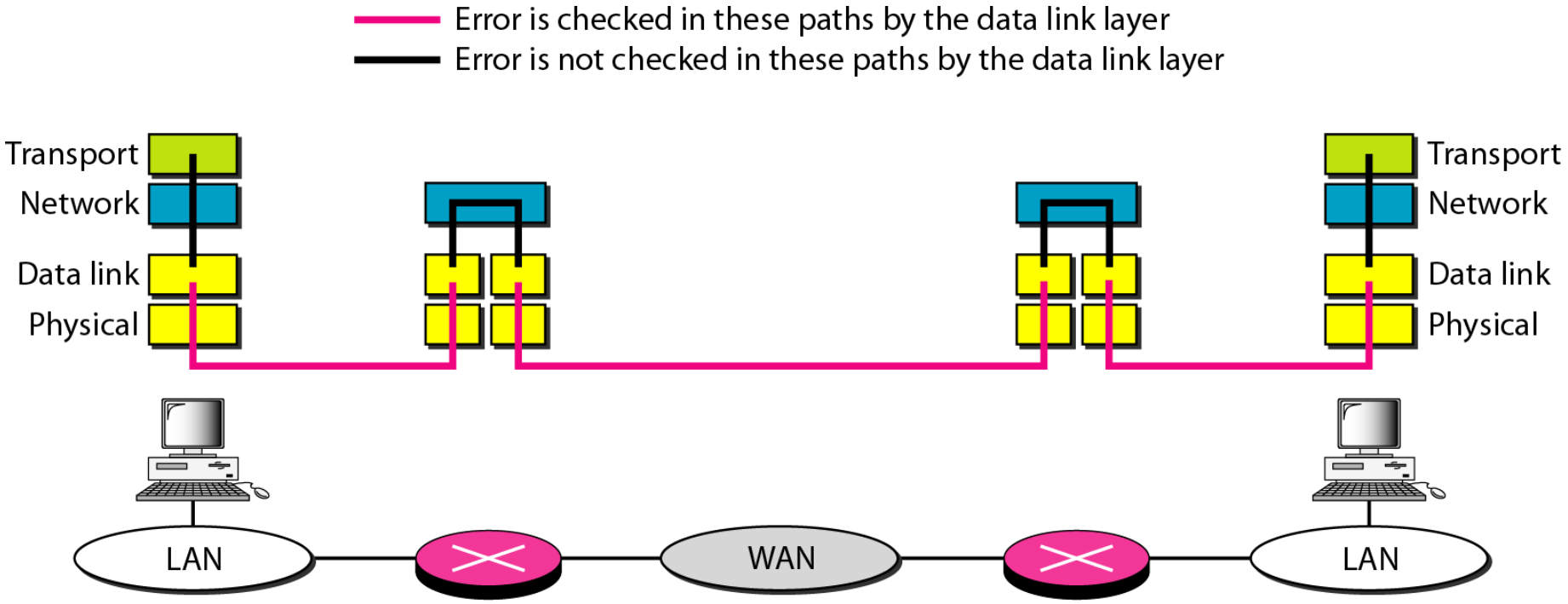
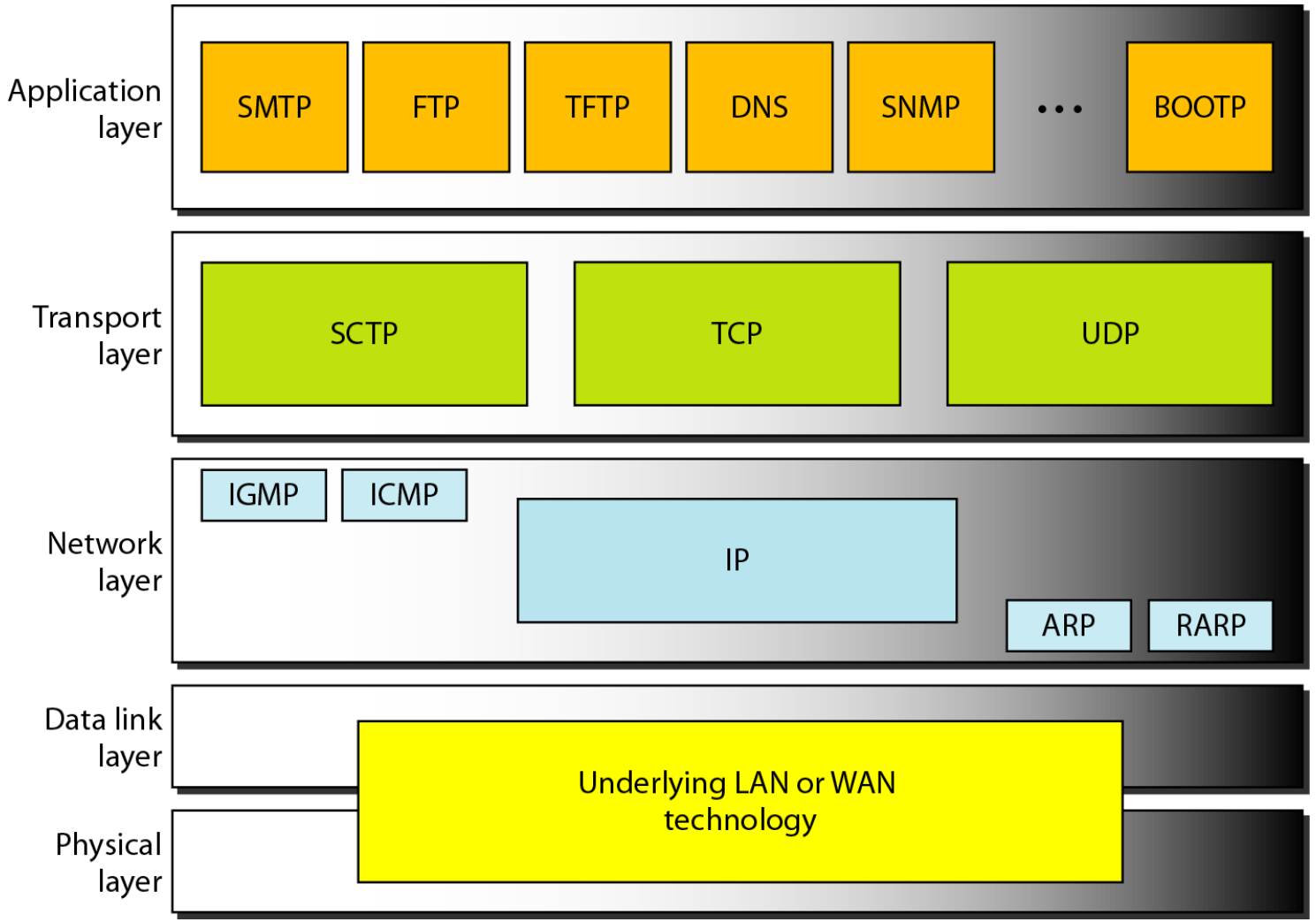


Figure 23.8 *Position of UDP, TCP, and SCTP in TCP/IP suite*



Transport Service Primitives

Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

The primitives for a simple transport service.

To start with, the server executes a LISTEN primitive, typically by calling a library procedure that makes a system call that blocks the server until a client turns up.

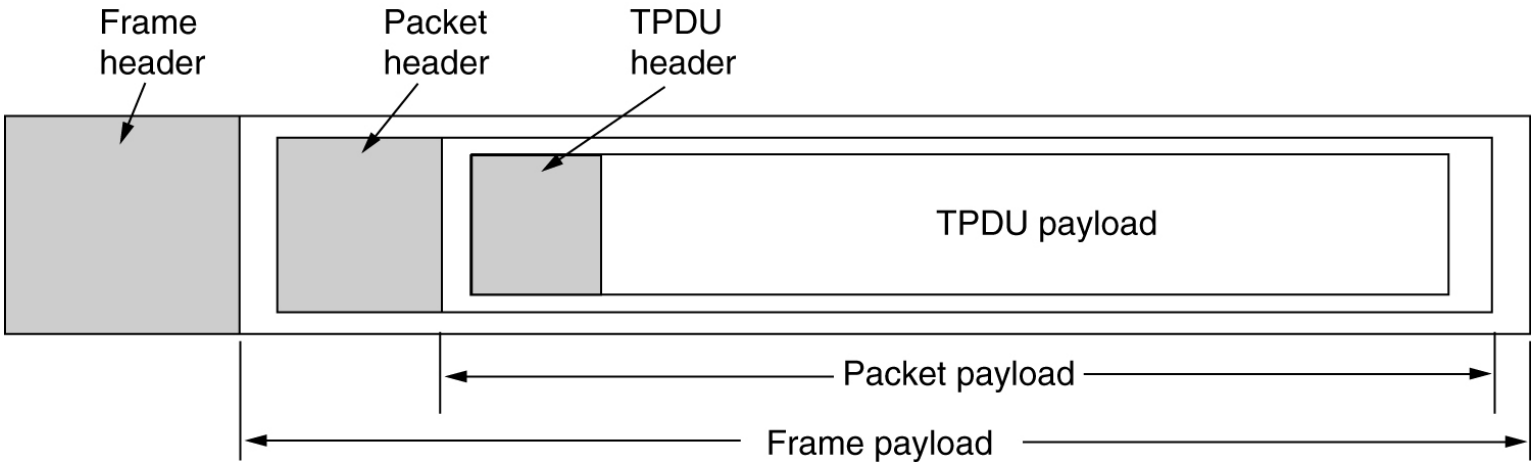
When a client wants to talk to the server, it executes a CONNECT primitive. The transport entity carries out this primitive by blocking thecaller and sending a packet to the server. The client's CONNECT call causes a CONNECTION REQUEST segment to be sent to the server. When it arrives, the transport entity checks to see that the server is blocked on a LISTEN (i.e., is interested in handling requests). If so, it then unblocks the server and sends a CONNECTION ACCEPTED segment back to the client. When this segment arrives, the client is unblocked and the connection is established.

Data can now be exchanged using the SEND and RECEIVE primitives. In the simplest form, either party can do a (blocking) RECEIVE to wait for the other party to do a SEND. When the segment arrives, the receiver is unblocked. It can then process the segment and send a reply. As long as both sides can keep track of whose turn it is to send, this scheme works fine.

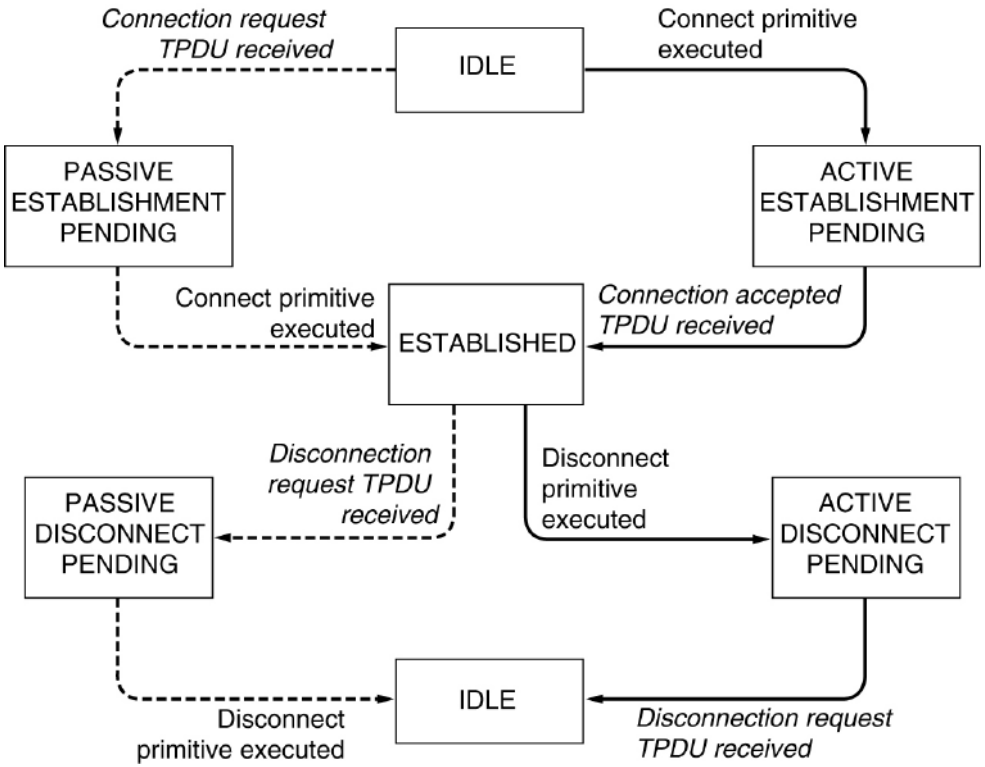
When a connection is no longer needed, it must be released to free up table space within the two transport entities. Disconnection has two variants: asymmetric and symmetric.

In the asymmetric variant, either transport user can issue a DISCONNECT primitive, which results in a DISCONNECT segment being sent to the remote transport entity. Upon its arrival, the connection is released.

In the symmetric variant, each direction is closed separately, independently of the other one. When one side does a DISCONNECT, that means it has no more data to send but it is still willing to accept data from its partner. In this model, a connection is released when both sides have done a DISCONNECT



Transport Service Primitives

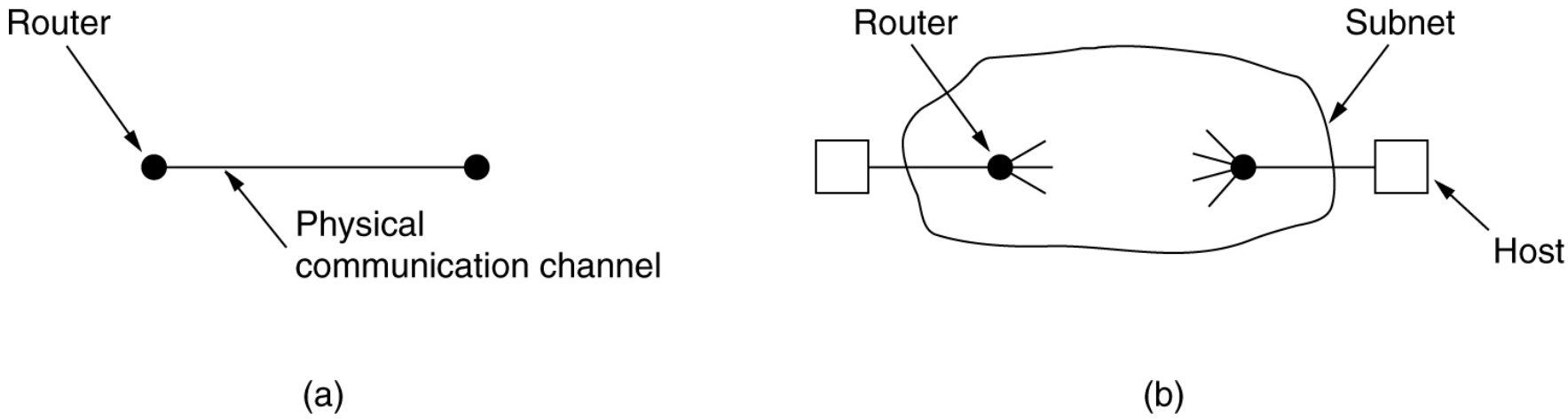


A state diagram for a simple connection management scheme. Transitions labeled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The dashed lines show the server's state sequence.

Elements of TransportProtocols

- Addressing
- Connection Establishment
- Connection Release
- Flow Control and Buffering
- Multiplexing
- Crash Recovery

Transport Protocol



(a) Environment of the data link layer.
(b) Environment of the transport layer.

1 Over point-to-point links such as wires or optical fiber, it is usually not necessary for a router to specify which router it wants to talk to—each outgoing line leads directly to a particular router. In the transport layer, explicit addressing of destinations is required.

2 The process of establishing a connection over the wire of Fig(a) is simple: the other end is always there (unless it has crashed, in which case it is not there). Either way, there is not much to do. Even on wireless links the process is not much different. Just sending a message is sufficient to have it reach all other destinations. If the message is not acknowledged due to an error, it can be resent. In the transport layer, initial connection establishment is complicated, as we will see.

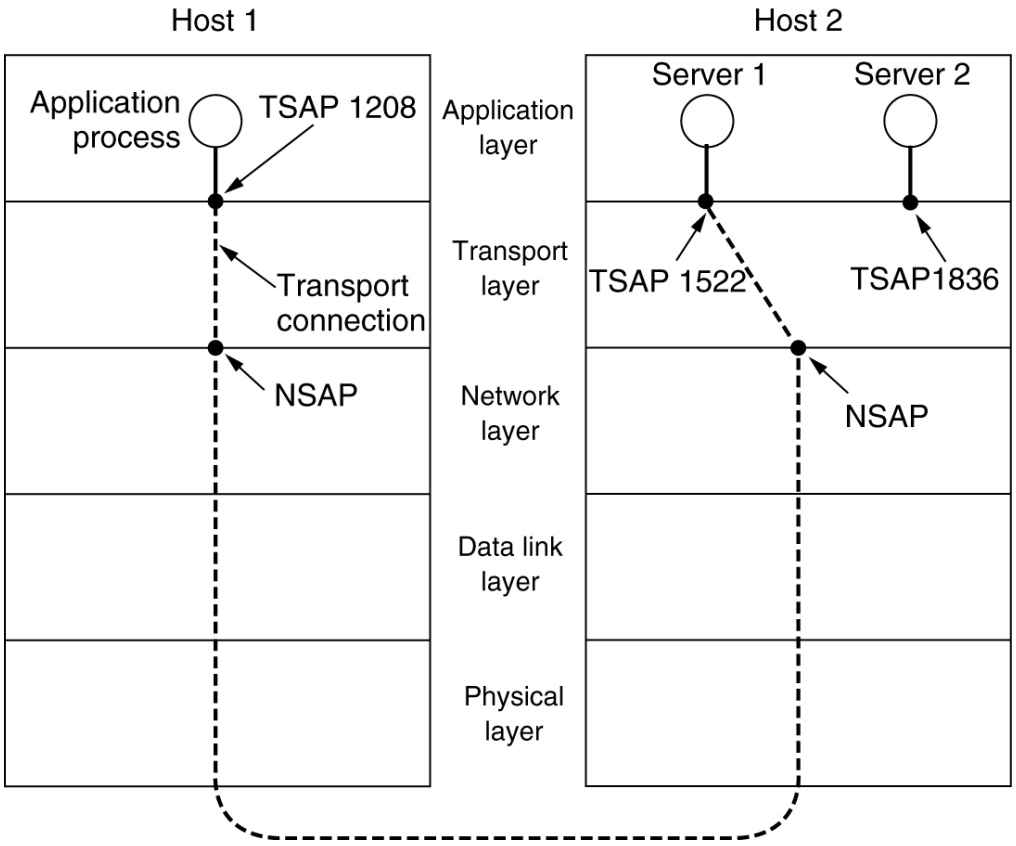
3 Another (exceedingly annoying) difference between the data link layer and the transport layer is the potential existence of storage capacity in the network. The consequences of the network's ability to delay and duplicate packets can sometimes be disastrous and can require the use of special protocols to correctly transport information.

4. Buffering and flow control are needed in both layers, but the presence in the transport layer of a large and varying number of connections with bandwidth that fluctuates as the connections compete with each other may require a different approach than we

Addressing

When an application (e.g., a user) process wishes to set up a connection to a remote application process, it must specify which one to connect to. (Connectionless transport has the same problem: to whom should each message be sent?) The method normally used is to define transport addresses to which processes can listen for connection requests. In the Internet, these endpoints are called **ports.** We will use the generic term **TSAP (Transport Service Access Point)** to mean a specific endpoint in the transport layer. The analogous endpoints in the network layer (i.e., network layer addresses) are not-surprisingly called **NSAPs** (**Network Service Access Points). IP addresses are examples of NSAPs.**

Addressing



TSAPs, NSAPs and transport connections.

A possible scenario for a transport connection is as follows:
1. A mail server process attaches itself to TSAP 1522 on host 2 to wait for an incoming call. A call such as our LISTEN might be used, for example.
2. An application process on host 1 wants to send an email message, so it attaches itself to TSAP 1208 and issues a CONNECT request. The request specifies TSAP 1208 on host 1 as the source and TSAP 1522 on host 2 as the destination. This action ultimately results in a transportconnection being established between the application process and the server.
3. The application process sends over the mail message.
4. The mail server responds to say that it will deliver the message.
5. The transport connection is released.

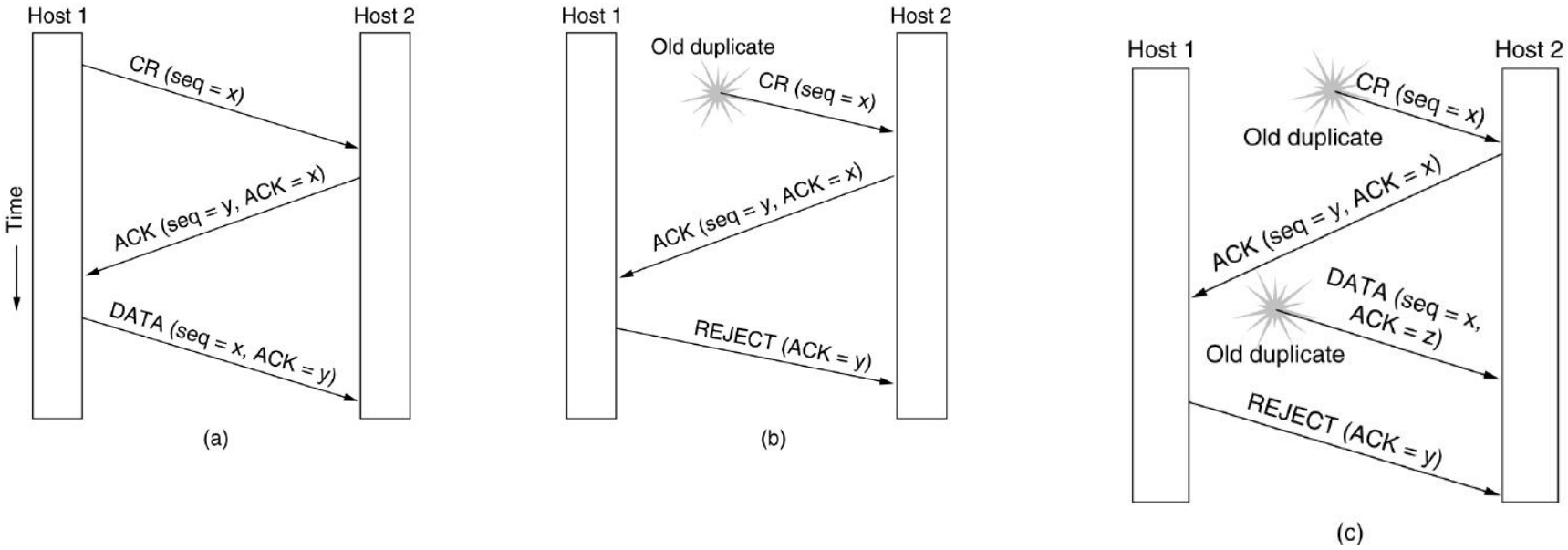
special process called a **portmapper**

CONNECTION ESTABLISHMENT

Establishing a connection sounds easy, but it is actually surprisingly tricky. At first glance, it would seem sufficient for one transport entity to just send a CONNECTION REQUEST segment to the destination and wait for a CONNECTION ACCEPTED reply. The problem occurs when the network can lose, delay, corrupt, and duplicate packets. This behavior causes serious complications

To solve this specific problem,(DELAYED DUPLICATES) Tomlinson (1975) introduced the **three-way handshake.** This establishment protocol involves one peer checking with the other that the connection request is indeed current. The normal setup procedure when host 1 initiates is shownin Fig. (a). Host 1 chooses a sequence number, *x, and sends a CONNECTION REQUEST segment containing it to host 2. Host 2* replies with an ACK segment acknowledging *x and announcing its own initial sequence* number, *y. Finally, host 1 acknowledges host 2's choice of an initial sequence* number in the first data segment that it sends.

Connection Establishment



Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST.

(a) Normal operation,

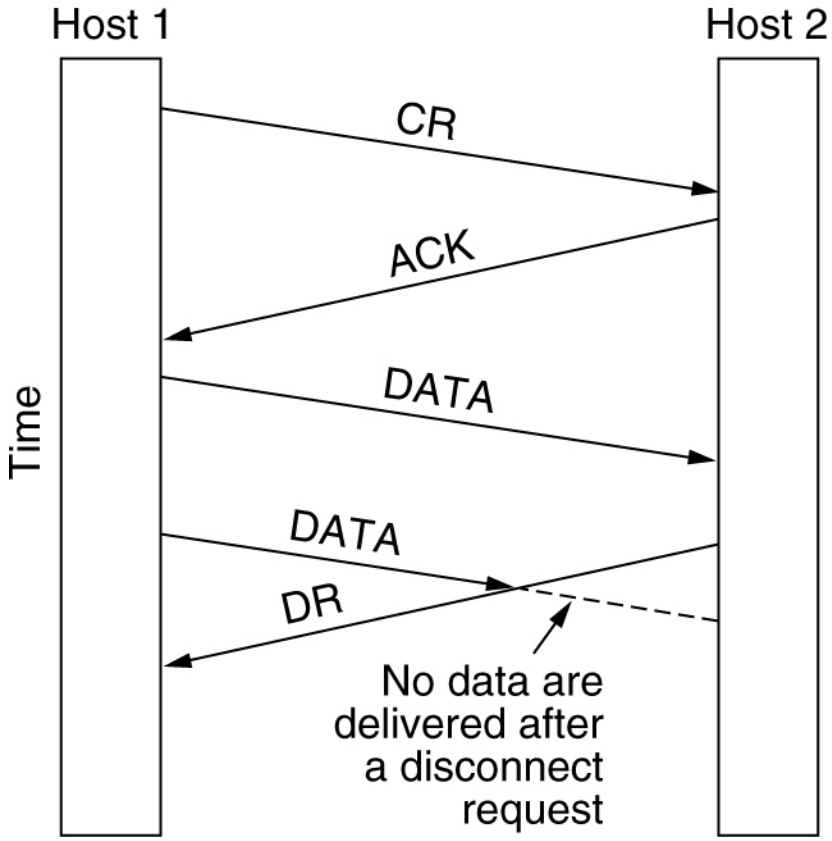
(b) Old CONNECTION REQUEST appearing out of nowhere.

(c) Duplicate CONNECTION REQUEST and duplicate ACK.

In Fig.(b), the first segment is a delayed duplicate CONNECTION REQUEST from an old connection. This segment arrives at host 2 without host 1's knowledge. Host 2 reacts to this segment by sending host 1 an ACK segment, in effect asking for verification that host 1 was indeed trying to set up a new connection. When host 1 rejects host 2's attempt toestablish a connection, host 2 realizes that it was tricked by a delayed duplicate and abandons the connection. In this way, a delayed duplicate does no damage

The worst case is when both a delayed CONNECTION REQUEST and an ACK are floating around in the subnet. This case is shown in Fig. (c). As in the previous example, host 2 gets a delayed CONNECTION REQUEST and replies to it. At this point, it is crucial to realize that host 2 has proposed using *y as the initial* sequence number for host 2 to host 1 traffic, knowing full well that no segments containing sequence number *y or acknowledgements to y are still in existence.* When the second delayed segment arrives at host 2, the fact that *z has been* acknowledged rather than *y tells host 2 that this, too, is an old duplicate. The important* thing to realize here is that there is no combination of old segments that can cause the protocol to fail and have a connection set up by accident when no one wants it.

Connection Release



Abrupt disconnection with loss of data.

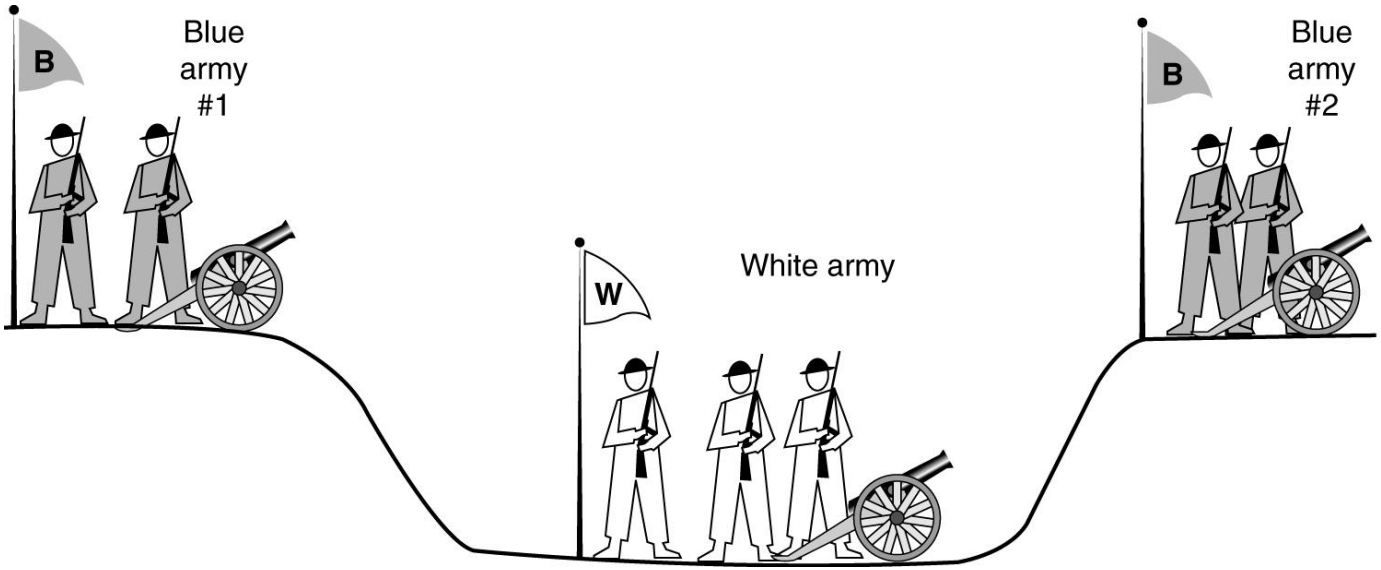
there are two styles of terminating a connection: asymmetric release and symmetric release Asymmetric release is the way the telephone system works: when one party hangs up, the connection is broken. Symmetric release treats the connection as two separate unidirectional connections and requires each one to be released separately

Asymmetric release is abrupt and may result in data loss. Consider the scenario of Fig. After the connection is established, host 1 sends a segment that arrives properly at host 2. Then host 1 sends another segment. Unfortunately, host 2 issues a DISCONNECT before the second segment arrives. The result is that the connection is released and data are lost.

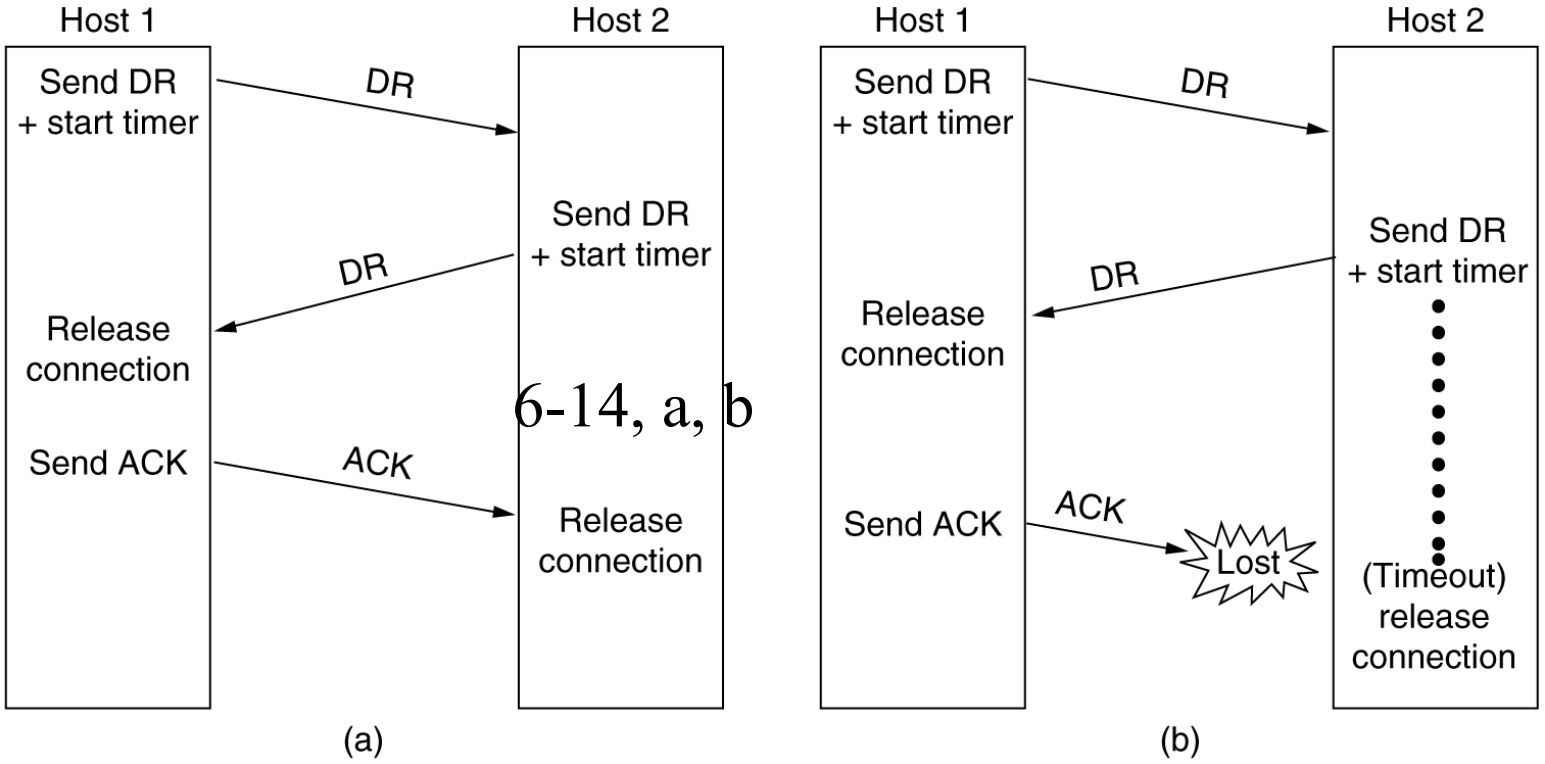
Clearly, a more sophisticated release protocol is needed to avoid data loss. One way is to use symmetric release, in which each direction is released independently of the other one. Here, a host can continue to receive data even after it has sent a DISCONNECT segment.
Symmetric release does the job when each process has a fixed amount of data to send and clearly knows when it has sent it. One can envision a protocol in which host 1 says ‘‘I am done. Are you done too?’’ If host 2 responds: ‘‘I am done too. Goodbye, the connection can be safely released.’’

Connection Release

The two-army problem.



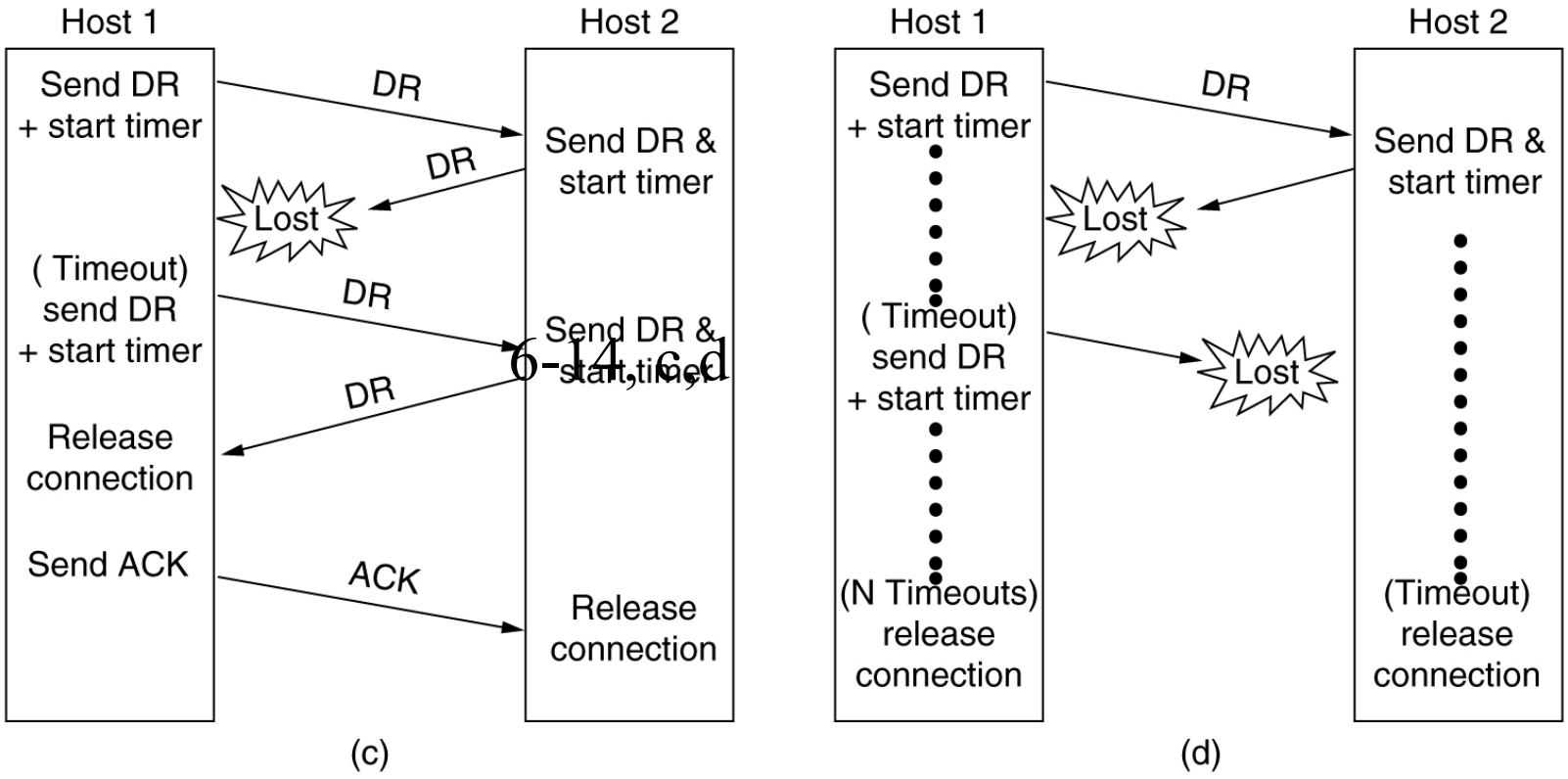
Connection Release



6-14, a, b

Four protocol scenarios for releasing a connection. (a) Normalcase of a three-way handshake. (b) final ACK lost.

Connection Release



6-14.c,d

(c) Response lost. (d) Response lost and subsequent DRs lost.

In Fig. (a), we see the normal case in which one of the users sends a DR (DISCONNECTION REQUEST) segment to initiate the connection release. When it arrives, the recipient sends back a DR segment and starts a timer, just in case its DR is lost. When this DR arrives, the original sender sends back an ACK segment and releases the connection. Finally, when the ACK segment arrives, the receiver also releases the connection.

If the final ACK segment is lost, as shown in Fig.(b), the situationis saved by the timer. When the timer expires, the connection is released anyway. Now consider the case of the second DR being lost. The user initiating the disconnection will not receive the expected response, will time out, and will start all over again.

In Fig.(c), we see how this works, assuming that the second time no segments are lost and all segments are delivered correctlyand on time.

Last scenario, Fig.(d), is the same as Fig. (c) except that now we assume all the repeated attempts to retransmit the DR also fail due to lost segments. After *N retries, the sender just gives up and releases the connection.* Meanwhile, the receiver times out and

TCP

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.

Topics discussed in this section:

TCP Services
TCP Features
Segment
A TCP Connection
Flow Control
Error Control

TCP Services

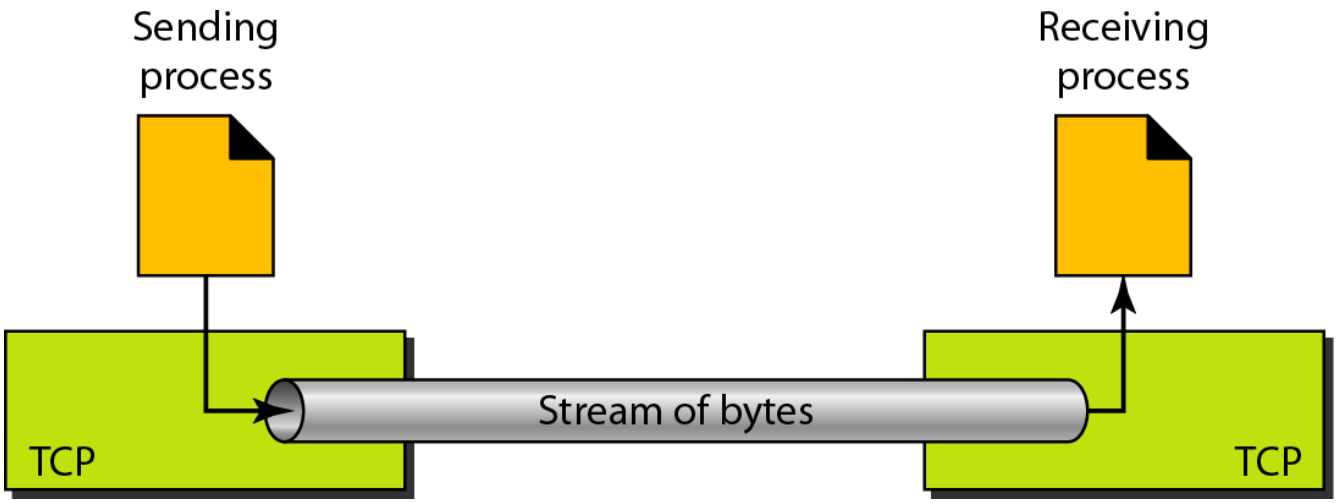
1 Process-to-Process Communication

TCP provides process-to-process communication using port numbers. Below Table lists some well-known port numbers used by TCP.

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

2 Stream Delivery Service

TCP, on the other hand, allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes. TCP creates an environment in which the two processes seem to be connected by an imaginary "tube“ that carries their dataacross the Internet. This imaginary environment is showed in below Figure. The sending process produces (writes to) the stream of bytes, andthe receiving process consumes (reads from) them



3 Sending and Receiving Buffers Because the sending and the receiving processes may not write or read data at the same speed, TCP needs buffers for storage. There are two buffers, the sending buffer and thereceiving buffer, one for each direction. One way to implement a buffer is to use a circular array of I-byte locations as shown in Figure. For simplicity, we have shown two buffers of 20 bytes each. Normally the buffers are hundreds or thousands of bytes, depending on the implementation. Wealso show the buffers as the same size, which is not always the case.

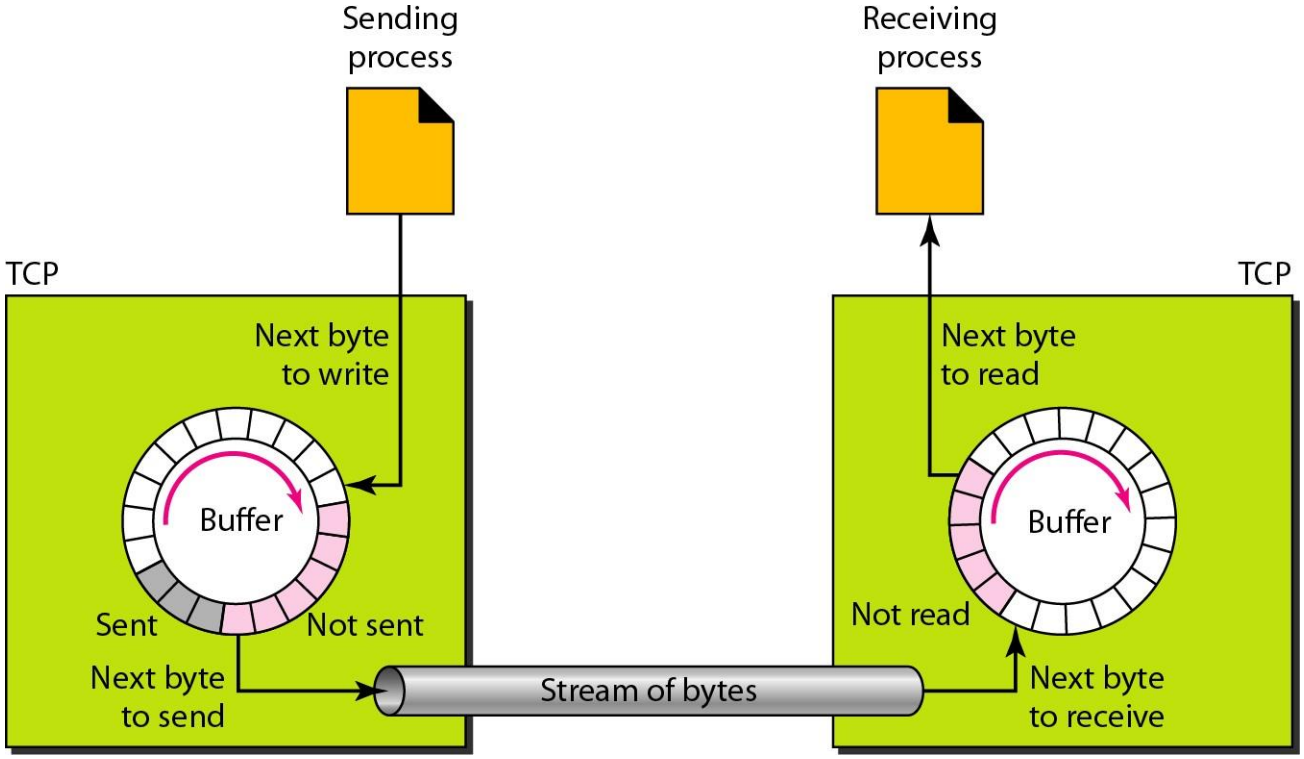
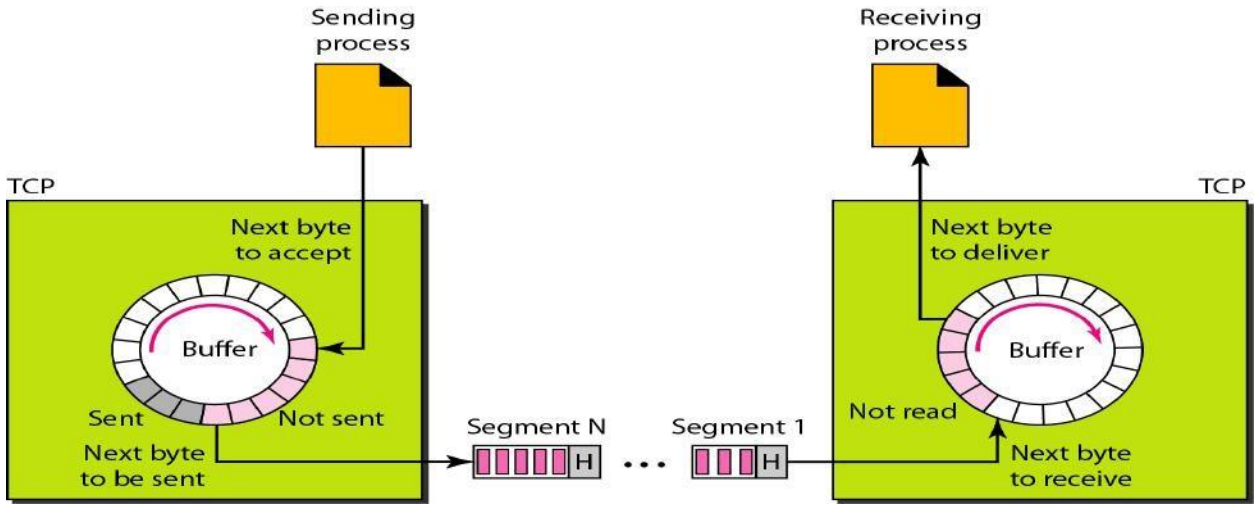


Figure shows the movement of the data in one direction. At the sending site, the buffer has three types of chambers. The white section contains empty chambers that can be filled by the sending process (producer). The gray area holds bytes that have been sent but not yet acknowledged. TCP keeps these bytes in the buffer until it receives an acknowledgment. The colored area contains bytes to be sent by the sending TCP.
However, as we will see later in this chapter, TCP may be able to send only part of this colored section. This could be due to the slowness of the receiving process or perhaps to congestion in the network. Also note that after the bytes in the gray chambers are acknowledged, the chambers are recycled and available for use by the sending process.
This is why we show a circular buffer.

The operation of the buffer at the receiver site is simpler. The circular buffer is divided into two areas (shown as white and colored). The white area contains empty chambers to be filled by bytes received from the network. The colored sections contain received bytes that can be read by the receiving process. When a byte is read by the receiving process, the chamber is recycled and added to the pool of empty chambers.

4 TCP segments



At the transport layer, TCP groups a number of bytes together into a packet called a segment. TCP adds a header to each segment (for control purposes) and delivers the segment to the IP layer for transmission. The segments are encapsulated in IP datagrams and transmitted.

This entire operation is transparent to the receiving process. Later we will see that segments may be received out of order, lost, or corrupted and resent. All these are handled by TCP with the receiving process unaware of any activities. Above fig shows how segments are created from the bytes in the buffers

5 Full-Duplex Communication

TCP offers full-duplex service, in which data can flow in both directions at the same time. Each TCP then has a sending and receiving buffer, and segments move in both directions

6 Connection-Oriented Service

TCP is a connection-oriented protocol. When a process at site A wants to send and receive data from another process at site B, the following occurs:

1. The two TCPs establish a connection between them.
2. Data are exchanged in both directions.
3. The connection is terminated.

7 Reliable Service

TCP is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe and sound arrival of data. We will discuss this feature further in the section on error control.

TCP Features

1 Numbering System

There are two fields called the sequence number and the acknowledgment number. These two fields refer to the byte number and not the segment number.

Byte Number The bytes of data being transferred in each connection are numbered by TCP. The numbering starts with a randomly generated number. For example, if the random number happens to be 1057 and the total data to be sent are 6000 bytes, the bytes are numbered from 1057 to 7056. We will see that byte numbering is used for flow and error control.

Sequence Number After the bytes have been numbered, TCP assigns a sequence number to each segment that is being sent. The sequence number for each segment is the number of the first byte carried in that segment.

Acknowledgment Number The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive. The acknowledgment number is cumulative.

2 Flow Control

TCP, provides *flow control. The receiver of the data controls the amount of* data that are to be sent by the sender. This is done to prevent the receiver from being overwhelmed with data. The numbering system allows TCP to use a byte-oriented flow control.

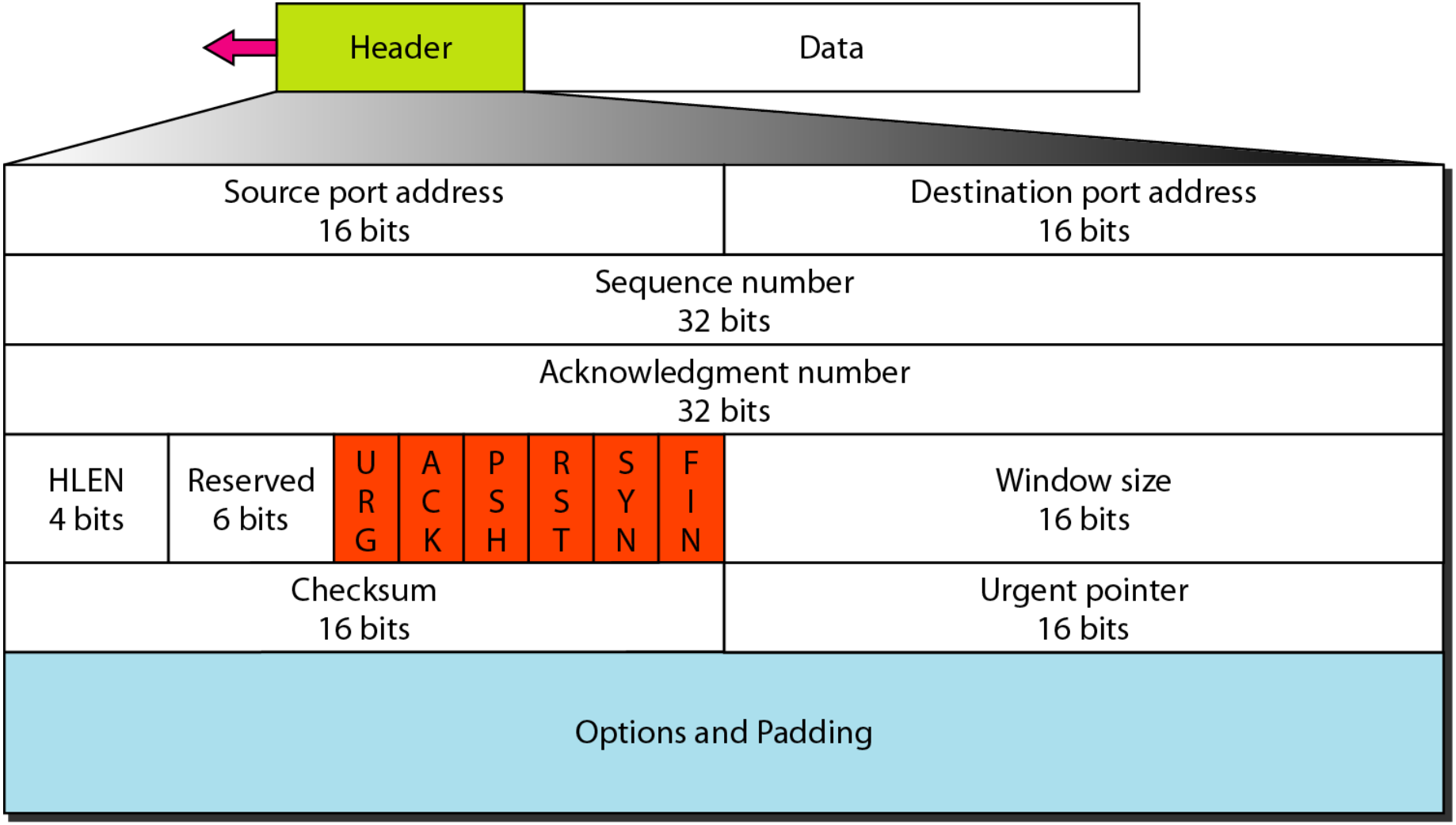
3 Error Control

To provide reliable service, TCP implements an error control mechanism. Although error control considers a segment as the unit of data for error detection (loss or corrupted segments), error control is byte-oriented, as we will see later.

4 Congestion Control

TCP takes into account congestion in the network. The amount of data sent by a sender is not only controlled by the receiver (flow control), but is also determined by the level of congestion in the network

TCP segment format



The segment consists of a 20- to 60-byte header,.

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.

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.

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 + I as the acknowledgment number. Acknowledgment and datacan be* piggybacked together.

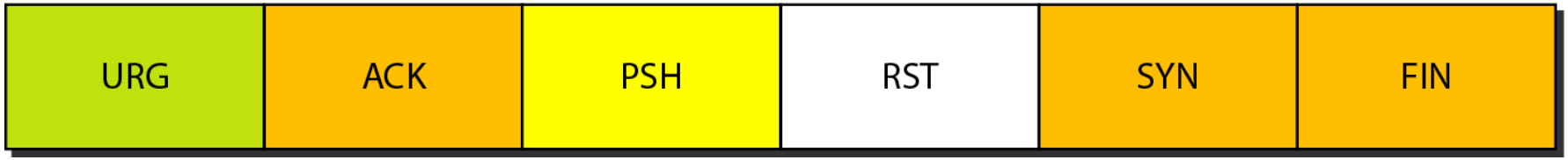
Header length. This 4-bit field indicates the number of 4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field can be between 5 (5 x 4 =20) and 15 (15 x 4 =60).

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.

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push

RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection



These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.

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 l6-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. We will not discuss these options here; please refer to the reference list for more information.

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.

In TCP, connection-oriented transmission requires three phases:

1. connection establishment,
2. data transfer,
3. connection termination.

TCP connection establishment(3 way handshaking)

1 The client sends the first segment, a SYN segment, in which only the SYN flag is set.

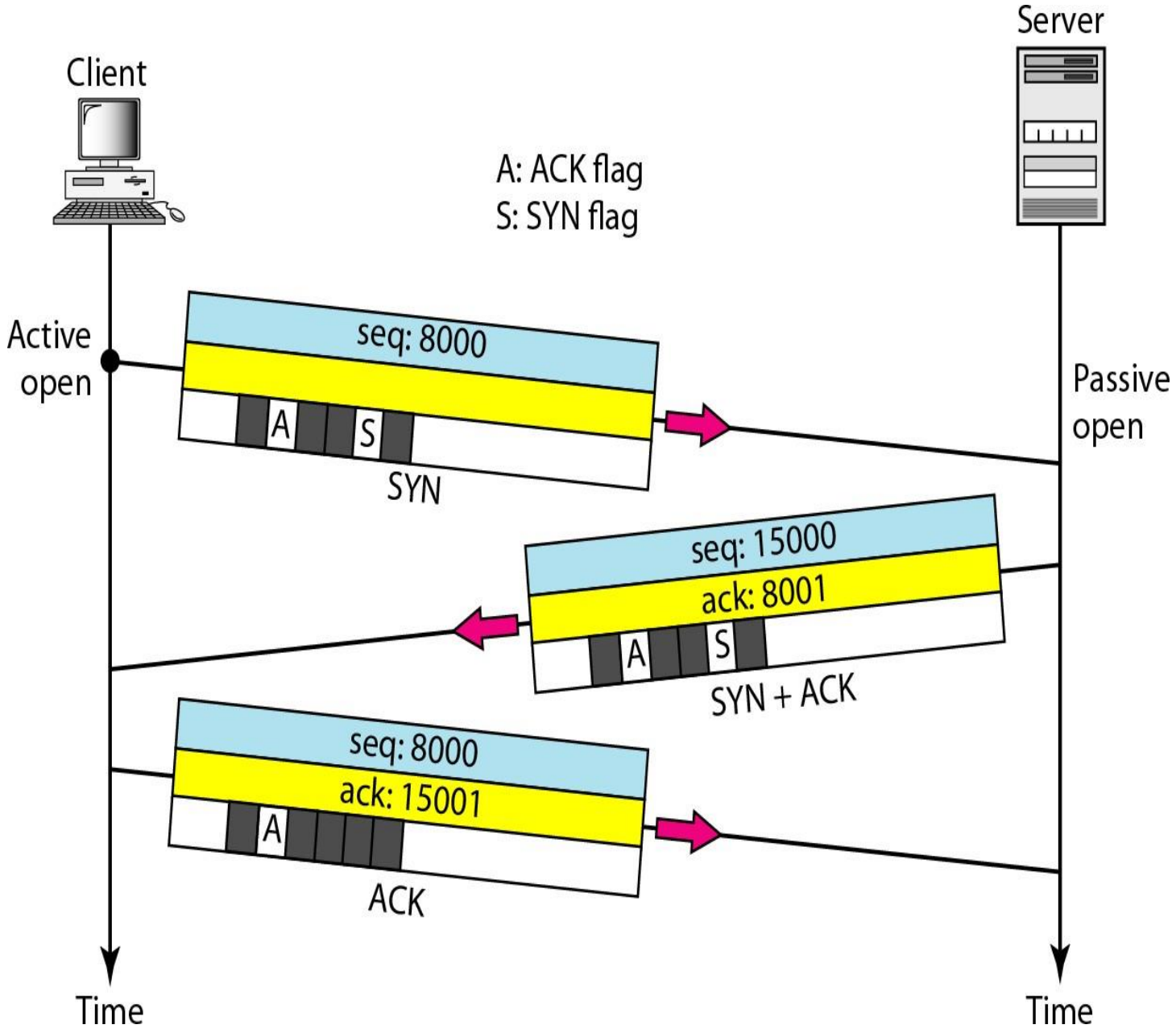
NOTE:A SYN segment cannot carry data, but it consumes one sequence number.

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.

NOTE:A SYN+ACK segment cannot carry data, but does consumeone 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.

NOTE: An ACK segment, if carrying no data, consumes no sequencenumber


Client
Server
A: ACK flag
S: SYN flag
Active open
Passive open
seq: 8000
A
S
SYN
seq: 15000
ack: 8001
A
S
SYN + ACK
seq: 8000
ack: 15001
A
ACK
Time
Time

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 corning from a different client by faking the source IP addresses in the datagram's.

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 anddenies service to every request.

SOLUTIONS:

1 Some have imposed a limit on connection requests during a specified period of
time.

2 Others filter out datagrams coming from unwanted source addresses.

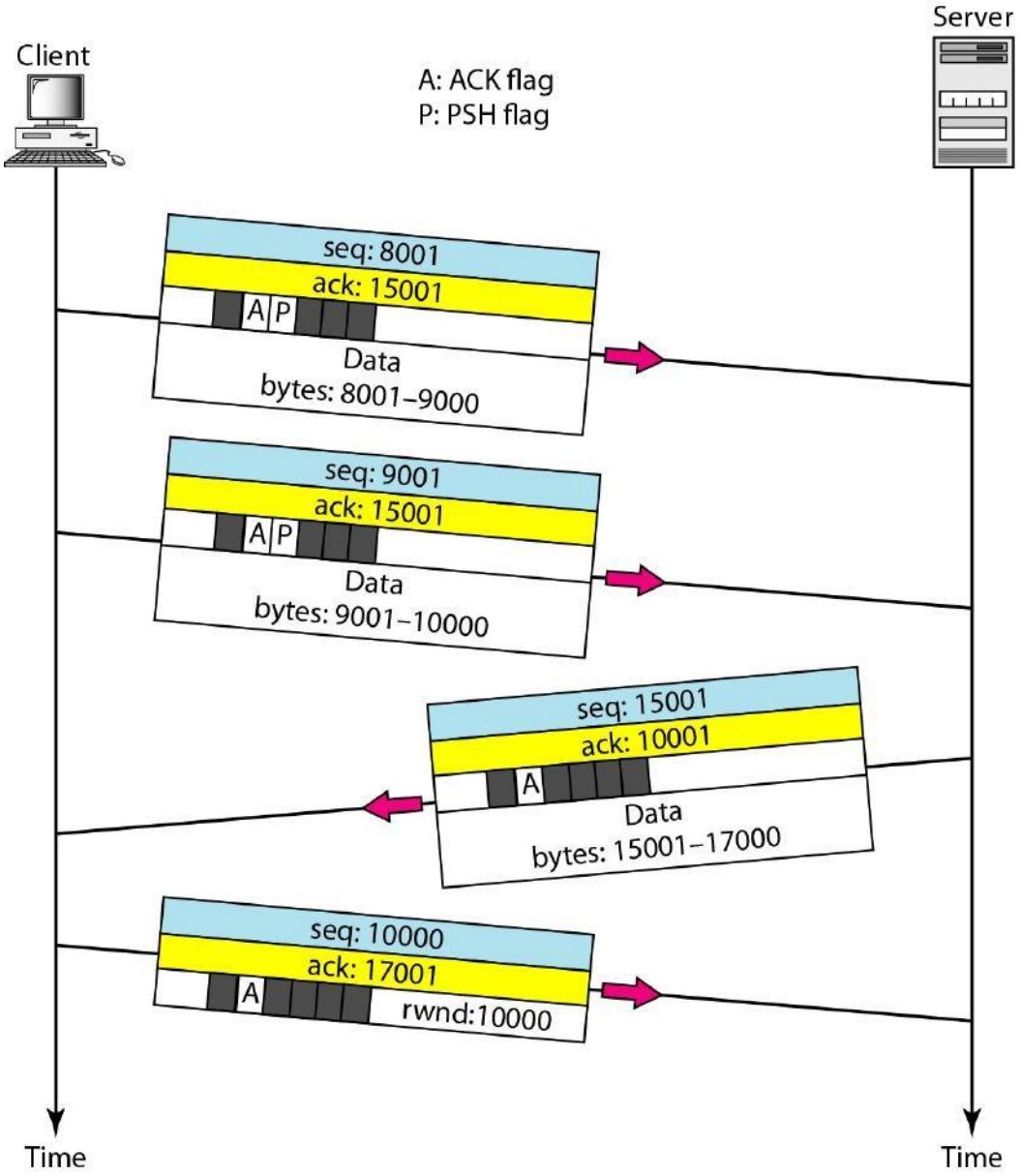
3 One recent strategy is to postpone resource allocation until the entire connection is set up, using what is called a cookie.

Data Transfer

After connection is established, bidirectional data transfer can take place. The client and server can both send data and acknowledgments. Data traveling in the same direction as an acknowledgment are carried on the same segment. The acknowledgment is piggybacked with the data

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 lastsegment carries only an acknowledgment because there are no more datato 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.

Data transfer



PUSHING DATA: 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.

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, sending application program wants a piece of data to be read out of order by the receiving application program.

Connection Termination *(*three-way handshaking and four-way handshaking with a half-close option.)

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 theFIN flag is set.
Note that a FIN segment can include the last chunk of data sent by the client, or it can be just a control segment as shown in Figure. If it is only a control segment, it consumes only one sequence number.
NOTE: The FIN segment consumes one sequence number ifit doesnot carry data.

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.
NOTE: The FIN +ACK segment consumes one sequence number ifit does not carry data.

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.

Half-Close In TCP, one end can stop sending data while still receiving data. This is called a half-close. Although either end can issue a half-close, it is normally initiated by the client. It can occur when the server needs all the data before processing can begin.

A good example is sorting. When the client sends data to the server to be sorted, the server needs to receive all the data before sorting can start. This means the client, after sending all the data, can close the connection in the outbound direction. However, the inbound direction must remain open to receive the sorted data. The server, after receiving the data, still needs time for sorting; its outbound direction must remain open

Connection termination using three-way handshaking

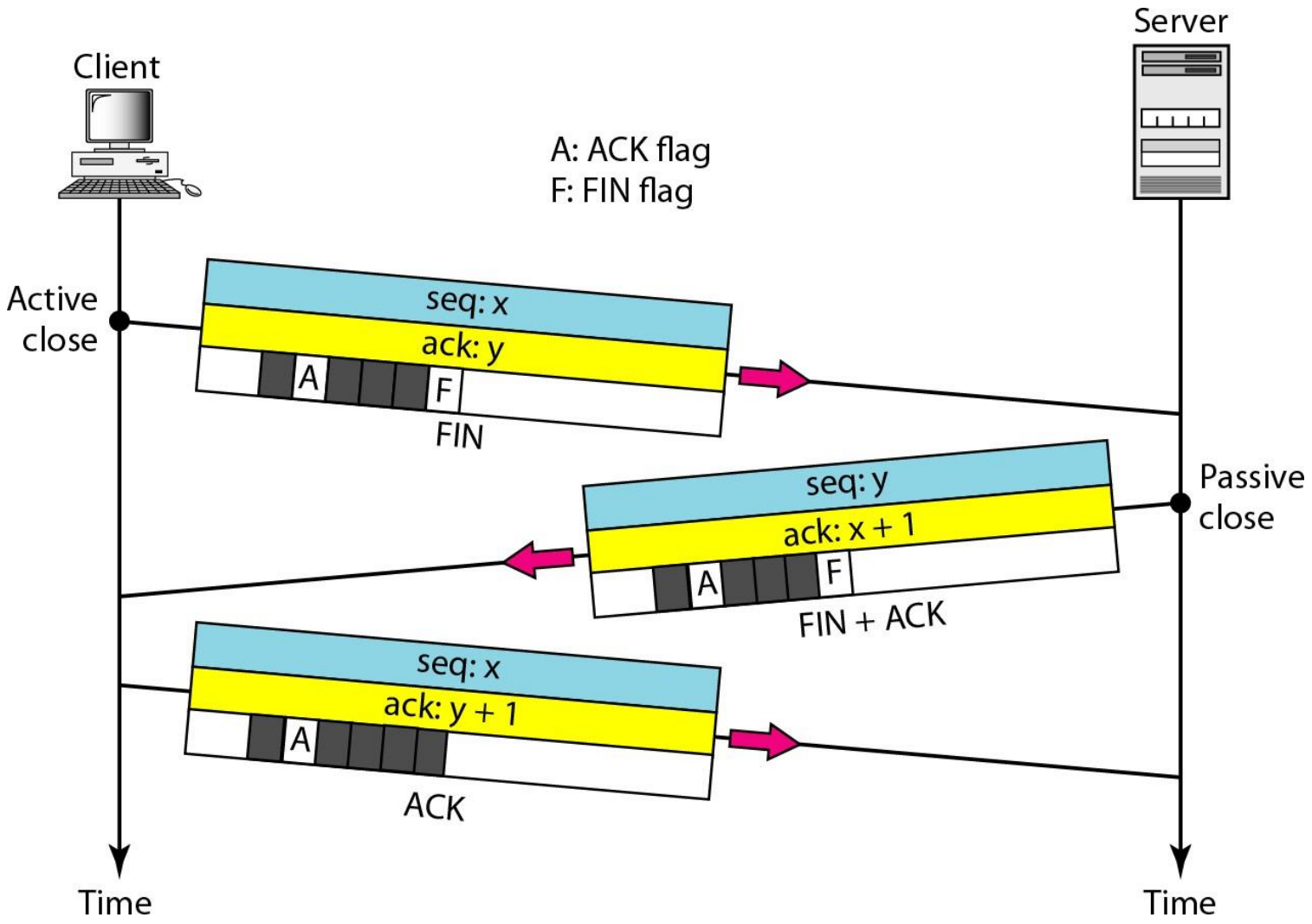
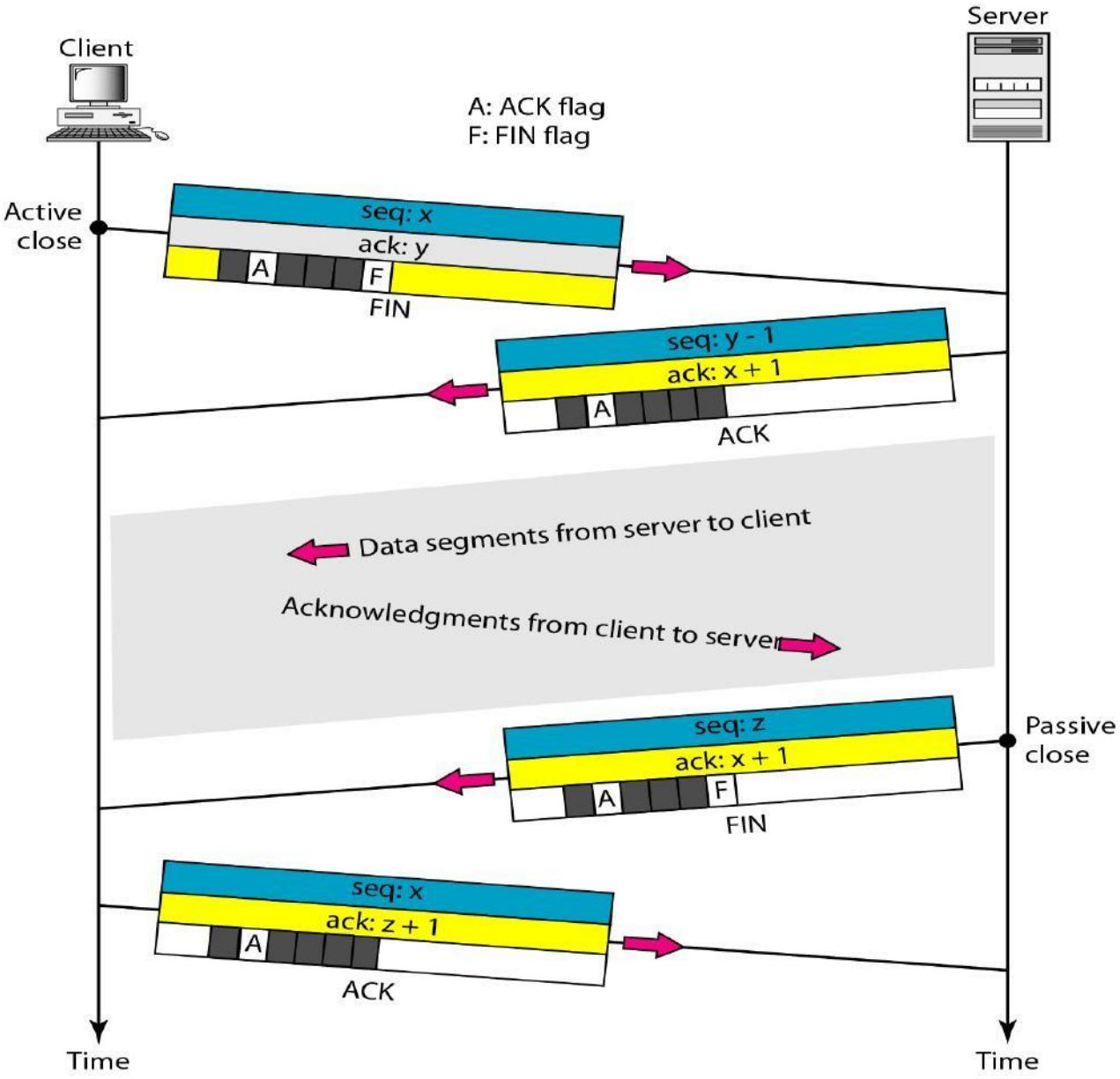


Figure 23.21 *Half-close*



Flow Control or TCP Sliding Window

TCP uses a sliding window, to handle flow control. The sliding window protocol used by TCP, however, is something between the *Go-Back-N* and Selective Repeat sliding window.

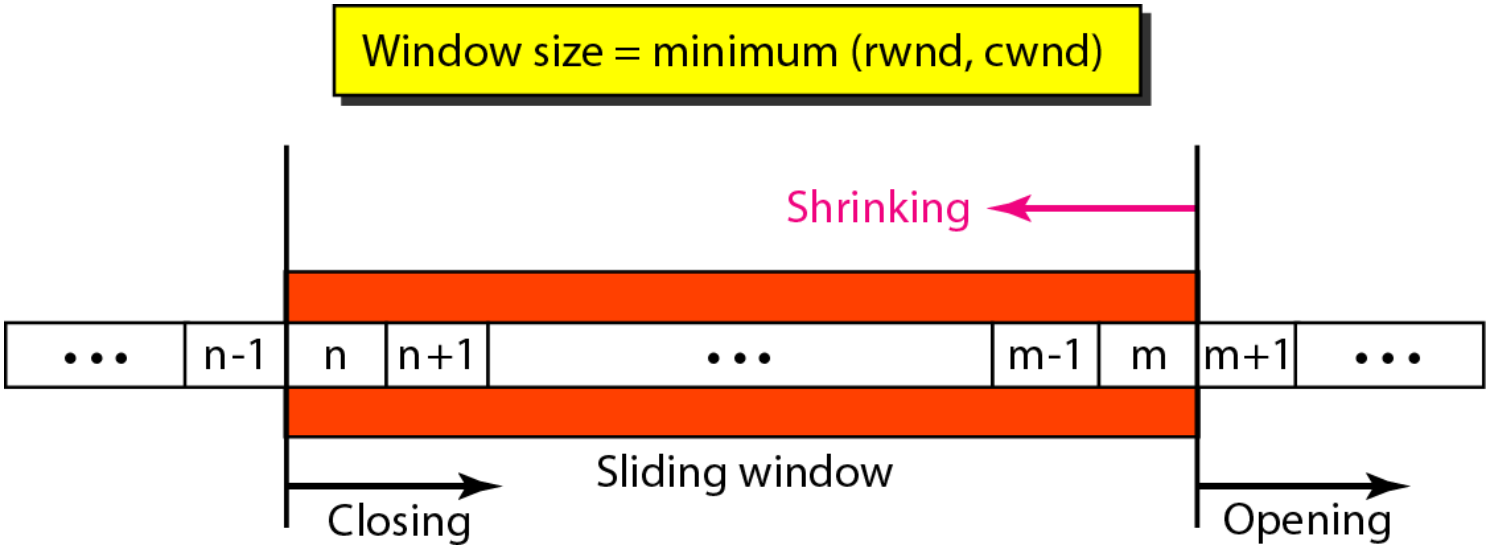
The sliding window protocol in TCP looks like the Go-Back-N protocol because it does not use NAKs;
it looks like Selective Repeat because the receiver holds the out-of-order segments until the missing ones arrive.

There are two big differences between this sliding window and the one we used at the data link layer.

1 the sliding window of TCP is byte-oriented; the one we discussed in the data link layer is frame-oriented.

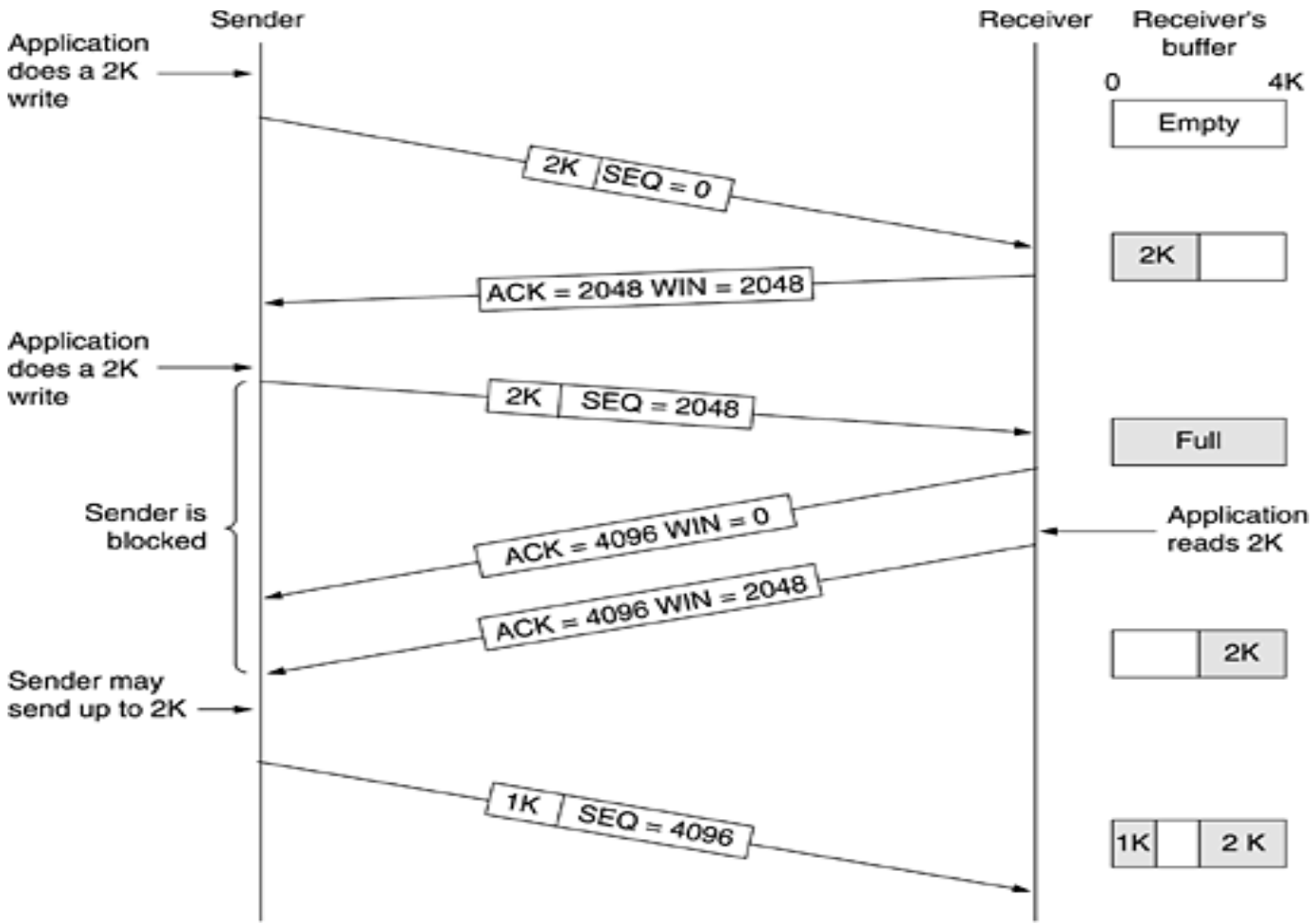
2 the TCP's sliding window is of variable size; the one we discussed in the data link layer was of fixed size

Sliding window



The window is *opened, closed, or shrunk*. *These three activities, as we will see, are* in the control of the receiver (and depend on congestion in the network), not the sender.
The sender must obey the commands of the receiver in this matter.
Opening a window means moving the right wall to the right. This allows more new bytes in the buffer that are eligible for sending.
Closing the window means moving the left wall to the right. This means that some bytes have been acknowledged and the sender need not worry about them anymore.
Shrinking the window means moving the right wall to the left.

The size of the window at one end is determined by the lesser of two values: *receiver window (rwnd) or congestion window (cwnd).*
The receiver window is the value advertised by the opposite end in a segment containing acknowledgment. It is the number of bytes the other end can accept before its buffer overflows and data are discarded.
The congestion window is a value determined by the network to avoid congestion



Window management in TCP

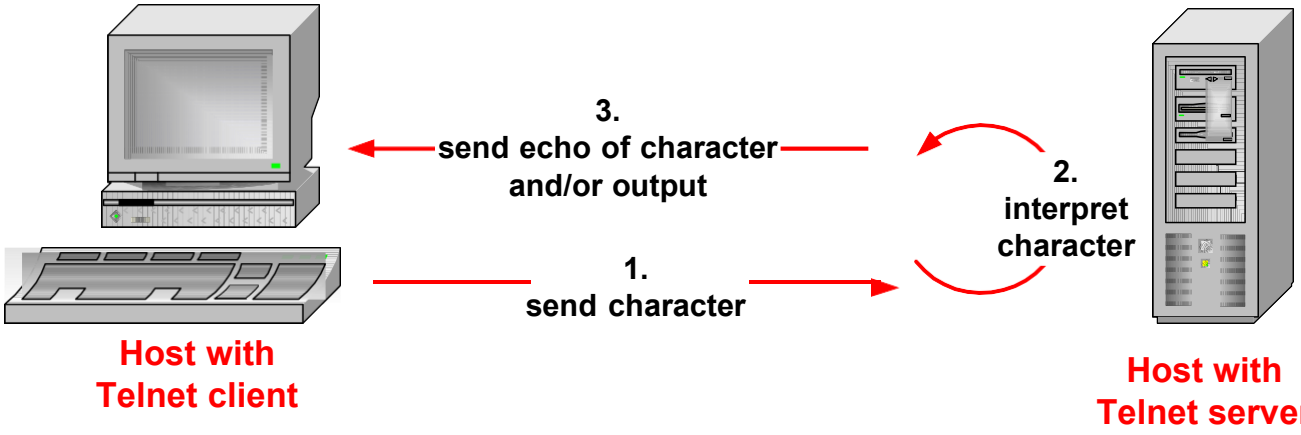
When the window is 0, the sender may not normally send segments, with two exceptions.

1) urgent data may be sent, for example, to allow the user to kill the process running on the remote machine.
2) the sender may send a 1-byte segment to force the receiver to reannounce the next byte expected and the window size. This packet is called a **window probe.**

The TCP standard explicitly provides this option to prevent deadlock if a window update ever gets lost.

Senders are not required to transmit data as soon as they come in from the application. Neither are receivers required to send acknowledgements as soon as possible.

For example, in Fig. when the first 2 KB of data came in, TCP, knowing that it had a 4-KB window, would have been completely correct in just buffering the data until another 2 KB came in, to be able to transmit a segment with a 4-KB payload. This freedom can be used to improve performance



Remote terminal applications (e.g., Telnet) send characters to a server.

The server interprets the character and sends the output at the server to the client.

For each character typed, you see three packets:

Client □ Server: Send typed character

Server □ Client: Echo of character (or user output) and acknowledgement for first packet

Client □ Server: Acknowledgement for second packet

Delayed Acknowledgement

- TCP delays transmission of ACKs for up to 500ms

- Avoid to send ACK packets that do not carry data.
  - The hope is that, within the delay, the receiver will have data ready to be sent to the receiver. Then, the ACK can be piggybacked with a data segment

Exceptions:

- ACK should be sent for every full sized segment
- Delayed ACK is not used when packets arrive out of order

Although delayed acknowledgements reduce the load placed on the network by the receiver, a sender that sends multiple short packets (e.g., 41-byte packets containing 1 byte of data) is still operating inefficiently. A way to reduce this usage is known as **Nagle's algorithm (Nagle, 1984).**

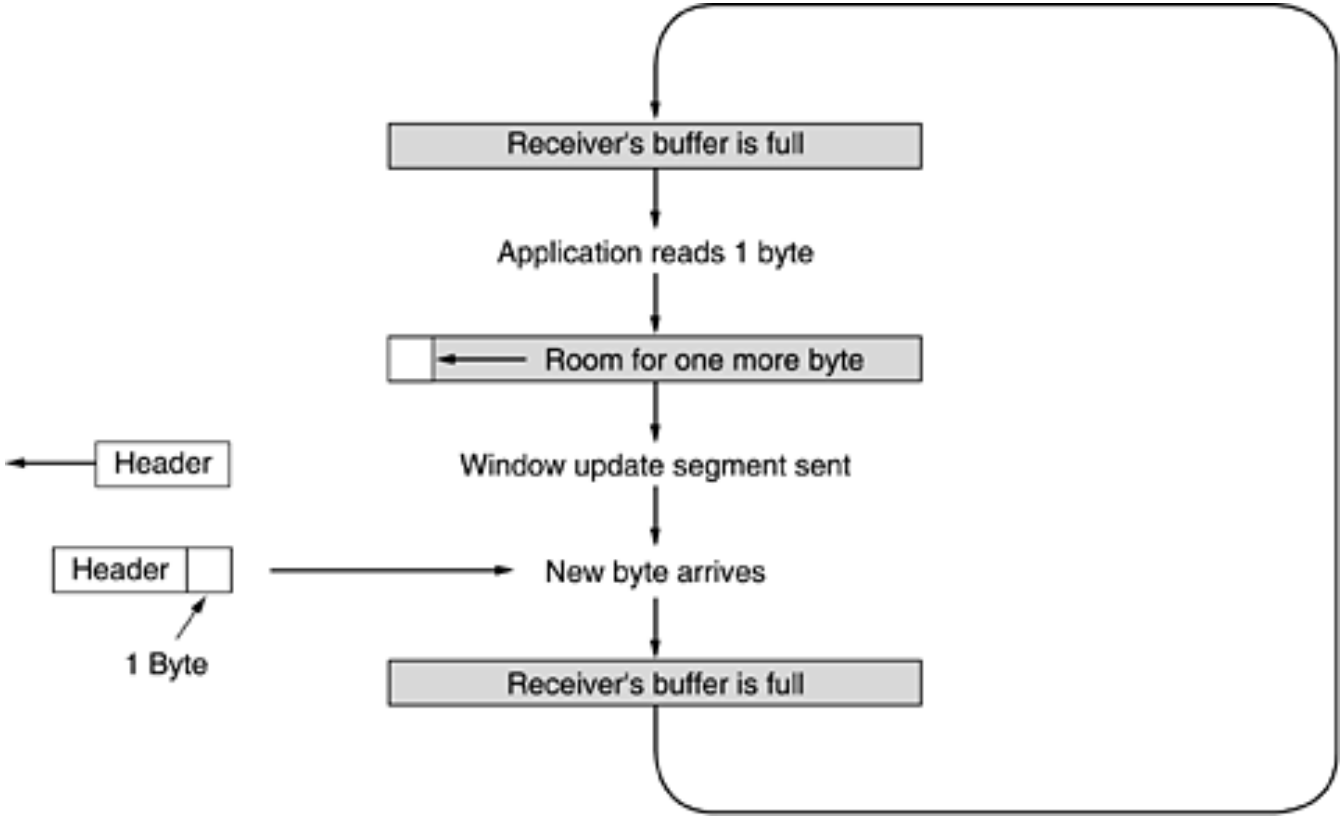
Nagel's Rule

Send one byte and buffer all subsequent bytes until acknowledgement is received. Then send all buffered bytes in a single TCP segment and start buffering again until the sent segment is acknowledged.
Nagle's algorithm will put the many pieces in one segment, greatly reducing the bandwidth used

Nagle's algorithm is widely used by TCP implementations, but there are times when it is better to disable it. In particular, in interactive games that arerun over the Internet.
A more subtle problem is that Nagle's algorithm can sometimes interact with delayed acknowledgements to cause a temporary deadlock: the receiver waits for data on which to piggyback an acknowledgement, and the sender waits on the acknowledgement to send more data.

Because of these problems, Nagle's algorithm can be disabled (which is called the *TCP NODELAY option).*

Another problem that can degrade TCP performance is the **silly window syndrome** (Clark, 1982).



Clark's solution is to prevent the receiver from sending a window update for 1 byte. Instead, it is forced to wait until it has a decent amount of space available and advertise that instead. Specifically, the receiver should not send a window update until it can handle the maximum segment size it advertised when the connection was established or until its buffer is half empty, whichever is smaller.

Furthermore, the sender can also help by not sending tiny segments. Instead, it should wait until it can send a full segment, or at least one containing half of the receiver's buffer size.

The goal is for the sender not to send small segments and the receiver not to ask for them. (Nagel + Clark). Both are used to improve TCP performance

The receiver will buffer the data until it can be passed up to the application in order (handling out of order segments)

Cumulative acknowledgements

Error Control

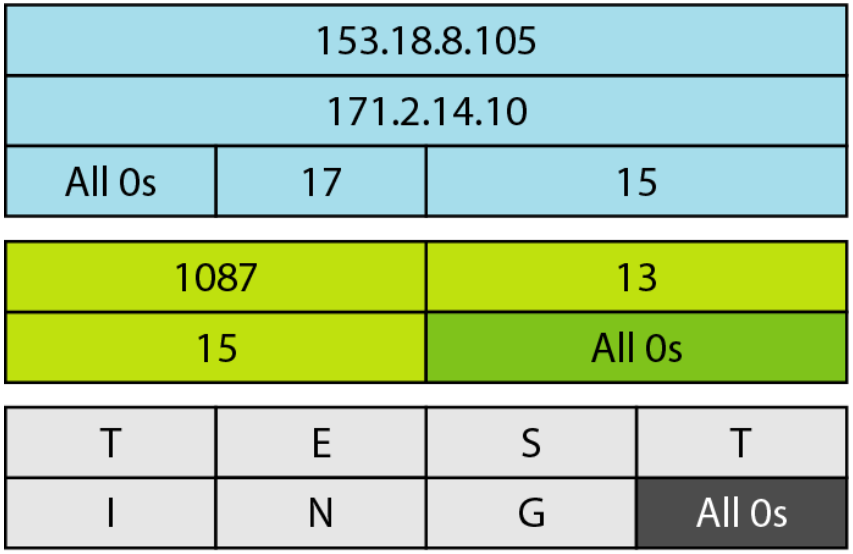
TCP is a reliable transport layer protocol. This means that an application program that delivers a stream of data to TCP relies on TCP to deliver the entire stream to the application program on the other end in order, without error, and without any part lost or duplicated.

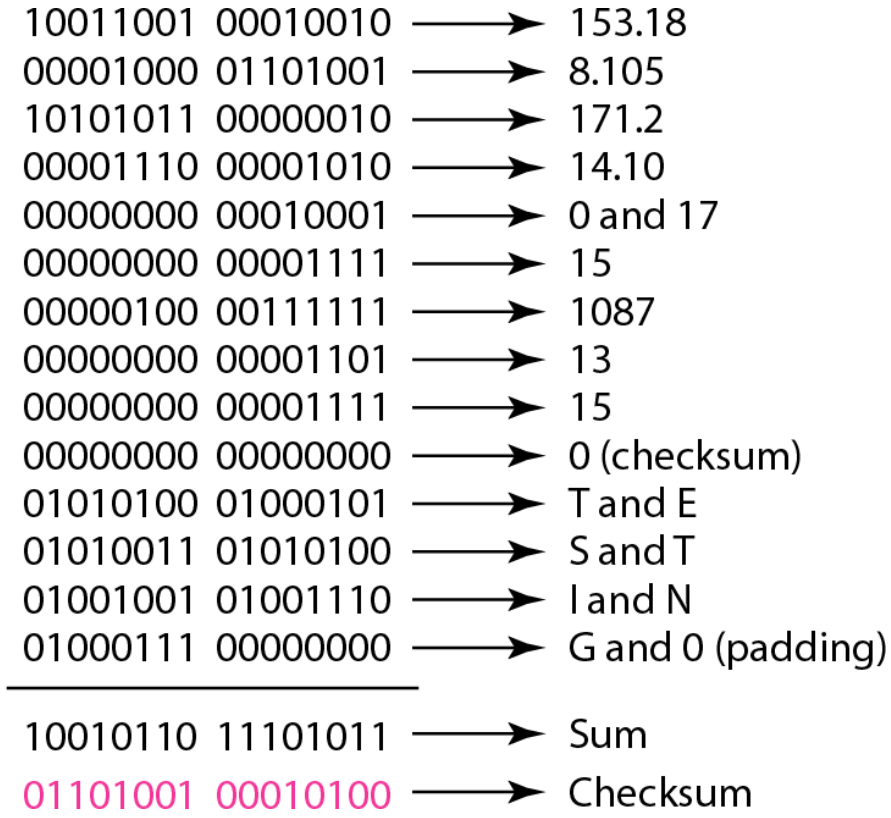
TCP provides reliability using error control. Error control includes mechanisms for detecting corrupted segments, lost segments, out-of-order segments, and duplicated segments. Error control also includes a mechanism for correcting errors after they are detected. Error detection and correction in TCP is achieved through the use of three simple tools: **checksum, acknowledgment, and time-out.**

Checksum

Each segment includes a checksum field which is used to check for acorrupted segment. If the segment is corrupted, it is discarded by the destination TCP and is considered as lost. TCP uses a 16-bit checksum that is mandatory in every segment

Figure 23.11 *Checksum calculation of a simple UDP user datagram*





Acknowledgment

TCP uses acknowledgments to confirm the receipt of data segments.Control segments that carry no data but consume a sequence number are also acknowledged. ACK segments are never acknowledged.

ACK segments do not consume sequence numbers and are not acknowledged.

Retransmission

The heart of the error control mechanism is the retransmission of segments. When a segment is corrupted, lost, or delayed, it isretransmitted.

In modern implementations, a retransmission occurs if the retransmission timer expires or three duplicate ACK segments have arrived.

Retransmission After RTO (retransmission time out)

Retransmission After Three Duplicate ACK Segments (also called fast retransmission)

Out-of-Order Segments

Data may arrive out of order and be temporarily stored by the receiving TCP, but yet guarantees that no out-of-order segment is delivered to the process

When the load offered to any network is more than it can handle, congestion builds up.
The network layer detects congestion when queues grow large at routers and tries to manage it, if only by dropping packets. It is up to the transport layer to receive congestion feedback from the network layer and slow down the rate of traffic that it is sending into the network.

For Congestion control, transport protocol uses an AIMD (Additive Increase Multiplicative Decrease) control law.

TCP congestion control is based on implementing this approach using a window called **congestion window.** TCP adjusts the size of the window according to the AIMD rule.

The window size at the sender is set as follows:

Send Window = MIN (flow control window,congestion window)

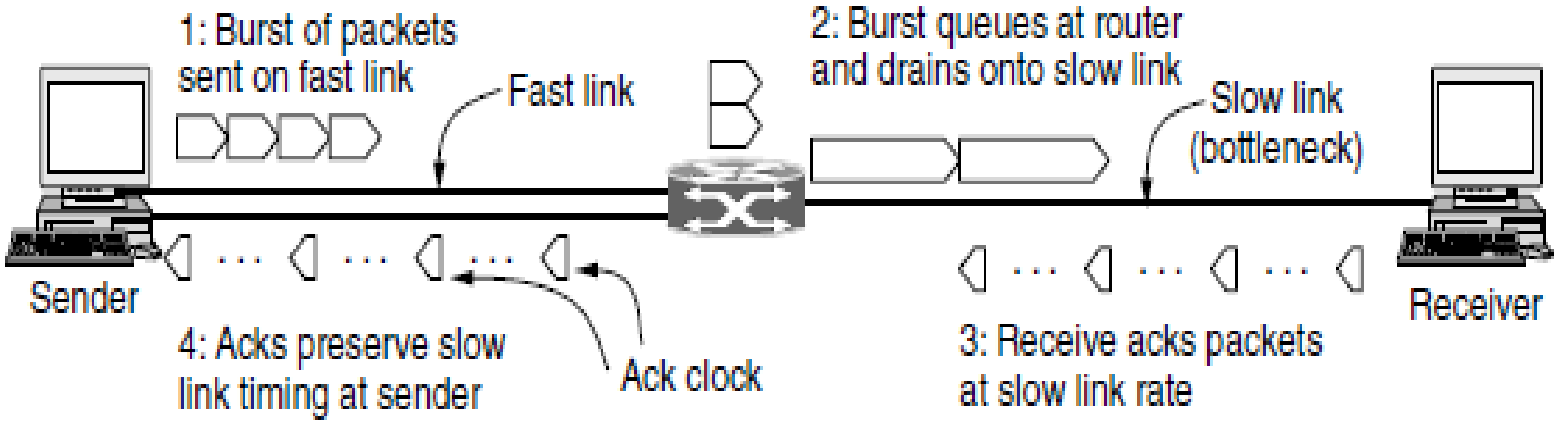
where

flow control window is advertised by the receiver (rwnd) congestion window is adjusted based on feedback from the

Modern congestion control was added to TCP largely through the efforts of Van Jacobson (1988). It is a fascinating story. Starting in 1986, the growing popularity of the early Internet led to the first occurrence of what became known as a **congestion collapse**, a prolonged period during which good put dropped suddenly (i.e., by more than a factor of 100) due to congestion in the network. Jacobson (and many others) set out to understand whatwas happening and remedy the situation.

To start, he observed that packet loss is a suitable signal of congestion. This signal comes a little late (as the network is already congested) but it is quite dependable

At the beginning how sender knows at what speed receiver can receive the packets?

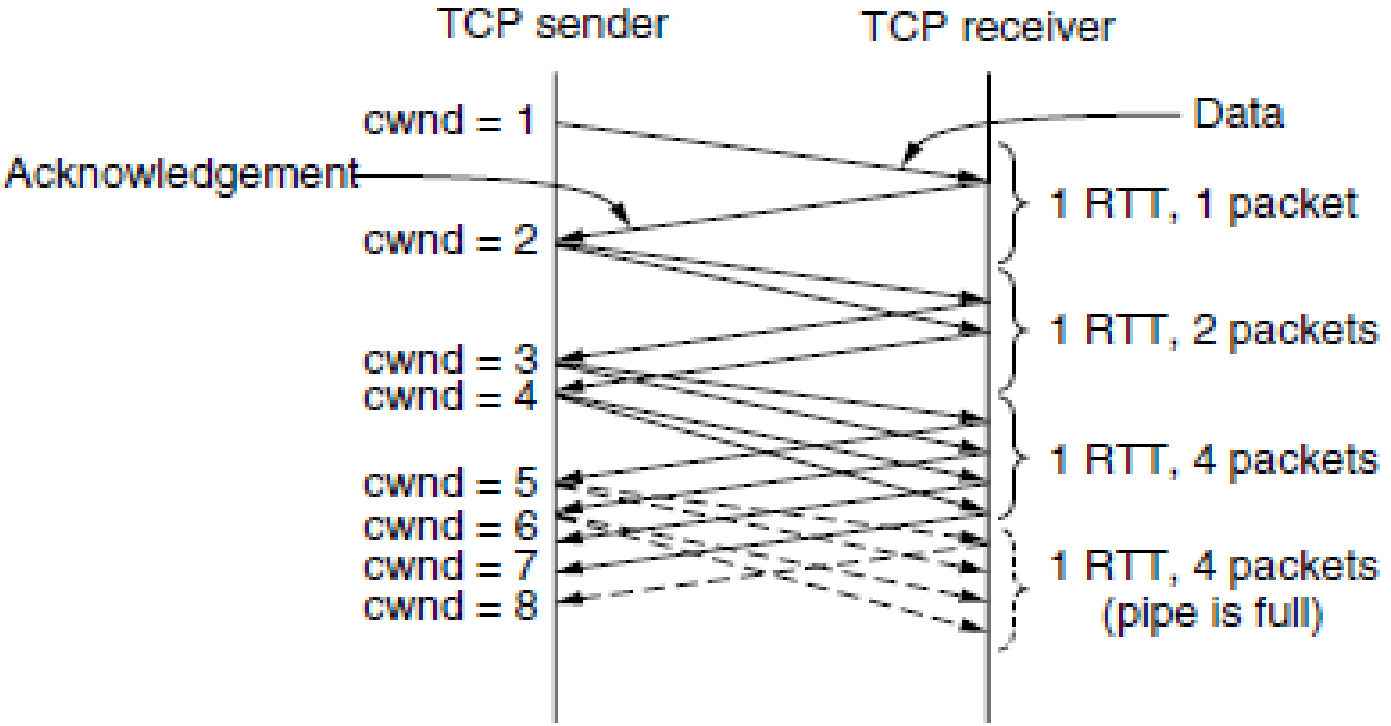


The key observation is this: the acknowledgements return to the sender at about the rate that packets can be sent over the slowest link in the path. This is precisely the rate that the sender wants to use. If it injects new packets into the network at this rate, they will be sent as fast as the slow link permits, but they will not queue up and congest any router along the path. This timing is known as an **ack clock**. It is an essential part of TCP. By using an ack clock, TCP smoothes out traffic and avoids unnecessary queues at routers. This is first consideration

A second consideration is that the AIMD rule will take a very long time to reach a good operating point on fast networks if the congestion window is started from a small size

Instead, the solution Jacobson chose to handle both of these considerations is a mix of linear and multiplicative increase.

SLOW-START



TCP Congestion Control

Slow Start

- Additive Increase / Multiplicative Decrease is only suitable for source, that is operating close to the available capacity of the network, but it takes too long to ramp up a connection when it is starting from scratch.
- slow start, that is used to increase the congestion window rapidly from a cold start.
- Slow start effectively **increases the congestion window exponentially**, rather than linearly.
  - the source starts out by setting CongestionWindow to one packet.
  - When the ACK for this packet arrives, TCP adds 1 to CongestionWindow and then sends two packets.
  - Upon receiving the corresponding two ACKs, TCP increments CongestionWindow by 2—one for each ACK—and next sends four packets.
  - The end result is that TCP effectively doubles the number of packets it has in transit every RTT.

Whenever a packet loss is detected, for example, by a timeout, the slow start threshold is set to be half of the congestion window and the entire process is restarted.

Congestion avoidance phase is started if cwnd has reached the slow start threshold value

Whenever the slow start threshold is crossed, TCP switches from slow start to additive increase. In this mode, the congestion window is increased by one segment every round-trip time.

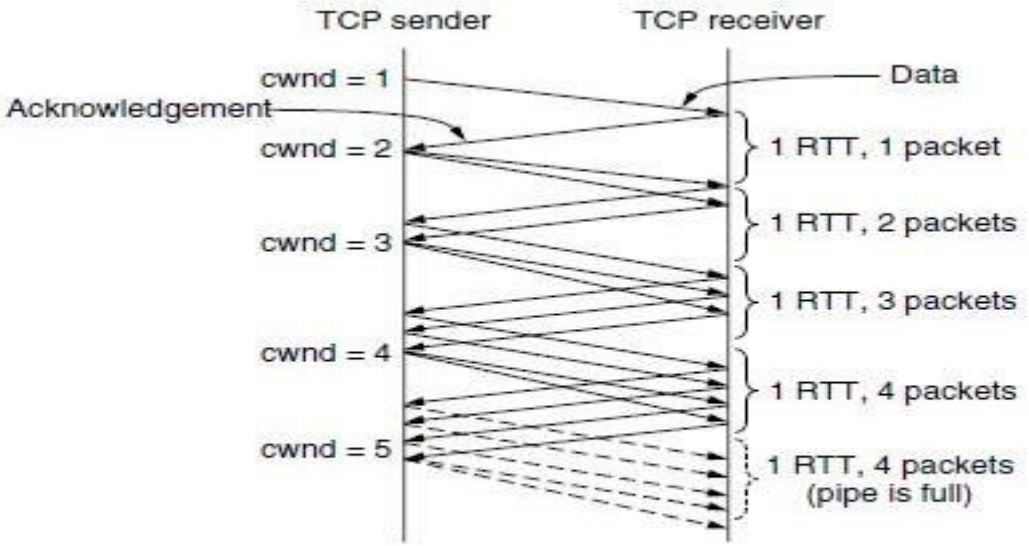


Figure 6-45. Additive increase from an initial congestion window of one segment.

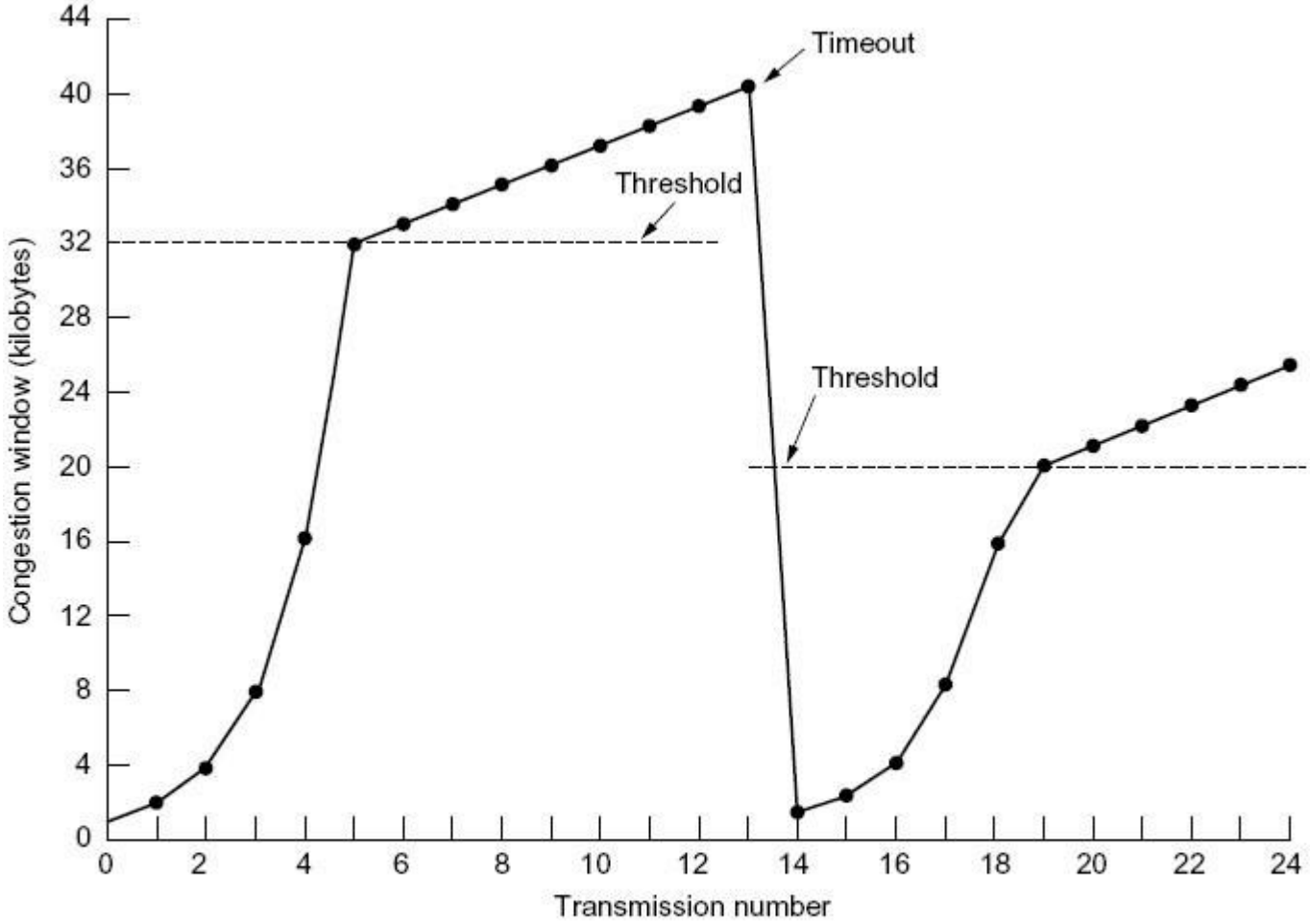


Fig. 6-37. An example of the Internet congestion algorithm.

Responses to Congestion

- So, TCP assumes there is congestion if it detects a packet loss
- A TCP sender can detect lost packets via:
  - Timeout of a retransmission timer
  - Receipt of a duplicate ACK

- TCP interprets a Timeout as a binary congestion signal. When a timeout occurs, the sender performs:
  - cwnd is reset to one:

    cwnd = 1
  - ssthresh is set to half the current size of the congestion window:

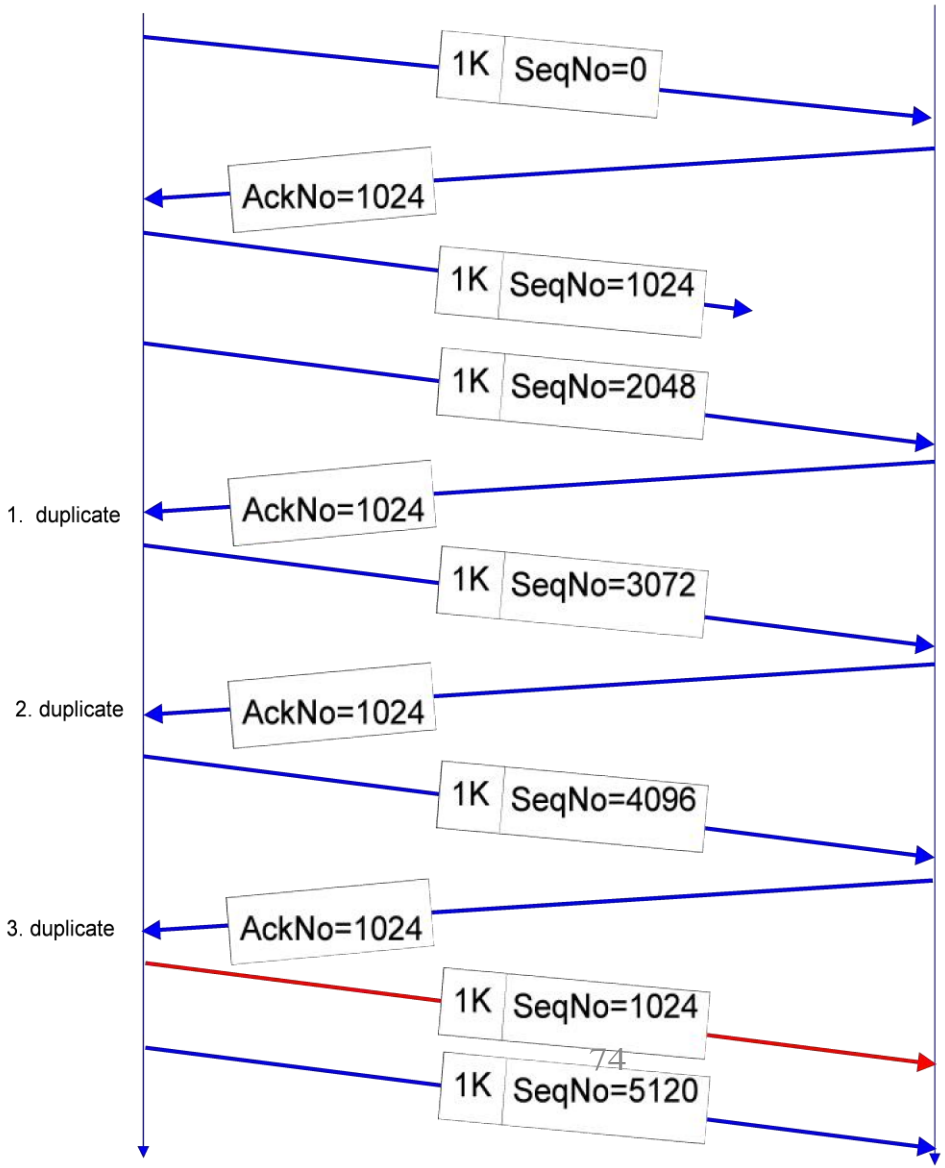
    ssthressh = cwnd / 2
  - and slow-start is entered

Fast Retransmit

- If three or more duplicate ACKs are received in a row, the TCP sender believes that a segment has been lost.
- Then TCP performs a retransmission of what seems to be the missing segment, without waitingfor a timeout to happen.
- Enter slow start:

  ssthresh = cwnd/2

  cwnd = 1



Flavors of TCP Congestion

Control

- **TCP Tahoe** (1988)
  - Slow Start
  - Congestion Avoidance
  - Fast Retransmit
- **TCP Reno** (1990) (TCP Tahoe+FR)
  - Fast Recovery
- **New Reno** (1996)
- **SACK** (1996) (**SACK (Selective ACKnowledgements)**)

- RED (Floyd and Jacobson 1993)

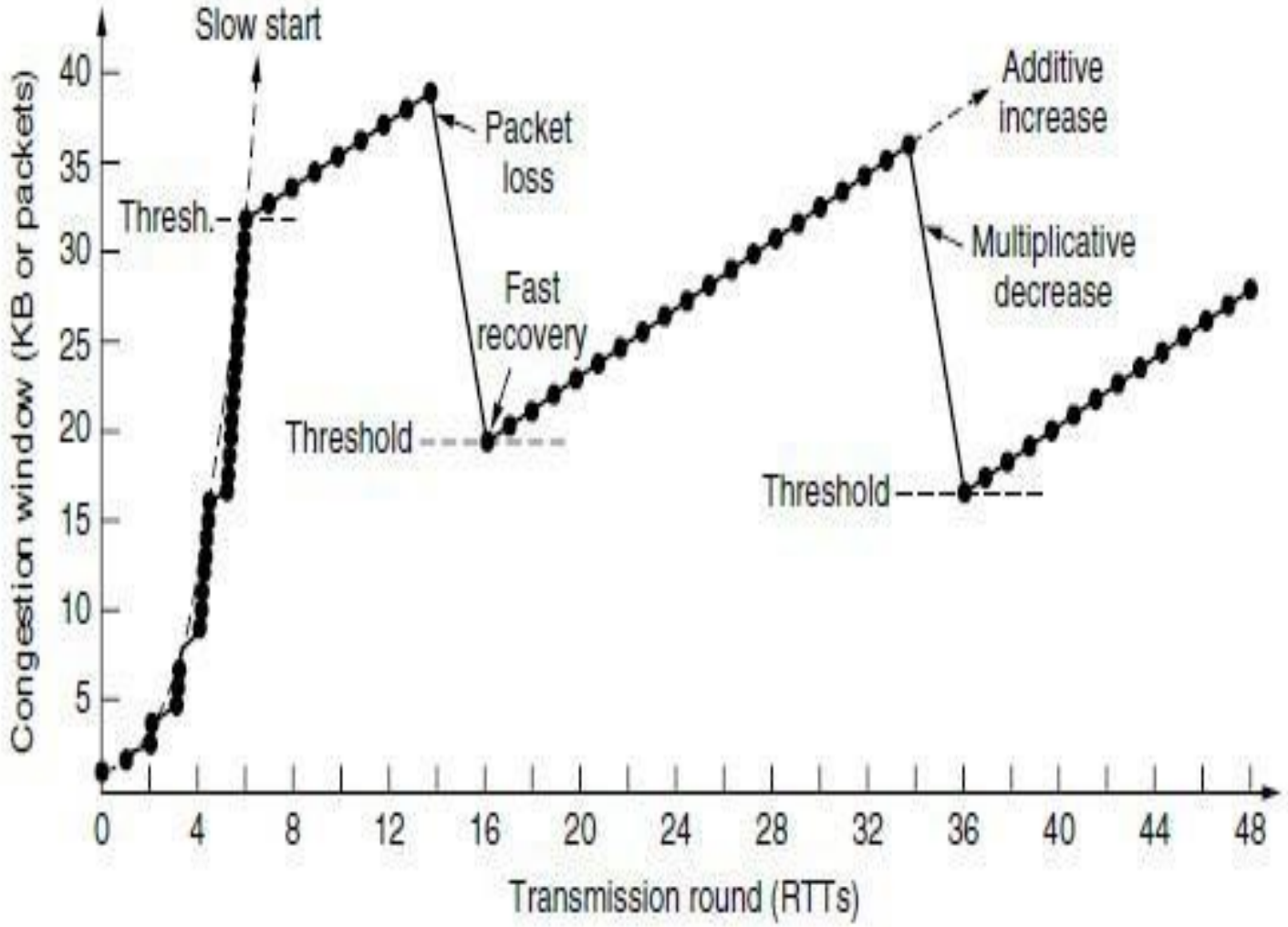


Figure 6-47. Fast recovery and the sawtooth pattern of TCP Reno.

The use of ECN (Explicit Congestion Notification) in addition to packet loss as a congestion signal. ECN is an IP layer mechanism to notify hosts of congestion.

The sender tells the receiver that it has heard the signal by using the *CWR (Congestion Window Reduced) flag.*

USER DATAGRAM PROTOCOL (UDP)

The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It doesnot add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication.

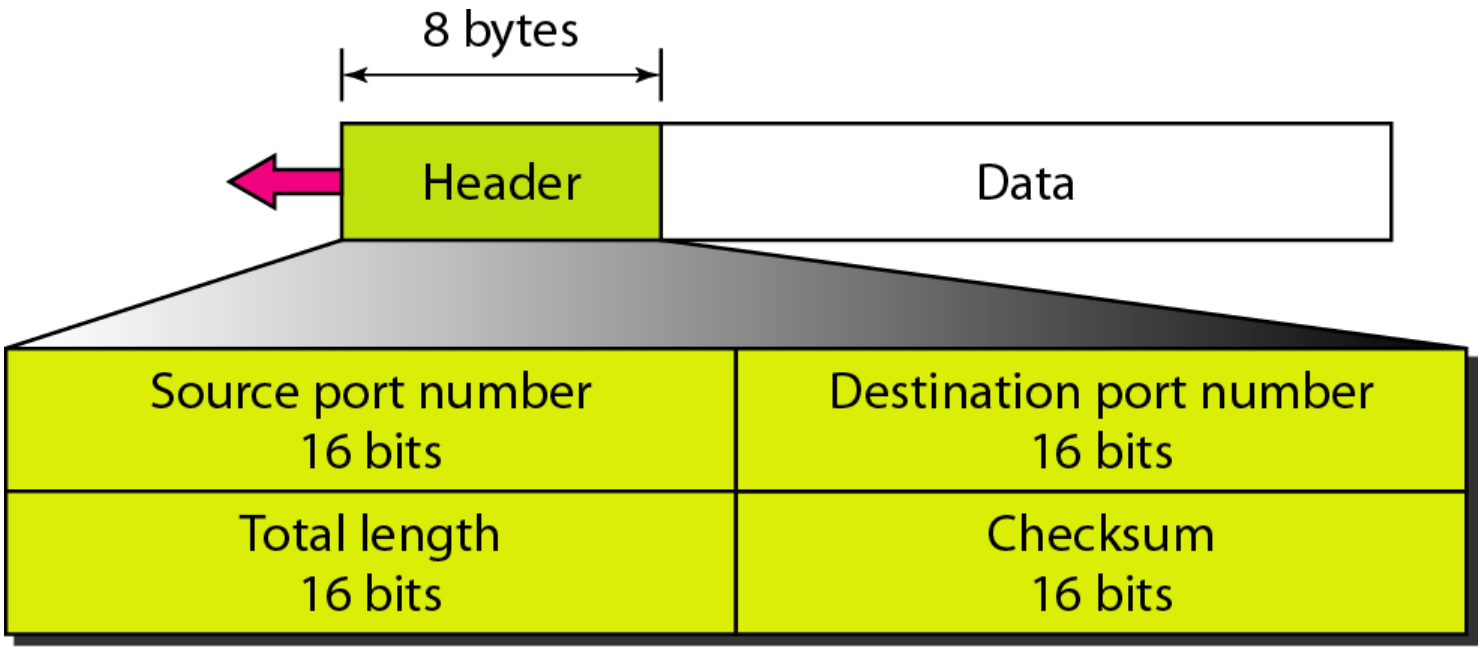
Topics discussed in this section:

Well-Known Ports for UDP
User Datagram
Checksum
UDP Operation
Use of UDP

Table 23.1 *Well-known ports used with UDP*

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

Figure 23.9 *User datagram format*

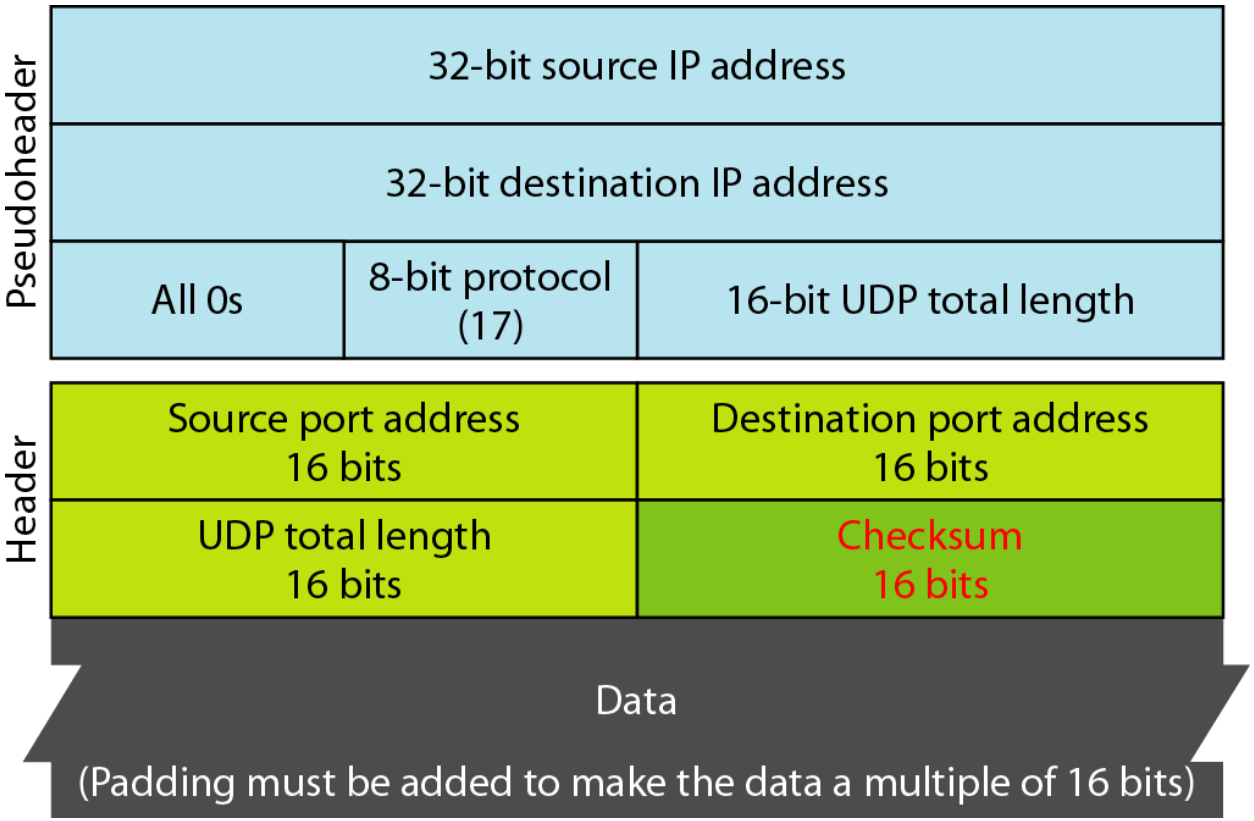


Checksum (OPTIONAL,IF NOT USED SET ALL 1'S DEFAULT)

The UDP checksum calculation is different from the one for IP and ICMP. Here the checksum includes three sections: **a pseudo header, the UDP header, and the data** coming from the application layer.
The pseudo header is the part of the header of the IP packet in which the user datagram is to be encapsulated with some fields filled with Os

If the checksum does not include the pseudo header, a user datagram may arrive safe and sound. However, if the IP header is corrupted, it may be delivered to the wrong host.
The protocol field is added to ensure that the packet belongs to UDP, and not to other transport-layer protocols.

Figure 23.10 *Pseudoheader for checksum calculation*



UDP Operation

Connectionless Services

UDP provides a connectionless service. This means that each user datagram sent by UDP is an independent datagram. There is no relationship between the different user datagrams even if they are comingfrom the same source process and going to the same destination program.The user datagrams are not numbered. Also, there is no connection establishment and no connection termination, as is the case for TCP. Thismeans that each user datagram can travel on a different path.

Flow and Error Control

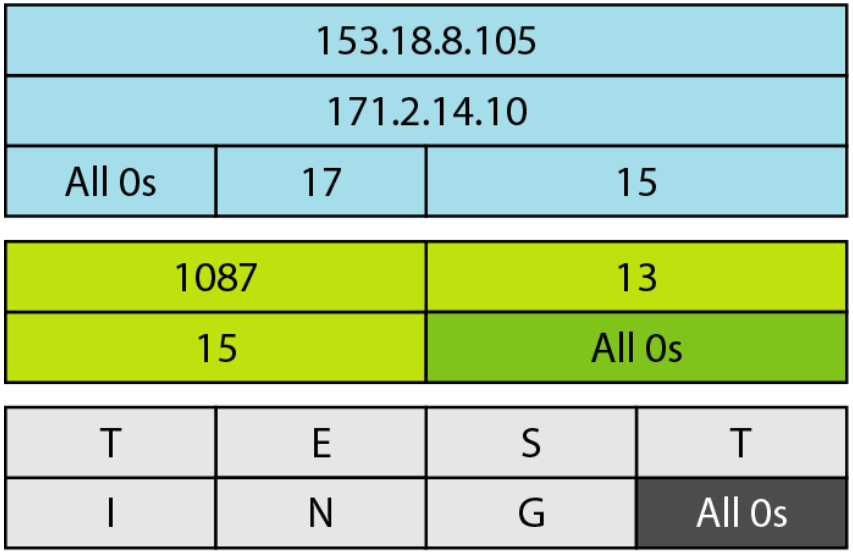
UDP is a very simple, unreliable transport protocol. There is no flow controland hence no window mechanism. The receiver may overflow with incoming messages. There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded. The lack of flow control and error control

Encapsulation and Decapsulation

To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages in an IP datagram.

Figure 23.11 shows the checksum calculation for a very small user datagram with only 7 bytes of data. Because the number of bytes of data is odd, padding is added for checksum calculation. The pseudoheader as well as thepadding will be dropped when the user datagram is delivered to IP.

Figure 23.11 *Checksum calculation of a simple UDP user datagram*



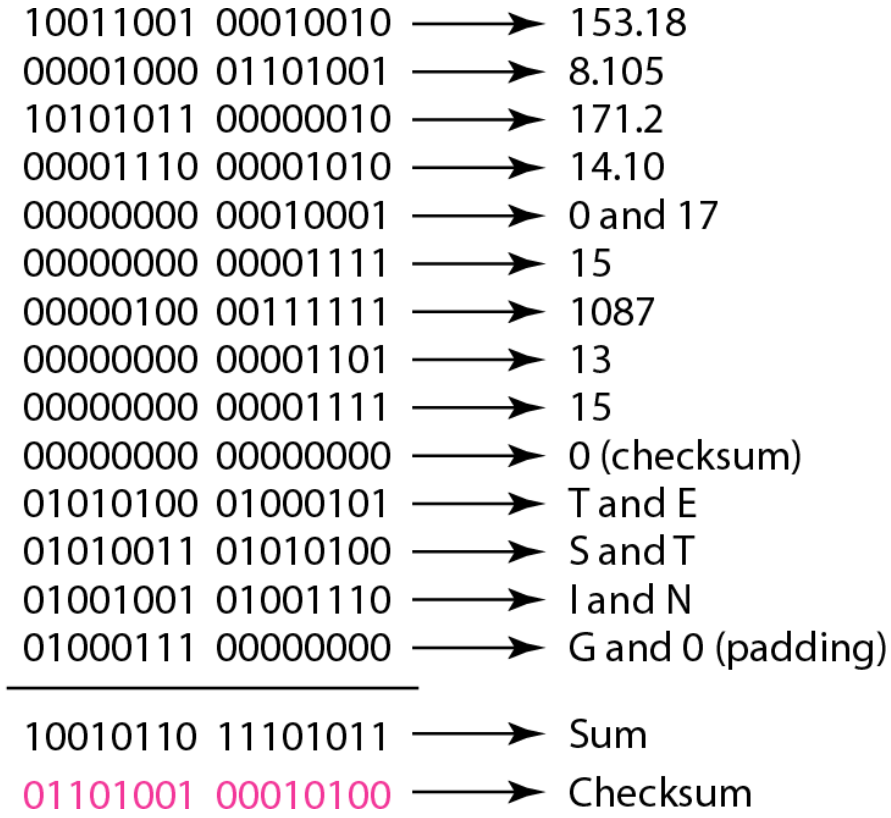
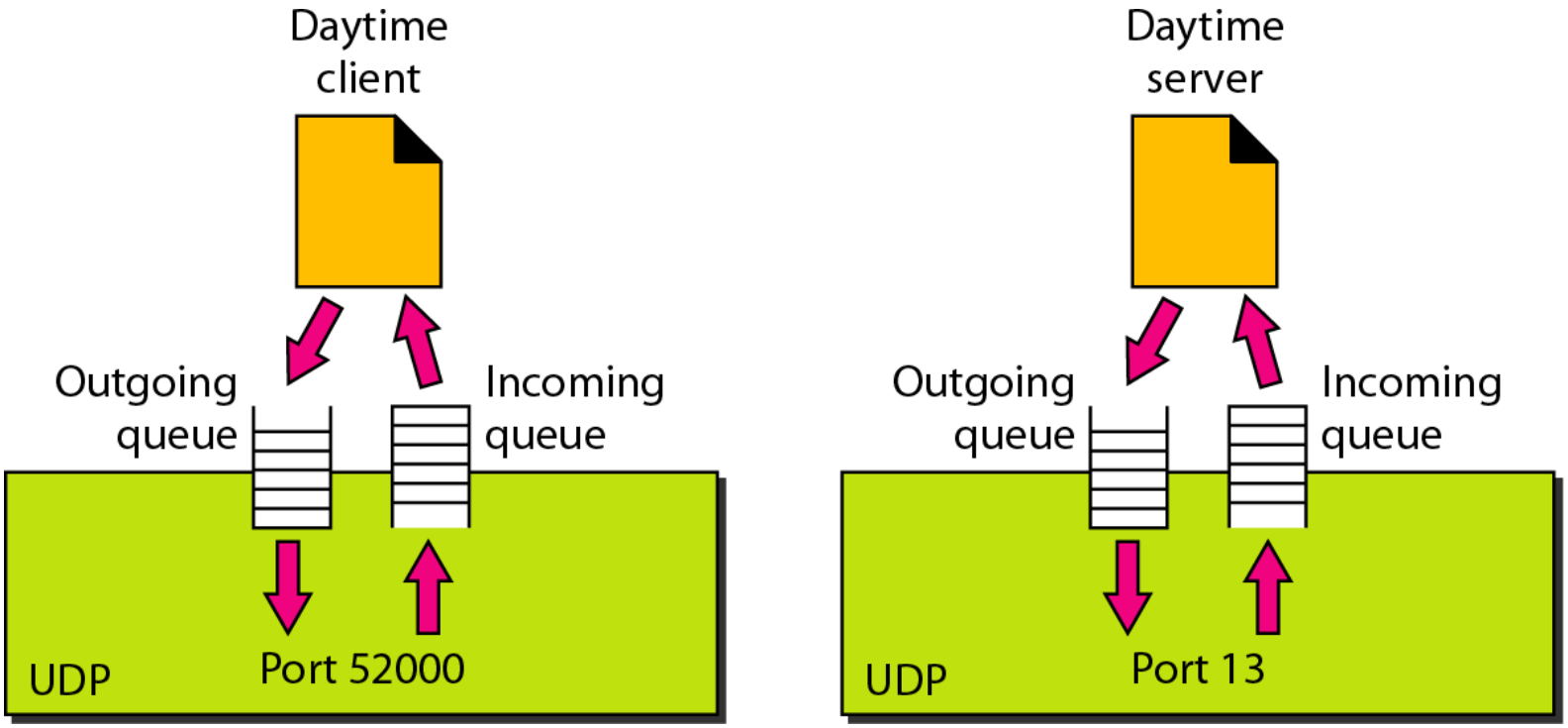


Figure 23.12 *Queues in UDP*



The key work was done by Birrell and Nelson (1984). In a nutshell, what Birrell and Nelson suggested was allowing programs to call procedures located on remote hosts. When a process on machine 1 calls a procedureon machine 2, the calling process on 1 is suspended and execution of the called procedure takes place on 2. Information can be transported from the caller to the callee in the parameters and can come back in the procedure result. No message passing is visible to the application programmer. This technique is known as **RPC** (**Remote Procedure Call).** Traditionally, the calling procedure is known as the client and the called procedure is known as the server, and we will use those names here too.

to call a remote procedure, the client program must be bound with a small library procedure, called the **client stub,** that represents the server procedure in the client's address space. Similarly, the server is bound with aprocedure called the **server stub.** These procedures hide the fact that the procedure call from the client to the server is not local

Step 1 is the client calling the client stub. This call is a local procedure call, with the parameters pushed onto the stack in the normal way.
Step 2 is the client stub packing the parameters into a message andmaking a system call to send the message. Packing the parameters is called **marshaling.**
Step 3 is the operating system sending the message from the client machine to the server machine.
Step 4 is the operating system passing the incoming packet to the server stub.
Finally, step 5 is the server stub calling the server procedure with the unmarshaled parameters.
The reply traces the same path in the other direction.

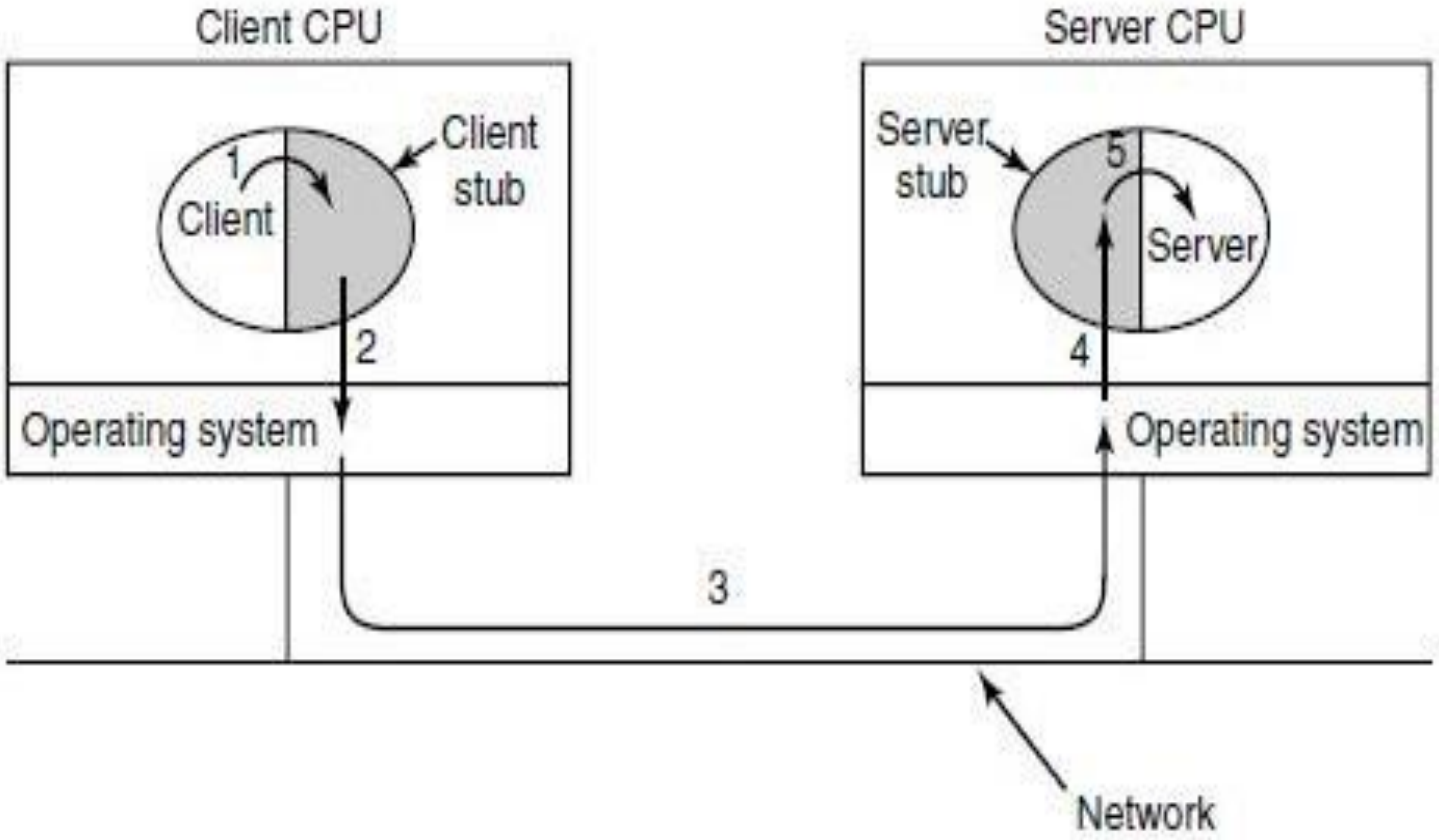


Figure 6-29. Steps in making a remote procedure call. The stubs are shaded.

Problems with RPC:

1 With RPC, passing pointers is impossible because the client and server are in different address spaces.
2 It is essentially impossible for the client stub to marshal the parameters: it has no way of determining how large they are.
3 A third problem is that it is not always possible to deduce the types of the parameters, not even from a formal specification or the code itself.(exa: printf)
4 A fourth problem relates to the use of global variables. Normally, the calling and called procedure can communicate by using global variables, in addition to communicating via parameters. But if the called procedure is moved to aremote machine, the code will fail because the global variables are no longer shared

TCP

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.

Topics discussed in this section:

TCP Services
TCP Features
Segment
A TCP Connection
Flow Control
Error Control

TCP Services

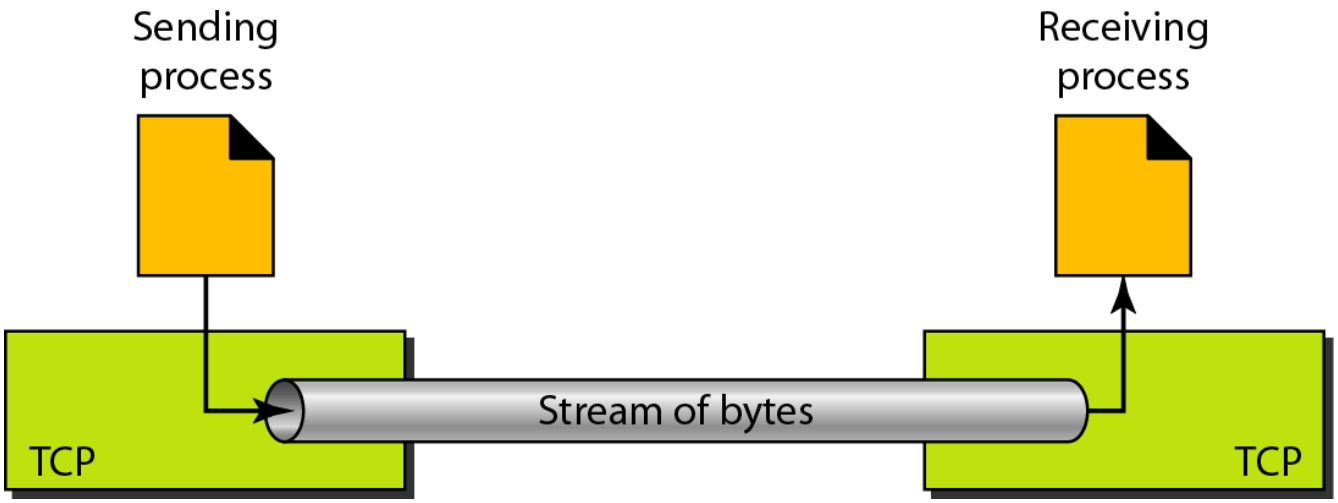
1 Process-to-Process Communication

TCP provides process-to-process communication using port numbers. Below Table lists some well-known port numbers used by TCP.

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

2 Stream Delivery Service

TCP, on the other hand, allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes. TCP creates an environment in which the two processes seem to be connected by an imaginary "tube“ that carries their dataacross the Internet. This imaginary environment is showed in below Figure. The sending process produces (writes to) the stream of bytes, andthe receiving process consumes (reads from) them



3 Sending and Receiving Buffers Because the sending and the receiving processes may not write or read data at the same speed, TCP needs buffers for storage. There are two buffers, the sending buffer and thereceiving buffer, one for each direction. One way to implement a buffer is to use a circular array of I-byte locations as shown in Figure. For simplicity, we have shown two buffers of 20 bytes each. Normally the buffers are hundreds or thousands of bytes, depending on the implementation. Wealso show the buffers as the same size, which is not always the case.

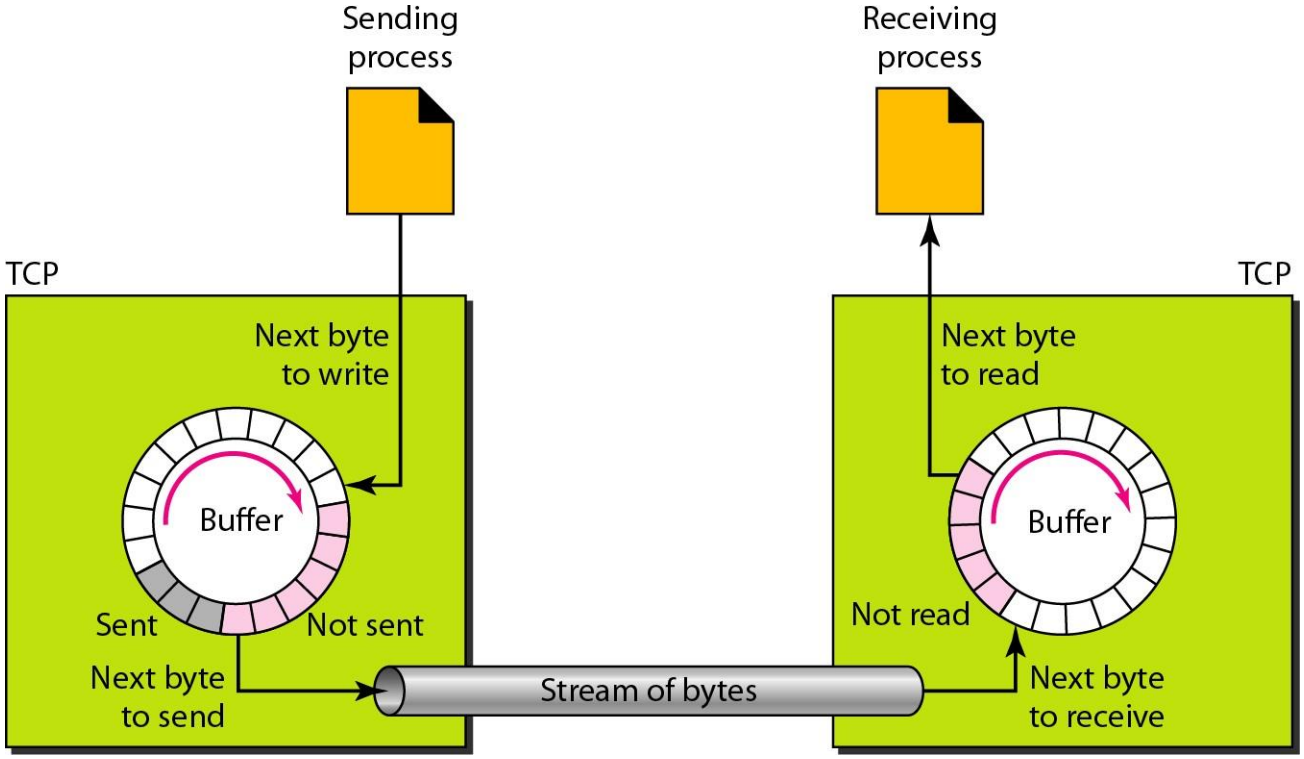
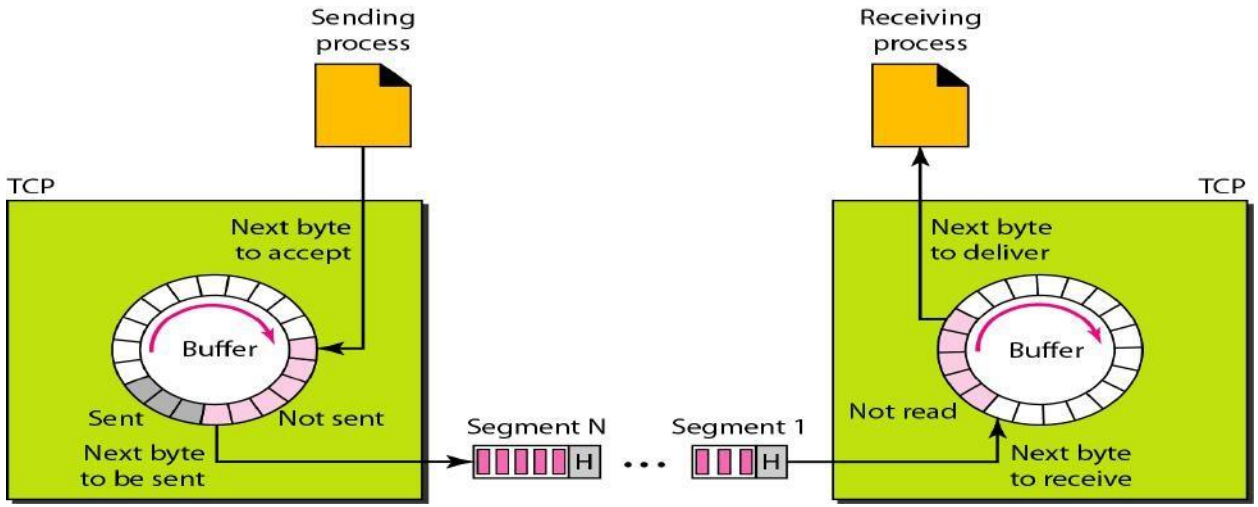


Figure shows the movement of the data in one direction. At the sending site, the buffer has three types of chambers. The white section contains empty chambers that can be filled by the sending process (producer). The gray area holds bytes that have been sent but not yet acknowledged. TCP keeps these bytes in the buffer until it receives an acknowledgment. The colored area contains bytes to be sent by the sending TCP.
However, as we will see later in this chapter, TCP may be able to send only part of this colored section. This could be due to the slowness of the receiving process or perhaps to congestion in the network. Also note that after the bytes in the gray chambers are acknowledged, the chambers are recycled and available for use by the sending process.
This is why we show a circular buffer.

The operation of the buffer at the receiver site is simpler. The circular buffer is divided into two areas (shown as white and colored). The white area contains empty chambers to be filled by bytes received from the network. The colored sections contain received bytes that can be read by the receiving process. When a byte is read by the receiving process, the chamber is recycled and added to the pool of empty chambers.

4 TCP segments



At the transport layer, TCP groups a number of bytes together into a packet called a segment. TCP adds a header to each segment (for control purposes) and delivers the segment to the IP layer for transmission. The segments are encapsulated in IP datagrams and transmitted.

This entire operation is transparent to the receiving process. Later we will see that segments may be received out of order, lost, or corrupted and resent. All these are handled by TCP with the receiving process unaware of any activities. Above fig shows how segments are created from the bytes in the buffers

5 Full-Duplex Communication

TCP offers full-duplex service, in which data can flow in both directions at the same time. Each TCP then has a sending and receiving buffer, and segments move in both directions

6 Connection-Oriented Service

TCP is a connection-oriented protocol. When a process at site A wants to send and receive data from another process at site B, the following occurs:

1. The two TCPs establish a connection between them.
2. Data are exchanged in both directions.
3. The connection is terminated.

7 Reliable Service

TCP is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe and sound arrival of data. We will discuss this feature further in the section on error control.

TCP Features

1 Numbering System

There are two fields called the sequence number and the acknowledgment number. These two fields refer to the byte number and not the segment number.

Byte Number The bytes of data being transferred in each connection are numbered by TCP. The numbering starts with a randomly generated number. For example, if the random number happens to be 1057 and the total data to be sent are 6000 bytes, the bytes are numbered from 1057 to 7056. We will see that byte numbering is used for flow and error control.

Sequence Number After the bytes have been numbered, TCP assigns a sequence number to each segment that is being sent. The sequence number for each segment is the number of the first byte carried in that segment.

Acknowledgment Number The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive. The acknowledgment number is cumulative.

2 Flow Control

TCP, provides *flow control. The receiver of the data controls the amount of* data that are to be sent by the sender. This is done to prevent the receiver from being overwhelmed with data. The numbering system allows TCP to use a byte-oriented flow control.

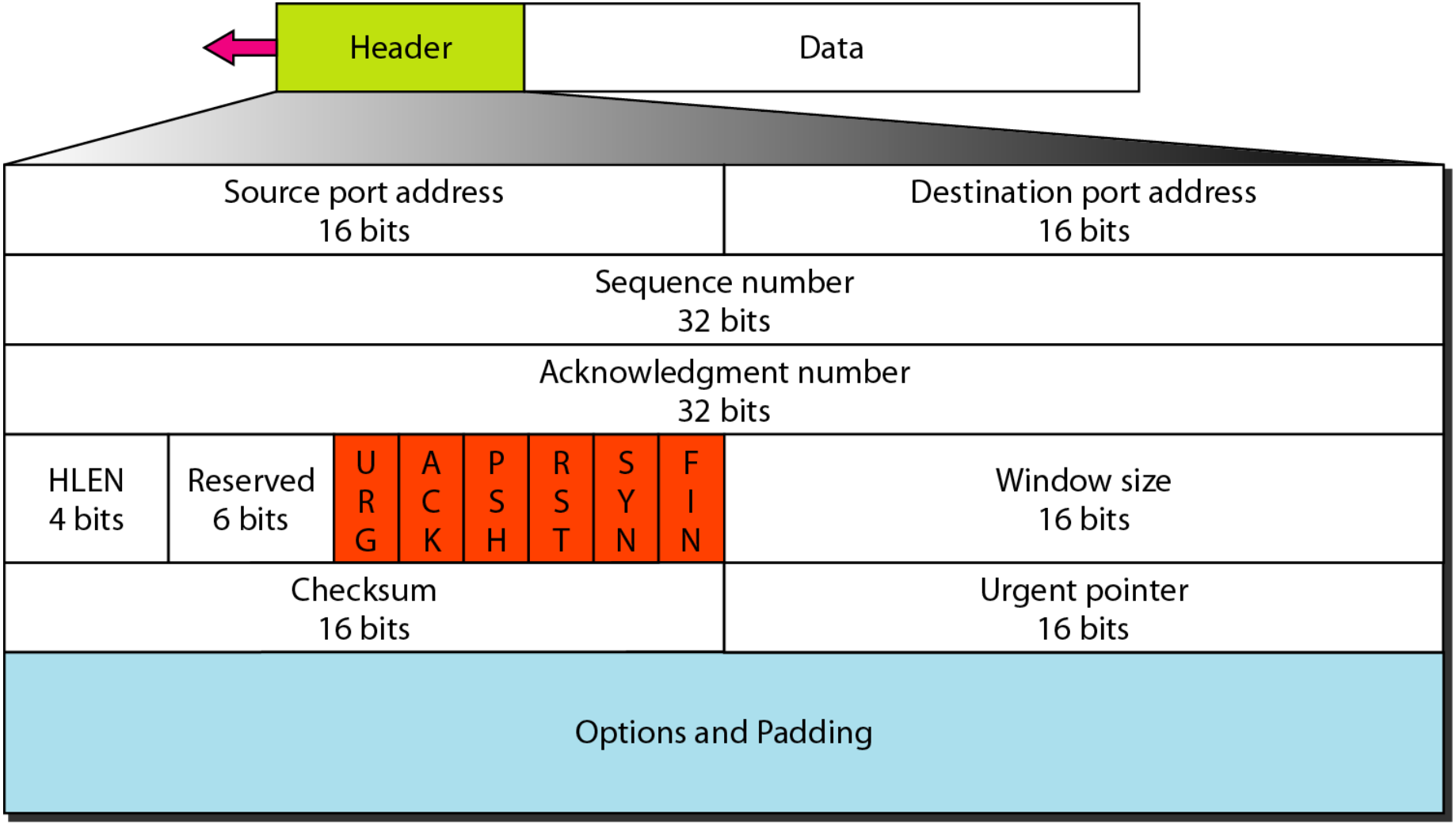
3 Error Control

To provide reliable service, TCP implements an error control mechanism. Although error control considers a segment as the unit of data for error detection (loss or corrupted segments), error control is byte-oriented, as we will see later.

4 Congestion Control

TCP takes into account congestion in the network. The amount of data sent by a sender is not only controlled by the receiver (flow control), but is also determined by the level of congestion in the network

TCP segment format



The segment consists of a 20- to 60-byte header,.

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.

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.

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 + I as the acknowledgment number. Acknowledgment and datacan be* piggybacked together.

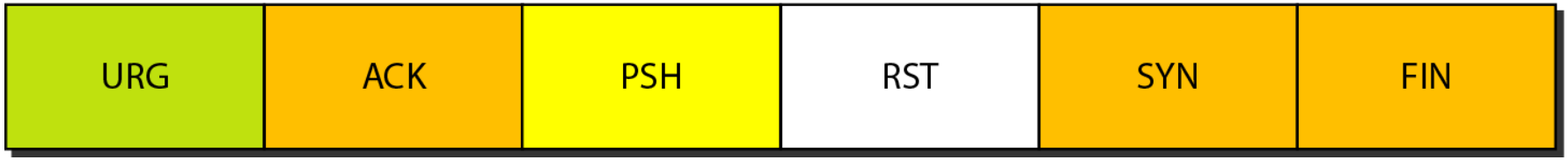
Header length. This 4-bit field indicates the number of 4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field can be between 5 (5 x 4 =20) and 15 (15 x 4 =60).

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.

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push

RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection



These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.

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 l6-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. We will not discuss these options here; please refer to the reference list for more information.

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.

In TCP, connection-oriented transmission requires three phases:

1. connection establishment,
2. data transfer,
3. connection termination.

TCP connection establishment(3 way handshaking)

1 The client sends the first segment, a SYN segment, in which only the SYN flag is set.

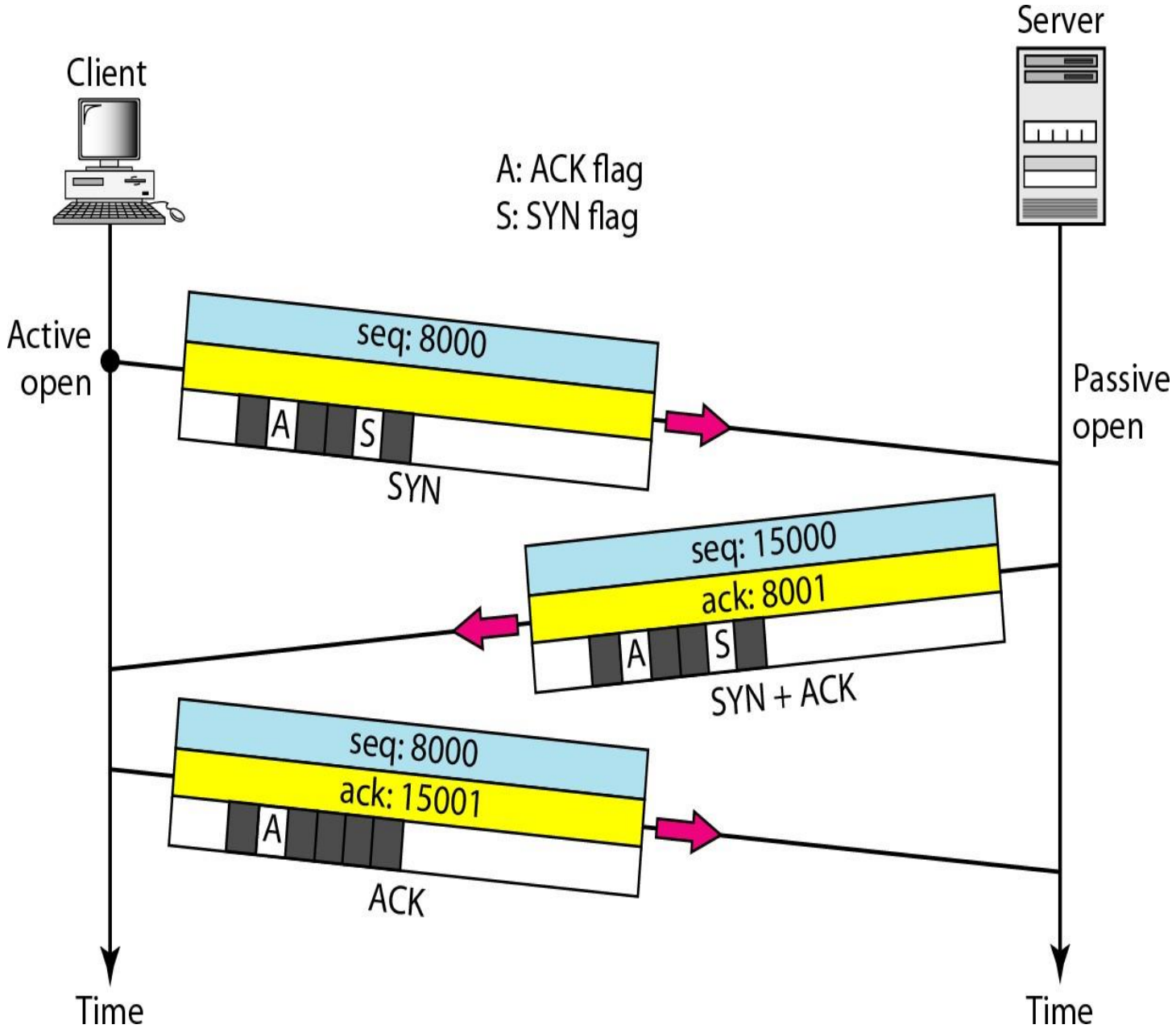
NOTE:A SYN segment cannot carry data, but it consumes one sequence number.

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.

NOTE:A SYN+ACK segment cannot carry data, but does consumeone 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.

NOTE: An ACK segment, if carrying no data, consumes no sequencenumber


Client
Server
A: ACK flag
S: SYN flag
Active open
Passive open
seq: 8000
A
S
SYN
seq: 15000
ack: 8001
A
S
SYN + ACK
seq: 8000
ack: 15001
A
ACK
Time
Time

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 corning from a different client by faking the source IP addresses in the datagram's.

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 anddenies service to every request.

SOLUTIONS:

1 Some have imposed a limit on connection requests during a specified period of
time.

2 Others filter out datagrams coming from unwanted source addresses.

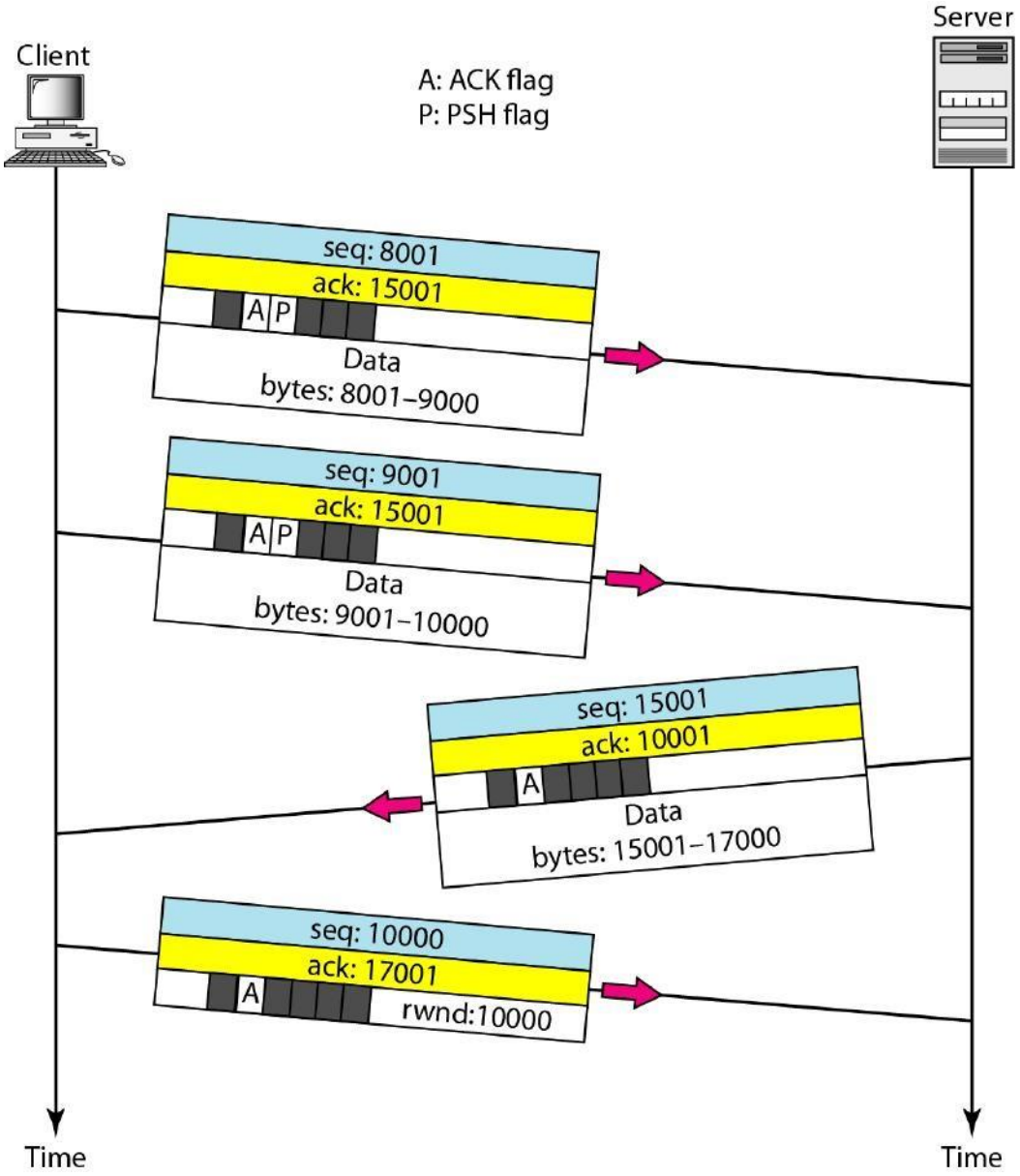
3 One recent strategy is to postpone resource allocation until the entire connection is set up, using what is called a cookie.

Data Transfer

After connection is established, bidirectional data transfer can take place. The client and server can both send data and acknowledgments. Data traveling in the same direction as an acknowledgment are carried on the same segment. The acknowledgment is piggybacked with the data

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 lastsegment carries only an acknowledgment because there are no more datato 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.

Data transfer



PUSHING DATA: 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.

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, sending application program wants a piece of data to be read out of order by the receiving application program.

Connection Termination *(*three-way handshaking and four-way handshaking with a half-close option.)

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 theFIN flag is set.
Note that a FIN segment can include the last chunk of data sent by the client, or it can be just a control segment as shown in Figure. If it is only a control segment, it consumes only one sequence number.
NOTE: The FIN segment consumes one sequence number ifit doesnot carry data.

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.
NOTE: The FIN +ACK segment consumes one sequence number ifit does not carry data.

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.

Half-Close In TCP, one end can stop sending data while still receiving data. This is called a half-close. Although either end can issue a half-close, it is normally initiated by the client. It can occur when the server needs all the data before processing can begin.

A good example is sorting. When the client sends data to the server to be sorted, the server needs to receive all the data before sorting can start. This means the client, after sending all the data, can close the connection in the outbound direction. However, the inbound direction must remain open to receive the sorted data. The server, after receiving the data, still needs time for sorting; its outbound direction must remain open

Connection termination using three-way handshaking

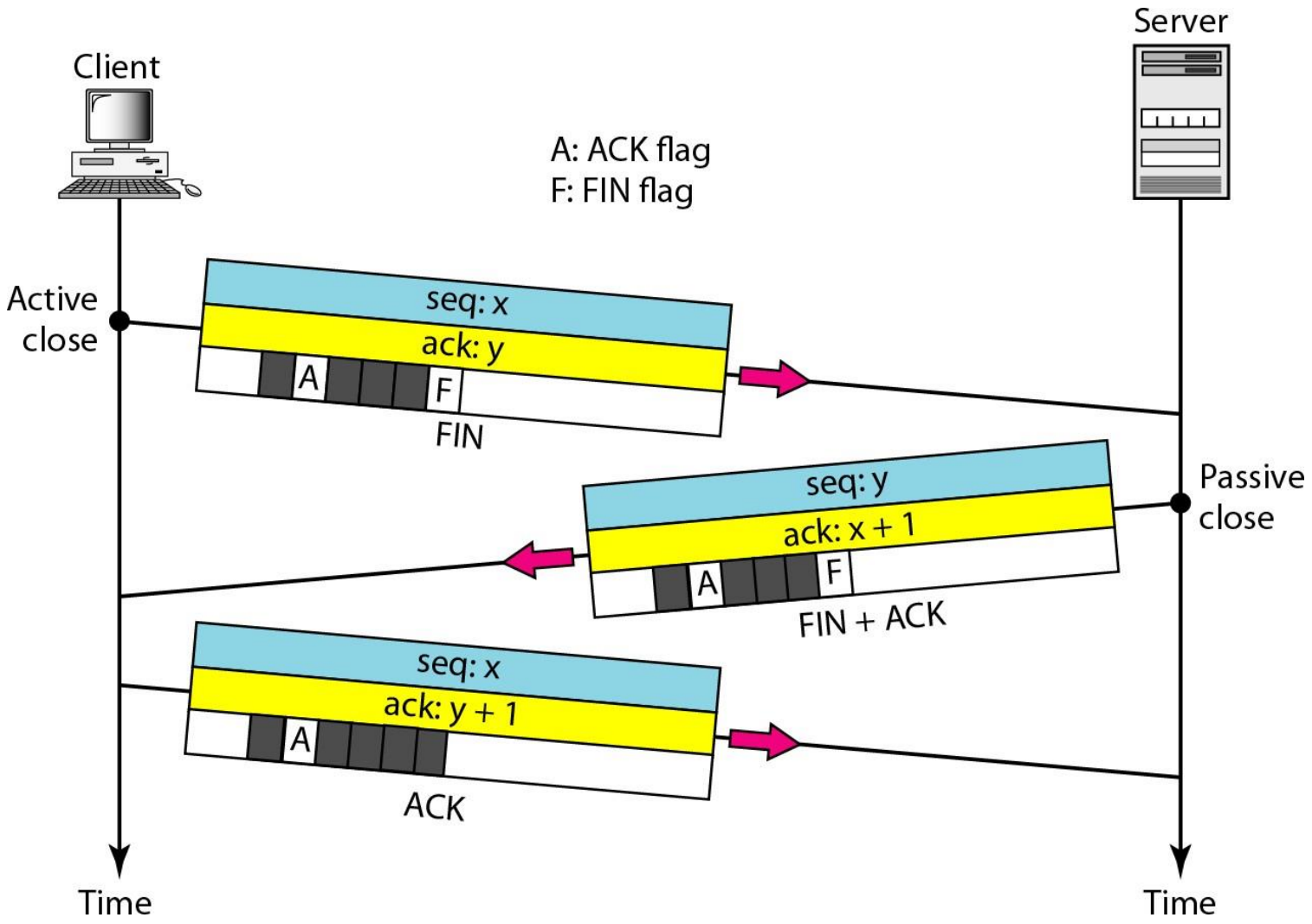
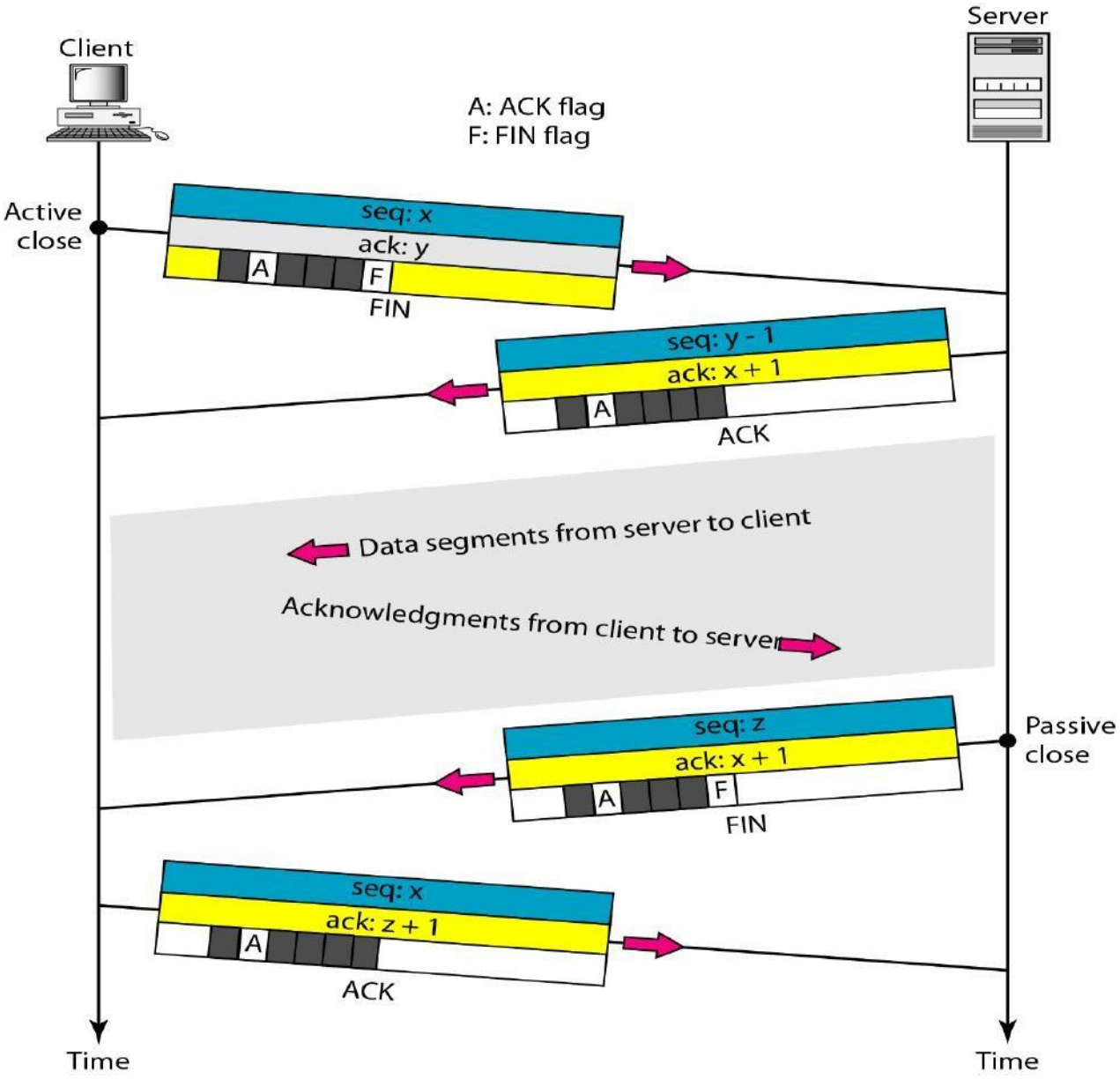


Figure 23.21 *Half-close*



Flow Control or TCP Sliding Window

TCP uses a sliding window, to handle flow control. The sliding window protocol used by TCP, however, is something between the *Go-Back-N* and Selective Repeat sliding window.

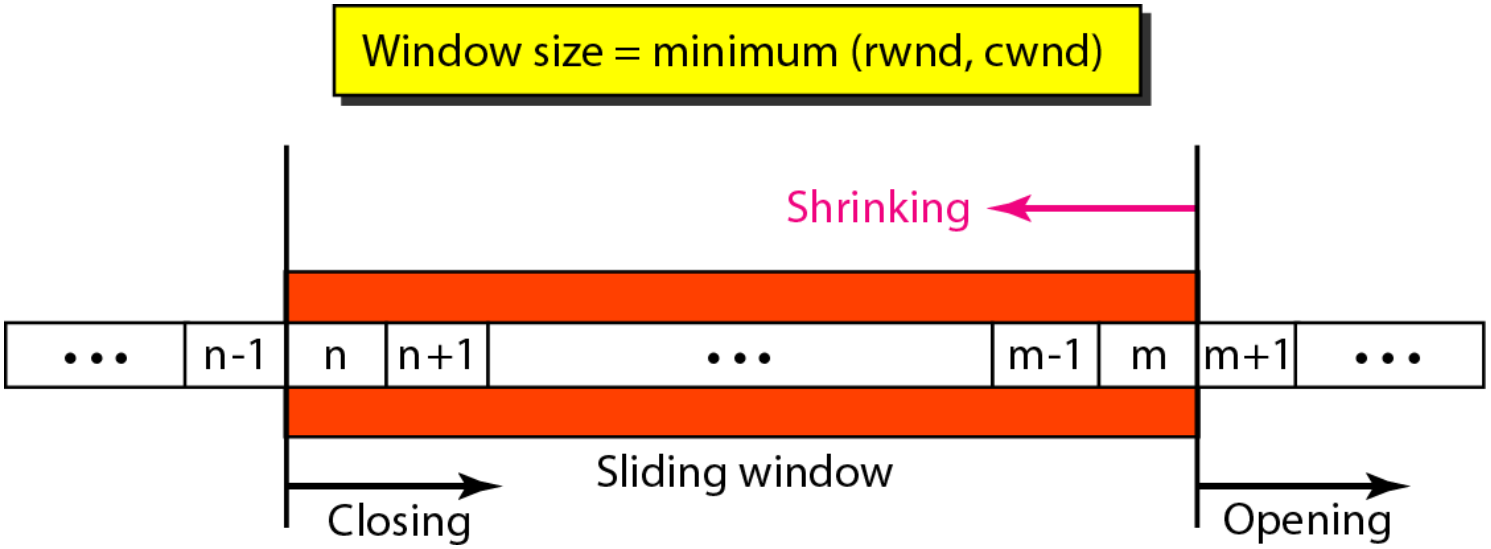
The sliding window protocol in TCP looks like the Go-Back-N protocol because it does not use NAKs;
it looks like Selective Repeat because the receiver holds the out-of-order segments until the missing ones arrive.

There are two big differences between this sliding window and the one we used at the data link layer.

1 the sliding window of TCP is byte-oriented; the one we discussed in the data link layer is frame-oriented.

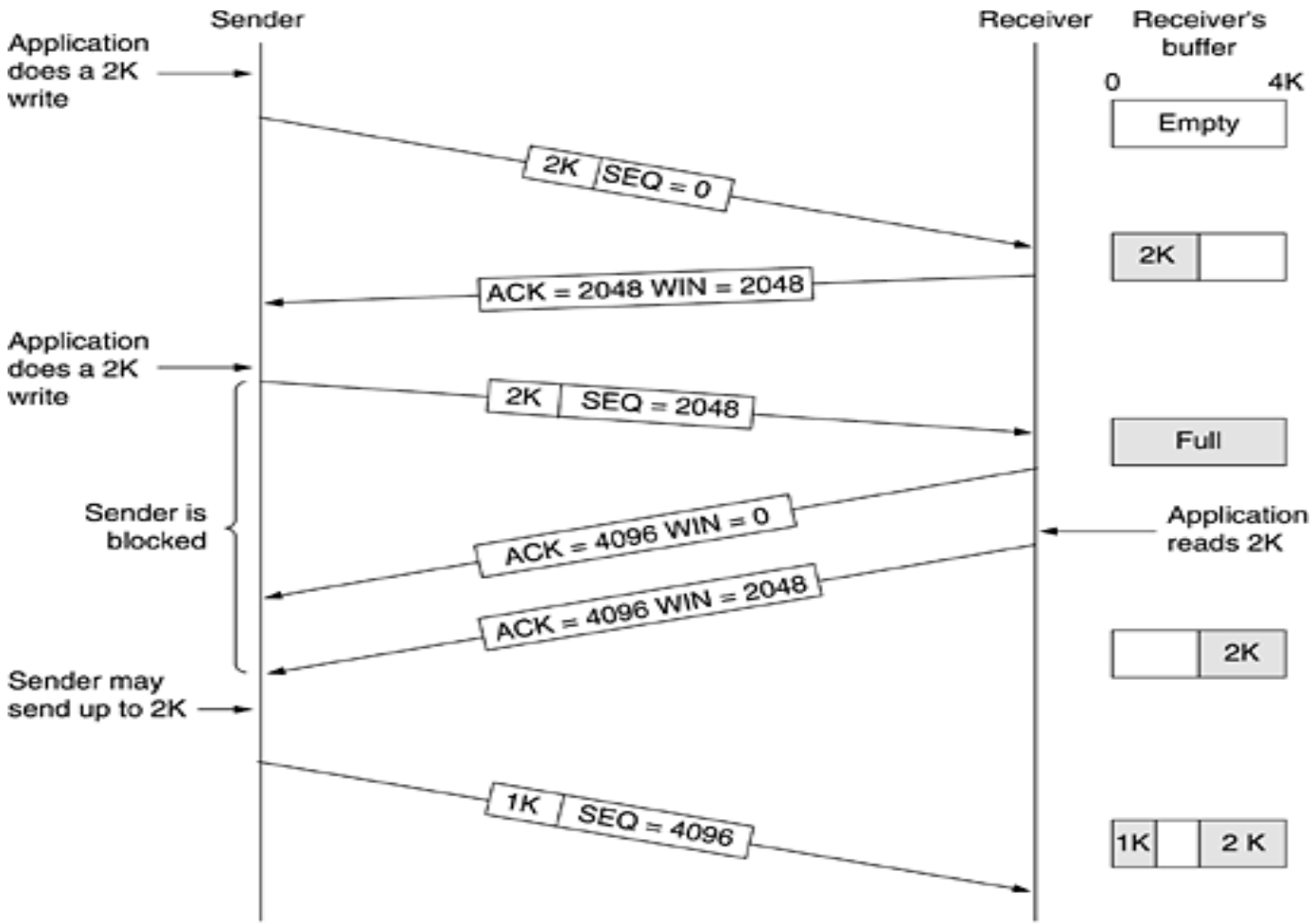
2 the TCP's sliding window is of variable size; the one we discussed in the data link layer was of fixed size

Sliding window



The window is *opened, closed, or shrunk*. *These three activities, as we will see, are* in the control of the receiver (and depend on congestion in the network), not the sender.
The sender must obey the commands of the receiver in this matter.
Opening a window means moving the right wall to the right. This allows more new bytes in the buffer that are eligible for sending.
Closing the window means moving the left wall to the right. This means that some bytes have been acknowledged and the sender need not worry about them anymore.
Shrinking the window means moving the right wall to the left.

The size of the window at one end is determined by the lesser of two values: *receiver window (rwnd) or congestion window (cwnd).*
The receiver window is the value advertised by the opposite end in a segment containing acknowledgment. It is the number of bytes the other end can accept before its buffer overflows and data are discarded.
The congestion window is a value determined by the network to avoid congestion



Window management in TCP

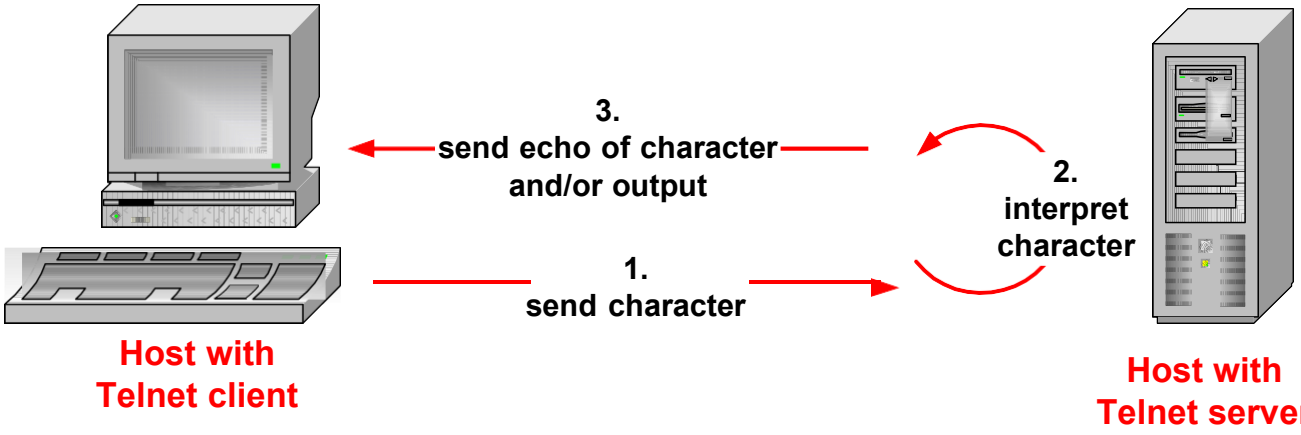
When the window is 0, the sender may not normally send segments, with two exceptions.

1) urgent data may be sent, for example, to allow the user to kill the process running on the remote machine.
2) the sender may send a 1-byte segment to force the receiver to reannounce the next byte expected and the window size. This packet is called a **window probe.**

The TCP standard explicitly provides this option to prevent deadlock if a window update ever gets lost.

Senders are not required to transmit data as soon as they come in from the application. Neither are receivers required to send acknowledgements as soon as possible.

For example, in Fig. when the first 2 KB of data came in, TCP, knowing that it had a 4-KB window, would have been completely correct in just buffering the data until another 2 KB came in, to be able to transmit a segment with a 4-KB payload. This freedom can be used to improve performance



Remote terminal applications (e.g., Telnet) send characters to a server.

The server interprets the character and sends the output at the server to the client.

For each character typed, you see three packets:

Client □ Server: Send typed character

Server □ Client: Echo of character (or user output) and acknowledgement for first packet

Client □ Server: Acknowledgement for second packet

Delayed Acknowledgement

- TCP delays transmission of ACKs for up to 500ms

- Avoid to send ACK packets that do not carry data.
  - The hope is that, within the delay, the receiver will have data ready to be sent to the receiver. Then, the ACK can be piggybacked with a data segment

Exceptions:

- ACK should be sent for every full sized segment
- Delayed ACK is not used when packets arrive out of order

Although delayed acknowledgements reduce the load placed on the network by the receiver, a sender that sends multiple short packets (e.g., 41-byte packets containing 1 byte of data) is still operating inefficiently. A way to reduce this usage is known as **Nagle's algorithm (Nagle, 1984).**

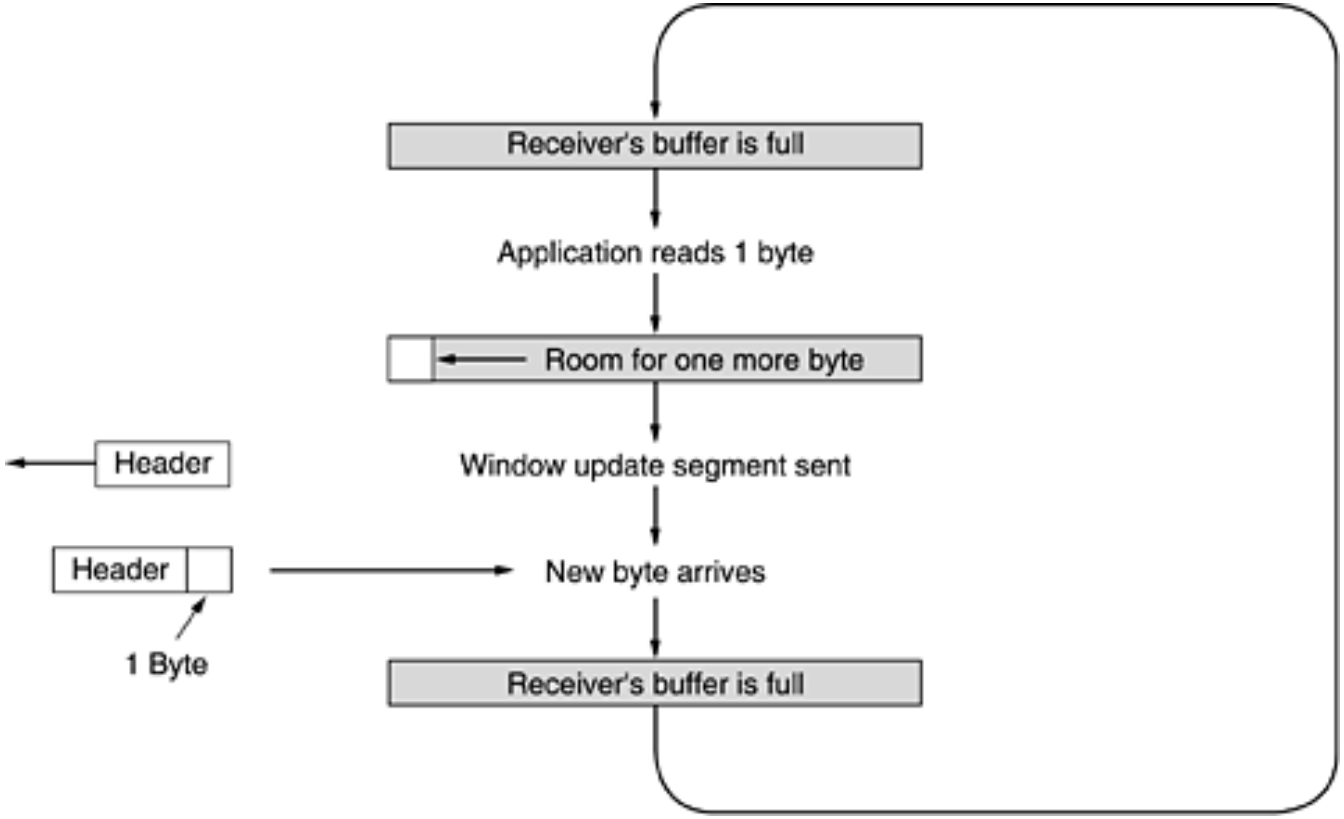
Nagel's Rule

Send one byte and buffer all subsequent bytes until acknowledgement is received. Then send all buffered bytes in a single TCP segment and start buffering again until the sent segment is acknowledged.
Nagle's algorithm will put the many pieces in one segment, greatly reducing the bandwidth used

Nagle's algorithm is widely used by TCP implementations, but there are times when it is better to disable it. In particular, in interactive games that arerun over the Internet.
A more subtle problem is that Nagle's algorithm can sometimes interact with delayed acknowledgements to cause a temporary deadlock: the receiver waits for data on which to piggyback an acknowledgement, and the sender waits on the acknowledgement to send more data.

Because of these problems, Nagle's algorithm can be disabled (which is called the *TCP NODELAY option).*

Another problem that can degrade TCP performance is the **silly window syndrome** (Clark, 1982).



Clark's solution is to prevent the receiver from sending a window update for 1 byte. Instead, it is forced to wait until it has a decent amount of space available and advertise that instead. Specifically, the receiver should not send a window update until it can handle the maximum segment size it advertised when the connection was established or until its buffer is half empty, whichever is smaller.

Furthermore, the sender can also help by not sending tiny segments. Instead, it should wait until it can send a full segment, or at least one containing half of the receiver's buffer size.

The goal is for the sender not to send small segments and the receiver not to ask for them. (Nagel + Clark). Both are used to improve TCP performance

The receiver will buffer the data until it can be passed up to the application in order (handling out of order segments)

Cumulative acknowledgements

Error Control

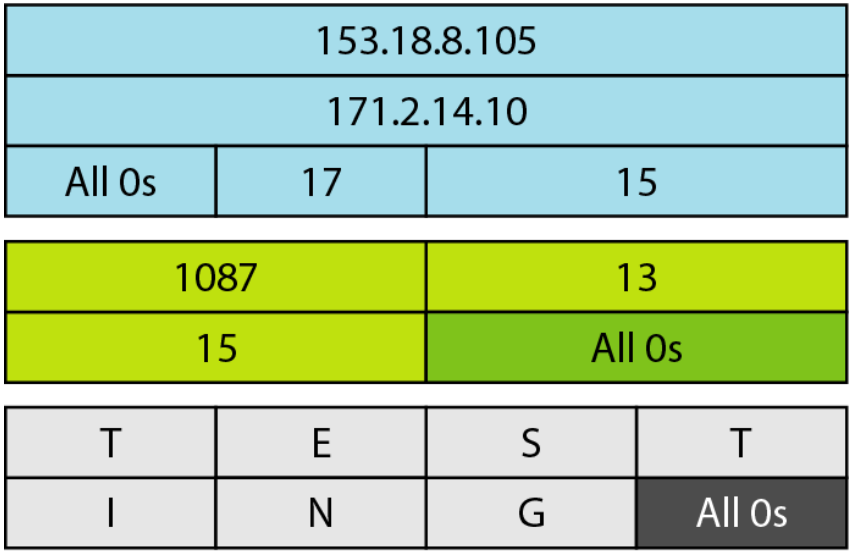
TCP is a reliable transport layer protocol. This means that an application program that delivers a stream of data to TCP relies on TCP to deliver the entire stream to the application program on the other end in order, without error, and without any part lost or duplicated.

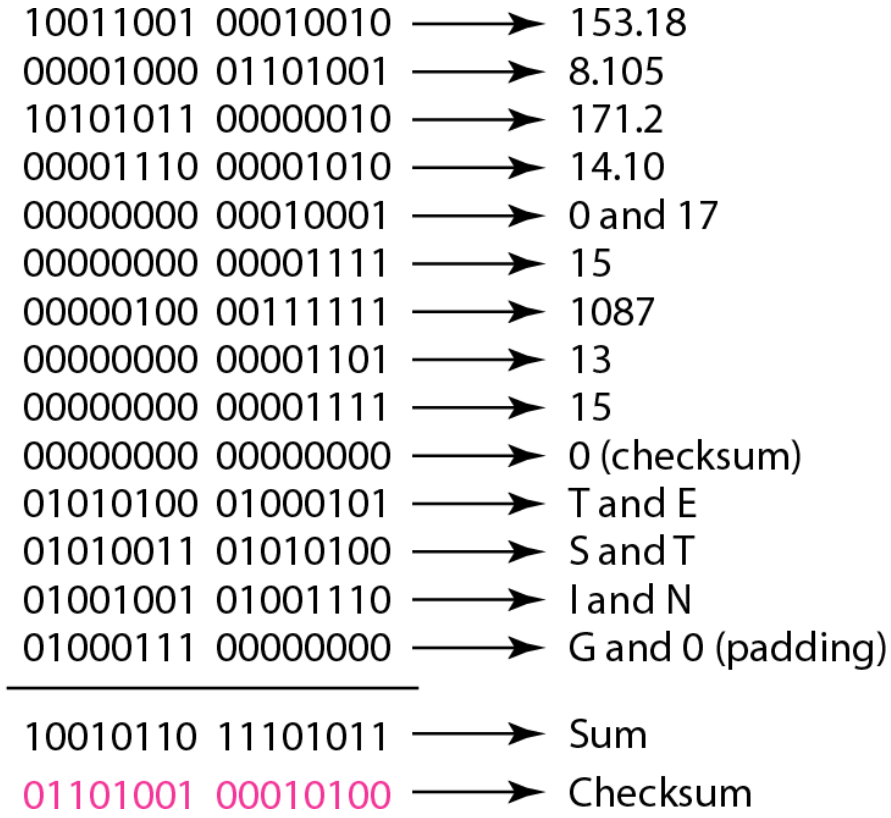
TCP provides reliability using error control. Error control includes mechanisms for detecting corrupted segments, lost segments, out-of-order segments, and duplicated segments. Error control also includes a mechanism for correcting errors after they are detected. Error detection and correction in TCP is achieved through the use of three simple tools: **checksum, acknowledgment, and time-out.**

Checksum

Each segment includes a checksum field which is used to check for acorrupted segment. If the segment is corrupted, it is discarded by the destination TCP and is considered as lost. TCP uses a 16-bit checksum that is mandatory in every segment

Figure 23.11 *Checksum calculation of a simple UDP user datagram*





Acknowledgment

TCP uses acknowledgments to confirm the receipt of data segments.Control segments that carry no data but consume a sequence number are also acknowledged. ACK segments are never acknowledged.

ACK segments do not consume sequence numbers and are not acknowledged.

Retransmission

The heart of the error control mechanism is the retransmission of segments. When a segment is corrupted, lost, or delayed, it isretransmitted.

In modern implementations, a retransmission occurs if the retransmission timer expires or three duplicate ACK segments have arrived.

Retransmission After RTO (retransmission time out)

Retransmission After Three Duplicate ACK Segments (also called fast retransmission)

Out-of-Order Segments

Data may arrive out of order and be temporarily stored by the receiving TCP, but yet guarantees that no out-of-order segment is delivered to the process

When the load offered to any network is more than it can handle, congestion builds up.
The network layer detects congestion when queues grow large at routers and tries to manage it, if only by dropping packets. It is up to the transport layer to receive congestion feedback from the network layer and slow down the rate of traffic that it is sending into the network.

For Congestion control, transport protocol uses an AIMD (Additive Increase Multiplicative Decrease) control law.

TCP congestion control is based on implementing this approach using a window called **congestion window.** TCP adjusts the size of the window according to the AIMD rule.

The window size at the sender is set as follows:

Send Window = MIN (flow control window,congestion window)

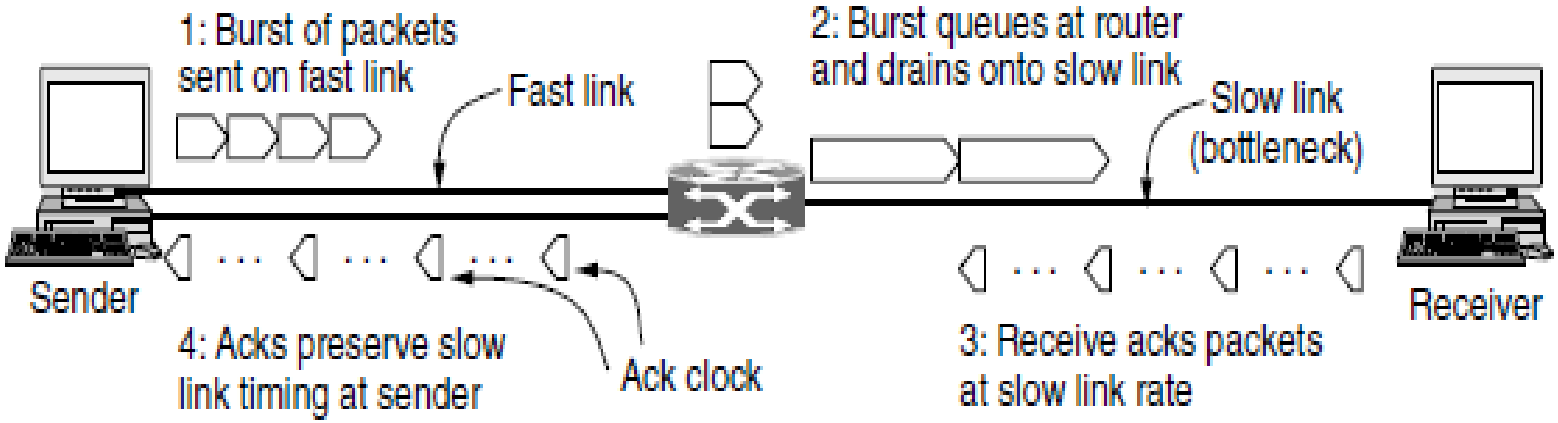
where

flow control window is advertised by the receiver (rwnd) congestion window is adjusted based on feedback from the

Modern congestion control was added to TCP largely through the efforts of Van Jacobson (1988). It is a fascinating story. Starting in 1986, the growing popularity of the early Internet led to the first occurrence of what became known as a **congestion collapse**, a prolonged period during which good put dropped suddenly (i.e., by more than a factor of 100) due to congestion in the network. Jacobson (and many others) set out to understand whatwas happening and remedy the situation.

To start, he observed that packet loss is a suitable signal of congestion. This signal comes a little late (as the network is already congested) but it is quite dependable

At the beginning how sender knows at what speed receiver can receive the packets?

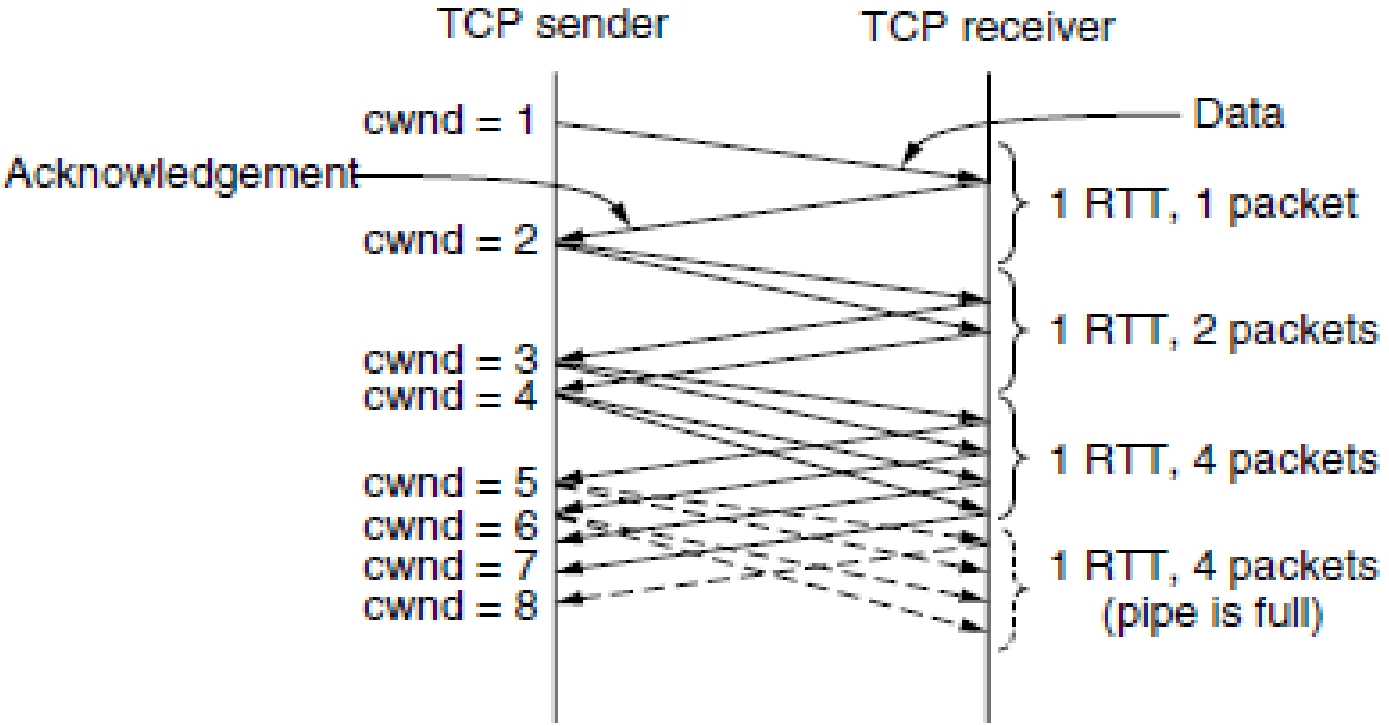


The key observation is this: the acknowledgements return to the sender at about the rate that packets can be sent over the slowest link in the path. This is precisely the rate that the sender wants to use. If it injects new packets into the network at this rate, they will be sent as fast as the slow link permits, but they will not queue up and congest any router along the path. This timing is known as an **ack clock**. It is an essential part of TCP. By using an ack clock, TCP smoothes out traffic and avoids unnecessary queues at routers. This is first consideration

A second consideration is that the AIMD rule will take a very long time to reach a good operating point on fast networks if the congestion window is started from a small size

Instead, the solution Jacobson chose to handle both of these considerations is a mix of linear and multiplicative increase.

SLOW-START



TCP Congestion Control

Slow Start

- Additive Increase / Multiplicative Decrease is only suitable for source, that is operating close to the available capacity of the network, but it takes too long to ramp up a connection when it is starting from scratch.
- slow start, that is used to increase the congestion window rapidly from a cold start.
- Slow start effectively **increases the congestion window exponentially**, rather than linearly.
  - the source starts out by setting CongestionWindow to one packet.
  - When the ACK for this packet arrives, TCP adds 1 to CongestionWindow and then sends two packets.
  - Upon receiving the corresponding two ACKs, TCP increments CongestionWindow by 2—one for each ACK—and next sends four packets.
  - The end result is that TCP effectively doubles the number of packets it has in transit every RTT.

Whenever a packet loss is detected, for example, by a timeout, the slow start threshold is set to be half of the congestion window and the entire process is restarted.

Congestion avoidance phase is started if cwnd has reached the slow start threshold value

Whenever the slow start threshold is crossed, TCP switches from slow start to additive increase. In this mode, the congestion window is increased by one segment every round-trip time.

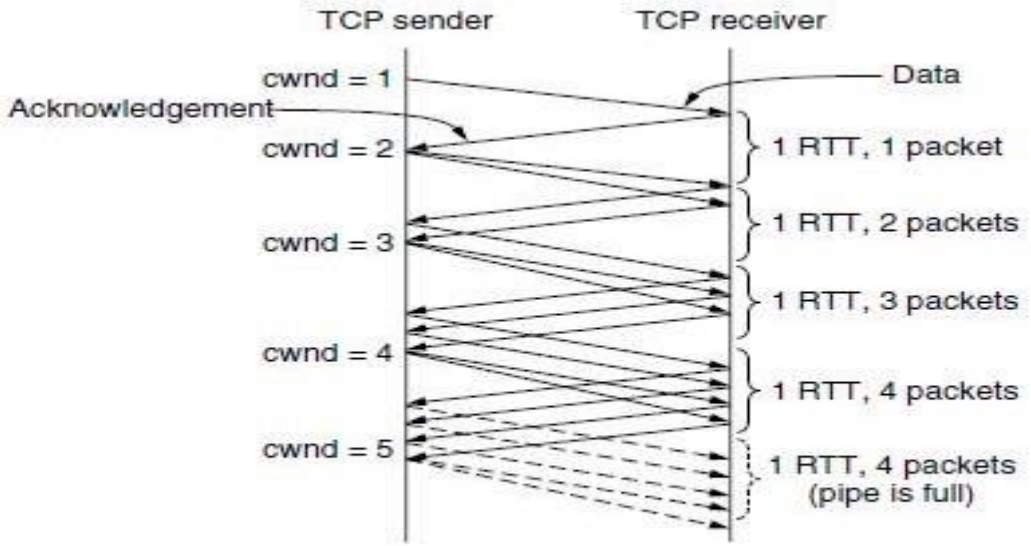


Figure 6-45. Additive increase from an initial congestion window of one segment.

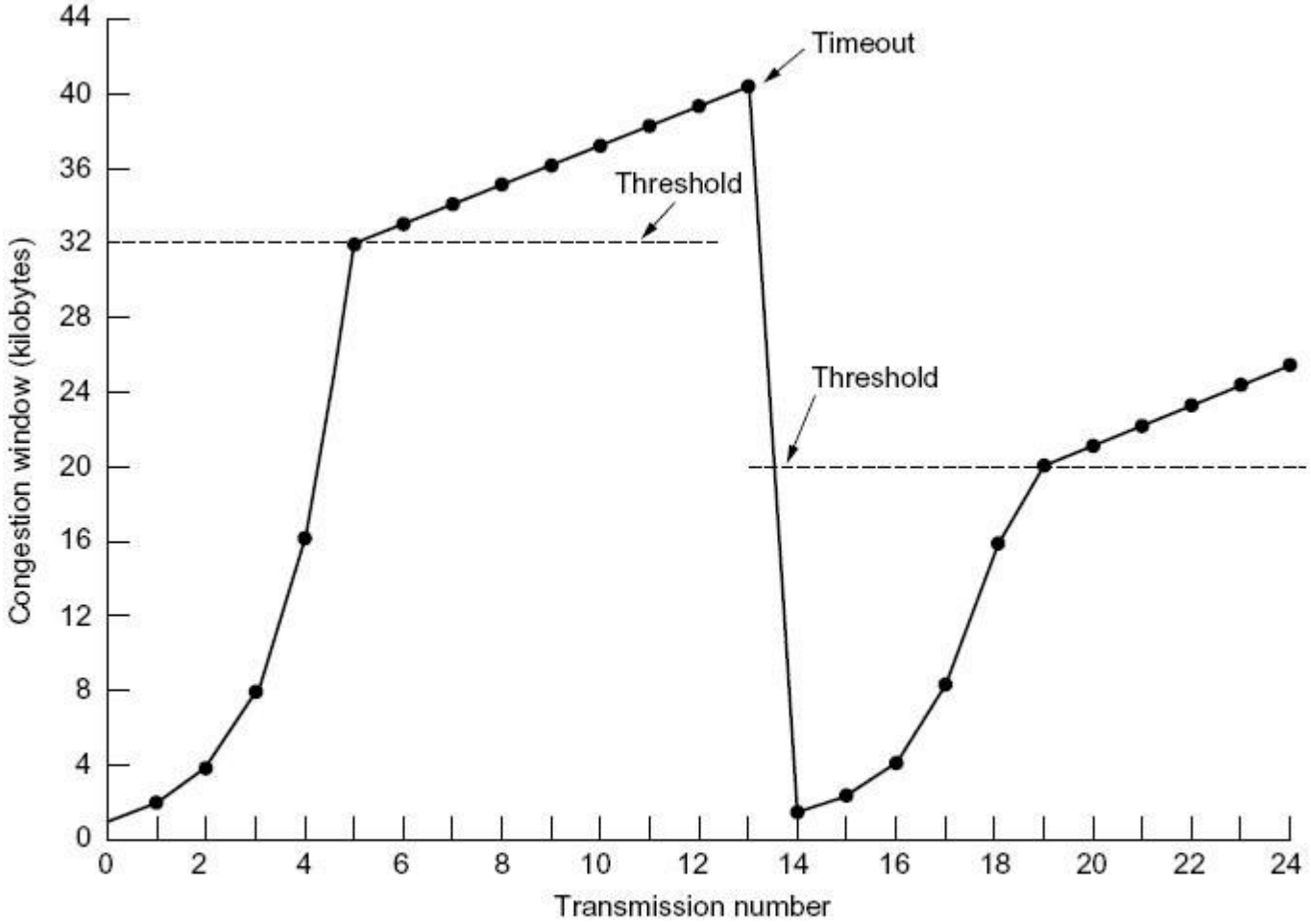


Fig. 6-37. An example of the Internet congestion algorithm.

Responses to Congestion

- So, TCP assumes there is congestion if it detects a packet loss
- A TCP sender can detect lost packets via:
  - Timeout of a retransmission timer
  - Receipt of a duplicate ACK

- TCP interprets a Timeout as a binary congestion signal. When a timeout occurs, the sender performs:
  - cwnd is reset to one:

    cwnd = 1
  - ssthresh is set to half the current size of the congestion window:

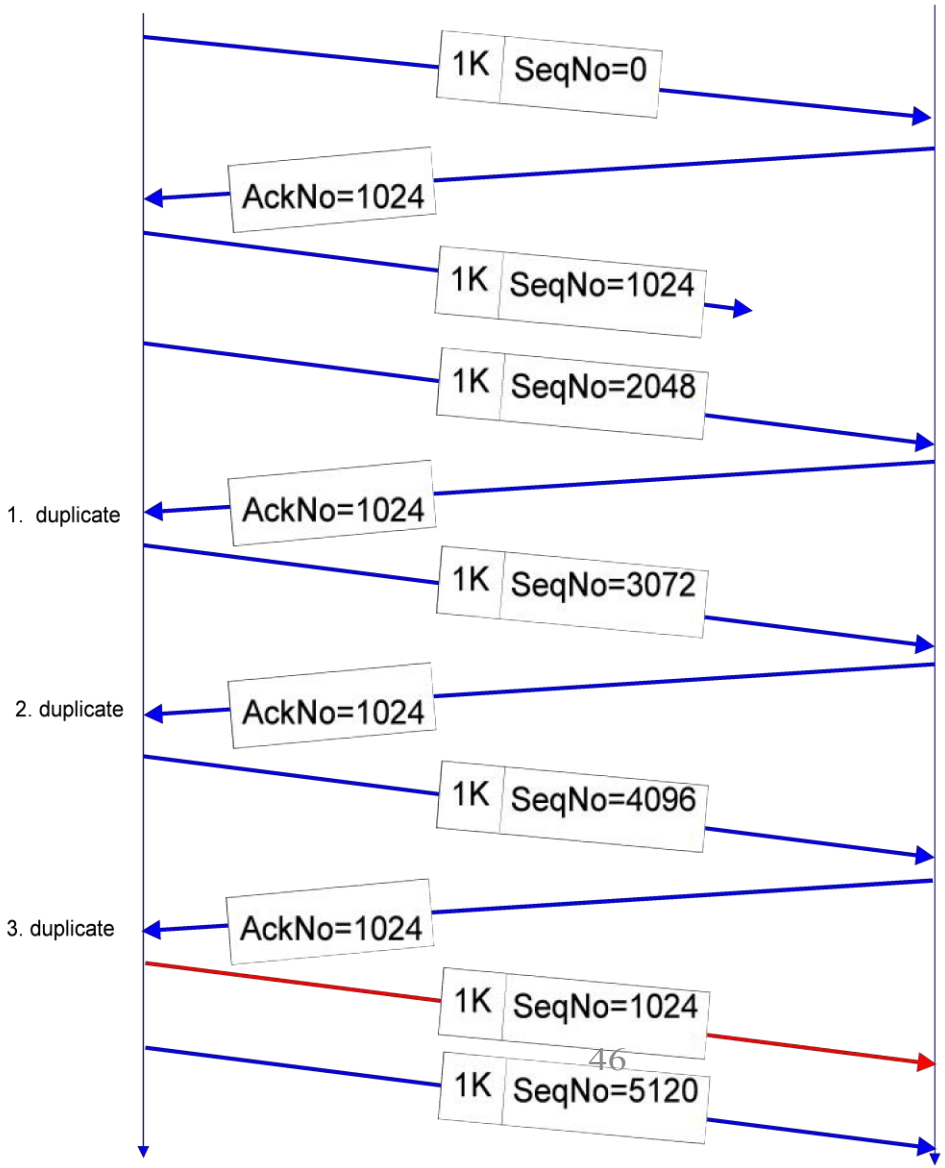
    ssthressh = cwnd / 2
  - and slow-start is entered

Fast Retransmit

- If three or more duplicate ACKs are received in a row, the TCP sender believes that a segment has been lost.
- Then TCP performs a retransmission of what seems to be the missing segment, without waitingfor a timeout to happen.
- Enter slow start:

  ssthresh = cwnd/2

  cwnd = 1



Flavors of TCP Congestion

Control

- **TCP Tahoe** (1988)
  - Slow Start
  - Congestion Avoidance
  - Fast Retransmit
- **TCP Reno** (1990) (TCP Tahoe+FR)
  - Fast Recovery
- **New Reno** (1996)
- **SACK** (1996) (**SACK (Selective ACKnowledgements)**)

- **RED** (Floyd and Jacobson 1993)

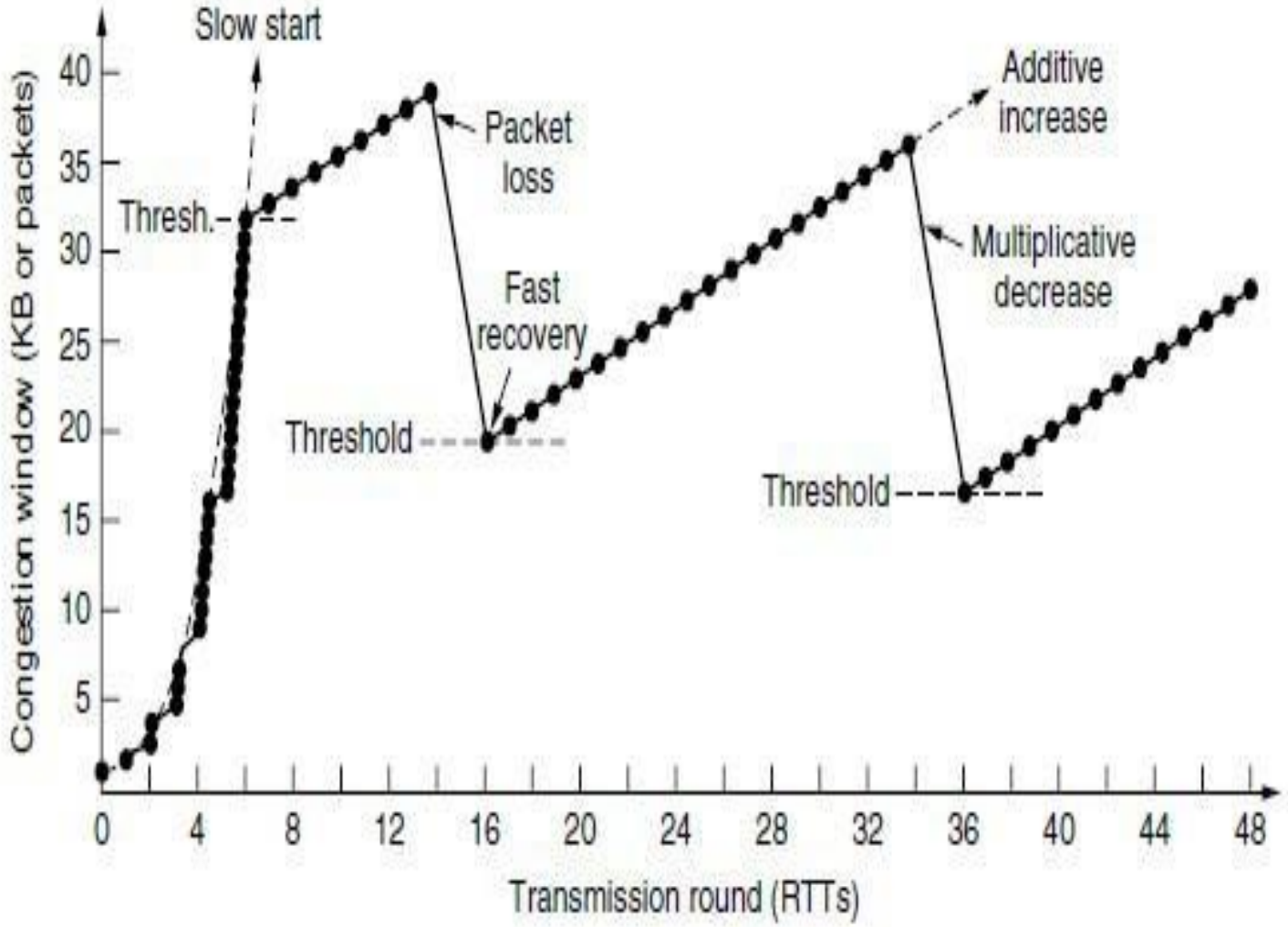


Figure 6-47. Fast recovery and the sawtooth pattern of TCP Reno.

The use of ECN (Explicit Congestion Notification) in addition to packet loss as a congestion signal. ECN is an IP layer mechanism to notify hosts of congestion.

The sender tells the receiver that it has heard the signal by using the *CWR (Congestion Window Reduced) flag.*

USER DATAGRAM PROTOCOL (UDP)

The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It doesnot add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication.

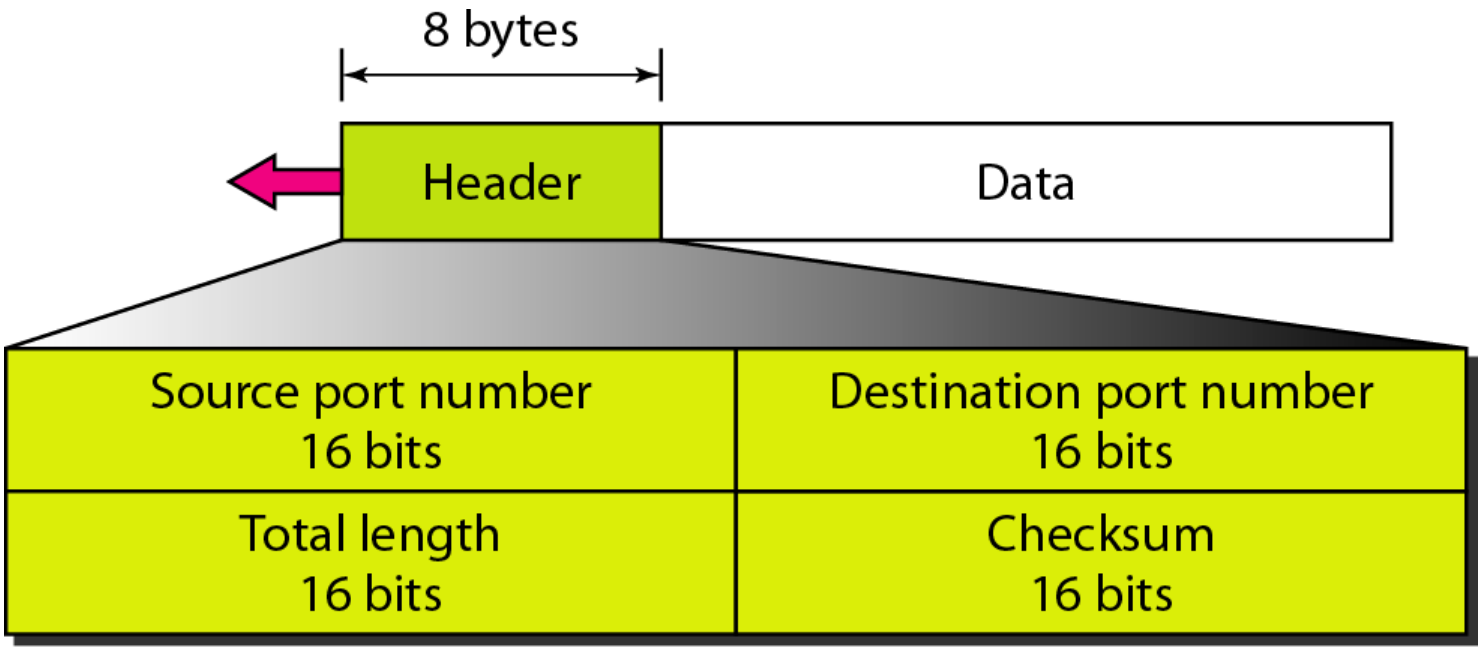
Topics discussed in this section:

Well-Known Ports for UDP
User Datagram
Checksum
UDP Operation
Use of UDP

Table 23.1 *Well-known ports used with UDP*

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

Figure 23.9 *User datagram format*

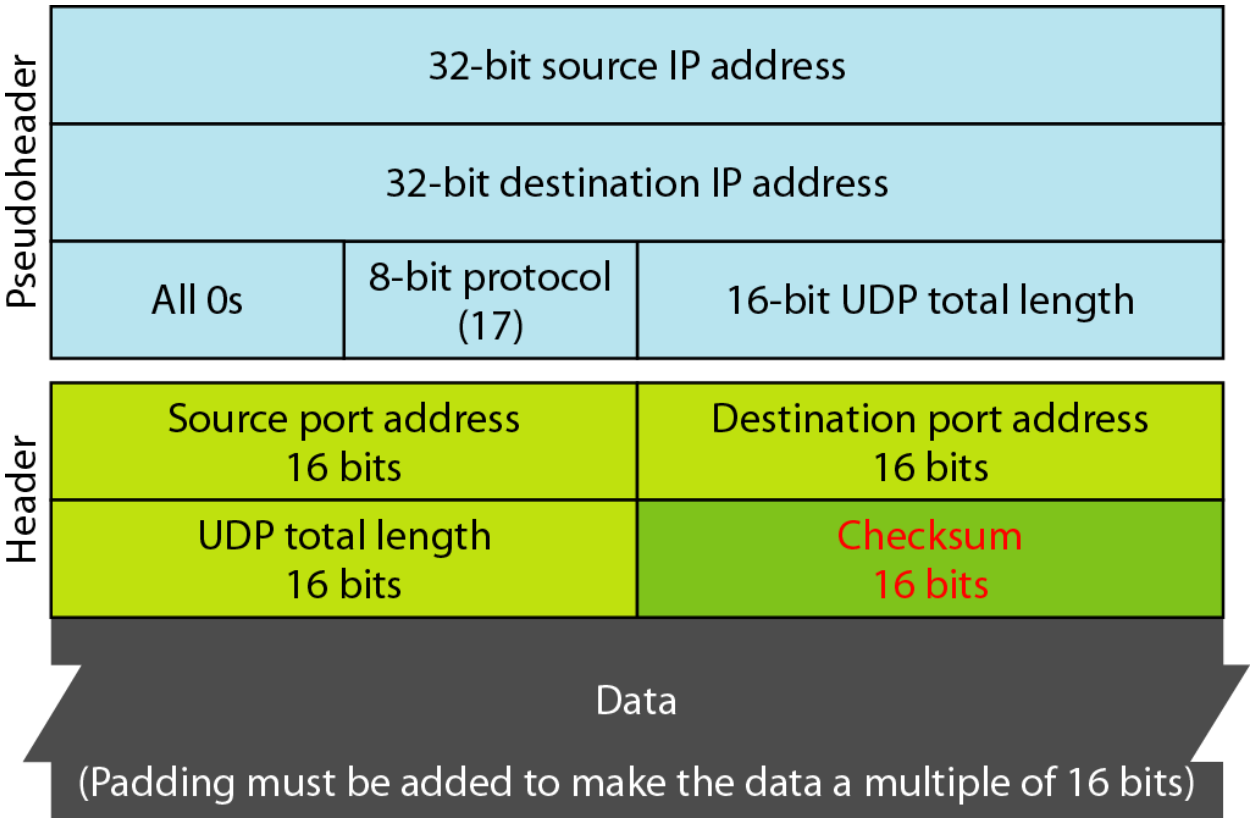


Checksum (OPTIONAL,IF NOT USED SET ALL 1'S DEFAULT)

The UDP checksum calculation is different from the one for IP and ICMP. Here the checksum includes three sections: **a pseudo header, the UDP header, and the data** coming from the application layer.
The pseudo header is the part of the header of the IP packet in which the user datagram is to be encapsulated with some fields filled with Os

If the checksum does not include the pseudo header, a user datagram may arrive safe and sound. However, if the IP header is corrupted, it may be delivered to the wrong host.
The protocol field is added to ensure that the packet belongs to UDP, and not to other transport-layer protocols.

Figure 23.10 *Pseudoheader for checksum calculation*



UDP Operation

Connectionless Services

UDP provides a connectionless service. This means that each user datagram sent by UDP is an independent datagram. There is no relationship between the different user datagrams even if they are comingfrom the same source process and going to the same destination program.The user datagrams are not numbered. Also, there is no connection establishment and no connection termination, as is the case for TCP. Thismeans that each user datagram can travel on a different path.

Flow and Error Control

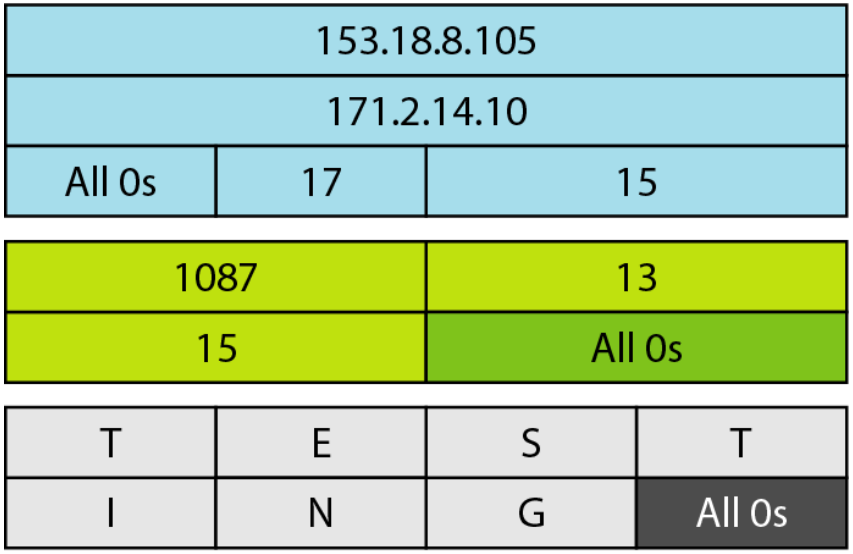
UDP is a very simple, unreliable transport protocol. There is no flow controland hence no window mechanism. The receiver may overflow with incoming messages. There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded. The lack of flow control and error control

Encapsulation and Decapsulation

To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages in an IP datagram.

Figure 23.11 shows the checksum calculation for a very small user datagram with only 7 bytes of data. Because the number of bytes of data is odd, padding is added for checksum calculation. The pseudoheader as well as thepadding will be dropped when the user datagram is delivered to IP.

Figure 23.11 *Checksum calculation of a simple UDP user datagram*



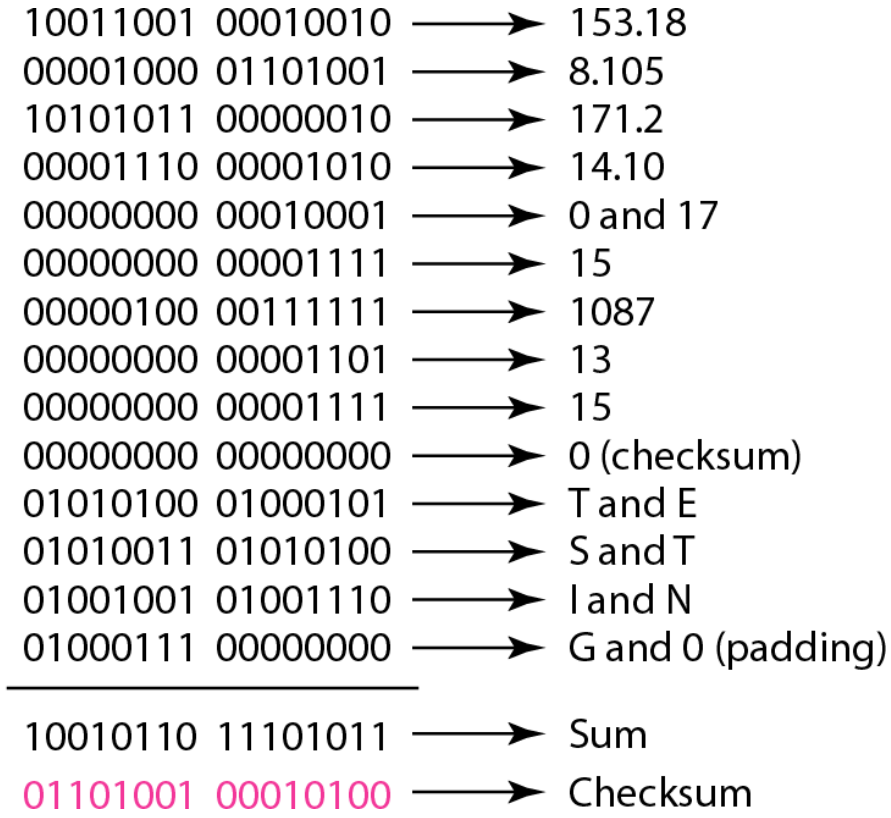
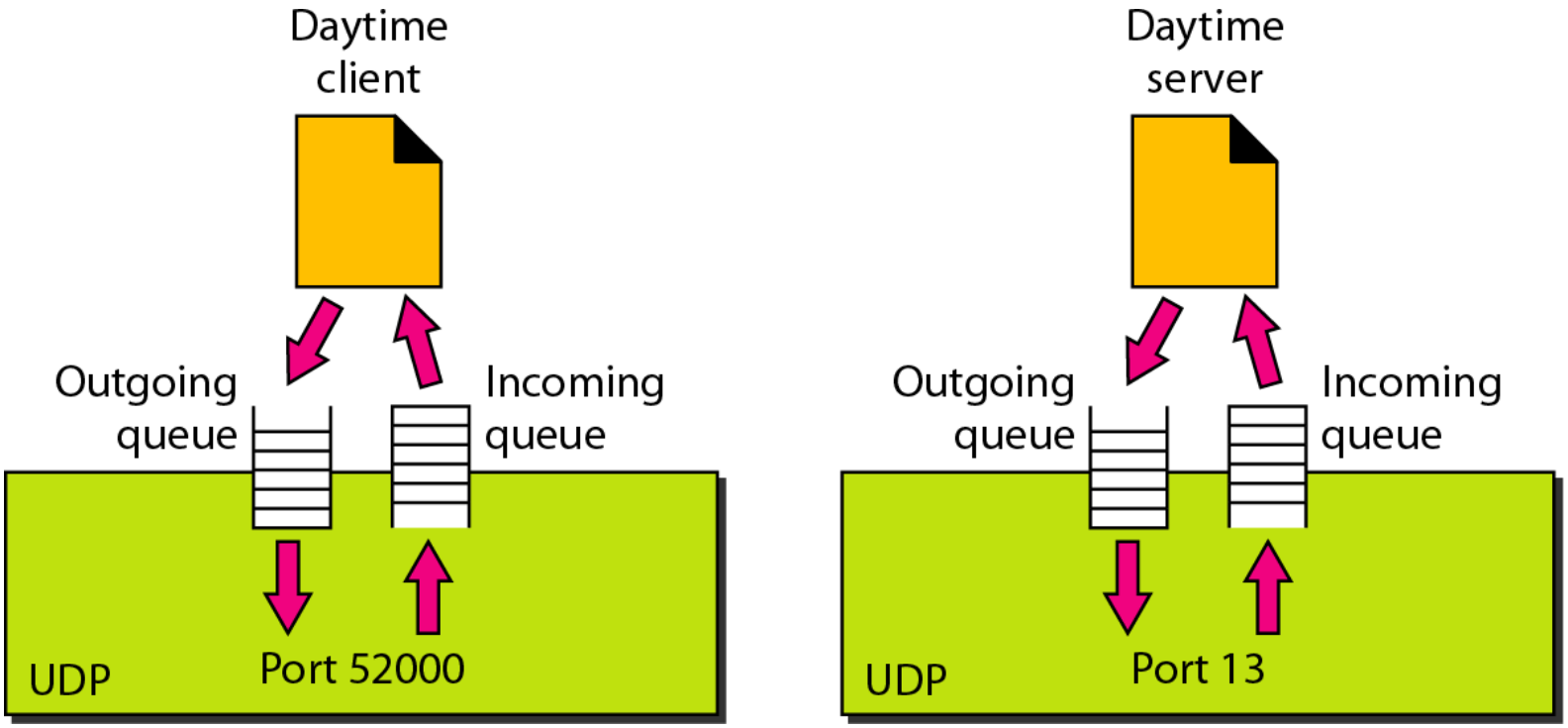


Figure 23.12 *Queues in UDP*



The key work was done by Birrell and Nelson (1984). In a nutshell, what Birrell and Nelson suggested was allowing programs to call procedures located on remote hosts. When a process on machine 1 calls a procedureon machine 2, the calling process on 1 is suspended and execution of the called procedure takes place on 2. Information can be transported from the caller to the callee in the parameters and can come back in the procedure result. No message passing is visible to the application programmer. This technique is known as **RPC** (**Remote Procedure Call).** Traditionally, the calling procedure is known as the client and the called procedure is known as the server, and we will use those names here too.

to call a remote procedure, the client program must be bound with a small library procedure, called the **client stub,** that represents the server procedure in the client's address space. Similarly, the server is bound with aprocedure called the **server stub.** These procedures hide the fact that the procedure call from the client to the server is not local

Step 1 is the client calling the client stub. This call is a local procedure call, with the parameters pushed onto the stack in the normal way.
Step 2 is the client stub packing the parameters into a message andmaking a system call to send the message. Packing the parameters is called **marshaling.**
Step 3 is the operating system sending the message from the client machine to the server machine.
Step 4 is the operating system passing the incoming packet to the server stub.
Finally, step 5 is the server stub calling the server procedure with the unmarshaled parameters.
The reply traces the same path in the other direction.

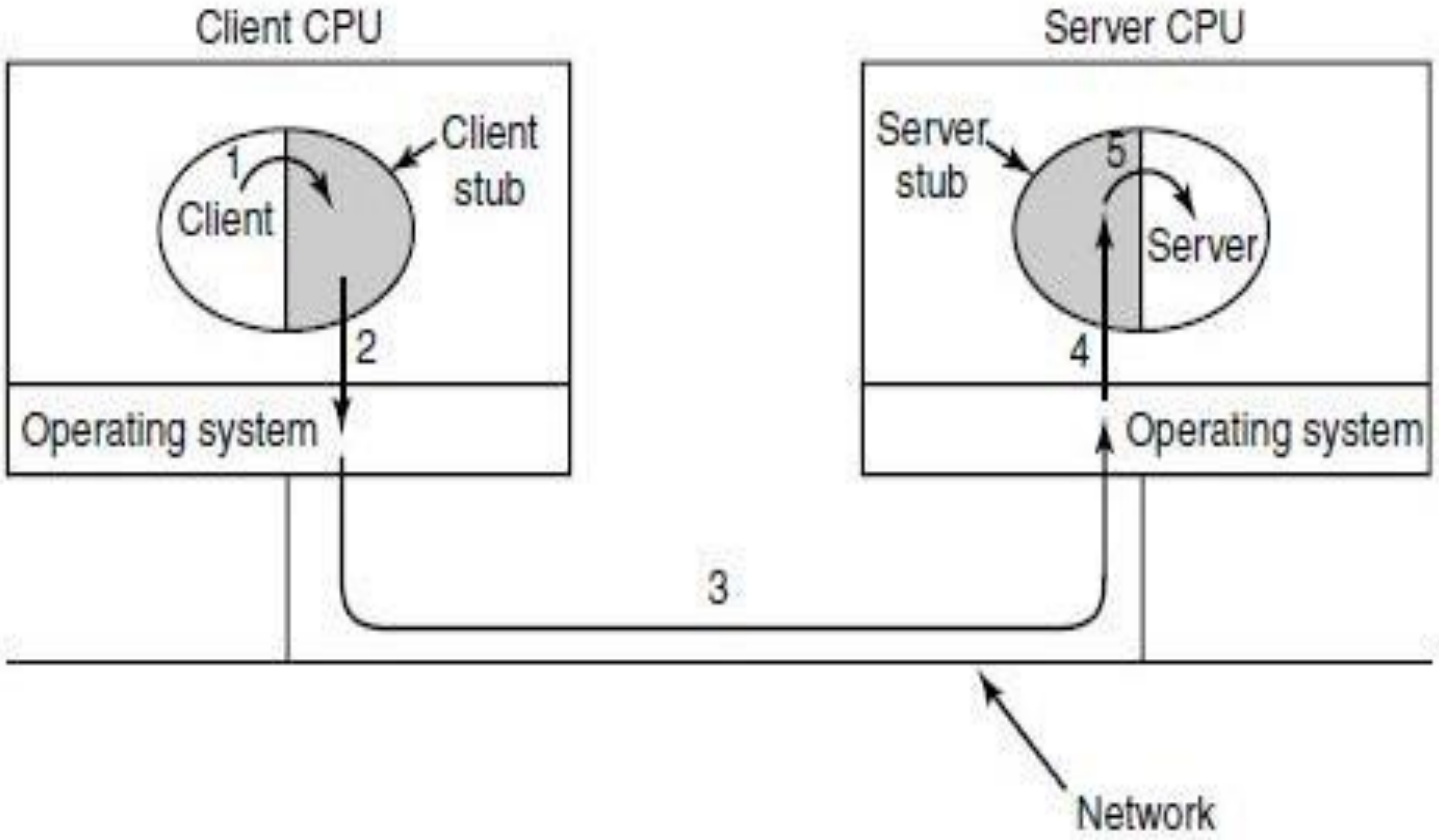


Figure 6-29. Steps in making a remote procedure call. The stubs are shaded.

Problems with RPC:

1 With RPC, passing pointers is impossible because the client and server are in different address spaces.
2 It is essentially impossible for the client stub to marshal the parameters: it has no way of determining how large they are.
3 A third problem is that it is not always possible to deduce the types of the parameters, not even from a formal specification or the code itself.(exa: printf)
4 A fourth problem relates to the use of global variables. Normally, the calling and called procedure can communicate by using global variables, in addition to communicating via parameters. But if the called procedure is moved to aremote machine, the code will fail because the global variables are no longer shared

Real-Time Transport Protocols

Client-server RPC is one area in which UDP is widely used.
Another one is for real-time multimedia applications.

Internet radio,
Internet telephony,
music-on-demand,
videoconferencing,
video-on-demand,

and other multimedia applications became more commonplace, people have discovered that each application was reinventing more or less the same real-time transport protocol.

It gradually became clear that having a generic real-time transport protocol for multiple applications would be a good idea.
Thus was **RTP (Real-time Transport Protocol)** born. It is described in RFC 3550 and is now in widespread use for multimedia applications. We will describe two aspects of real-time transport.
The first is the RTP protocol for transporting audio and video data in packets.
The second is the processing that takes place, mostly at the receiver, to play out the audio and video at the right time..

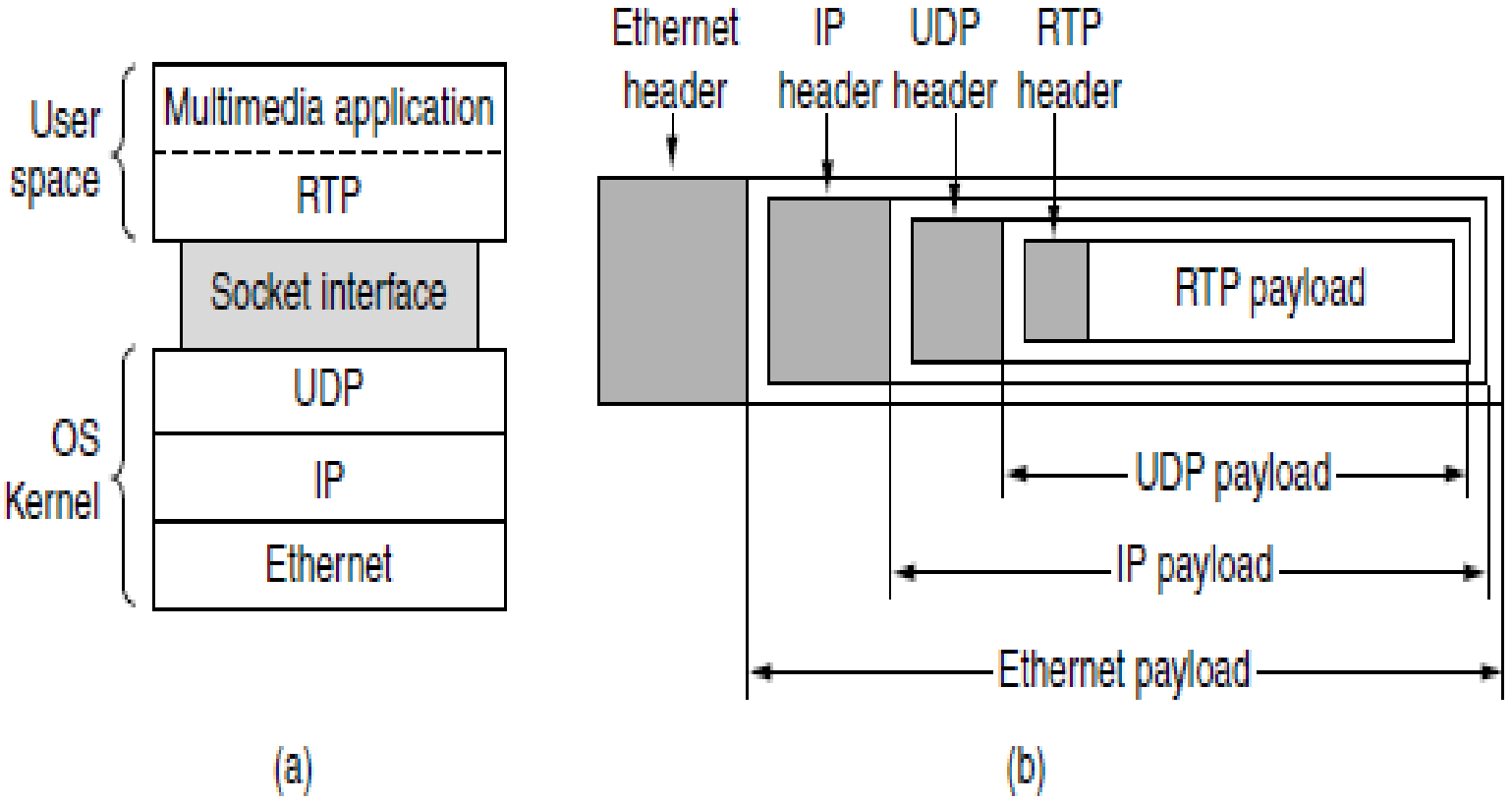


Figure 6-30. (a) The position of RTP in the protocol stack. (b) Packet nesting.

RTP normally runs in user space over UDP (in the operating system).
It operates as follows. The multimedia application consists of multiple audio, video, text, and possibly other streams. These are fed into the RTP library, which is in user space along with the application. This library multiplexes the streams and encodes them in RTP packets, which it stuffs into a socket.
On the operating system side of the socket, UDP packets are generated to wrap the RTP packets and handed to IP for transmission over a link such as Ethernet.
The reverse process happens at the receiver. The multimedia application eventually receives multimedia data from the RTP library. It is responsible for playing out the media. The protocol stack for this situation is shown in Fig. 6-30(a). The packet nesting is shown in Fig. 6-30(b).

RTP—The Real-time Transport Protocol

The basic function of RTP is to multiplex several real-time data streams onto
a single stream of UDP packets. The UDP stream can be sent to a single destination (unicasting) or to multiple destinations (multicasting).
Because RTP just uses normal UDP, its packets are not treated specially by the routers unless some normal IP quality-of-service features are enabled. In particular, there are no special guarantees about delivery, and packets may be lost, delayed, corrupted, etc.

The RTP format contains several features.
Each packet sent in an RTP stream is given a number one higher than its predecessor. This numbering allows the destination to determine if any packets are missing.
RTP has no acknowledgements, and no mechanism to request retransmissions.
Each RTP payload may contain multiple samples, and they may be coded any way that the application wants. To allow for interworking, RTP defines several profiles (e.g., a single audio stream), and for each profile, multiple encoding formats may be allowed
Another facility many real-time applications need is time stamping. Not only does time stamping reduce the effects of variation in network delay, but it

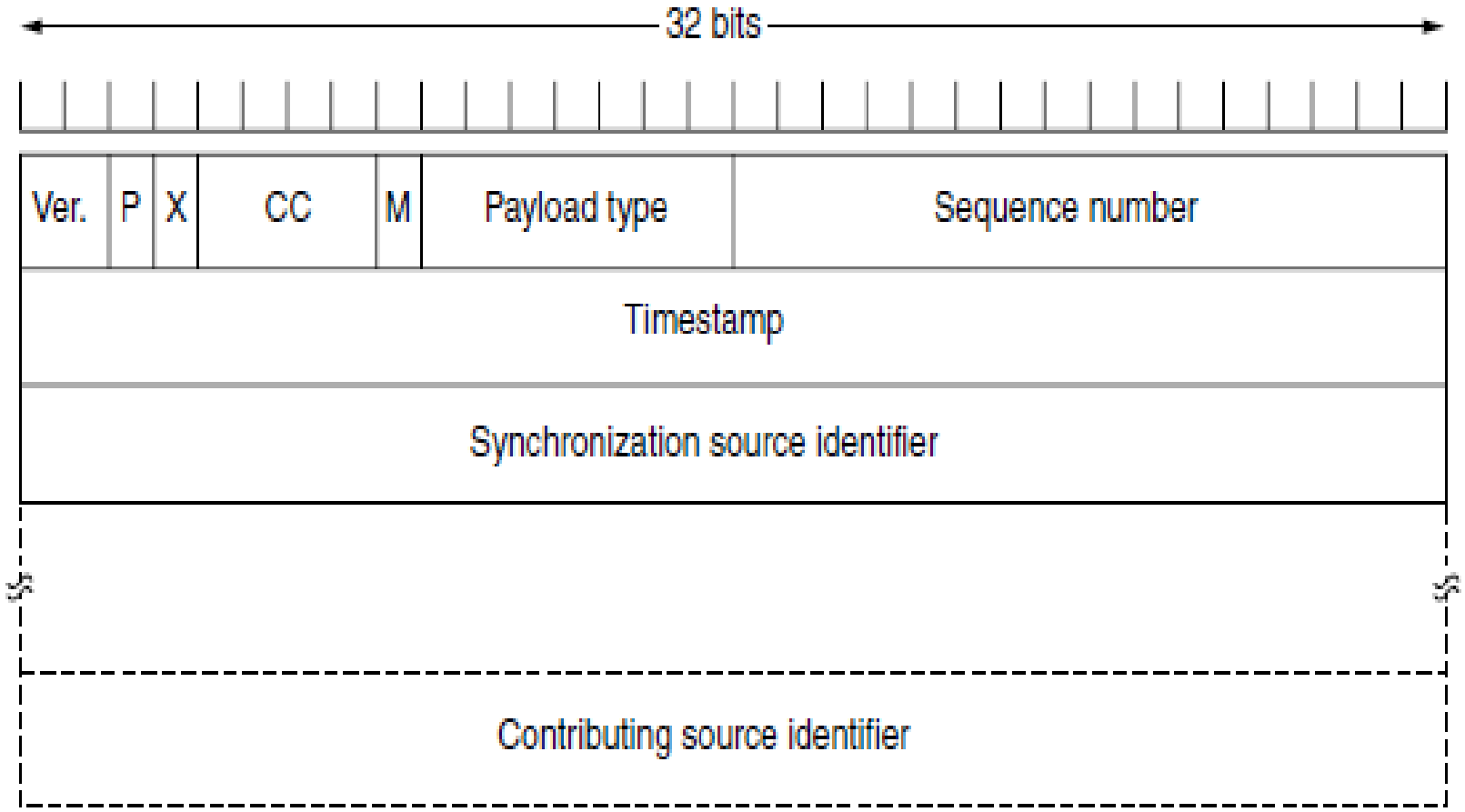


Figure 6-31. The RTP header.

It consists of three 32-bit words and potentially some extensions.The first word contains the *Version field, which* is already at 2.

The *P bit indicates that the packet has been padded to a multiple of 4bytes.* The last padding byte tells how many bytes were added.

The *X bit indicates that* an extension header is present.

The *CC field tells how many contributing sources are present, from 0 to 15*

The *M bit is an application-specific marker bit. It can be used to* mark the start of a video frame, the start of a word in an audio channel, or something else that the application understands.

The *Payload type field tells which encoding* algorithm has been used (e.g., uncompressed 8-bit audio, MP3, etc.). Since every packet carries this field, the encoding can change during transmission.

The *Sequence number is just a counter that is incremented on each RTP packet* sent. It is used to detect lost packets.

The *Timestamp, this* value can help reduce timing variability called jitter at the receiver by decoupling the playback from the packet arrival time.

The *Synchronization source identifier tells which stream the packet belongs to. It is* the method used to multiplex and demultiplex multiple data streams onto a single stream of UDP packets.

Finally, the *Contributing source identifiers, if any, are* used when

RTCP—The Real-time Transport Control Protocol

RTP has a little sister protocol (little sibling protocol?) called RTCP (Real time Transport Control Protocol). It is defined along with RTP in RFC 3550 and handles feedback, synchronization, and the user interface. It does not transport any media samples.

The first function can be used to provide feedback on delay, variation in delay or jitter, bandwidth, congestion, and other network properties to the sources. This information can be used by the encoding process to increase the data rate (and give better quality) when the network is functioning well and to cut back the data rate when there is trouble in the network. By providing continuous feedback, It provides the best quality

The *Payload type field is used to tell the destination* what encoding algorithm is used for the current packet, making it possible to vary it on demand.

RTCP also handles inter stream synchronization. The problem is that different streams may use different clocks, with different granularities and different drift rates. RTCP can be used to keep them in sync.

Finally, RTCP provides a way for naming the various sources (e.g., in ASCII text). This information can be displayed on the receiver'sscreen to indicate who is talking at the moment.

Playout with Buffering and Jitter Control

Once the media information reaches the receiver, it must be played out at the right time. Even if the packets are injected with exactly the right intervals between them at the sender, they will reach the receiver with differentrelative times. This variation in delay is called **jitter.** Even a small amount of packet jitter can cause distracting media artifacts, such as jerky video frames and unintelligible audio, if the media is simply played out as itarrives.

The solution to this problem is to **buffer** packets at the receiver before they are played out to reduce the jitter.

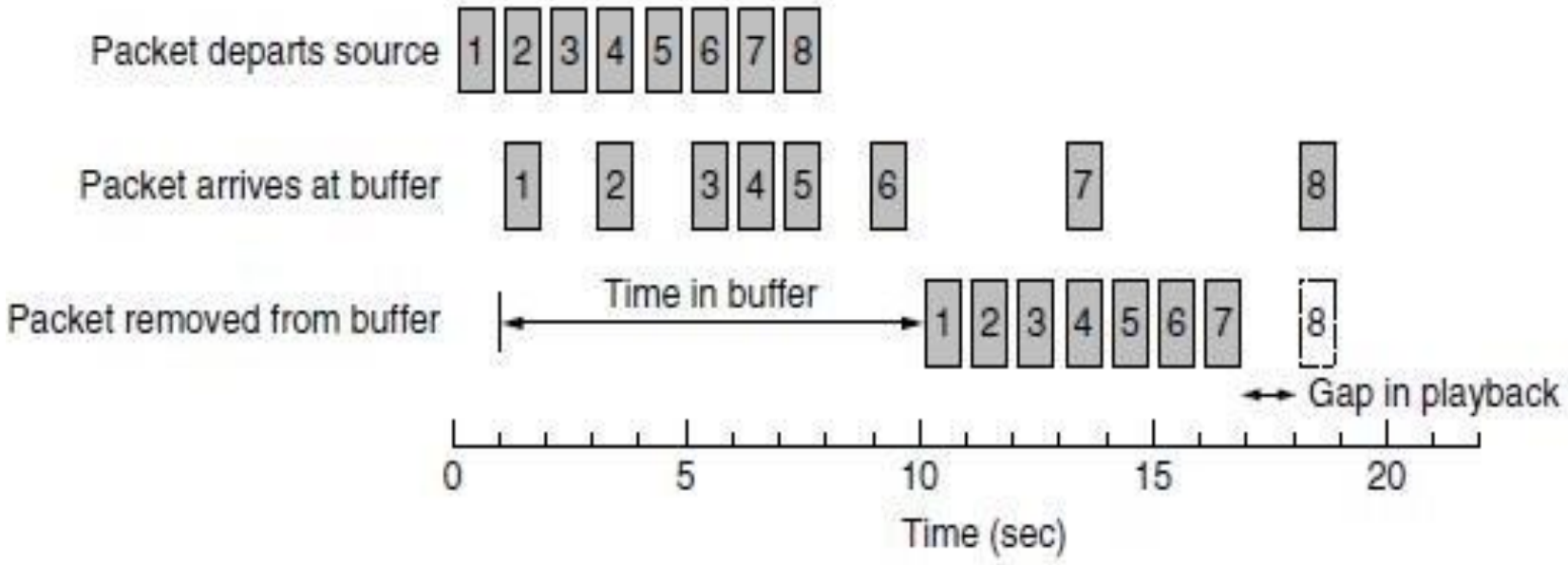


Figure 6-32. Smoothing the output stream by buffering packets.

A key consideration for smooth playout is the **playback point,** or how long to wait at the receiver for media before playing it out. Deciding how long to wait depends on the jitter. The difference between a low-jitter and high-jitter connection is shown in Fig. The average delay may not differ greatly between the two, but if there is high jitter the playback point may need to be much further out to capture 99% of the packets than if there is low jitter.

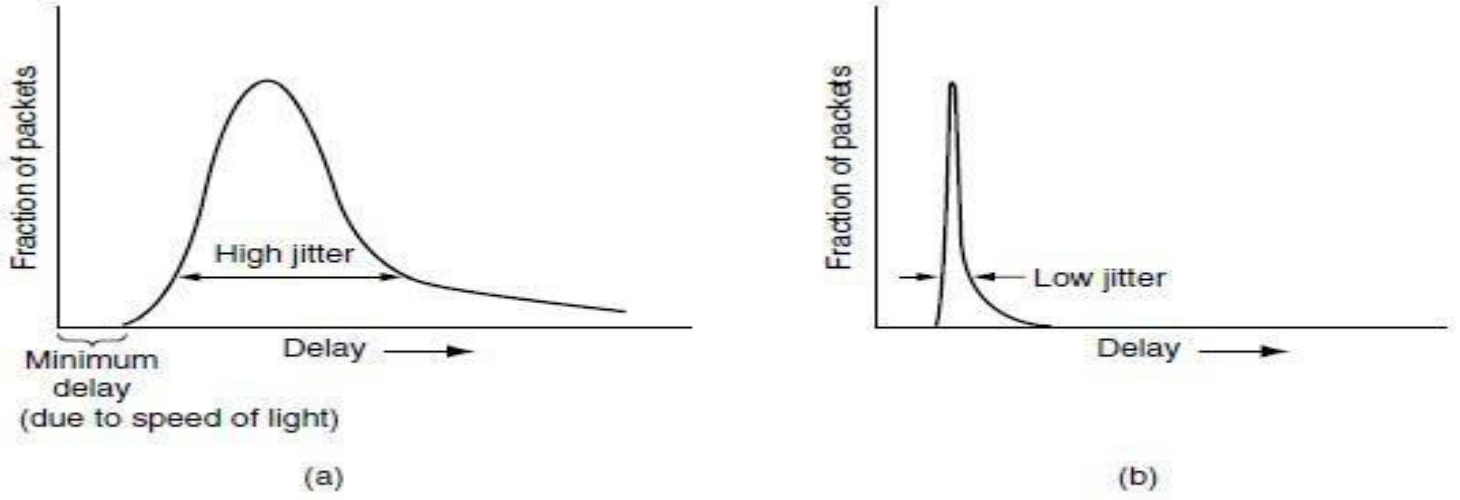


Figure 6-33. (a) High jitter. (b) Low jitter.

One way to avoid this problem for audio is to adapt the playbackpoint between **talk spurts,** in the gaps in a conversation. No one will notice the difference between a short and slightly longer silence

TELNET

It is client/server application program. TELNET is an abbreviation for *TErminaL NETwork.* TELNET enables the establishment of a connection to a remote system in such a way that the local terminal appears to be a terminal at the remote system.

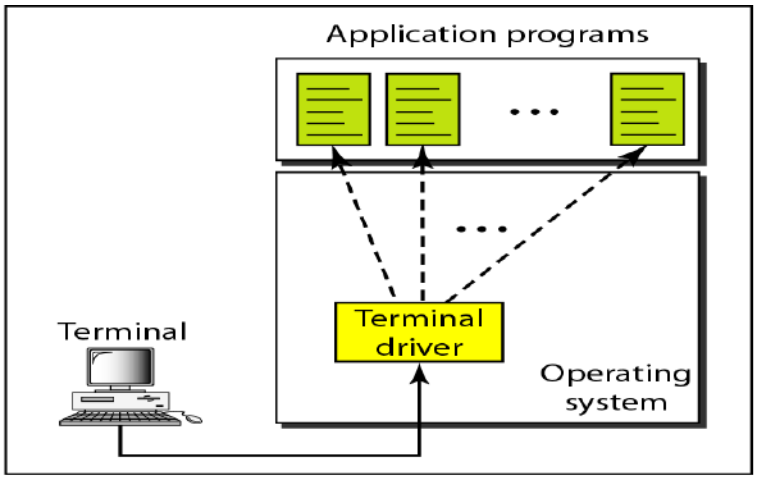
Timesharing Environment

A large computer supports multiple users. The interaction between auser and the computer occurs through a terminal, which is usually a combination of keyboard, monitor, and mouse.

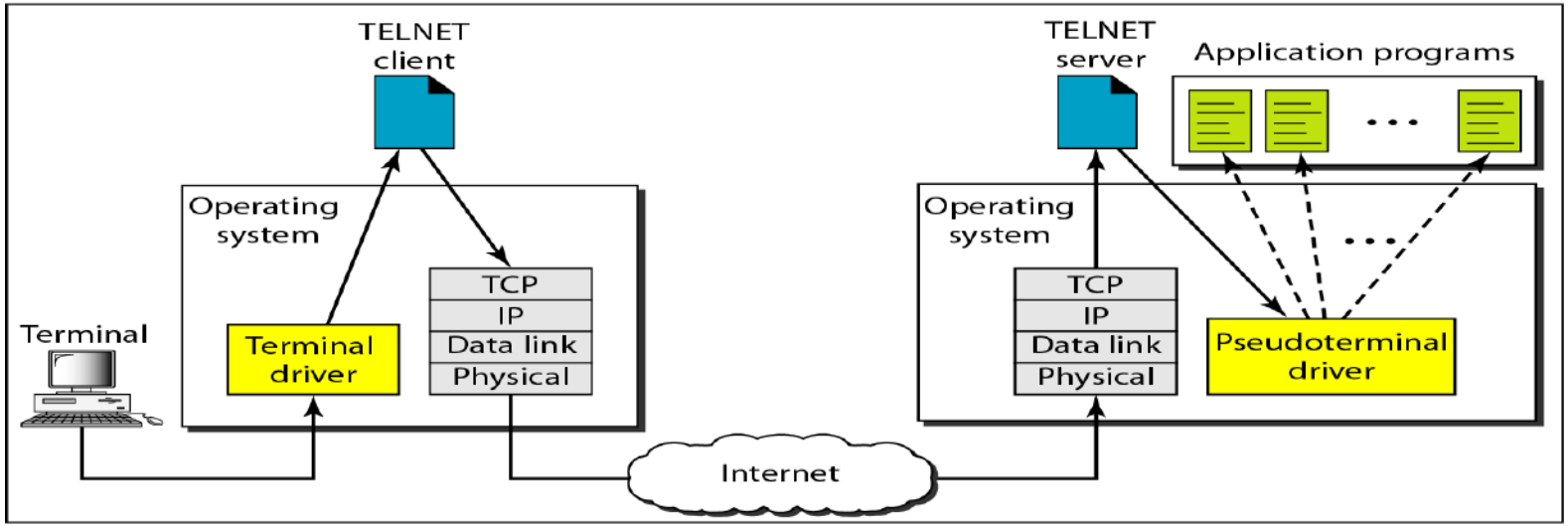
Logging

To access the system, the user logs into the system with a user id or log-in name. The system also includes password checking to prevent an unauthorized user from accessing the resources.

Local login
Remote login



a. Local log-in



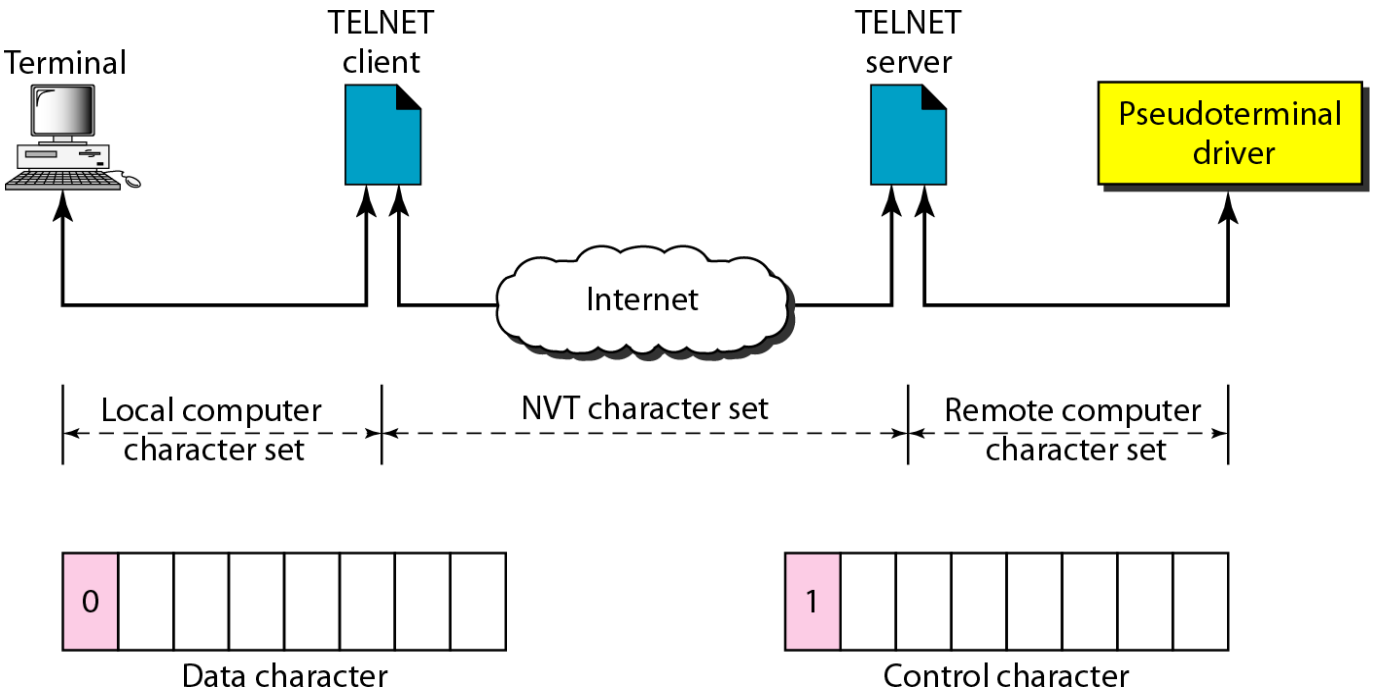
b. Remote log-in

When a user logs into a local timesharing system, it is called **local log-in**. As a user types at a terminal or at a workstation running a terminalemulator, the keystrokes are accepted by the terminal driver. The terminal driver passes the characters to the operating system. The operating system, in turn, interprets the combination of characters and invokes the desired application program or utility.

When a user wants to access an application program or utility located on a remote Machine, it is called **remote log-in**. Here the TELNET client and server programs come into use. The user sends the keystrokes to theterminal driver, where the local operating system accepts the characters but does not interpret them. The characters are sent to the TELNET client, which transforms the characters to a universal character set called *network virtual terminal (NVT) characters and delivers them to the local TCP/IP* protocol stack.

The commands or text, in NVT form, travel through the Internet and arriveat the TCP/IP stack at the remote machine. Here the characters are delivered to the operating system and passed to the TELNET server, whichchanges the characters to the corresponding characters understandable bythe remote computer. However, the characters cannot be passed directlyto the operating system because the remote operating system is not designed to receive characters from a TELNET server: It is designed to receive characters from a terminal driver. The solution is to add a piece of

Concept of NVT(network virtual terminal)



WWW and HTTP

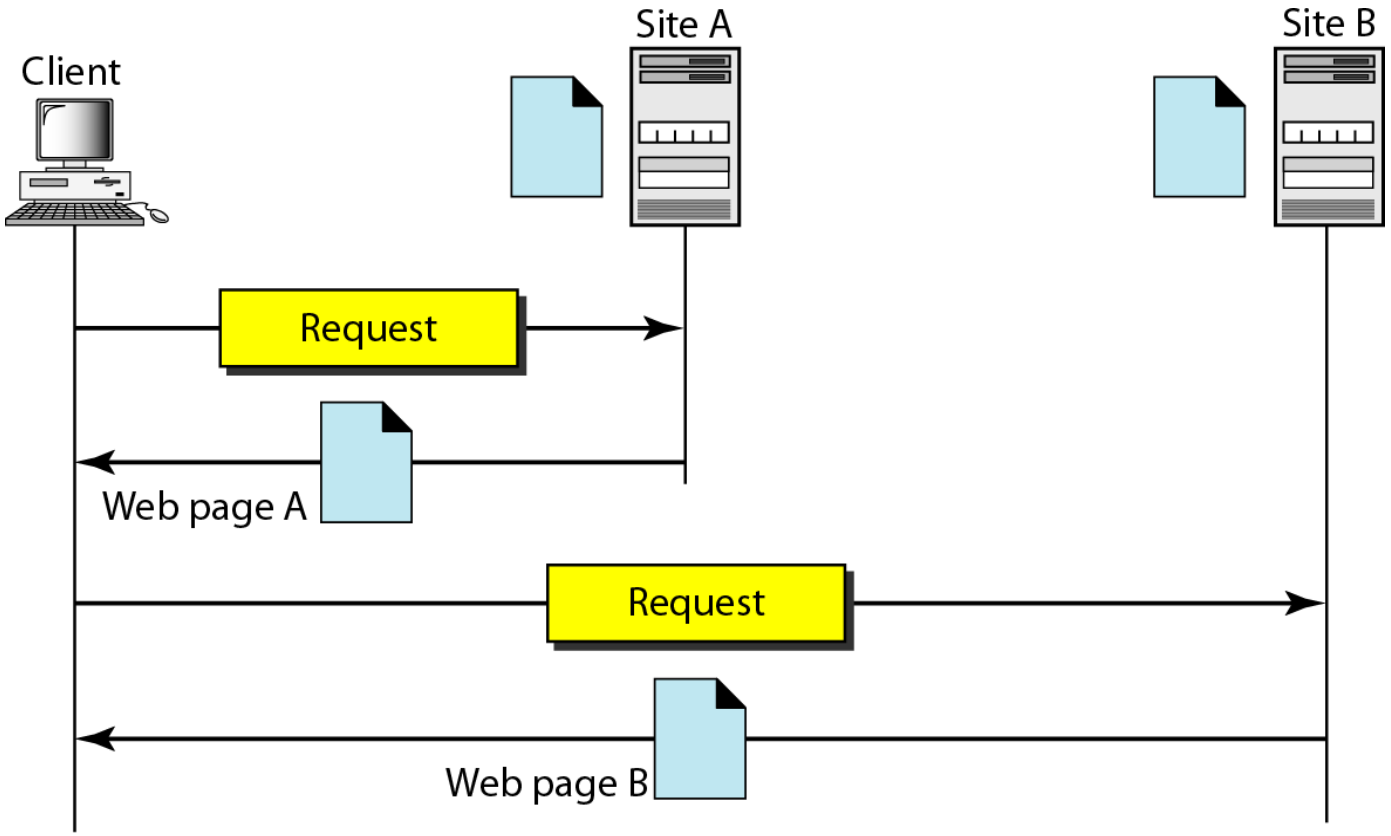
27-1 ARCHITECTURE

The WWW today is a distributed client/server service, in which a client using a browser can access a service using a server. However, the service provided is distributed over many locations called sites as shown infig.

Topics discussed in this section:

Client (Browser)
Server
Uniform Resource Locator
Cookies

Figure 27.1 *Architecture of WWW*



Client (Browser)

A variety of vendors offer commercial browsers that interpret and display a Web document,and all use nearly the same architecture.

Each browser usually consists of three parts: a controller, client protocol, and interpreters.

The controller receives input from the keyboard or the mouse and uses the client programs to access the document.

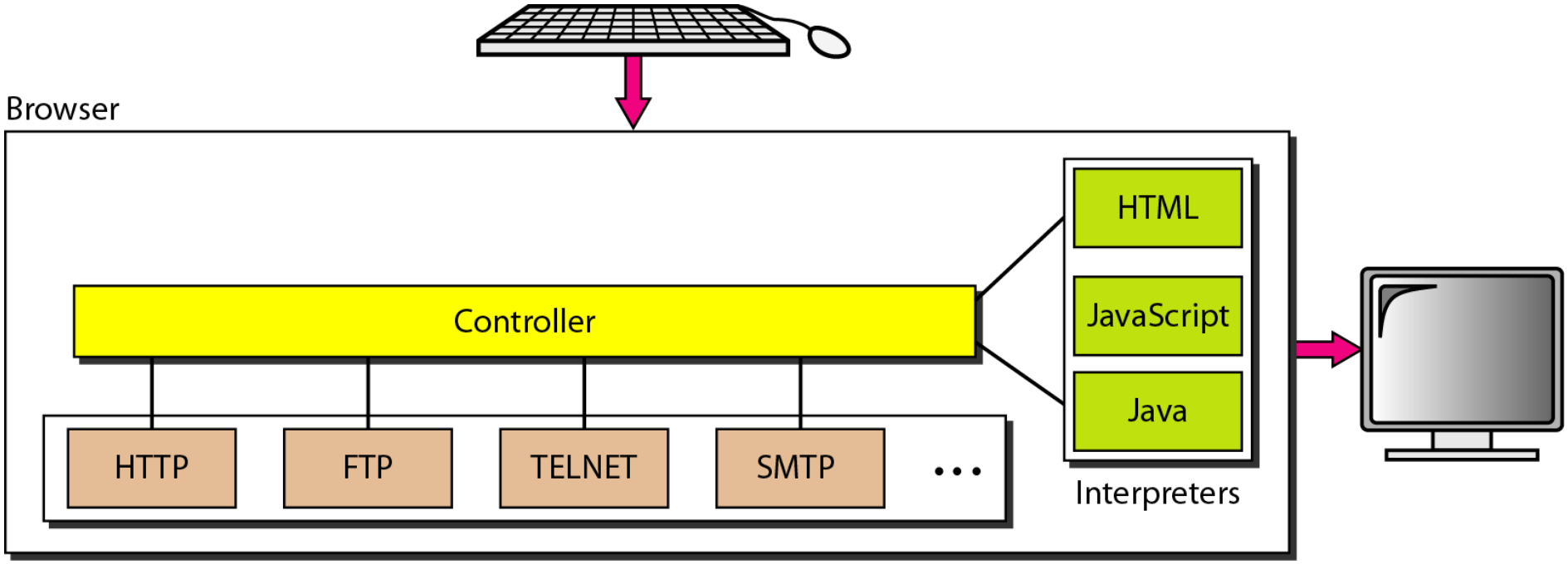
After the document has been accessed, the controller uses one of the interpreters to display the document on the screen. The interpreter can be HTML, Java, or JavaScript, depending on the type of document

The client protocol can be one of the protocols described previously such as FTP or HTTP.

Server

The Web page is stored at the server. Each time a client request arrives, the corresponding document is sent to the client. Toimprove efficiency, servers normally store requested files in a cache in memory; memory is faster to access than disk. A server can also become more efficient through multithreading or multiprocessing. In this case, a server can answer more than one

Figure 27.2 *Browser*



Uniform Resource Locator

A client that wants to access a Web page needs the address. To facilitate the access of documents distributed throughout the world, HTTP uses locators. The uniform resource locator (URL) is a standard for specifying any kind of information on the Internet. The URL defines four things: protocol, host computer, port, and path.

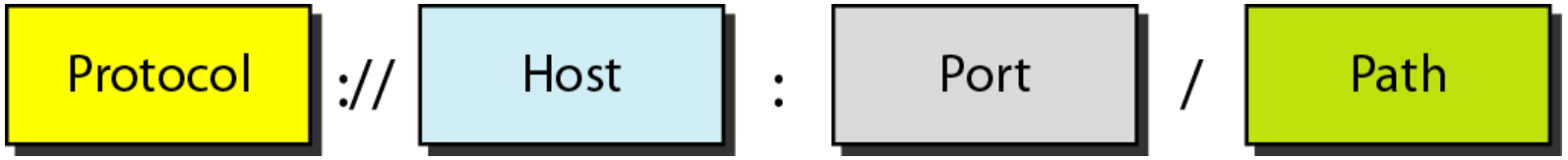
The *protocol is the client/server program used to retrieve the document. Many* different protocols can retrieve a document; among them are FTP or HTTP. The most common today is HTTP.

The host is the computer on which the information is located, although the name of the computer can be an alias. Web pages are usually stored in computers, and computers are given alias namesthat usually begin with the characters "www".

The URL can optionally contain the port number of the server. If the *port is* included, it is inserted between the host and the path, and it is separated from the host by a colon.

Path is the pathname of the file where the information is located. Note that the path can itself contain slashes that, in the UNIX operating system, separate the directories from the subdirectories and files.

Figure 27.3 *URL*



An HTTP cookie (also called web cookie, Internetcookie,browser cookie or simply cookie, the latter which is not to be confused with the literal definition), is a small piece of data sent from a website and stored in a user's web browser while the user is browsing that website

27-2 WEB DOCUMENTS

The documents in the WWW can be grouped into three broad categories: static, dynamic, and active. The category is based on the time at which the contents of the document are determined.

Topics discussed in this section:

Static Documents
Dynamic Documents
Active Documents

Static Documents

Static documents are fixed-content documents that are createdand stored in a server. The client can get only a copy of the document. When a client accesses the document, a copy of the document is sent. The user can then use a browsing program to display the document

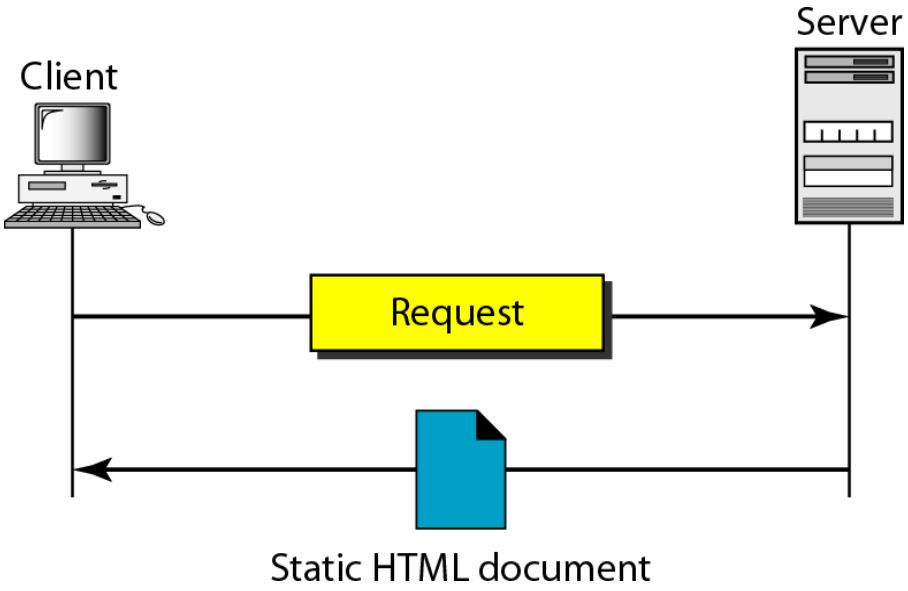


Figure 27.5 *Boldface tags*

HTML

Hypertext Markup Language (HTML) is a language for creatingWeb pages.

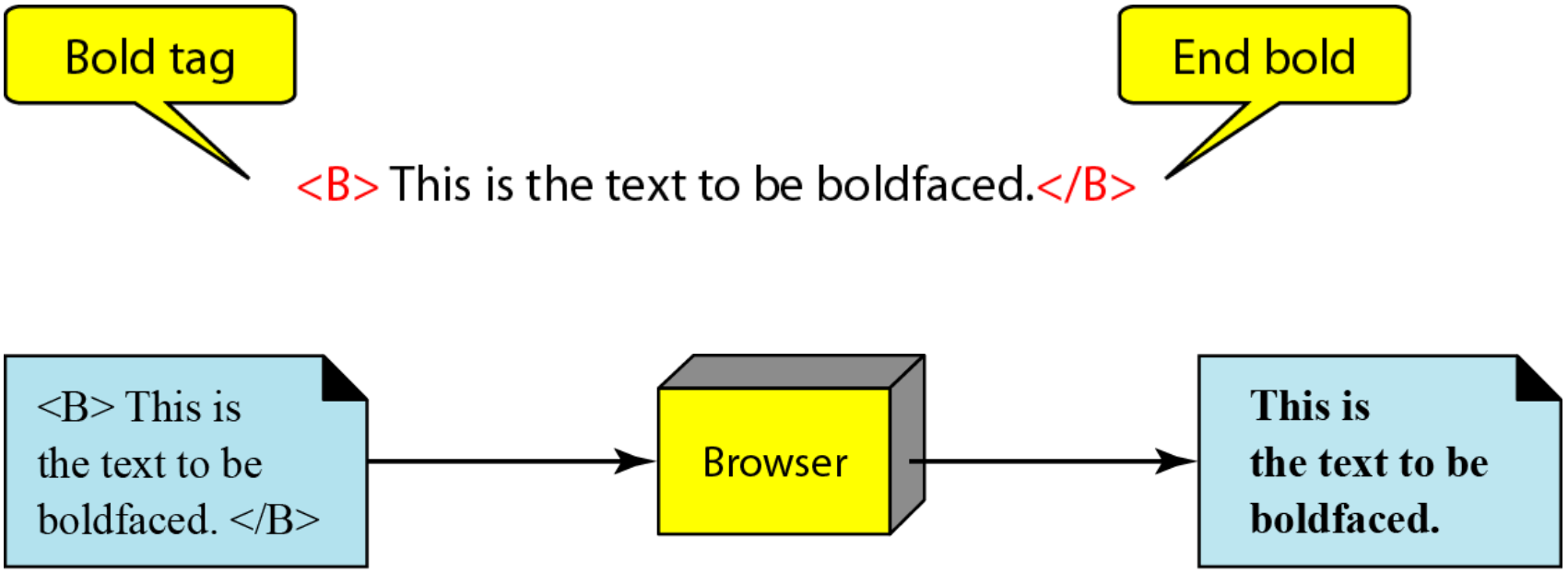
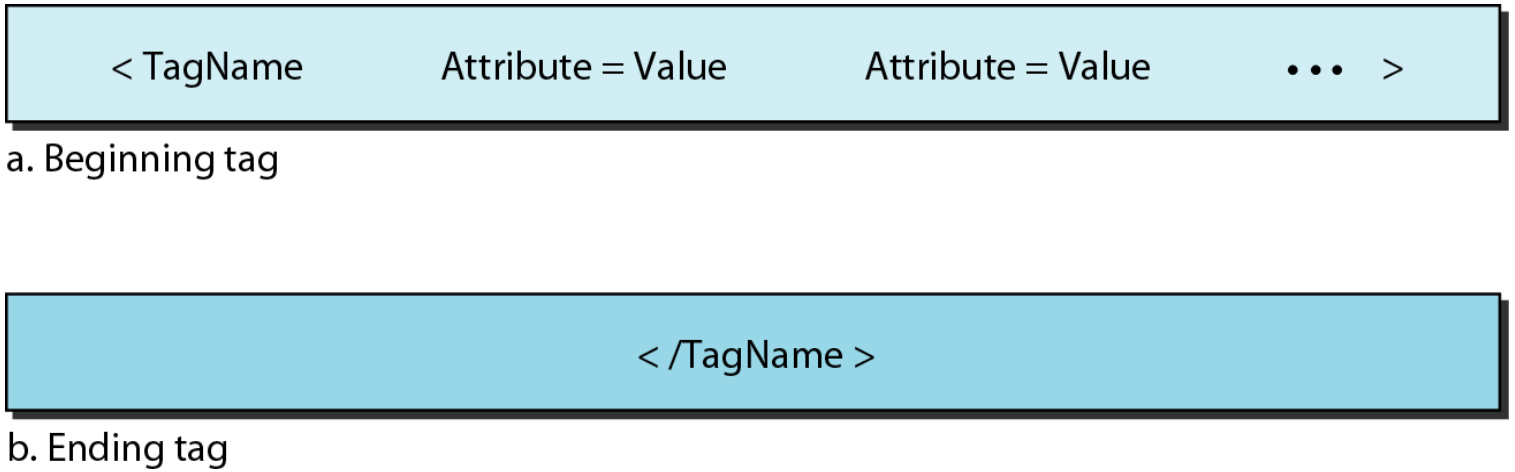


Figure 27.7 *Beginning and ending tags*



Dynamic Documents

A dynamic document is created by a Web server whenever a browser requests the document. When a request arrives, the Webserver runs an application program or a script that creates the dynamic document. The server returns the output of the programor script as a response to the browser that requested the document.

A very simple example of a dynamic document is the retrieval of the time and date from a server. Time and date are kinds of information that are dynamic in that they change from moment to moment. The client can ask the server to run a program such asthe *date program in UNIX and send the result of the program to* the client.

Common Gateway Interface (CGI)

The Common Gateway Interface (CGI) is a technology that creates and handles dynamic documents.

Hypertext Preprocessor (pHP), which uses the Perl language; Java Server Pages (JSP), which uses the Java language for scripting; Active Server Pages (ASP), a Microsoft product which uses VisualBasic language for scripting; and ColdFusion, which embeds SQLdatabase queries in the HTML document

Figure 27.8 *Dynamic document using CGI*

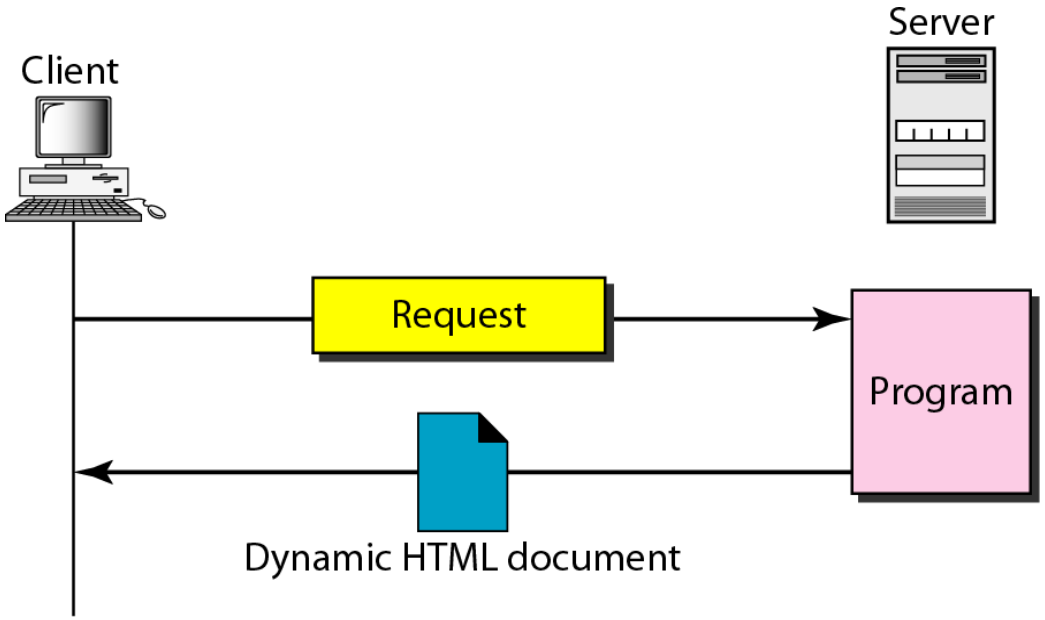
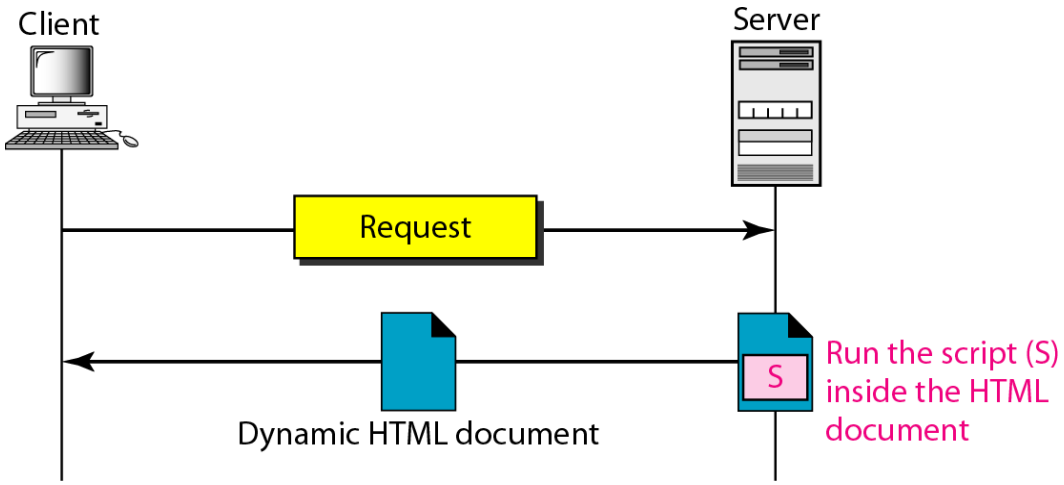
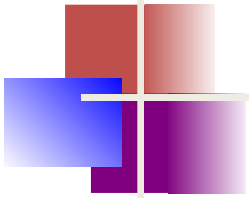


Figure 27.9 *Dynamic document using server-site script*





Note

Dynamic documents are sometimes referred to as server-sitedynamic documents.

Figure 27.10 *Active document using Java applet*

Active Documents

For many applications, we need a program or a script to be runat the client site. These are called active documents

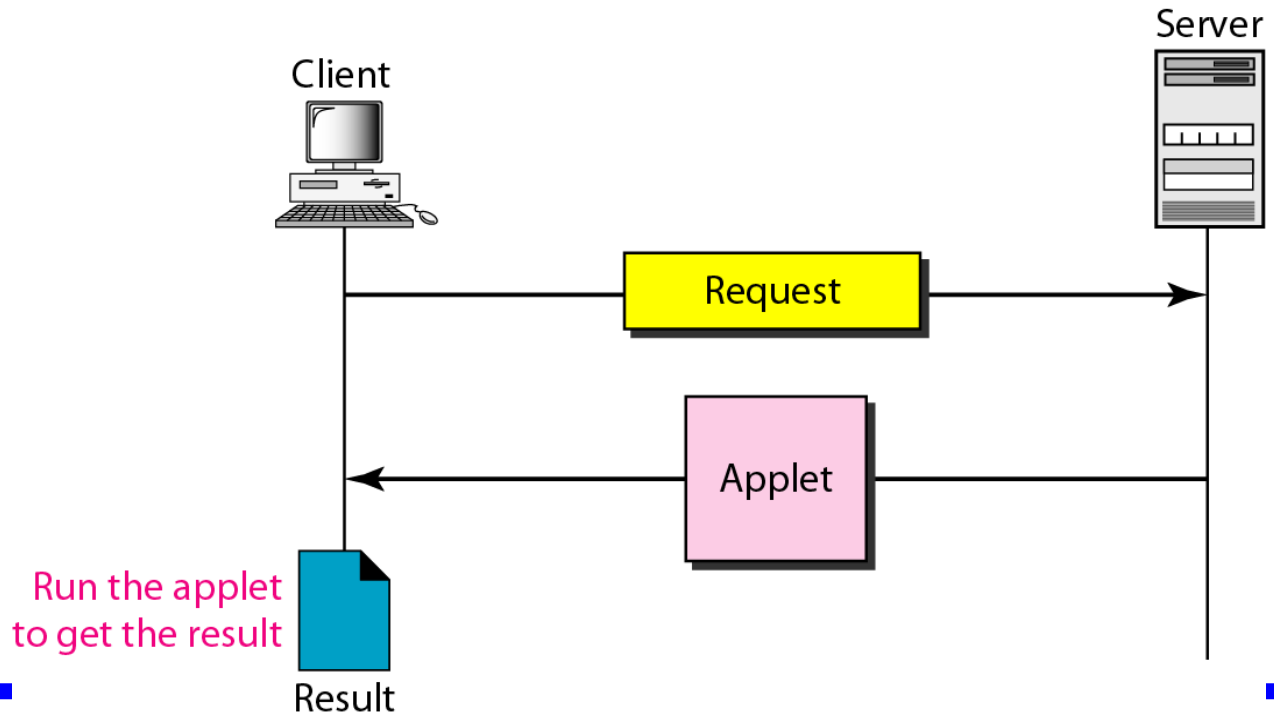
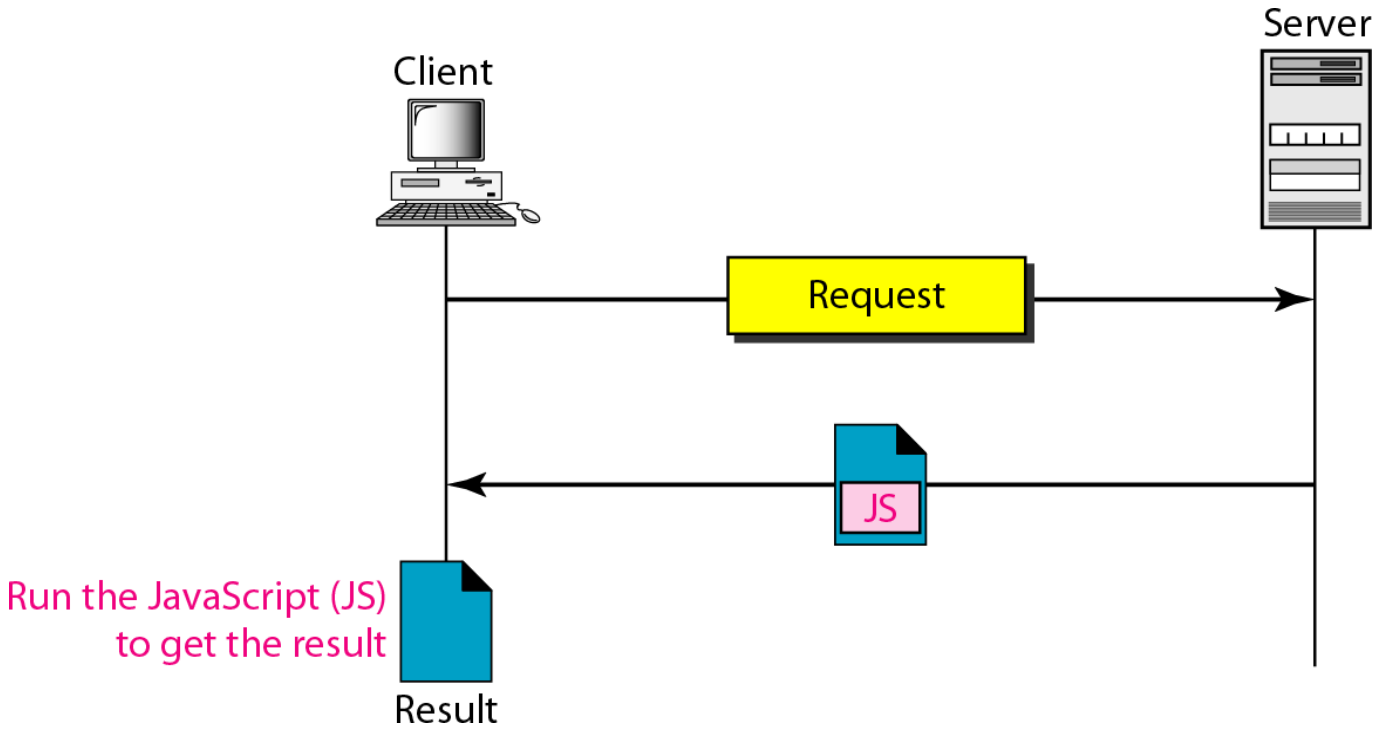


Figure 27.11 *Active document using client-site script*



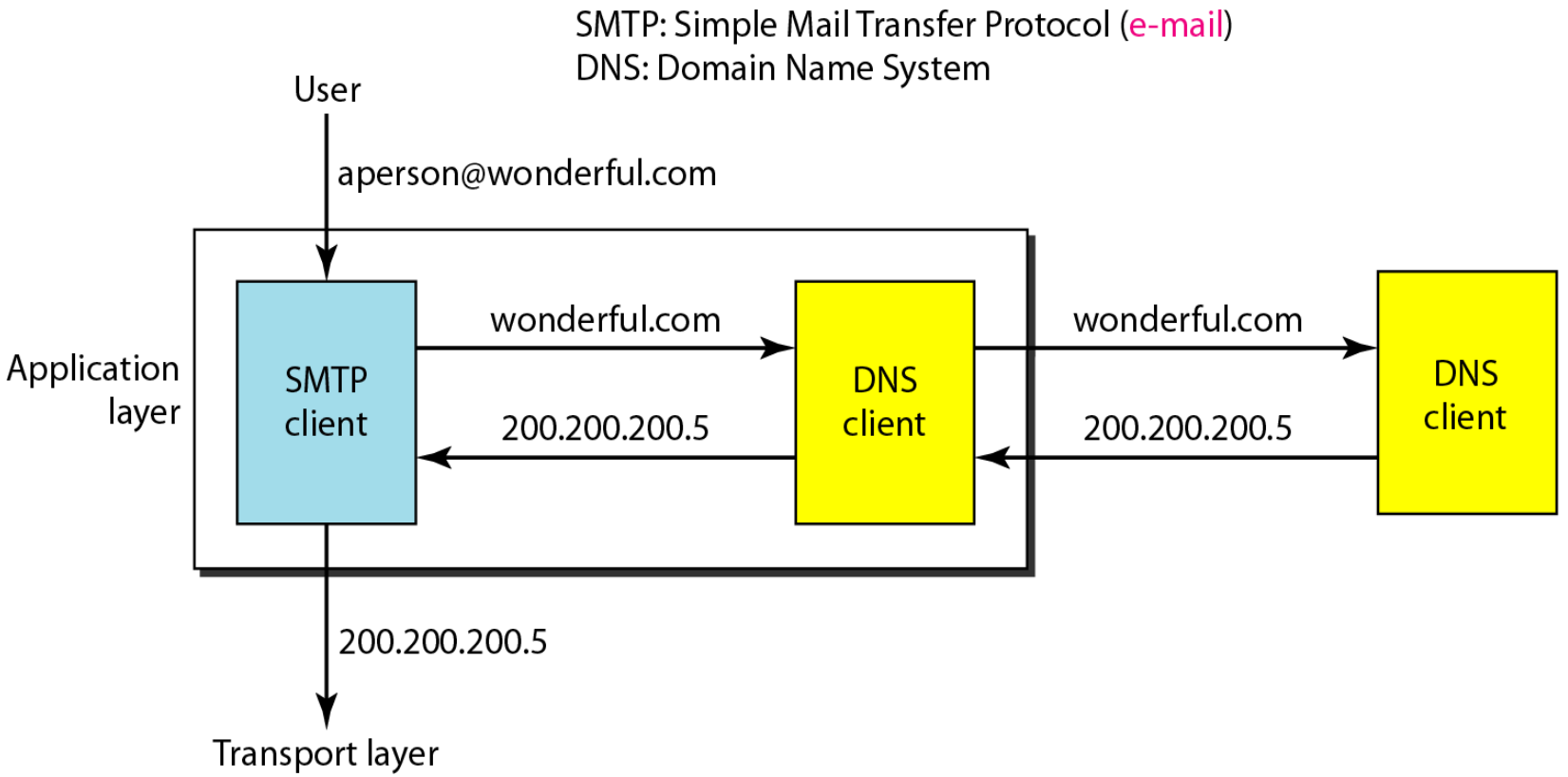
Note

Active documents are sometimes referred to as client-site dynamic documents.

HTTP version 1.1 specifies a persistent connection by default.

DNS (Domain Name System)

To identify an entity, TCP/IP protocols use the IP address, which uniquely identifies the connection of a host to the Internet. However, people prefer to use names instead of numeric addresses. Therefore, we need a system that can map a name to an address or an address to a name.



NAME SPACE

A name space that maps each address to a unique name can be organized in two ways: fiat or hierarchical.

Flat Name Space

In a flat name space, a name is assigned to an address. A name in this space is a sequence of characters without structure.

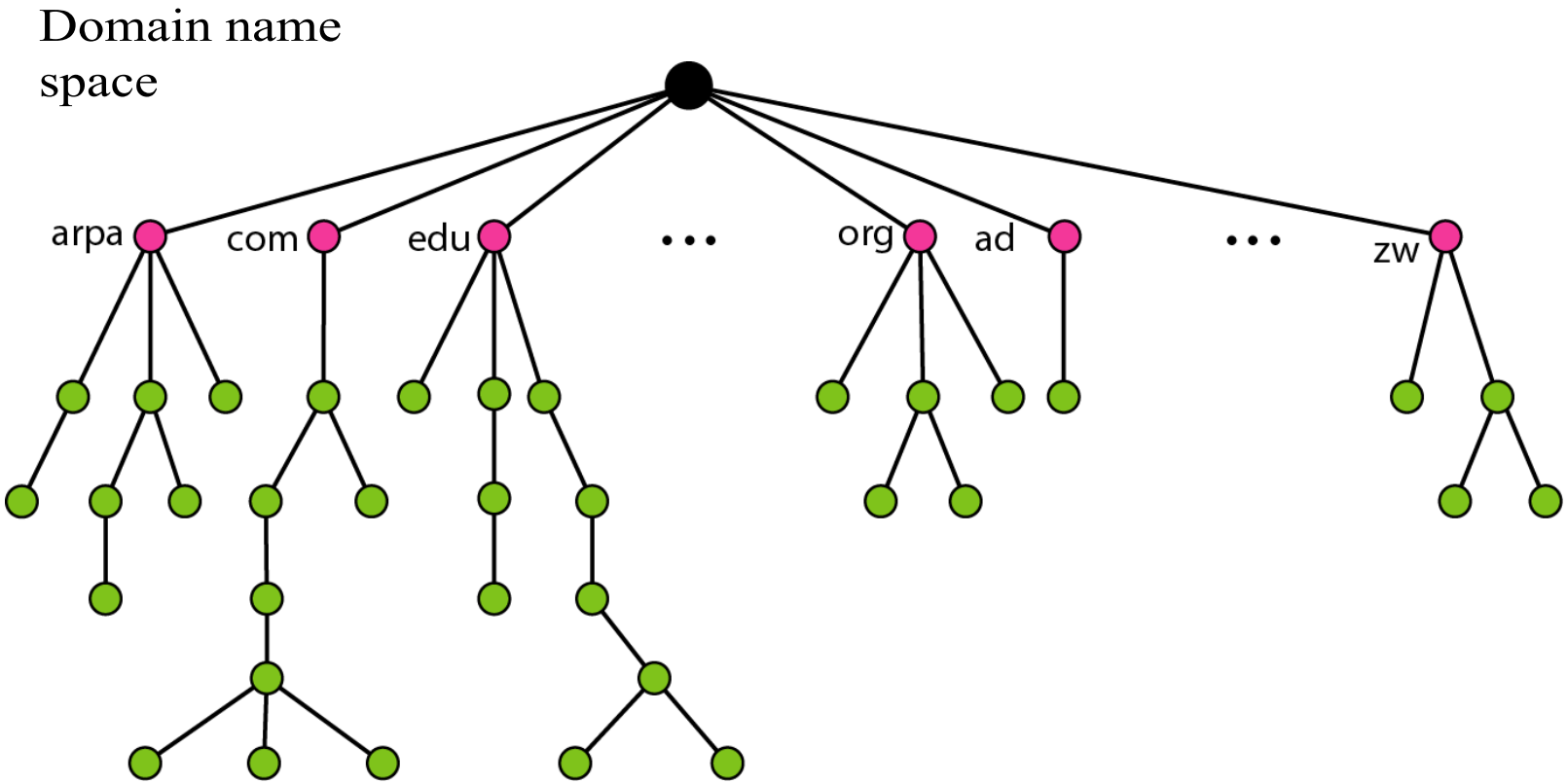
Hierarchical Name Space

In a hierarchical name space, each name is made of several parts. The first part can define the nature of the organization, the second part can define the name of an organization, the third part can define departments in the organization, and so on.

Exa: *challenger.jhda.edu, challenger.berkeley.edu, and challenger.smart.com*

DOMAIN NAME SPACE

To have a hierarchical name space, a domain name space was designed. In this design the names are defined in an inverted-tree structure with the root at the top. The tree can have only 128 levels: level 0 (root) to level 127.

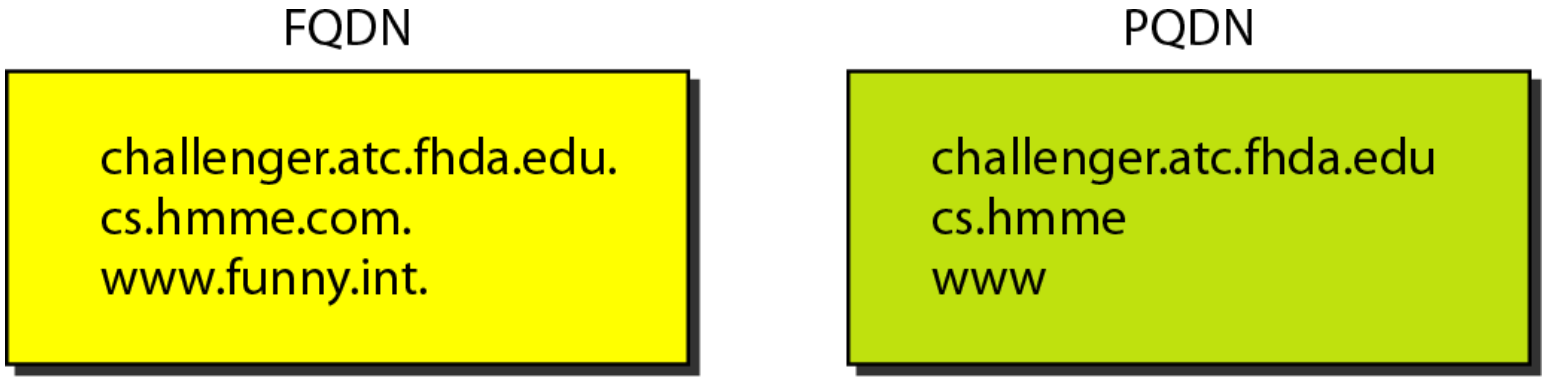


Label

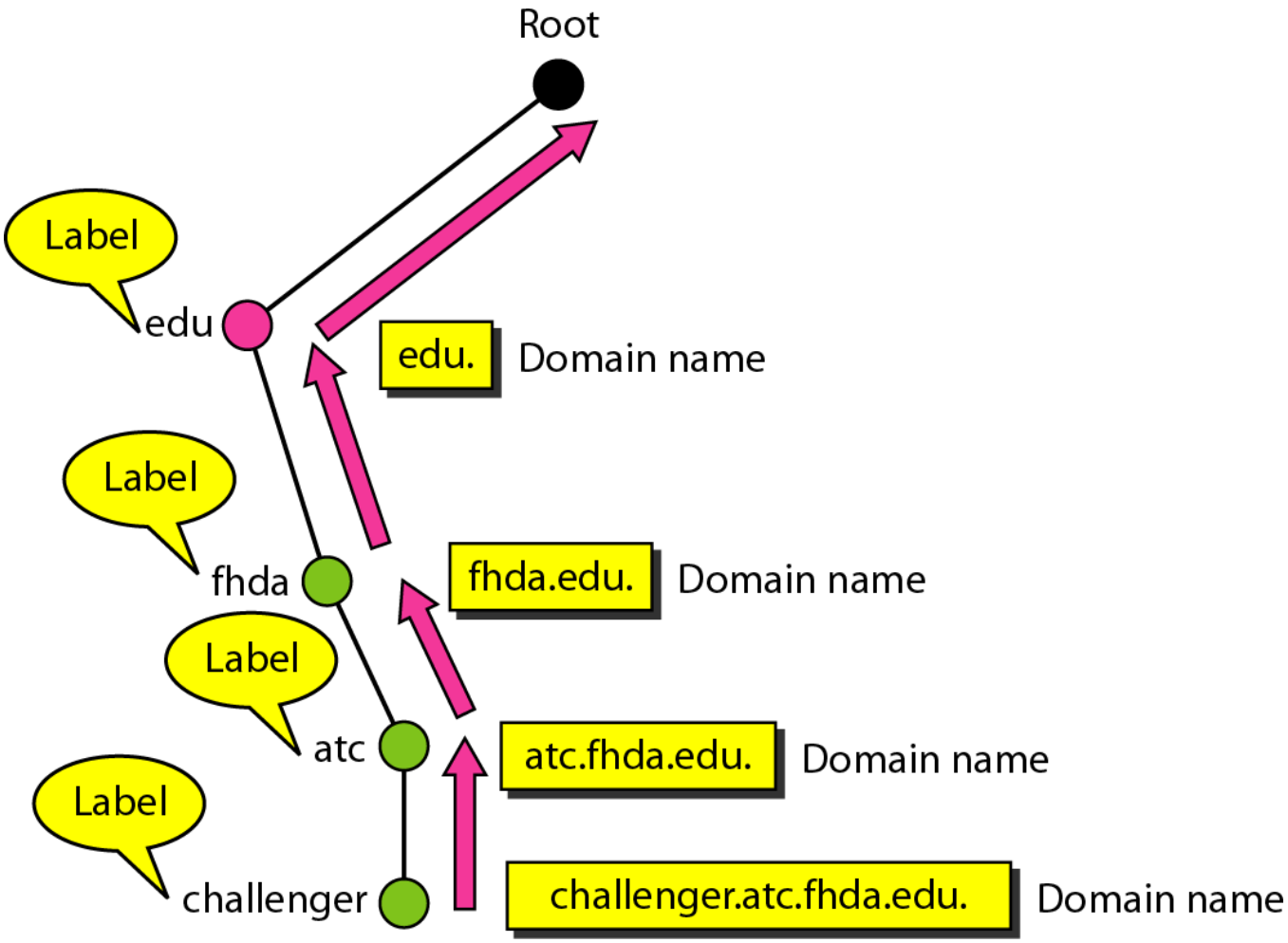
Each node in the tree has a label, which is a string with a maximum of 63 characters. The root label is a null string (empty string). DNS requires that children of a node (nodes that branch from the same node) have different labels, which guarantees the uniqueness of the domain names.

Domain Name

Each node in the tree has a domain name. A full domain name is a sequence of labels separated by dots (.). The domain names are always read from the node up to the root. The last label is the label of the root (null). This means that a full domain name always ends in a null label, which means the last character is a dot because the null string is nothing. Below Figure shows some domain names

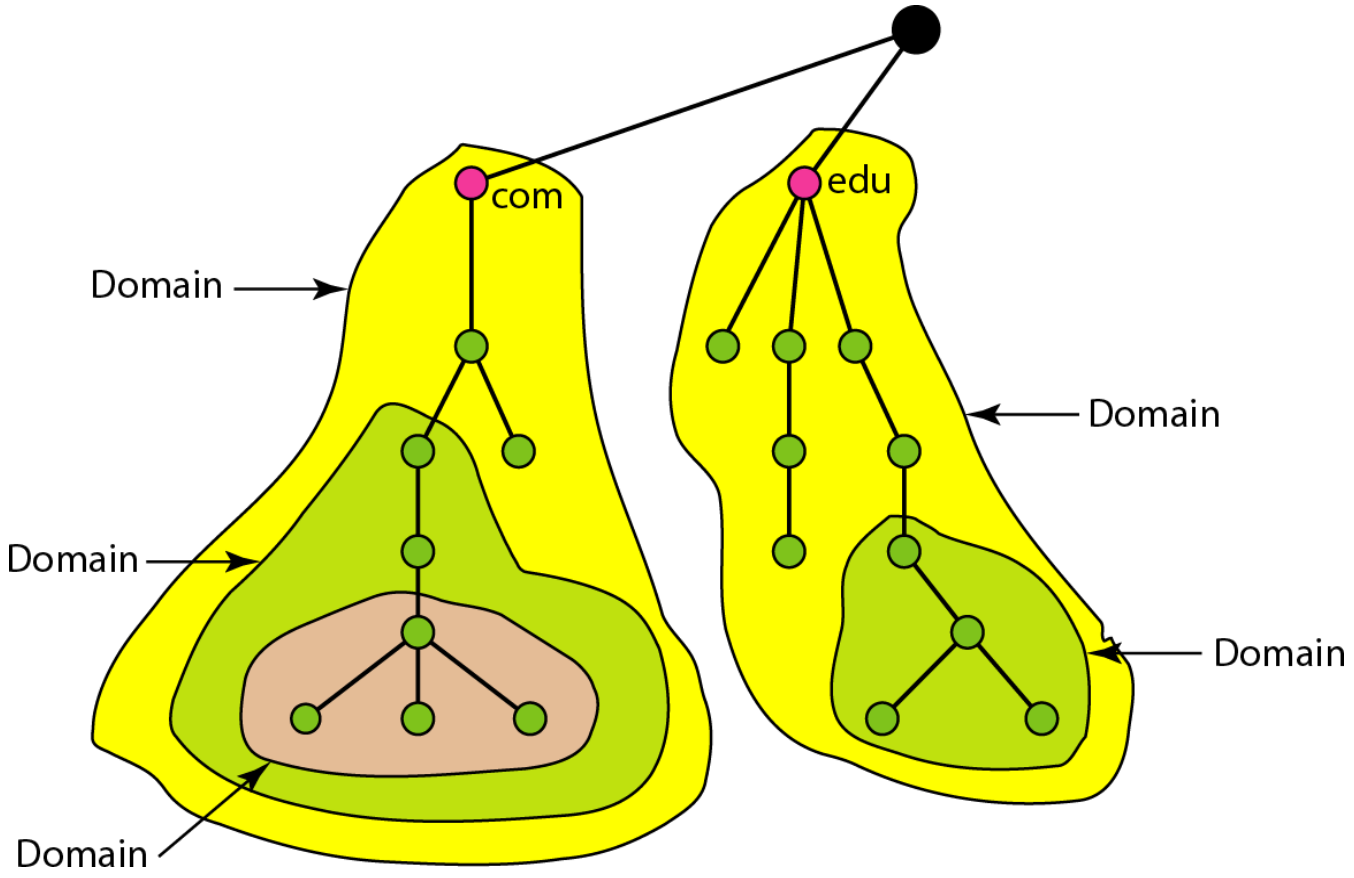


Domain names and labels



Domain

A domain is a subtree of the domain name space. The name of the domain is the domain name of the node at the top of the subtree.

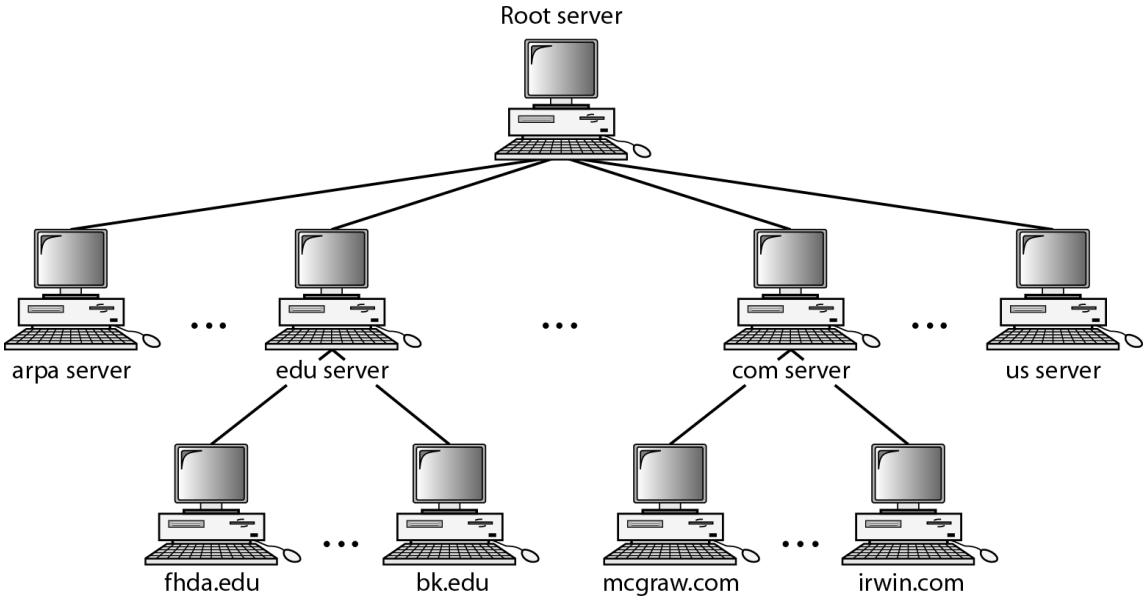


DISTRIBUTION OF NAME SPACE:

The information contained in the domain name space must be stored. However, it is very inefficient and also unreliable to have just one computer store such a huge amount of information. In this section, we discuss the distribution of the domain name space

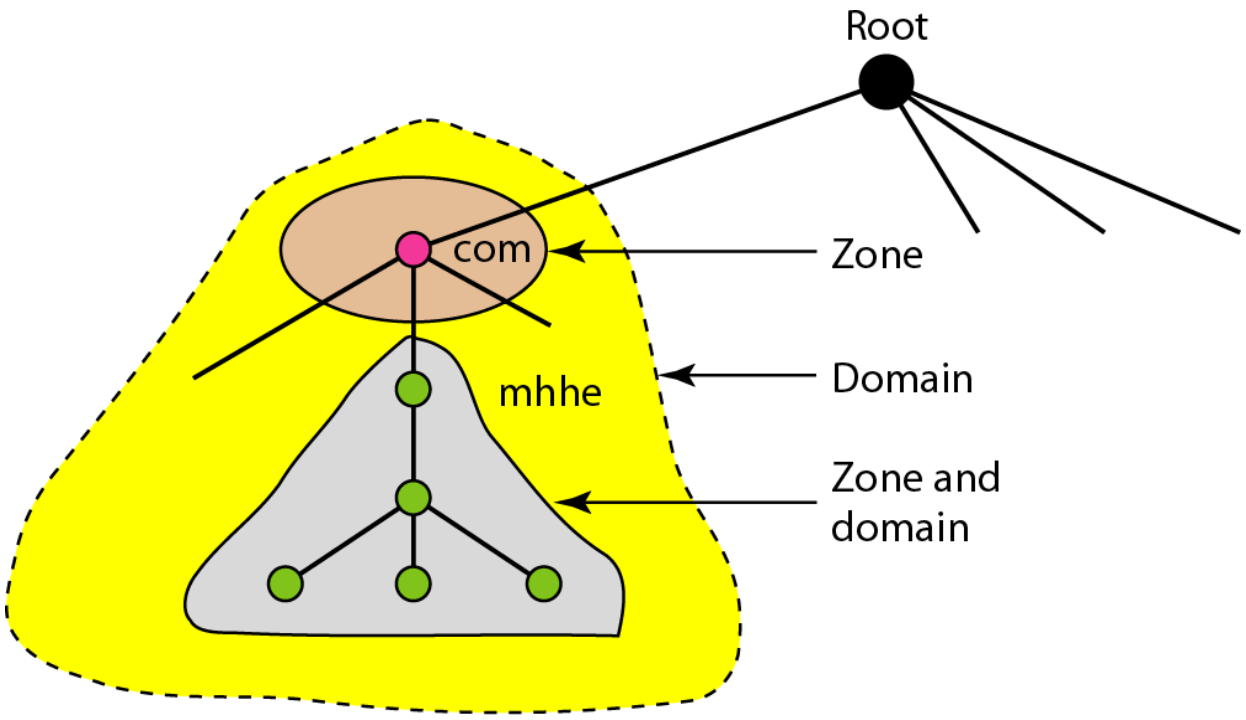
1 Hierarchy of Name Servers

distribute the information among many computers called DNS servers. we let the root stand alone and create as many domains (subtrees) as there are first-level nodes



2 Zone

Since the complete domain name hierarchy cannot be stored on a single server, it is divided among many servers. What a server is responsible for or has authority over is called a zone. We can define a zone as a contiguous part of the entire tree



3 Root Server

A root server is a server whose zone consists of the whole tree. A root server usually does not store any information about domains but delegates its authority to other servers, keeping references to those servers. There are several root servers, each covering the whole domain name space. The servers are distributed all around the world.

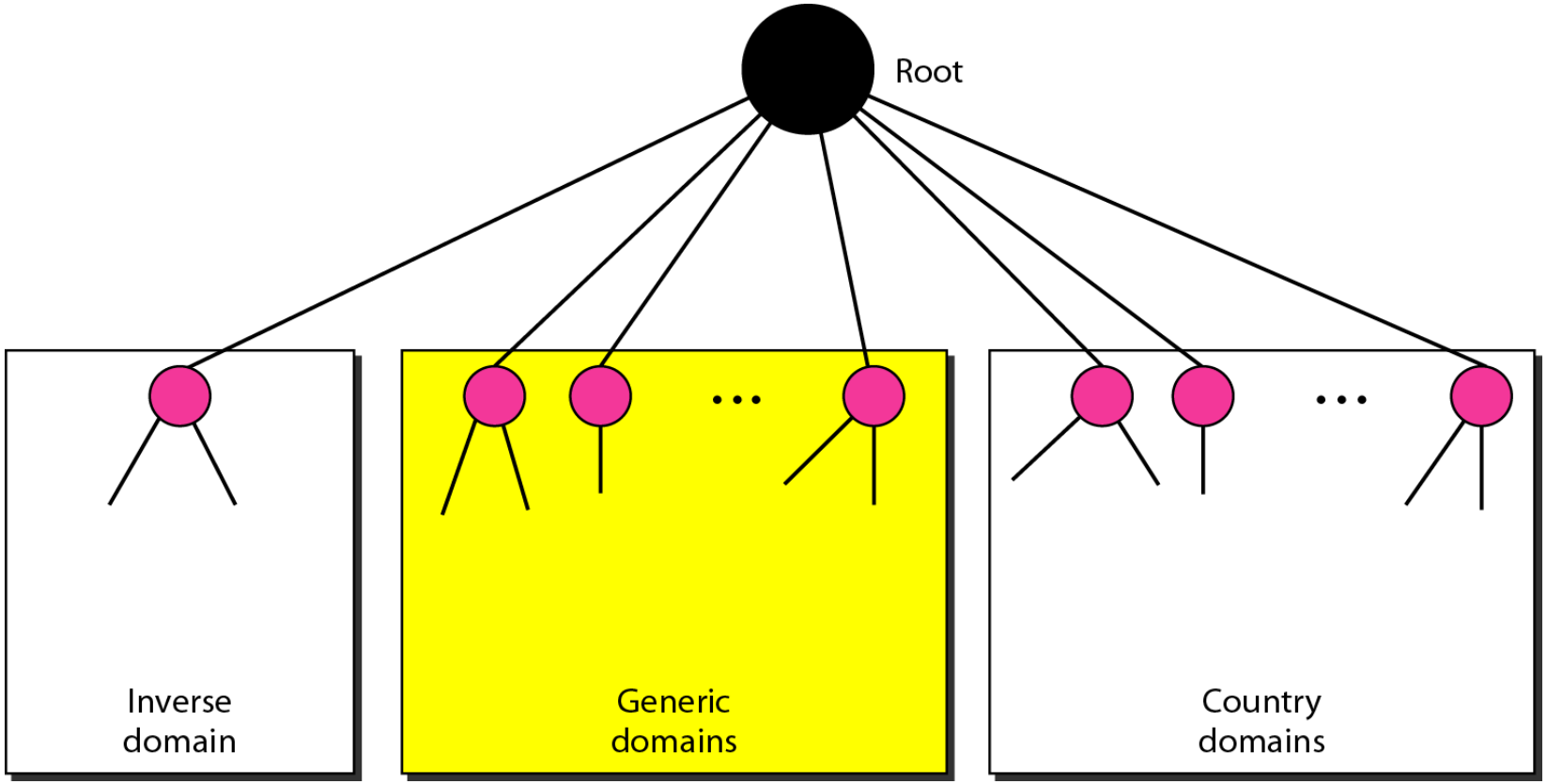
4 Primary and Secondary Servers

A primary server is a server that stores a file about the zone for which it is an authority. It is responsible for creating, maintaining, and updating the zone file. It stores the zone file on a local disk

A secondary server is a server that transfers the complete information about a zone from another server (primary or secondary) and stores the file on its local disk. The secondary server neither creates nor updates the zone files

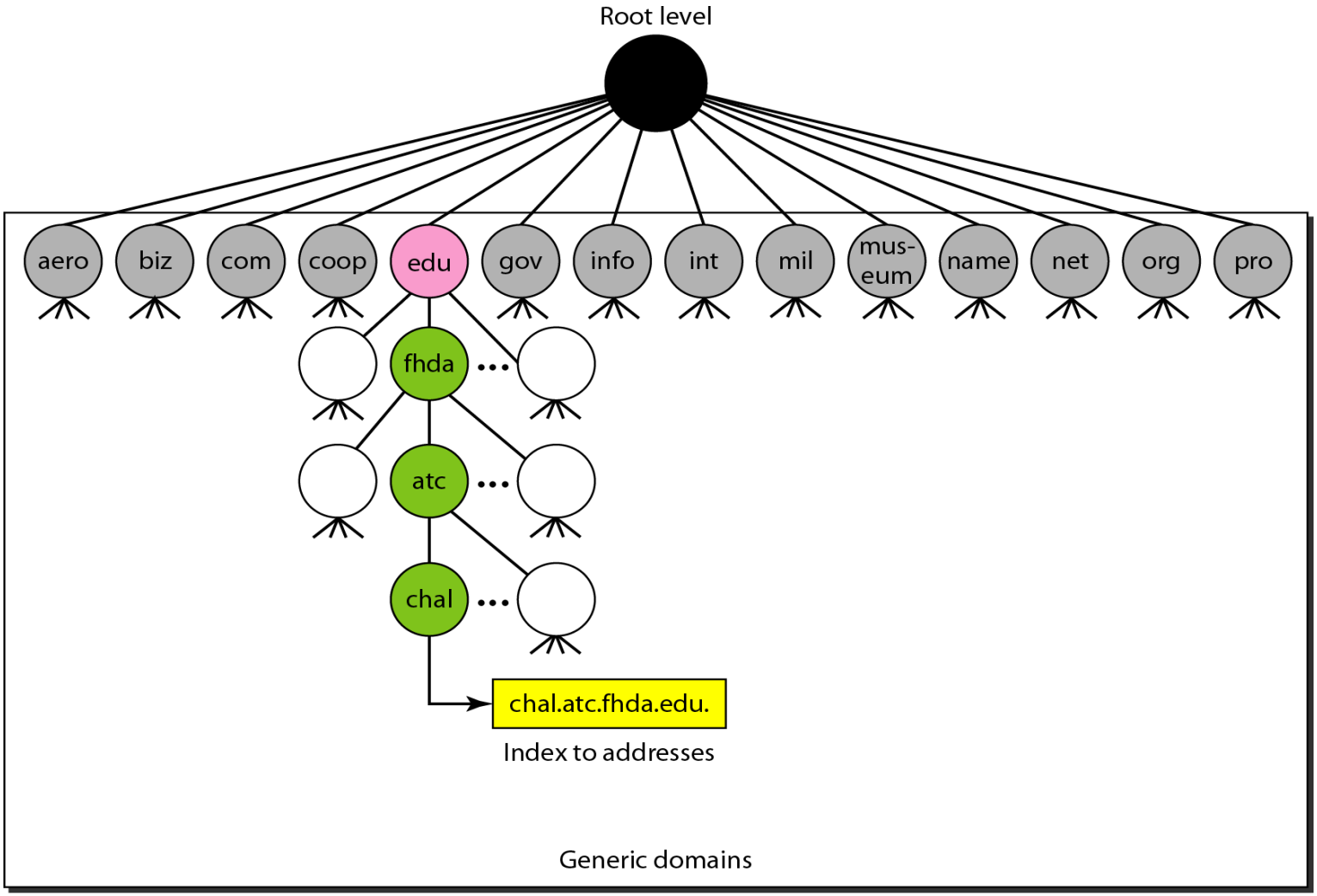
DNS IN THE INTERNET

DNS is a protocol that can be used in different platforms. In the Internet, the domain name space (tree) is divided into three different sections: generic domains, country domains, and the inverse domain



1 Generic Domains

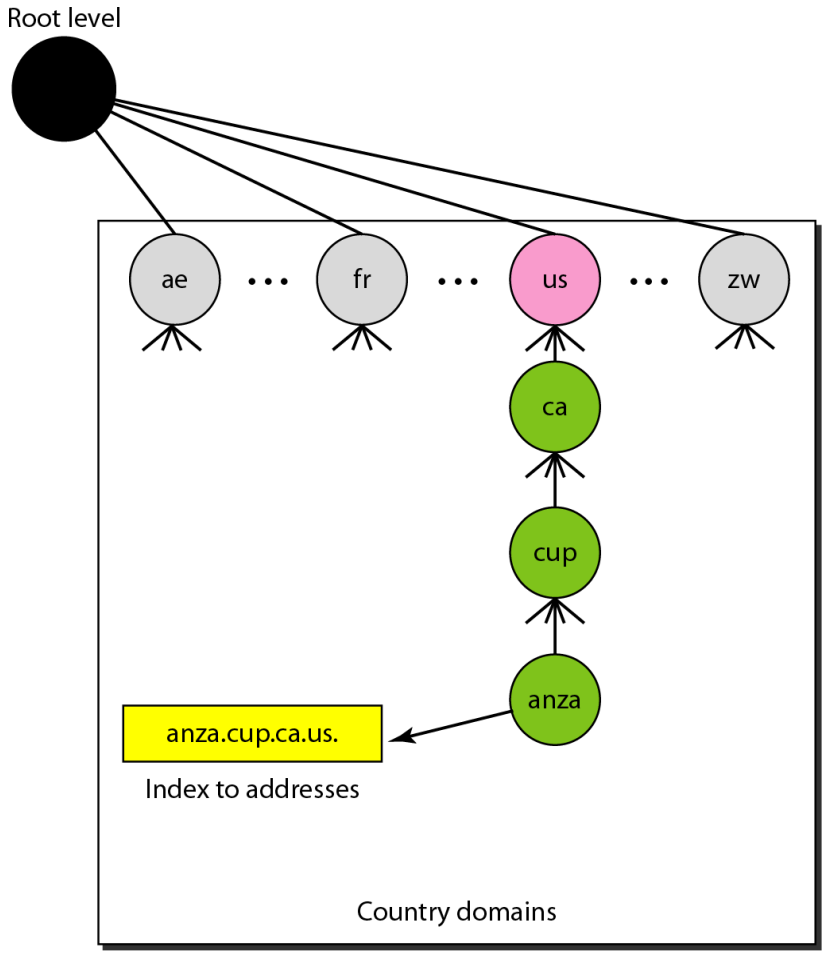
The generic domains define registered hosts according to their generic behavior. Each node in the tree defines a domain, which is an index to the domain name space database



Label	*Description*
aero	Airlines and aerospace companies
biz	Businesses or firms (similar to “com”)
com	Commercial organizations
coop	Cooperative business organizations
edu	Educational institutions
gov	Government institutions
info	Information service providers
int	International organizations
mil	Military groups
museum	Museums and other nonprofit organizations
name	Personal names (individuals)
net	Network support centers
org	Nonprofit organizations
pro	Professional individual organizations

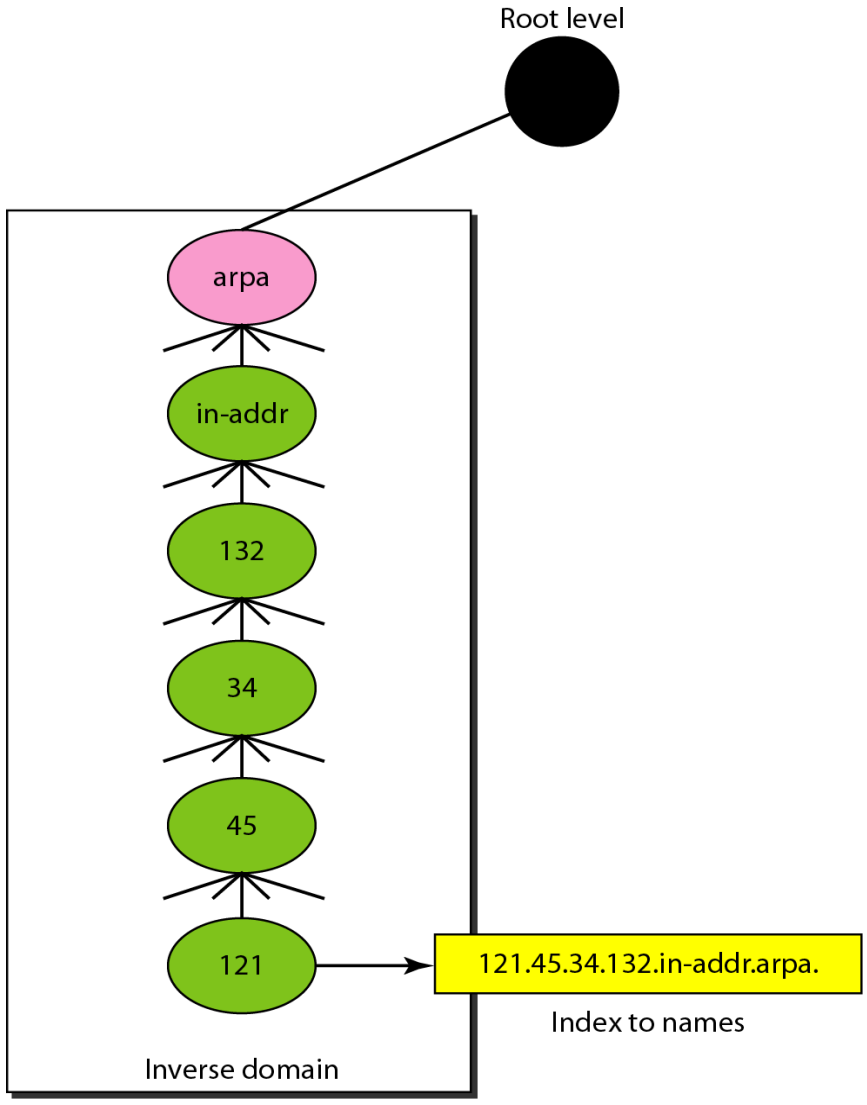
2 Country Domains

The country domains section uses two-character country abbreviations (e.g., us for United States). Second labels can be organizational, or theycan be more specific, national designations. The United States, for example,uses state abbreviations as a subdivision of us (e.g., ca.us.).



3 Inverse Domain

The inverse domain is used to map an address to a name.



RESOLUTION

Mapping a name to an address or an address to a name is called name-address resolution

1 Resolver

DNS is designed as a client/server application. A host that needs to map an address to a name or a name to an address calls a DNSclient called a resolver. The resolver accesses the closest DNS server with a mapping request. If the server has the information, it satisfies the resolver; otherwise, it either refers the resolver to other servers or asks other servers to provide the information.

2 Mapping Names to Addresses

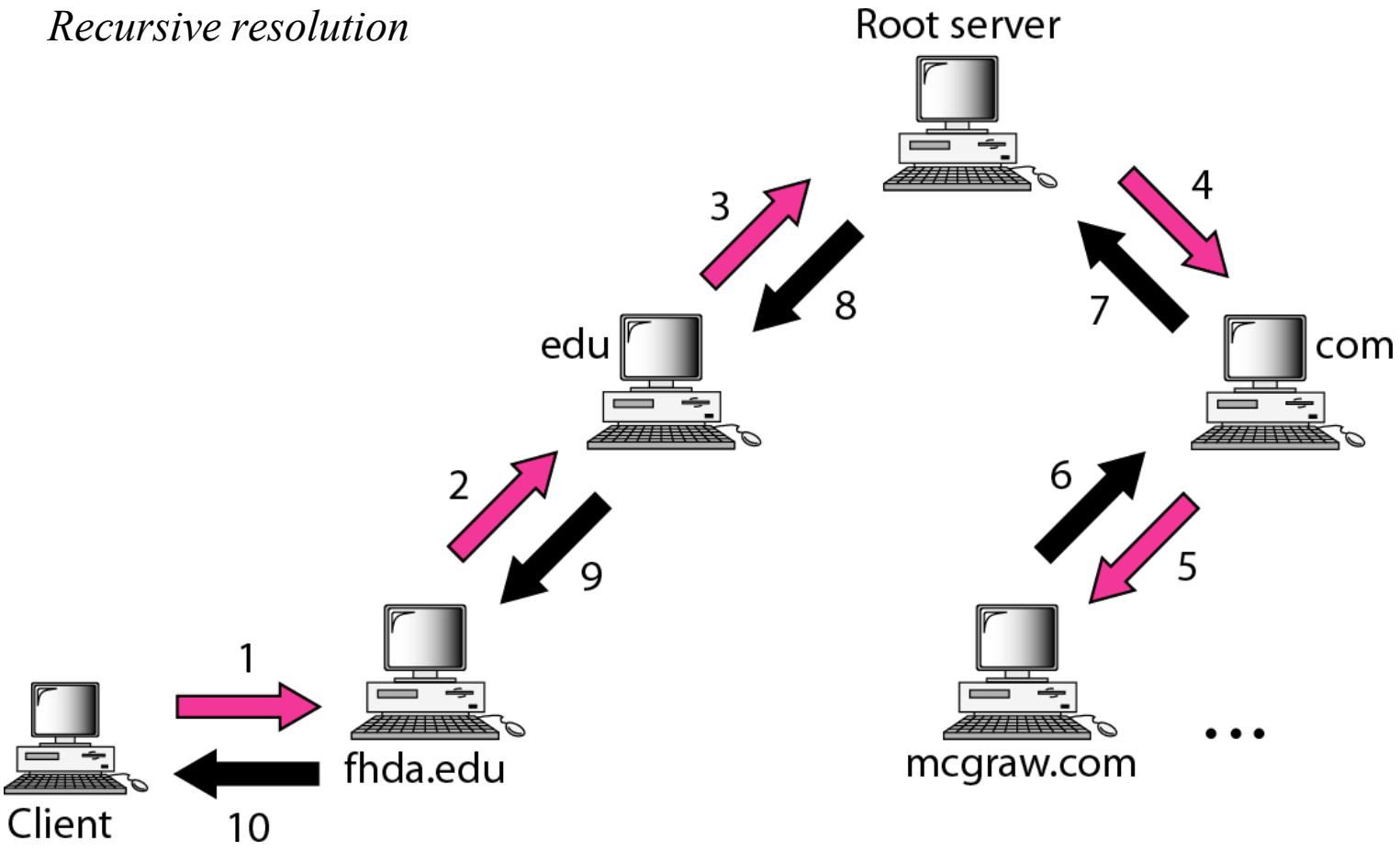
In this case, the server checks the generic domains or the country domains to find the mapping.

3 Mapping Addresses to Names

To answer queries of this kind, DNS uses the inverse domain4 Recursive Resolution

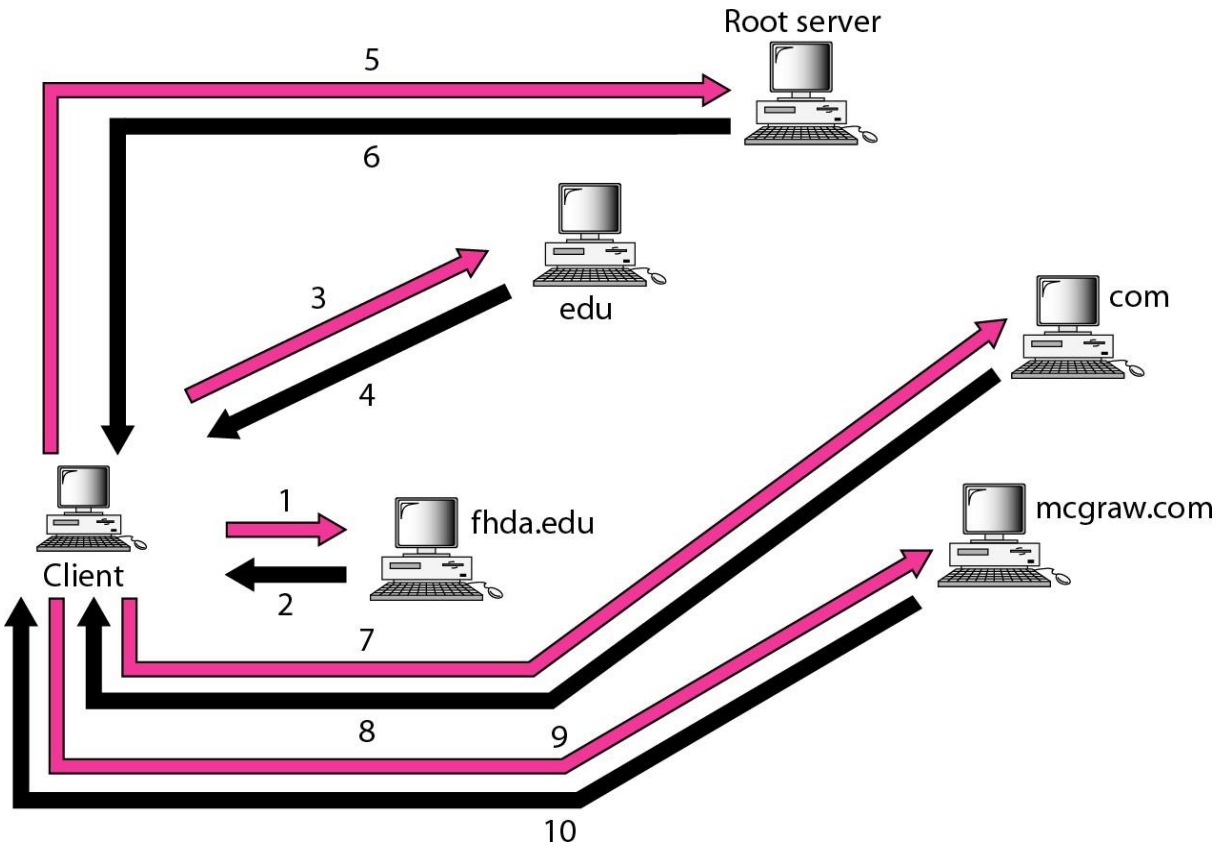
The client (resolver) can ask for a recursive answer from a name server. This means that the resolver expects the server to supply the final answer.. When the query is finally resolved, the response travels back until it finally reaches the requesting client. This is called recursive resolution and is shown in FIG

Recursive resolution



5 Iterative Resolution

If the client does not ask for a recursive answer, the mapping can be done iteratively. If the server is an authority for the name, it sends the answer. If it is not, it returns (to the client) the IP address of the server that it thinks can resolve the query



6 Caching

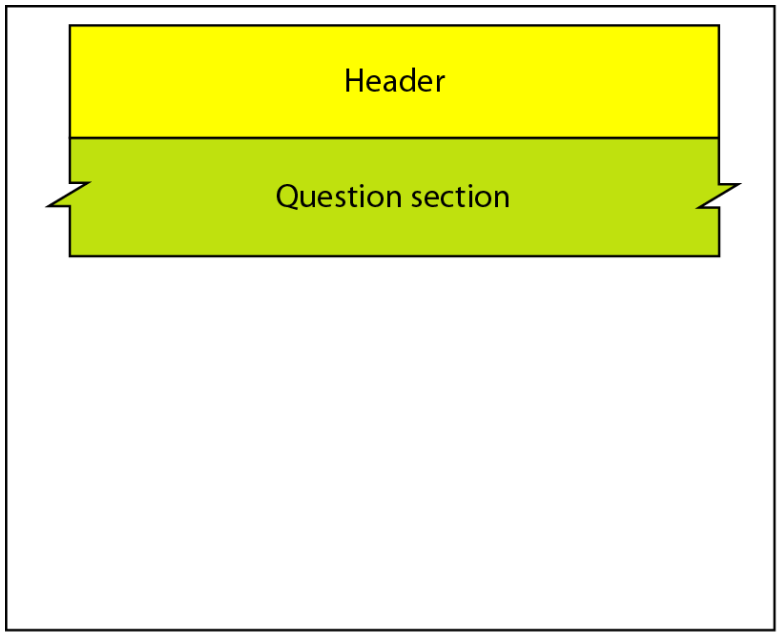
Each time a server receives a query for a name that is not in its domain, it needs to search its database for a server IP address. Reduction of this search time would increase efficiency. DNS handles this with a mechanism called caching

DNS MESSAGES

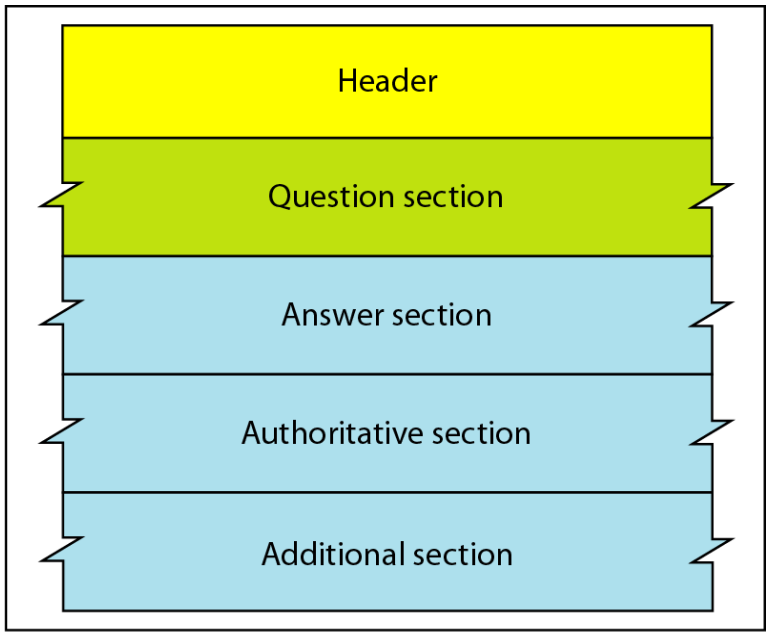
DNS has two types of messages: query and response. Both types have the same format.

The query message consists of a header,
and question records;

the response message consists of a header,
question records,
answer records,
authoritative records,
and additional records



a. Query



b. Response

Header

Both query and response messages have the same header format with some fields set to zero for the query messages. The header is 12 bytes,

Identification	Flags
Number of question records	Number of answer records (all 0s in query message)
Number of authoritative records (all 0s in query message)	Number of additional records (all 0s in query message)

TYPES OF RECORDS

The question records are used in the question section of the query and response messages. The resource records are used in the answer, authoritative, and additional information sections of the responsemessage.

Question Record

A question record is used by the client to get information from a server..

Resource Record

Each domain name (each node on the tree) is associated with a record called the resource record. The server database consists of resource records. Resource records are also what is returned by the server to the client.

REGISTRARS

How are new domains added to DNS? This is done through a registrar, a commercial entity accredited by ICANN. A registrar first verifies that the requested domain name is unique and then enters it into the DNS database. A fee is charged. Today, there are many registrars; their names and addresses can be found at http://www.intenic.net

DYNAMIC DOMAIN NAME SYSTEM (DDNS)

In DNS, when there is a change, such as adding a new host, removing a host, or changing an IP address, the change must be made to the DNS master file. The size of today's Internet does not allow for this kind of manual operation.

The DNS master file must be updated dynamically. The Dynamic Domain Name System (DDNS) therefore was devised to respond to this need.

ENCAPSULATION

DNS can use either UDP or TCP. In both cases the well-known port used bythe server is port 53.