UNIT - IV

The Transport Layer

Contents

Transport Layer:

•Services provided to the upper layers
•elements of transport protocol
addressing, connection establishment, Connection release
Error Control & Flow Control
•Crash Recovery.

The Internet Transport Protocols:

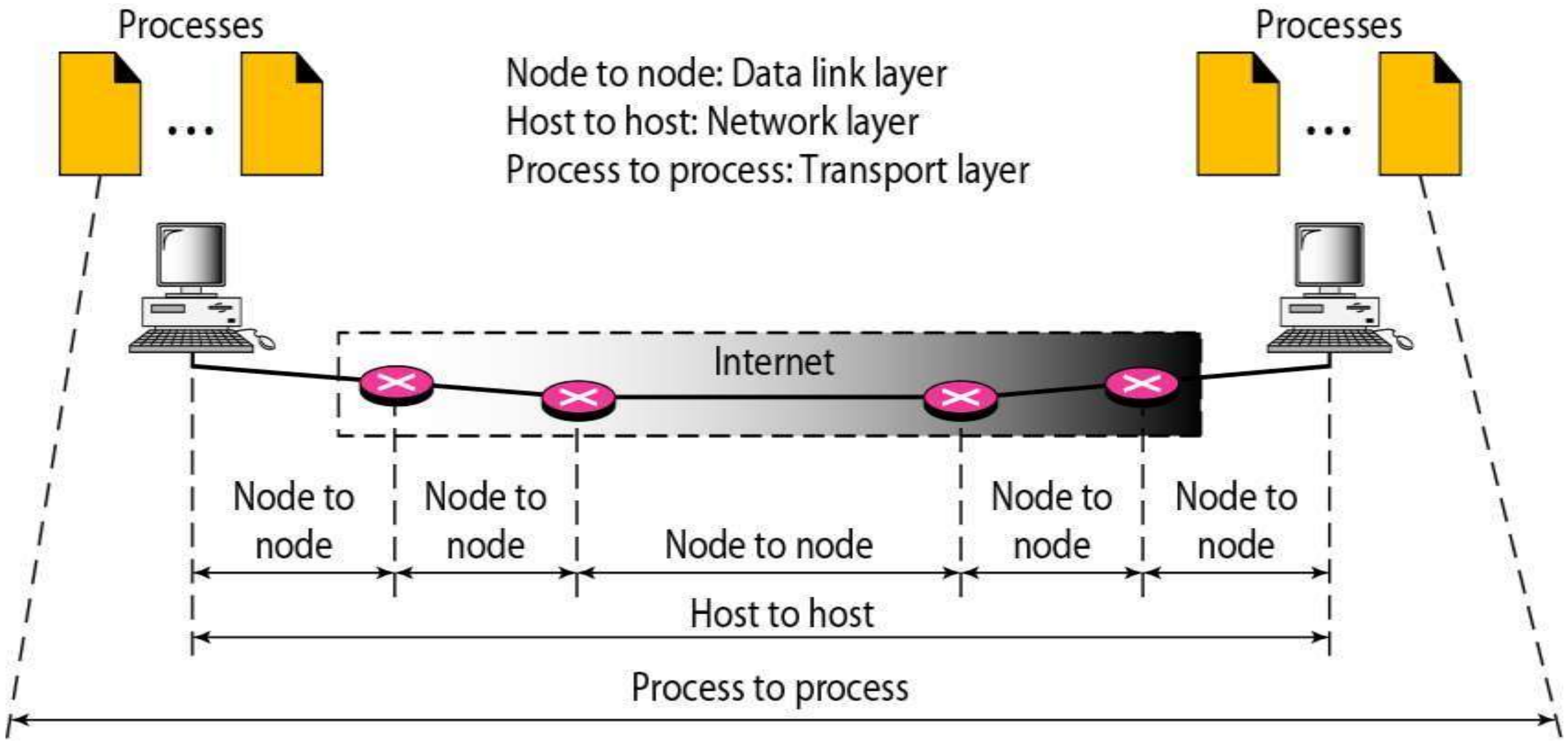
•UDP, Introduction to TCP, The TCP Service Model,
The TCP Segment Header
•The Connection Establishment
•The TCP Connection Release
•The TCP Sliding Window
•The TCP Congestion Control Algorithm.

Transport Layer

- The main role of the transport layer is to provide the communication services directly to the application processes running on different hosts.
- The transport layer provides a logical communication between application processes running on different hosts. Although the application processes on different hosts are not physically connected, application processes use the logical communication provided by the transport layer to send the messages to each other.
- The transport layer protocols are implemented in the end systems but not in the network routers.

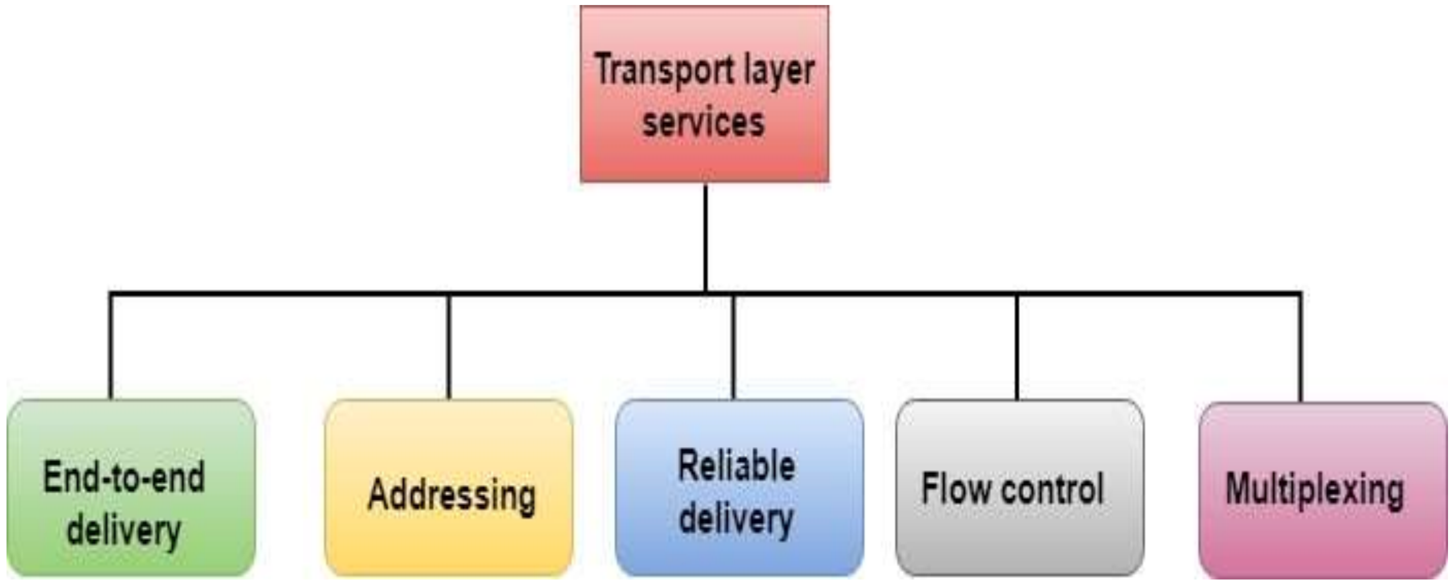
PROCESS-TO-PROCESS DELIVERY

- The data link layer is responsible for delivery of frames between two neighboring nodes over a link. This is called node-to-node delivery.
- The network layer is responsible for delivery of datagrams between two hosts. This is called host-to-host delivery.
- Communication on the Internet is not defined as the exchange of data between two nodes or between two hosts. Real communication takes place between two processes (application programs). We need process-to-process delivery.
- The transport layer is responsible for process-to-process delivery- the delivery of a packet, part of a message, from one process to another.


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

Services provided by the Transport Layer

- The services provided by the transport layer are similar to those of the data link layer. The data link layer provides the services within a single network while the transport layer provides the services across an internetwork made up of many networks. The data link layer controls the physical layer while the transport layer controls all the lower layers.



Services Provided to the Upper Layers

- The ultimate goal of the transport layer is to provide efficient, reliable, and cost-effective service to its users, normally processes in the application layer.
- To achieve this goal, the transport layer makes use of the services provided by the network layer.
- The hardware and/or software within the transport layer that does the work is called the transport entity.
- The (logical) relationship of the network, transport, and application layers is illustrated in below figure

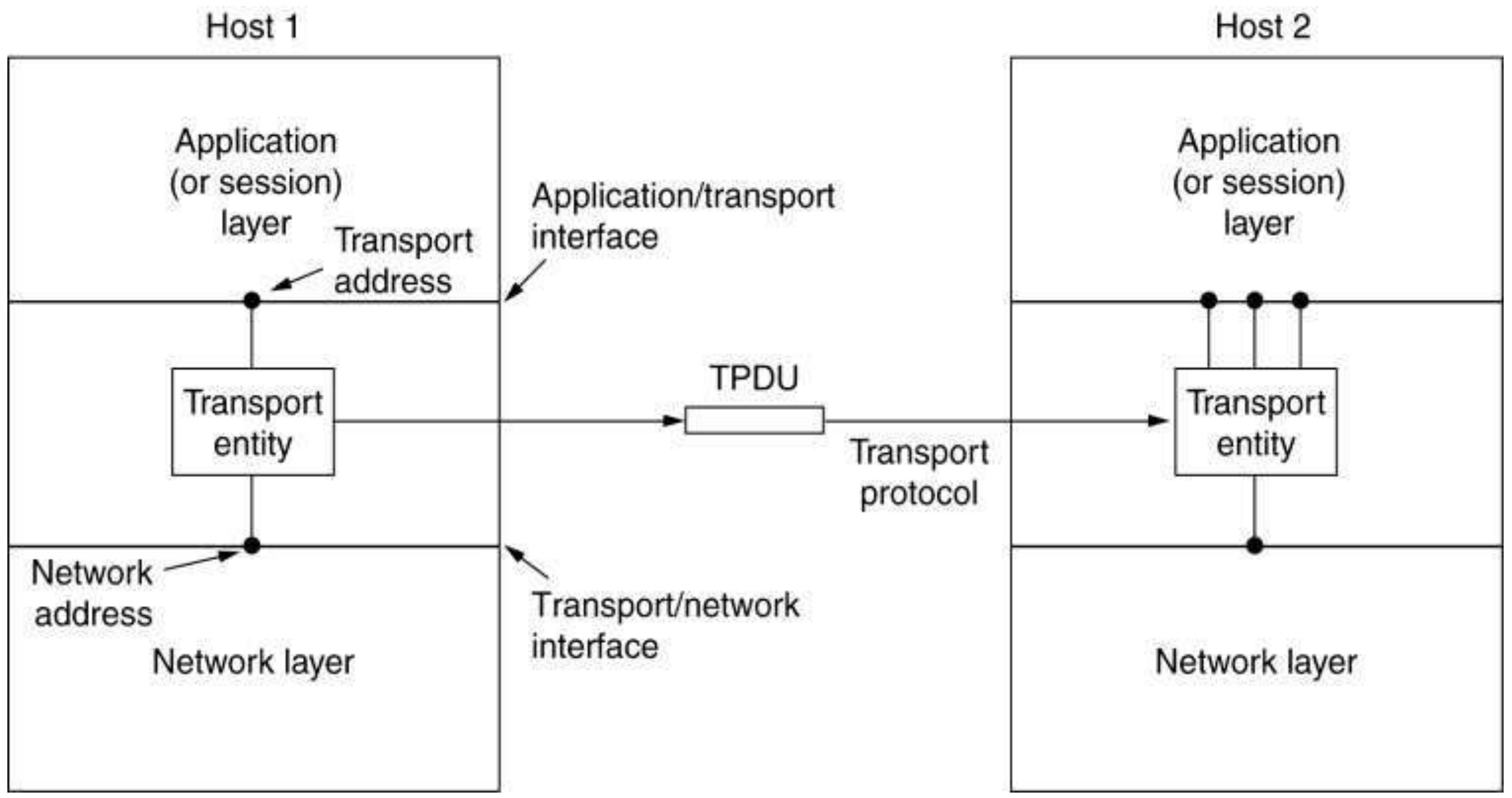
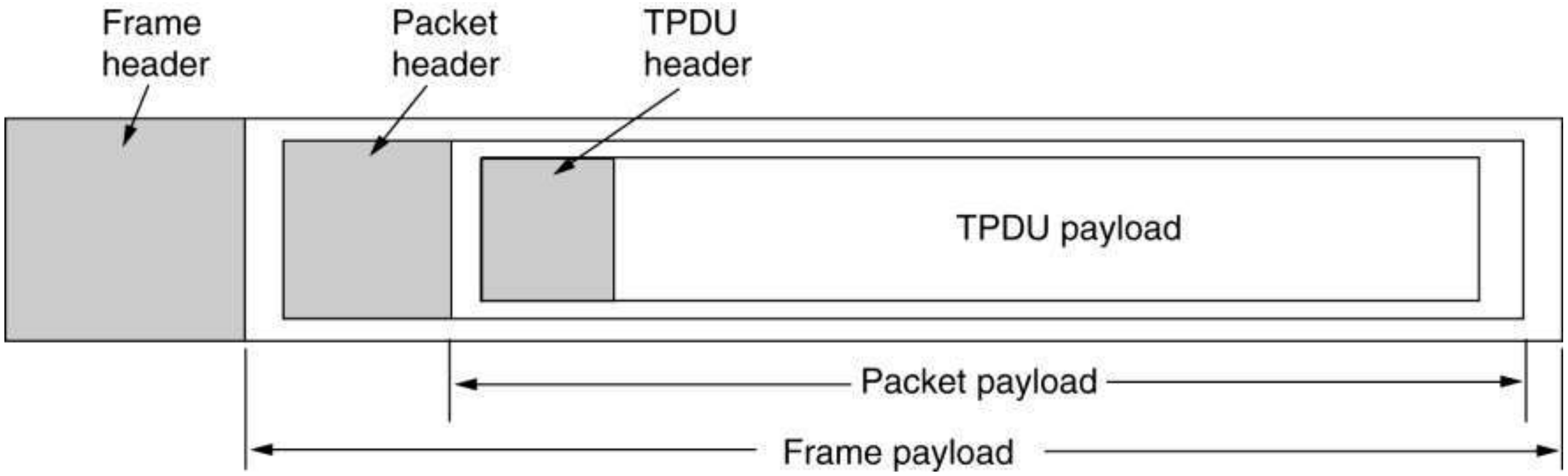


Fig. 6-1 The network, transport, and application layers.

TPDU

TPDU (Transport Protocol Data Unit) is a term used for messages sent from transport entity to transport entity. Thus, TPDUs (exchanged by the transport layer) are contained in packets (exchanged by the network layer). In turn, packets are contained in frames (exchanged by the data link layer). When a frame arrives, the data link layer processes the frame header and passes the contents of the frame payload field up to the network entity. The network entity processes the packet header and passes the contents of the packet payload up to the transport entity. This nesting is illustrated in below figure



The nesting of TPDUs, packets, and frames.

- **The connection-oriented and the connectionless**
- **Why need the transport layer.**

- Just as there are two types of network service, connection-oriented and connectionless, there are also two types of transport service. The transport service is similar to the network service in many ways.

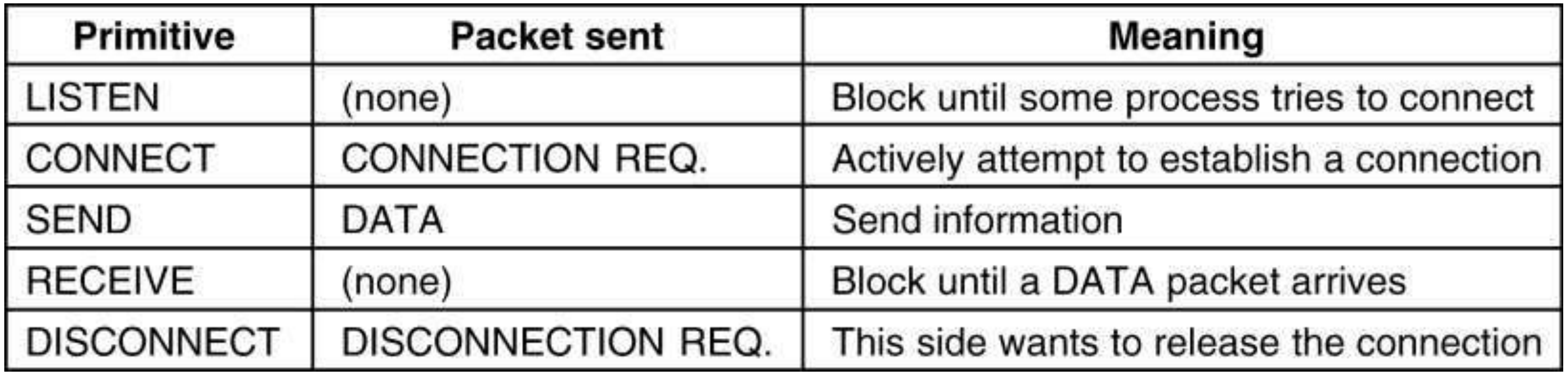
- The transport code runs entirely on the users' machines, but the network layer mostly runs on the routers, which are operated by the carrier (at least for a wide area network). What happens if the network layer offers inadequate service? Suppose that it **frequently loses packets**? What happens if routers **crash from time to time**?

- Problems occur, that's what. **The users have no real control over the network layer,** so they cannot solve the problem of poor service by using better routers or putting more error handling in the data link layer. The only possibility is to put on top of the network layer another layer that improves the quality of the service.

In essence, the existence of the transport layer makes it possible for the transport service to be more reliable than the underlying network service. Lost packets and mangled data can be detected and compensated for by the transport layer. Furthermore, the transport service primitives can be implemented as calls to library procedures in order to make them independent of the network service primitives.

Transport Service Primitives

Transport primitives are very important, because many programs (and thus programmers) see the transport primitives. Consequently, the transport service must be convenient and easy to use.



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

Fig. The primitives for a simple transport service.

Elements of Transport Protocols

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

Addressing

- Whenever we need to deliver something to one specific destination among many, we need an address. At the data link layer, we need a MAC address to choose one node among several nodes if the connection is not point-to-point.
- A frame in the data link layer needs a destination MAC address for delivery and a source address for the next node's reply.
- At the network layer, we need an IP address to choose one host among millions.
- A datagram in the network layer needs a destination IP address for delivery and a source IP address for the destination's reply.
- At the transport layer, we need a transport layer address, called a port number, to choose among multiple processes running on the destination host. The destination port number is needed for delivery; the source port number is needed for the reply.

- In the Internet model, the port numbers are 16-bit integers between 0 and 65,535. The client program defines itself with a port number, chosen randomly by the transport layer software running on the client host. This is the ephemeral port number

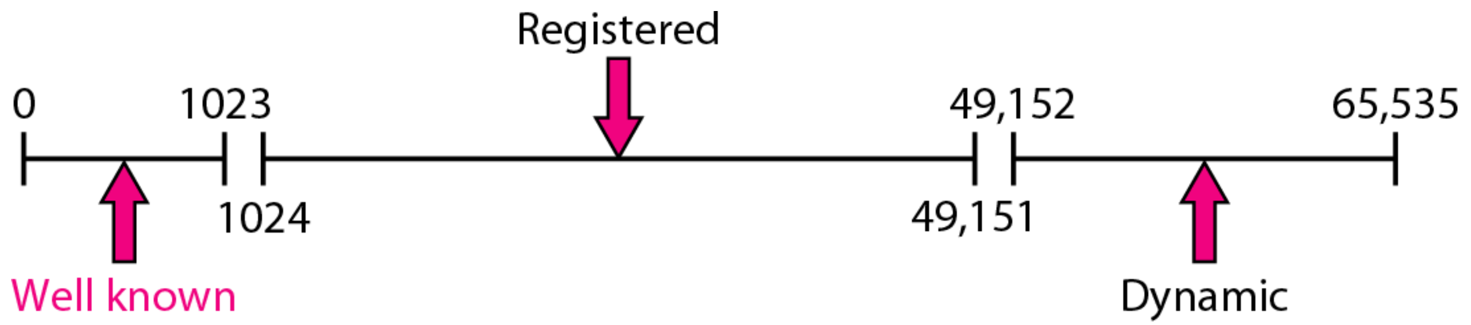


Figure : lANA (Internet Assigned Number Authority) ranges

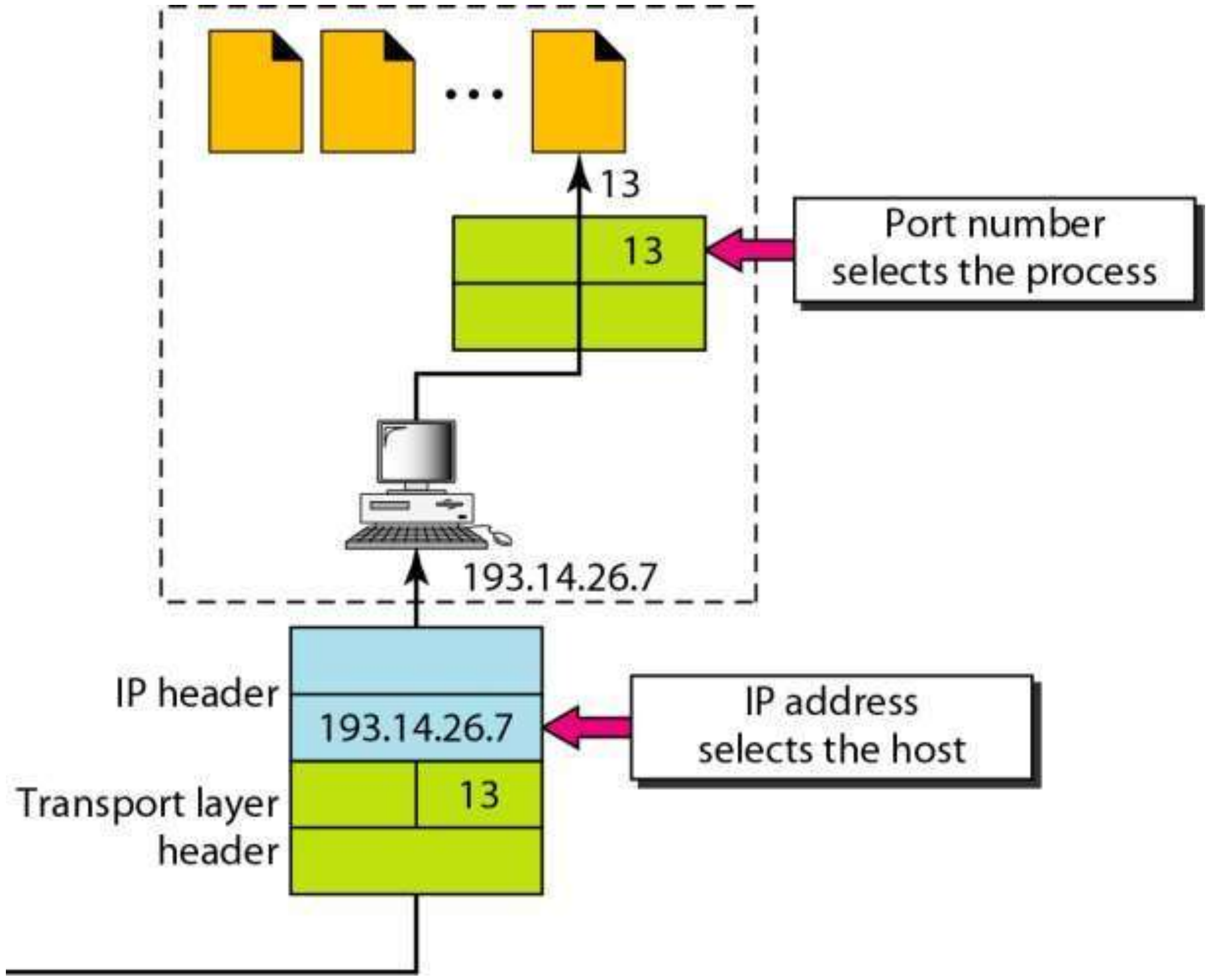


Figure : IP addresses versus port numbers

Addressing

- The method normally used is to define **transport addresses** to which processes can listen for connection requests. In the Internet, these end points are called **ports.** We will use the generic term **TSAP, (Transport Service Access Point)**.
- The analogous end points in the network layer (i.e., network layer addresses) are then called **NSAPs**. IP addresses are examples of NSAPs.

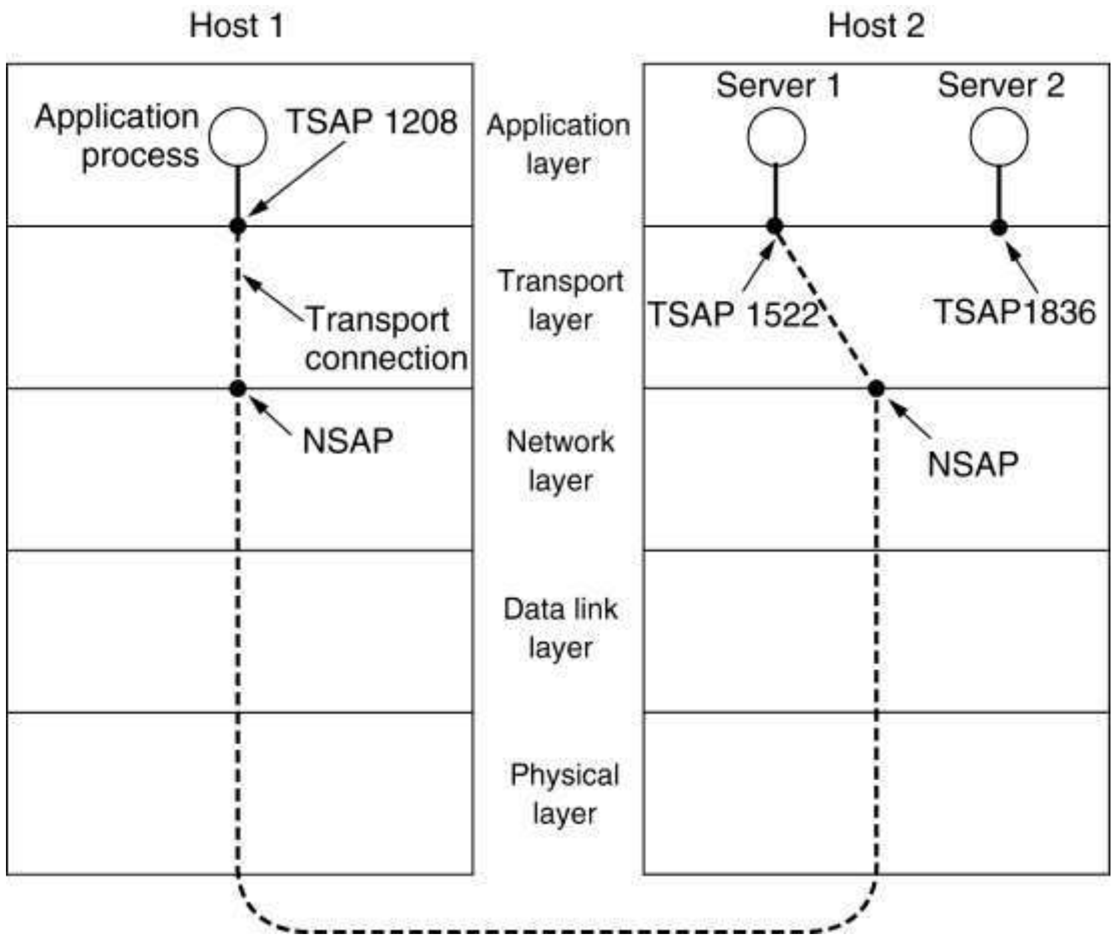


Figure :TSAPs, NSAPs and transport connections.

How does the client process know the server’s TSAP?

Two ways:

1. One is that the server’s TSAP is **well known** TSAP.
2. Another one is that the server host run a special process called a **name server** or sometimes a **directory server**.

•A better scheme is needed for rarely used servers. It is known as the **initial connection protocol**. Each machine that wishes to offer services to remote users has a special **process server** that acts as a proxy for less heavily used servers.

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 TPDU to the destination and wait for a CONNECTION ACCEPTED reply. But the problem occurs when the network can lose, store, and duplicate packets.

- To solve this specific problem,(DELAYED DUPLICATES) Tomlinson (1975) introduced the **three-way handshake.**

- The normal setup procedure when host 1 initiates is shown in 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.

Three-way handshake

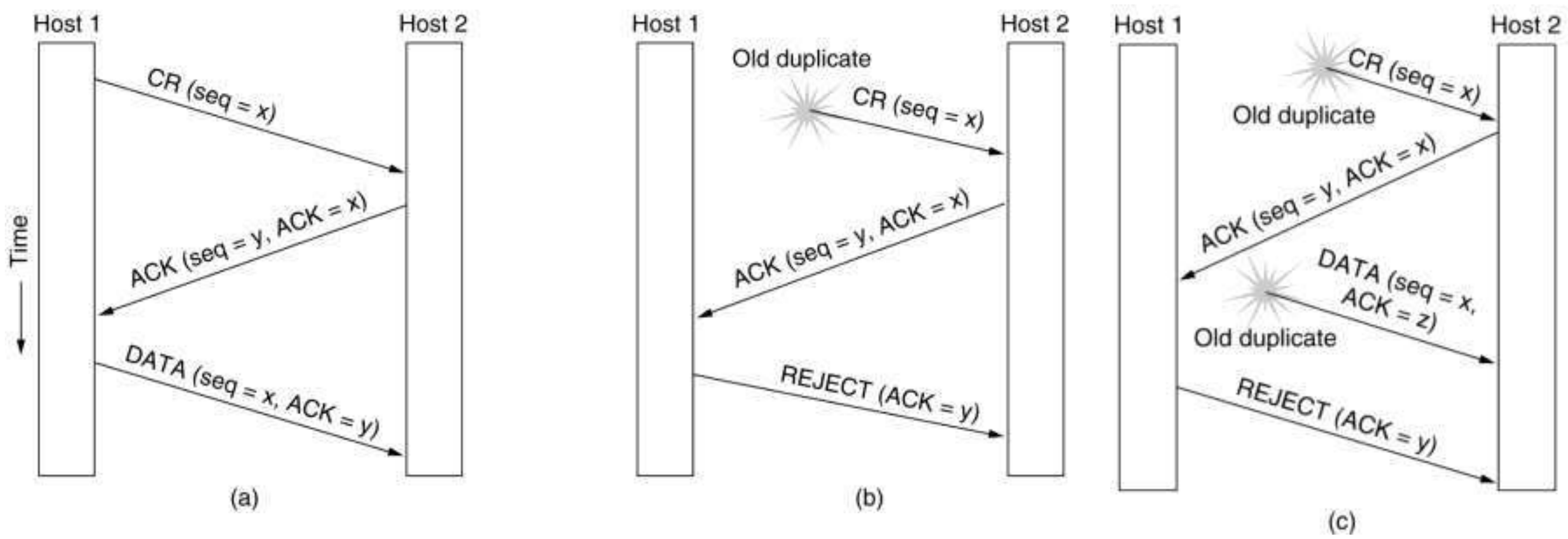


Figure 6-11. Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST.
(a) Normal operation,
(b) Old CONNECTION REQUEST appearing out of nowhere.
(c) Duplicate CONNECTION REQUEST and duplicate ACK.

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 to establish 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

- Asymmetric release
- Symmetric release

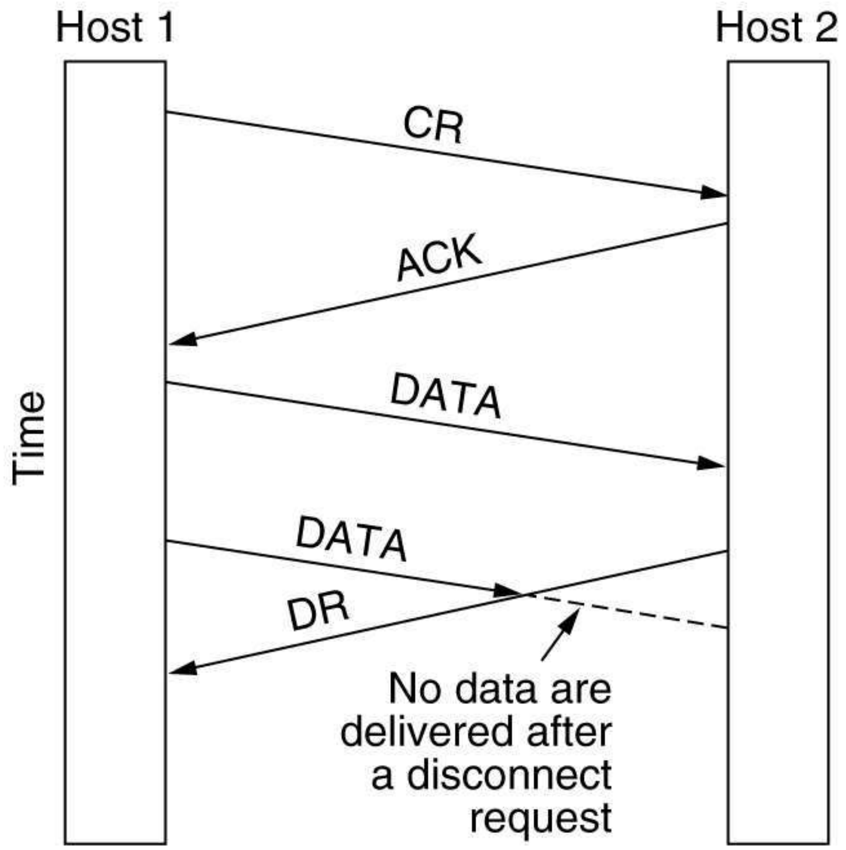


Figure 6-12. Abrupt disconnection with loss of data.

- 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.”

Release Connection Using a Three-way Handshake

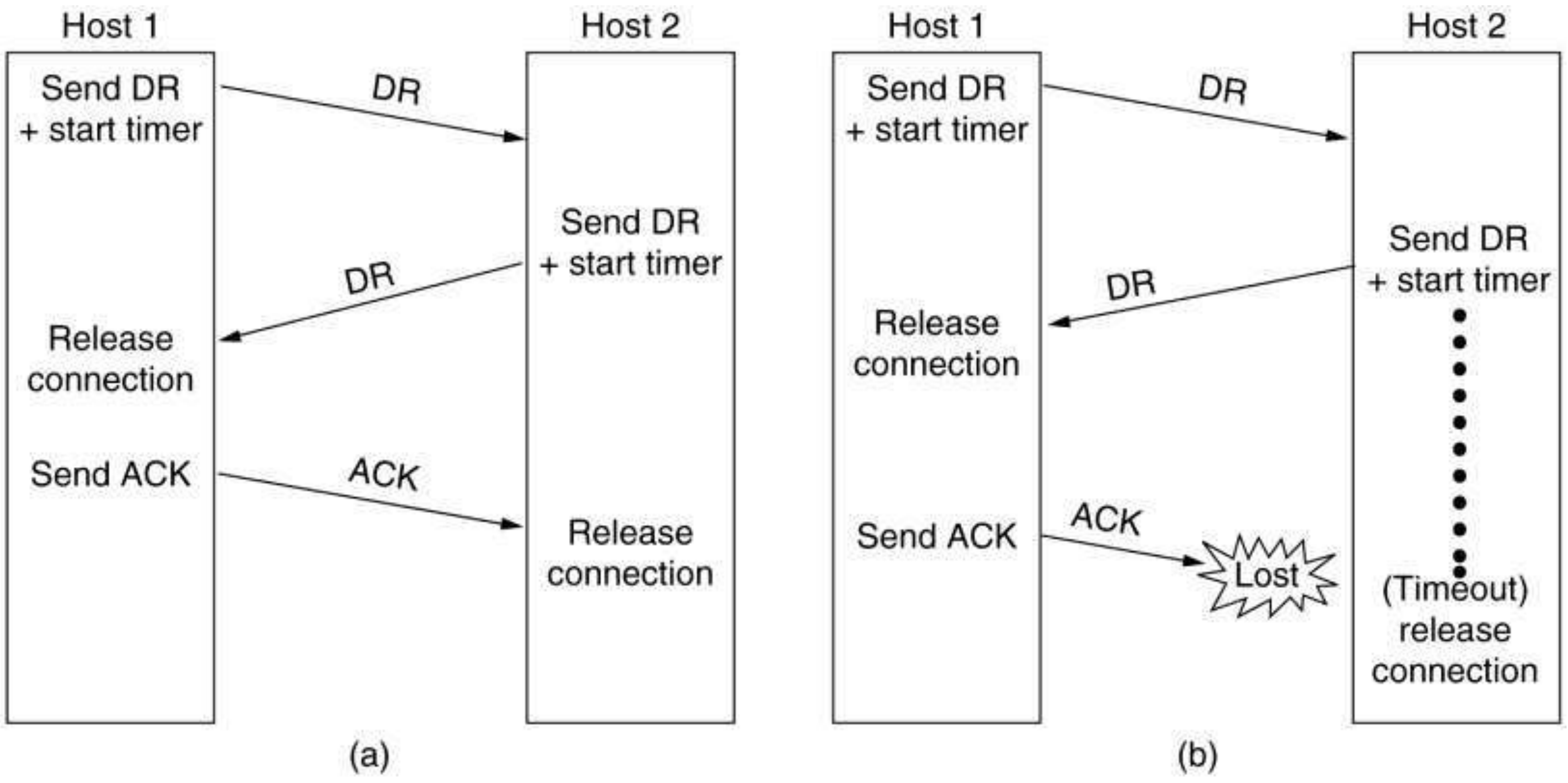
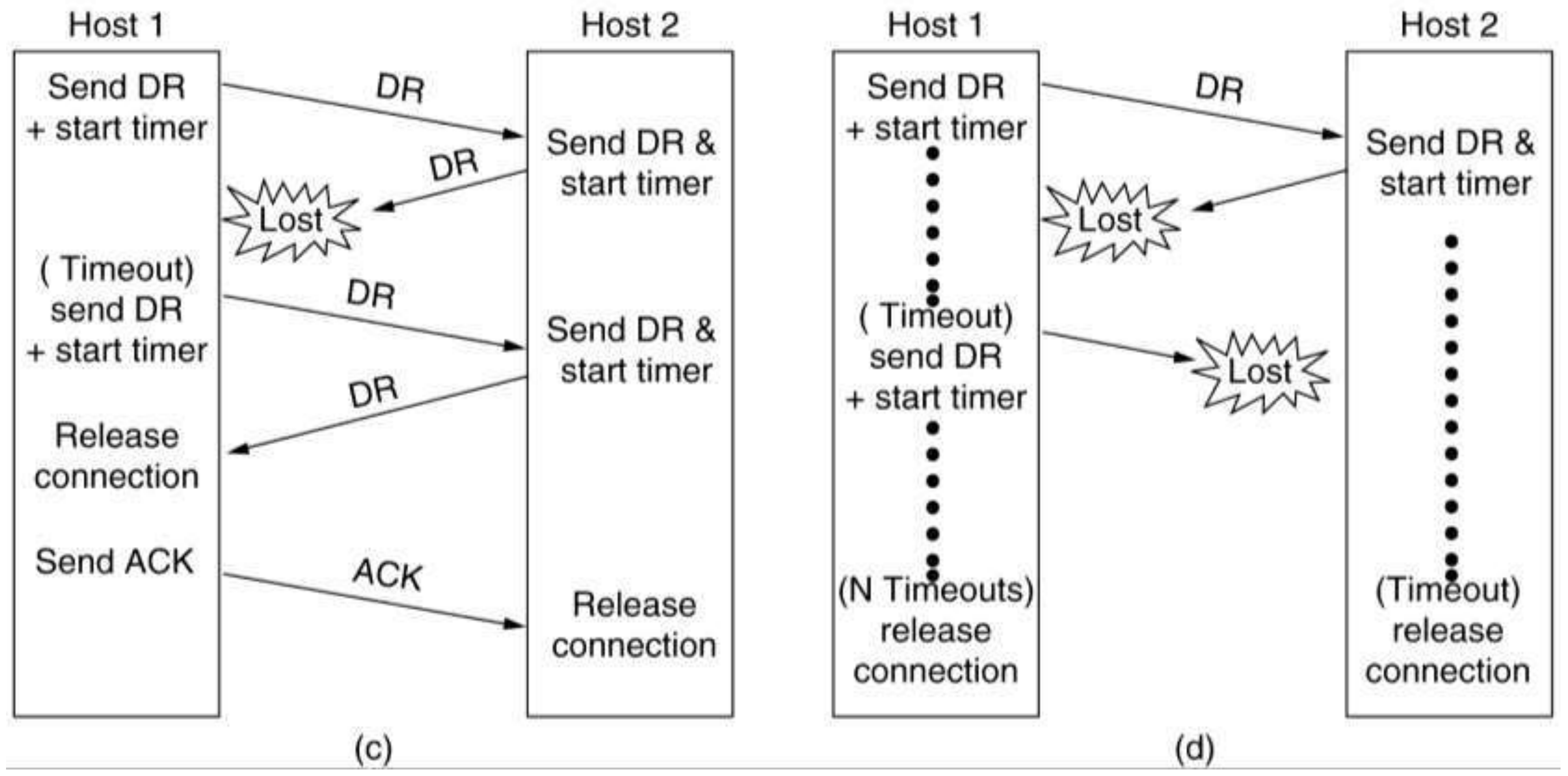


Figure 6-14. Four protocol scenarios for releasing a connection. (a) Normal case of a three-way handshake. (b) final ACK lost.



(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.

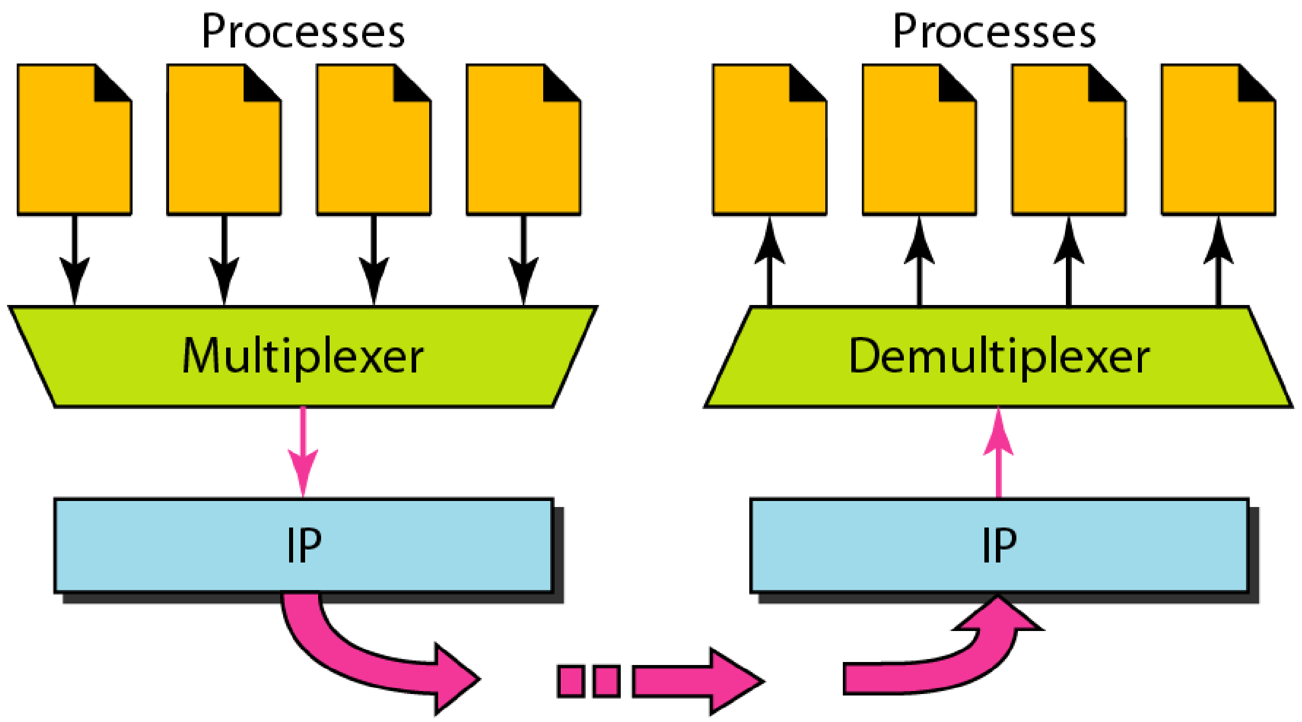
If the final ACK segment is lost, as shown in Fig.(b), the situation is saved by the timer. When the timer expires, the connection is released anyway. Now consider the case of the second DR being lost. The user initiating the disconnection will not receive the expected response, will time out, and will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

Multiplexing and Demultiplexing

The addressing mechanism allows multiplexing and demultiplexing by the transport layer, as shown in below figure.



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

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

Multiplexing

Multiple transport connections use one network connection, called **upward multiplexing**. One transport connection use multiple network connection, called **downward multiplexing**.

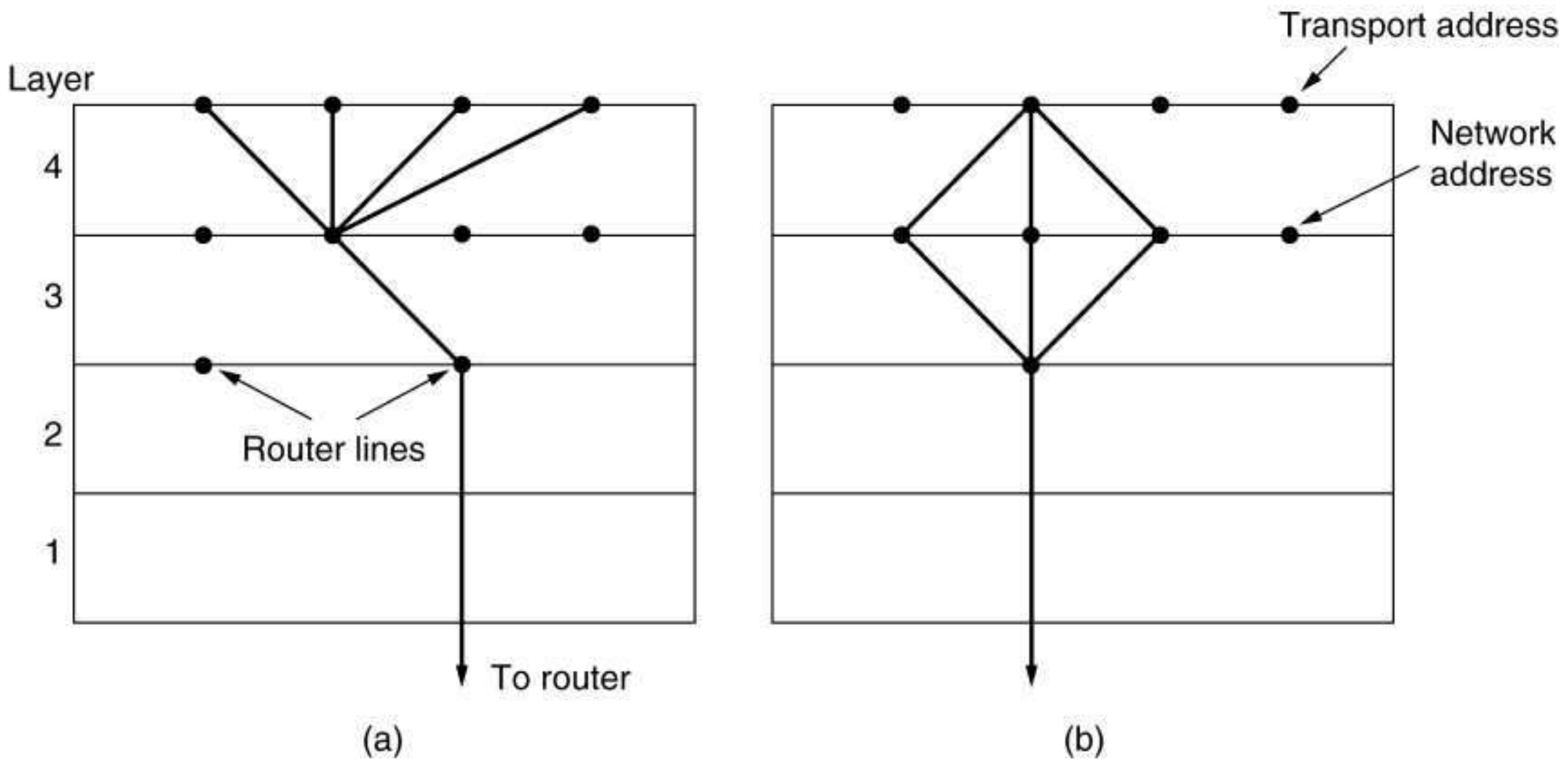


Figure (a) Upward multiplexing. (b) Downward multiplexing.

Crash Recovery

If hosts and routers are subject to crashes, recovery from these crashes becomes an issue. If the transport entity is entirely within the hosts, recovery from network and router crashes is straightforward. A more troublesome problem is how to recover from host crashes.

The host must decide whether to retransmit the most recent TPDU after recovery from a crash.

Eg. A client and a server communication, then the server crashes, see fig.6-18.

No matter how the transport entity is programmed, there are always situations where the protocol fails to recover properly, because the acknowledgement and the write can’t be done at the same time.

Conclusion: When a crash occurred in layer N, only the layer N+1 can recovery.

	Strategy used by receiving host					
	First ACK, then write			First write, then ACK		
Strategy used by sending host	AC(W)	AWC	C(AW)	C(WA)	W AC	WC(A)
Always retransmit	OK	DUP	OK	OK	DUP	DUP
Never retransmit	LOST	OK	LOST	LOST	OK	OK
Retransmit in S0	OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1	LOST	OK	OK	OK	OK	DUP

OK = Protocol functions correctly
DUP = Protocol generates a duplicate message
LOST = Protocol loses a message

Figure Different combinations of client and server strategy.

Part-II

The Internet Transport Protocols

Introduction to UDP

- The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to process communication instead of host-to-host communication. Also, it performs very limited error checking.

- If UDP is so powerless, why would a process want to use it? With the disadvantages come some advantages. UDP is a very simple protocol using a minimum of overhead. If a process wants to send a small message and does not care much about reliability, it can use UDP. Sending a small message by using UDP takes much less interaction between the sender and receiver than using TCP or SCTP.

User Datagram

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

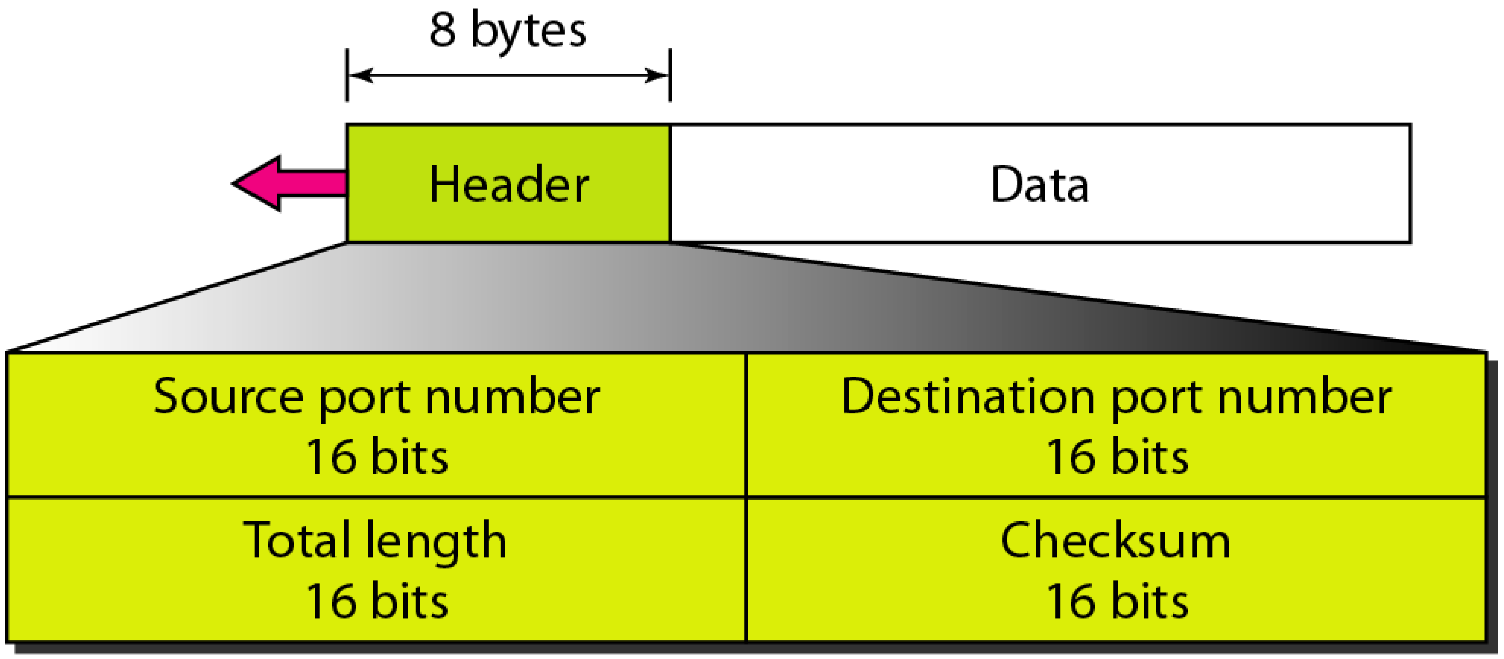


Figure. User Datagram format

Checksum

•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.

Optional Use of the Checksum : The calculation of the checksum and its inclusion in a user datagram are optional. If the checksum is not calculated, the field is filled with 1s. Note that a calculated checksum can never be all 1s because this implies that the sum is all Os, which is impossible because it requires that the value of fields to be Os.

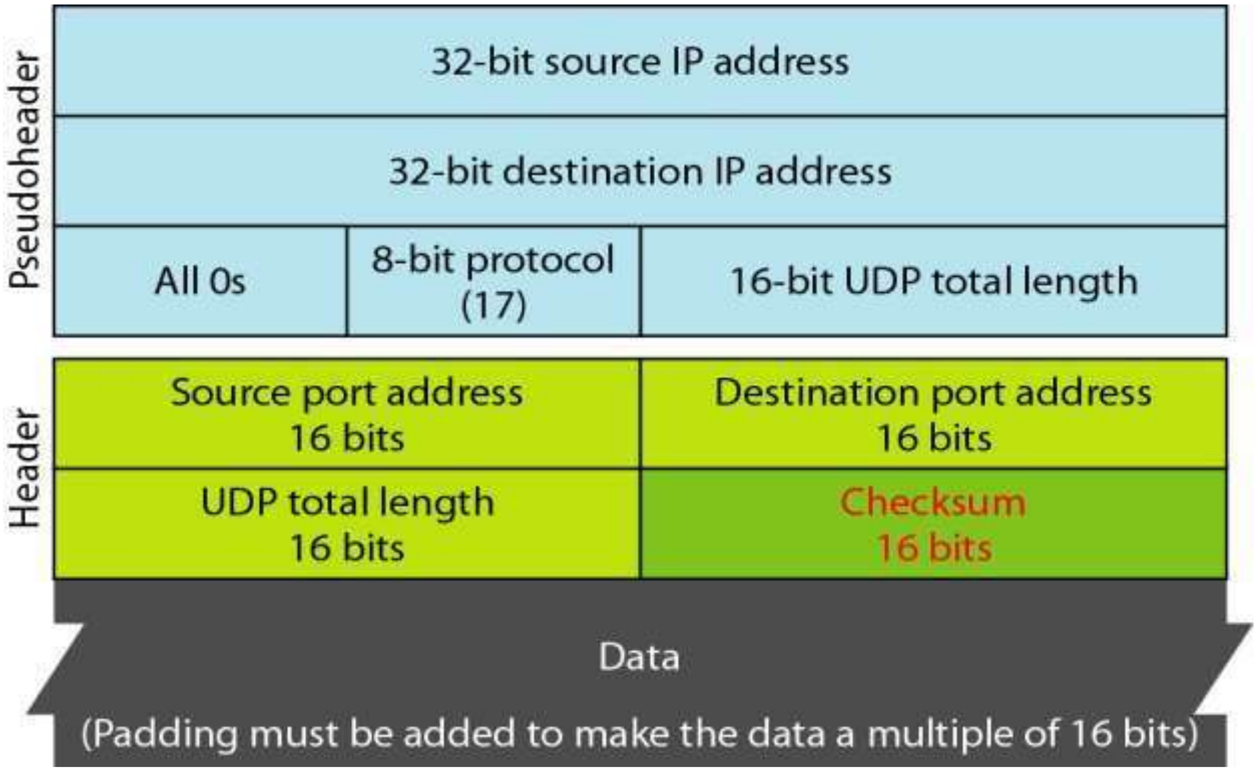

Pseudoheader
32-bit source IP address
32-bit destination IP address
All 0s
8-bit protocol (17)
16-bit UDP total length
Header
Source port address 16 bits
Destination port address 16 bits
UDP total length 16 bits
Checksum 16 bits
Data
(Padding must be added to make the data a multiple of 16 bits)

Figure *Pseudoheader for checksum calculation*

Below Figure 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 the padding will be dropped when the user datagram is delivered to IP.

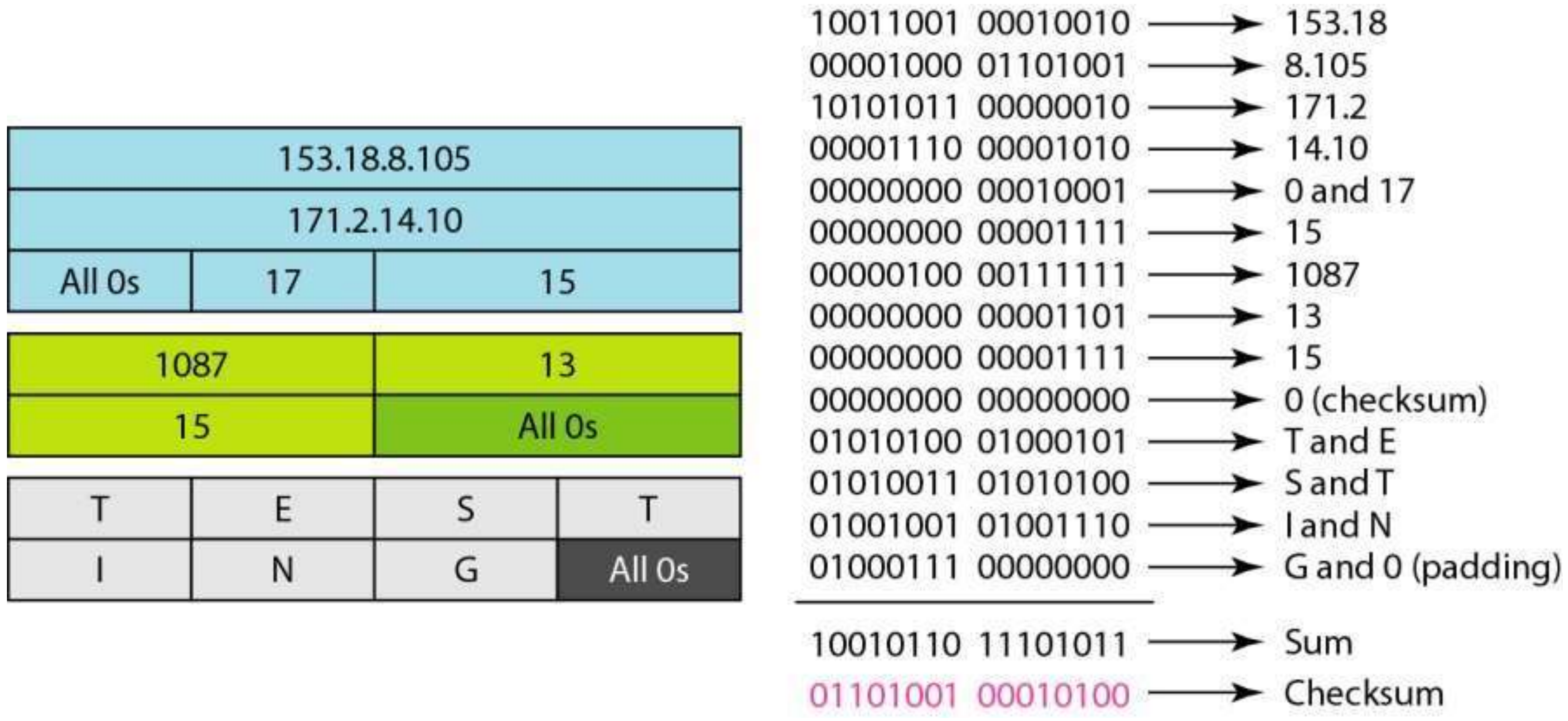


Figure *Checksum calculation of a simple UD user datagram*

UDP Operation

UDP uses concepts common to the transport layer. These concepts will be discussed here briefly, and then expanded in the next section on the TCP protocol

Connectionless Services : As mentioned previously, 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 coming from 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. This means 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 control and 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.

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

Queuing

At the client site, when a process starts, it requests a port number from the operating system. Some implementations create both an incoming and an outgoing queue associated with each process. Other implementations create only an incoming queue associated with each process.

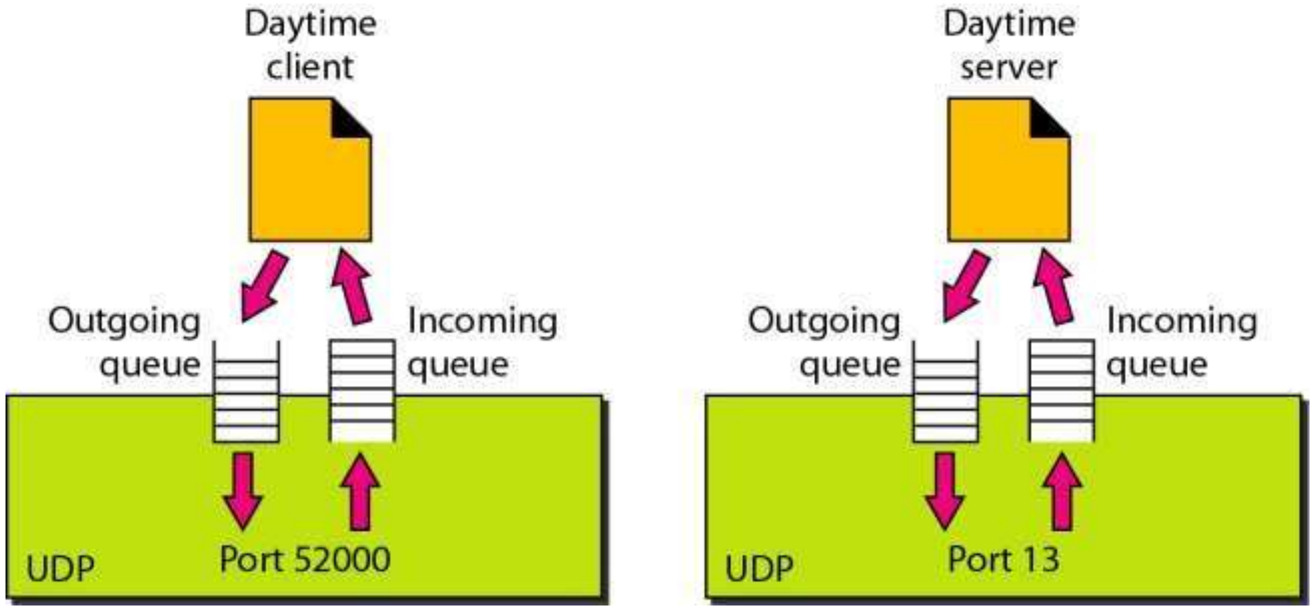


Figure *Queues in UDP*

Introduction to TCP

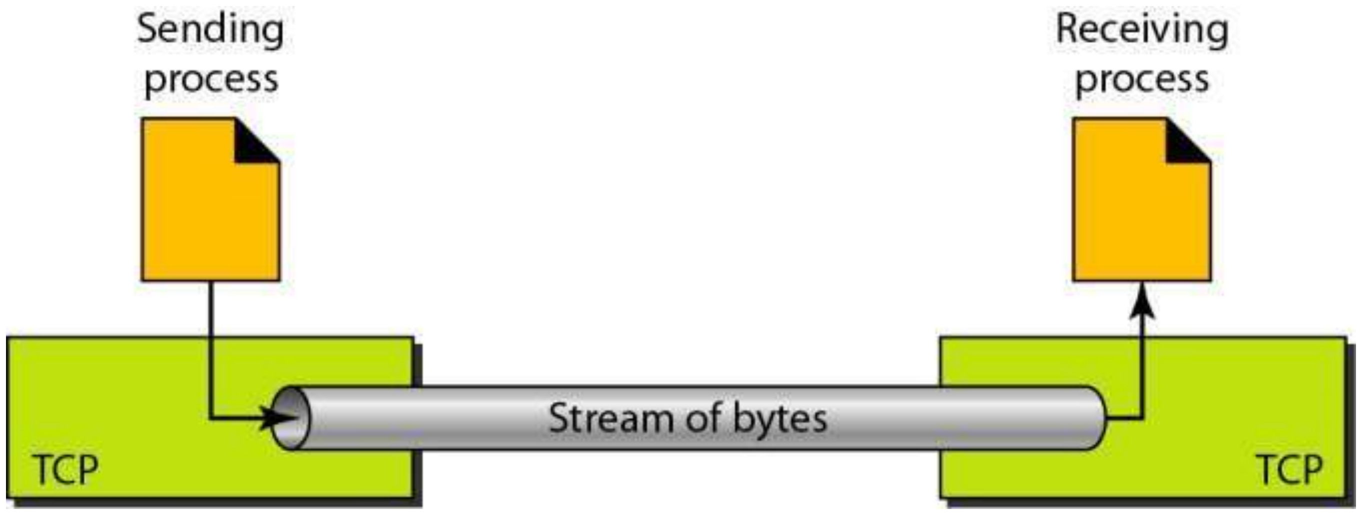
- It was specifically designed to provide a reliable end-to end byte stream over an unreliable network. It was designed to adapt dynamically to properties of the inter network and to be robust in the face of many kinds of failures.

- Each machine supporting TCP has a TCP transport entity, which accepts user data streams from local processes, breaks them up into pieces not exceeding 64kbytes and sends each piece as a separate IP datagram. When these datagrams arrive at a machine, they are given to TCP entity, which reconstructs the original byte streams. It is up to TCP to time out and retransmits them as needed, also to reassemble datagrams into messages in proper sequence.

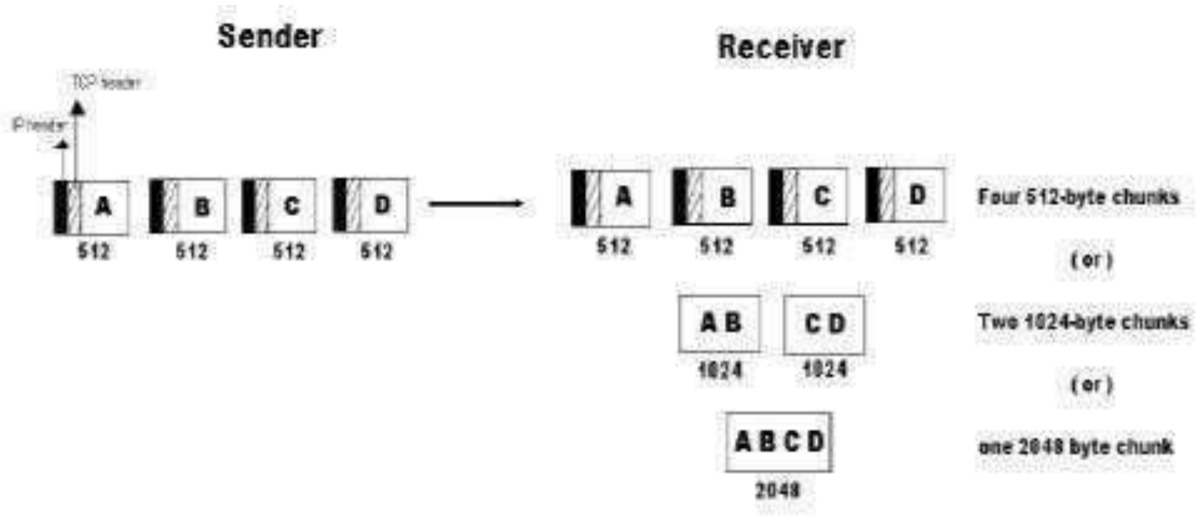
The TCP Service Model

- TCP service is obtained by having both the sender and receiver create end points called SOCKETS.
- Each socket has a socket number(address)consisting of the IP address of the host, called a “PORT” (= TSAP)
- To obtain TCP service a connection must be explicitly established between a socket on the sending machine and a socket on the receiving machine
- All TCP connections are full duplex and point to point i.e., multicasting or broadcasting is not supported.
- A TCP connection is a byte stream, not a message stream i.e., the data is delivered as chunks

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 data across the Internet. This imaginary environment is showed in below Figure. The sending process produces (writes to) the stream of bytes, and the receiving process consumes (reads from) them

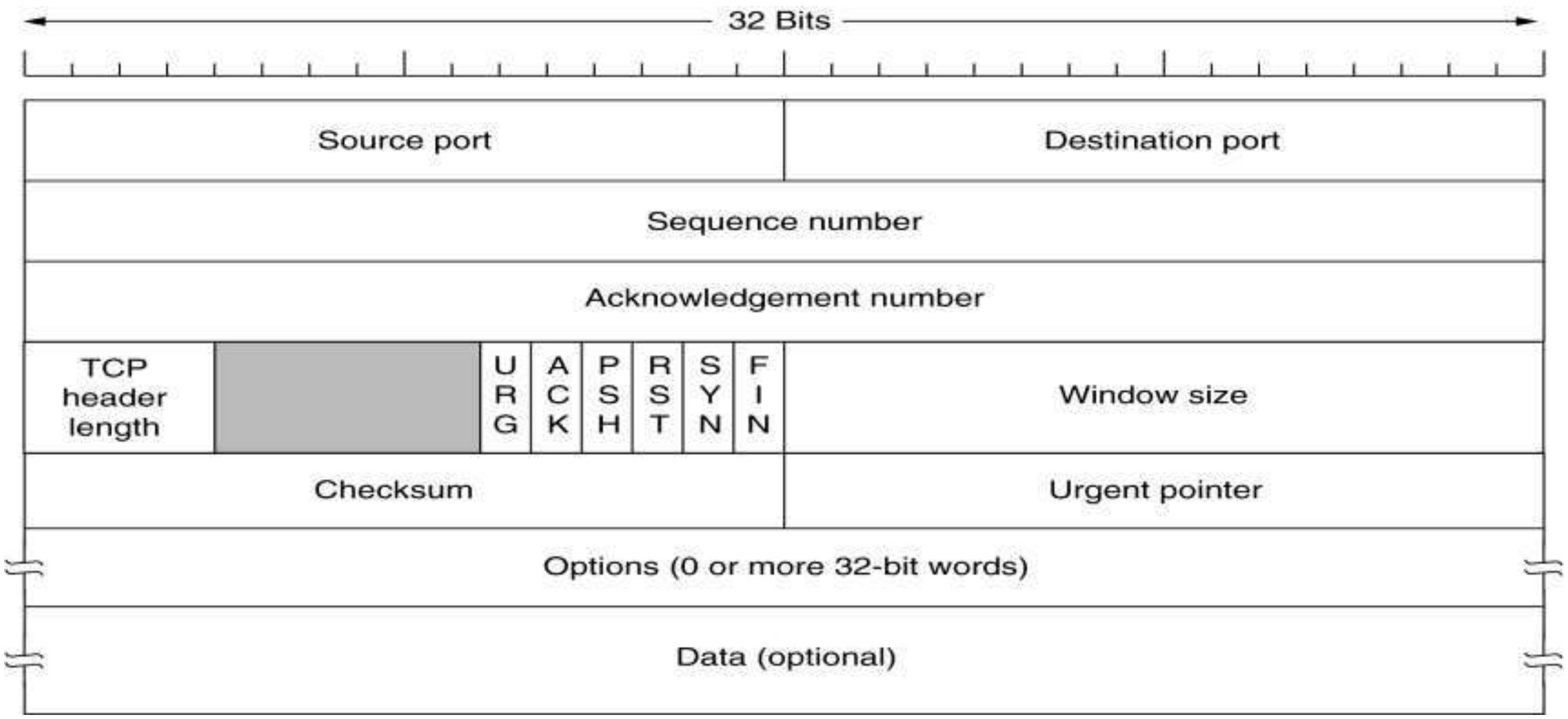


E.g.: 4 * 512 bytes of data is to be transmitted.



The TCP Segment Header

Every segment begins with a fixed-format, 20-byte header. The fixed header may be followed by header options. After the options, if any, up to 65,535 - 20 - 20 = 65,495 data bytes may follow, where the first 20 refer to the IP header and the second to the TCP header. Segments without any data are legal and are commonly used for acknowledgements and control messages.



- **Source Port, Destination Port :** Identify local end points of the connections
- **Sequence number:** Specifies the sequence number of the segment
- **Acknowledgement Number:** Specifies the next byte expected.
- **TCP header length:** Tells how many 32-bit words are contained in TCP header
- **URG:** It is set to 1 if URGENT pointer is in use, which indicates start of urgent data.
- **ACK:** It is set to 1 to indicate that the acknowledgement number is valid.
- **PSH:** Indicates pushed data
- **R ST:** It is used to reset a connection that has become confused due to reject an invalid segment or refuse an attempt to open a connection.
- **FIN:** Used to release a connection.
- **SYN:** Used to establish connections.

Checksum

A Checksum is also provided for extra reliability. It checksums the header, the data, and the conceptual pseudoheader shown in Fig. 6-30.

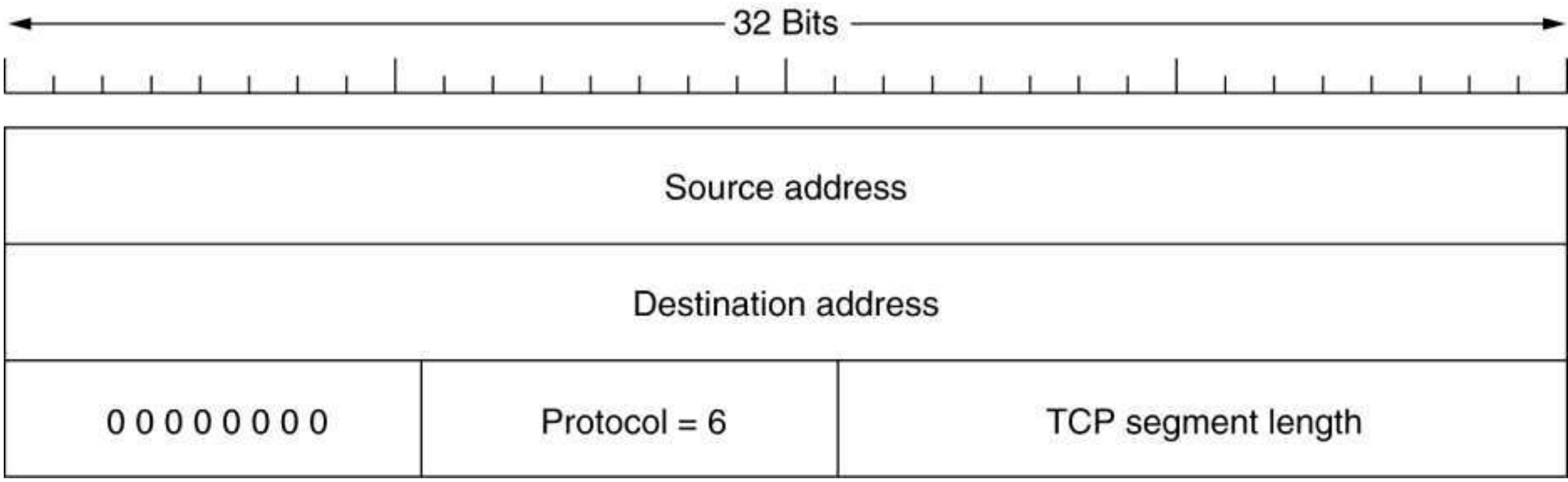


Figure 6-30. The pseudoheader included in the TCP checksum.

TCP Connection Establishment

- To establish a connection, one side, say, the server, passively waits for an incoming connection by executing the LISTEN and ACCEPT primitives, either specifying a specific source or nobody in particular.

- The other side, say, the client, executes a CONNECT primitive, specifying the IP address and port to which it wants to connect, the maximum TCP segment size it is willing to accept, and optionally some user data (e.g., a password).

- The CONNECT primitive sends a TCP segment with the *SYN* bit on and *ACK* bit off and waits for a response.

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:

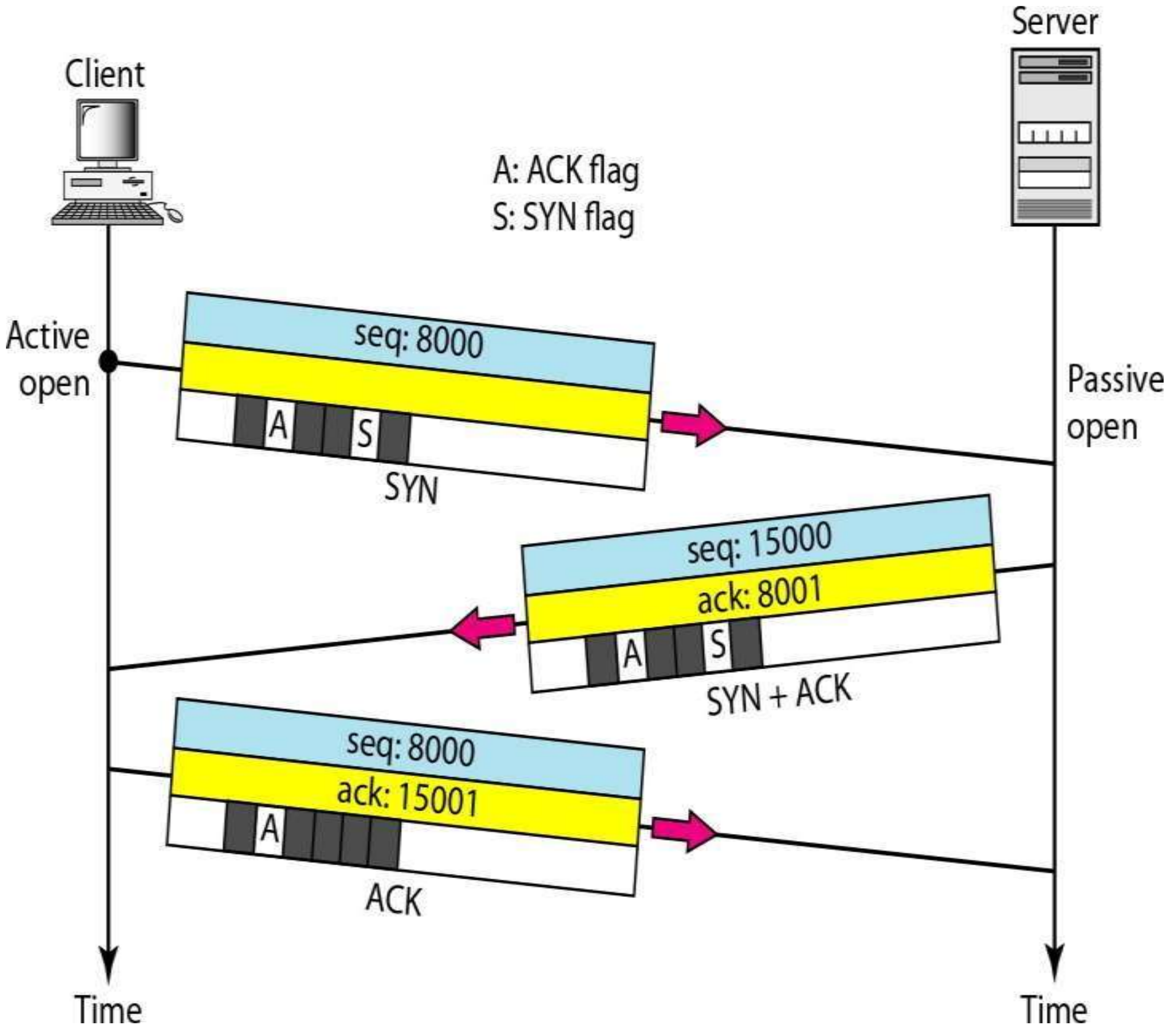
1. connection establishment
2. data transfer
3. connection termination.

TCP connection establishment(3 way handshaking)

1. The client sends the first segment, a SYN segment, in which only the SYN flag is set.. It consumes one sequence number. When the data transfer starts, the sequence number is incremented by 1. We can say that the SYN segment carries no real data, but we can think of it as containing 1 imaginary byte

2. The server sends the second segment, a SYN + ACK segment, with 2 flag bits set: SYN and ACK. This segment has a dual purpose. It is a SYN segment for communication in the other direction and serves as the acknowledgment for the SYN segment. It consumes one sequence number.

3. The client sends the third segment. This is just an ACK segment. It acknowledges the receipt of the second segment with the ACK flag and acknowledgment number field. Note that the sequence number in this segment is the same as the one in the SYN segment; the ACK segment does not consume any sequence numbers.


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 and denies service to every request.

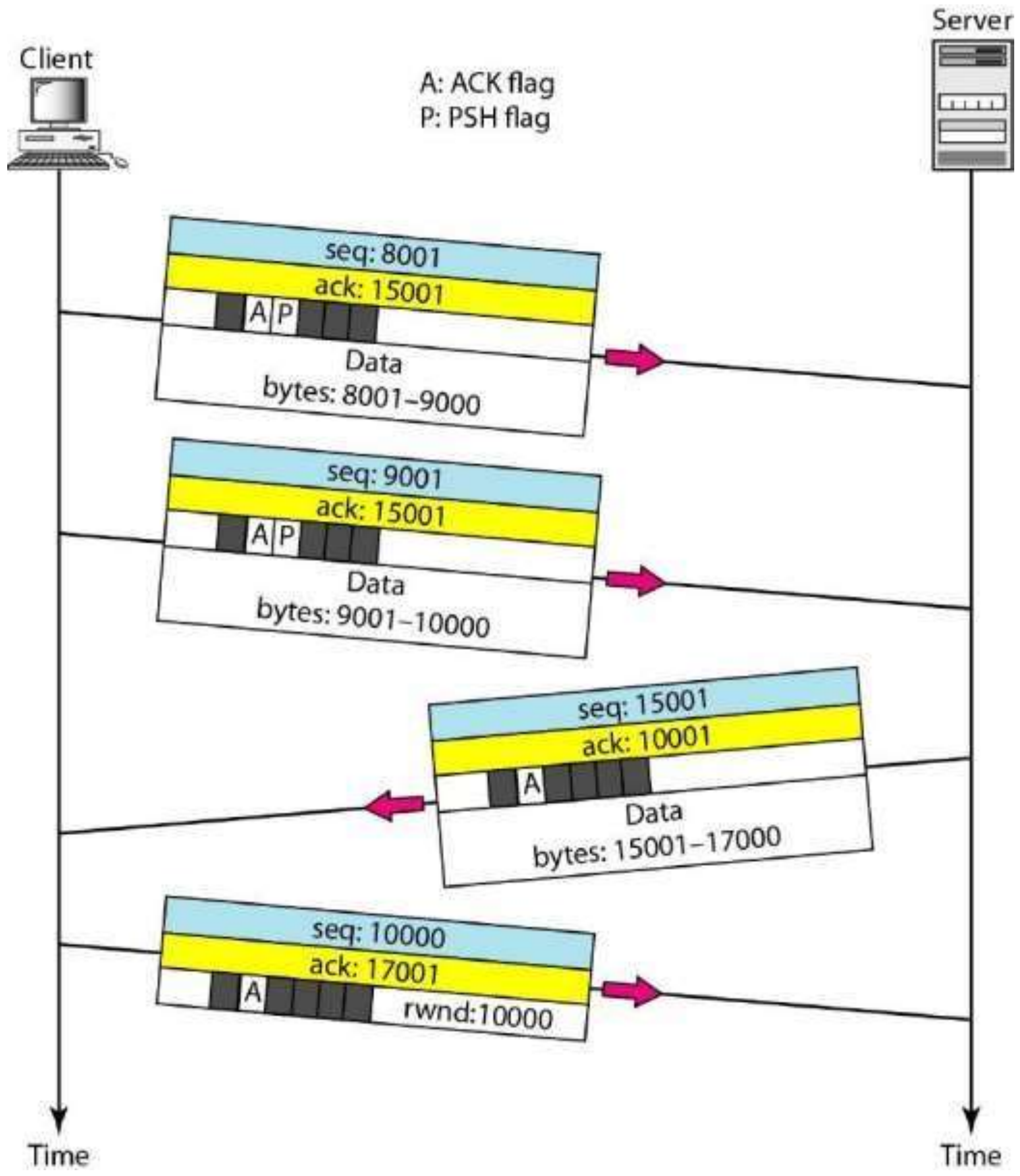
SOLUTIONS:

- Some have imposed a limit on connection requests during a specified period of time.
- Others filterout datagrams coming from unwanted source addresses.
- 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 , the client sends 2000 bytes of data in two segments. The server then sends 2000 bytes in one segment. The client sends one more segment. The first three segments carry both data and acknowledgment, but the last segment carries only an acknowledgment because there are no more data to be sent.
- Note the values of the sequence and acknowledgment numbers. The data segments sent by the client have the PSH (push) flag set so that the server TCP knows to deliver data to the server process as soon as they are received.

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.

- The PSH flag in the TCP header informs the receiving host that the data should be pushed up to the receiving application immediately.

The URG Flag

The URG flag is used to inform a receiving station that certain data within a segment is urgent and should be prioritized. If the URG flag is set, the receiving station evaluates the urgent pointer, a 16-bit field in the TCP header. This pointer indicates how much of the data in the segment, counting from the first byte, is urgent.

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 the FIN flag is set.

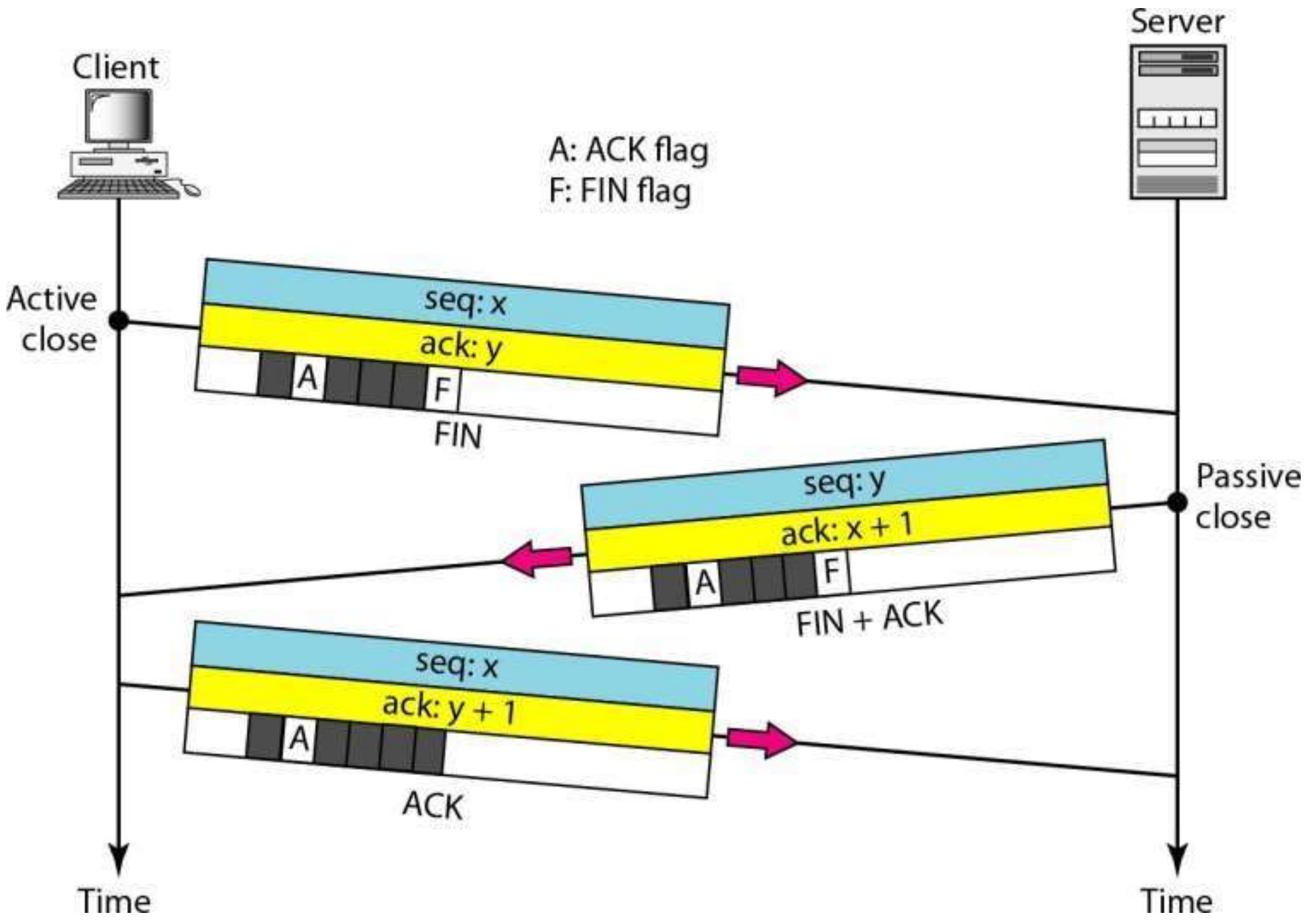
   Note that a FIN segment can include the last chunk of data sent by the client, or it can be just a control segment as shown in Figure. If it is only a control segment, it consumes only one sequence number.
   NOTE: The FIN segment consumes one sequence number if it does not carry data.

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 if it 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.

Connection termination using three-way handshaking



6.5.6 TCP Connection Release

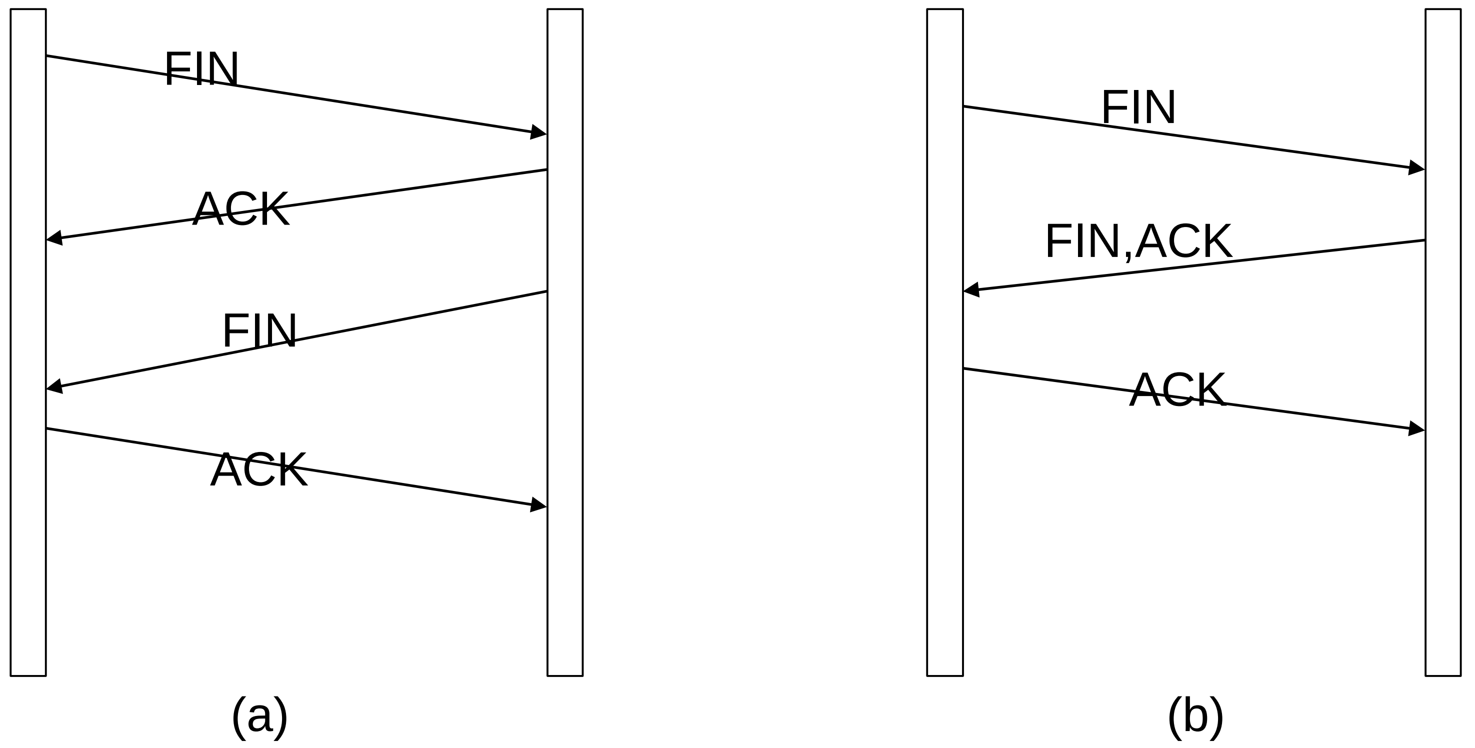


Figure 6-32. (a) Normally, four TCP segments are needed to release a connection .
(b) It is possible for the first ACK and the second FIN to be contained in the same segment .

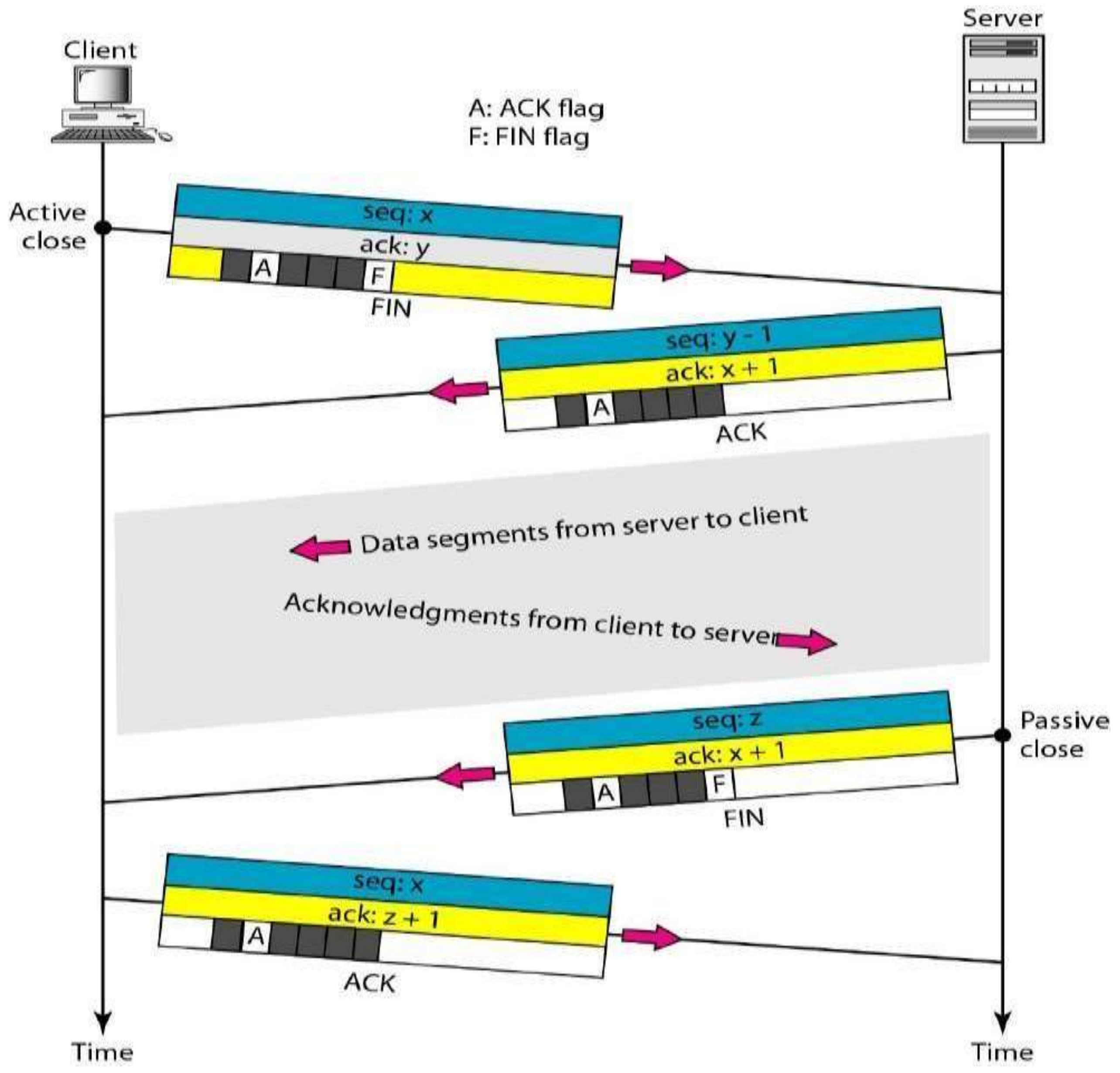
Half-Close

- 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

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.

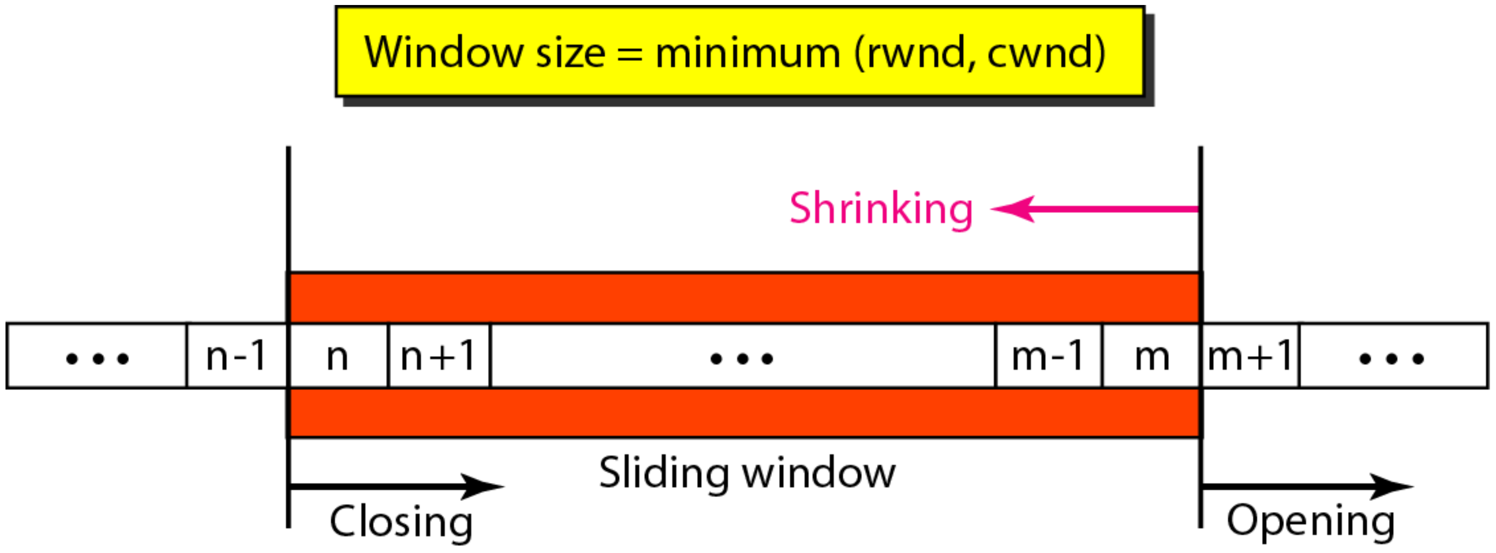
There are two big differences between this sliding window and the one we used at the data link layer.

- The sliding window of TCP is byte-oriented; the one we discussed in the data link layer is frame-oriented.
- The TCP's sliding window is of variable size; the one we discussed in the data link layer was of fixed size

- The sending system cannot send more bytes than space that is available in the receive buffer on the receiving system. TCP on the sending system must wait to send more data until all bytes in the current send buffer are acknowledged by TCP on the receiving system.
- On the receiving system, TCP stores received data in a receive buffer. TCP acknowledges receipt of the data, and *advertises* (communicates) a new *receive window* to the sending system. The receive window represents the number of bytes that are available in the receive buffer. If the receive buffer is full, the receiving system advertises a receive window size of zero, and the sending system must wait to send more data.
- After the receiving application retrieves data from the receive buffer, the receiving system can then advertise a receive window size that is equal to the amount of data that was read. Then, TCP on the sending system can resume sending data.

The available space in the receive buffer depends on how quickly data is read from the buffer by the receiving application. TCP keeps the data in its receive buffer until the receiving application reads it from that buffer. After the receiving application reads the data, that space in the buffer is available for new data. The amount of free space in the buffer is advertised to the sending system, as described in the previous paragraph.

Sliding window



- The window is <u>*opened, closed, or shrunk*</u>. *These three activities 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.

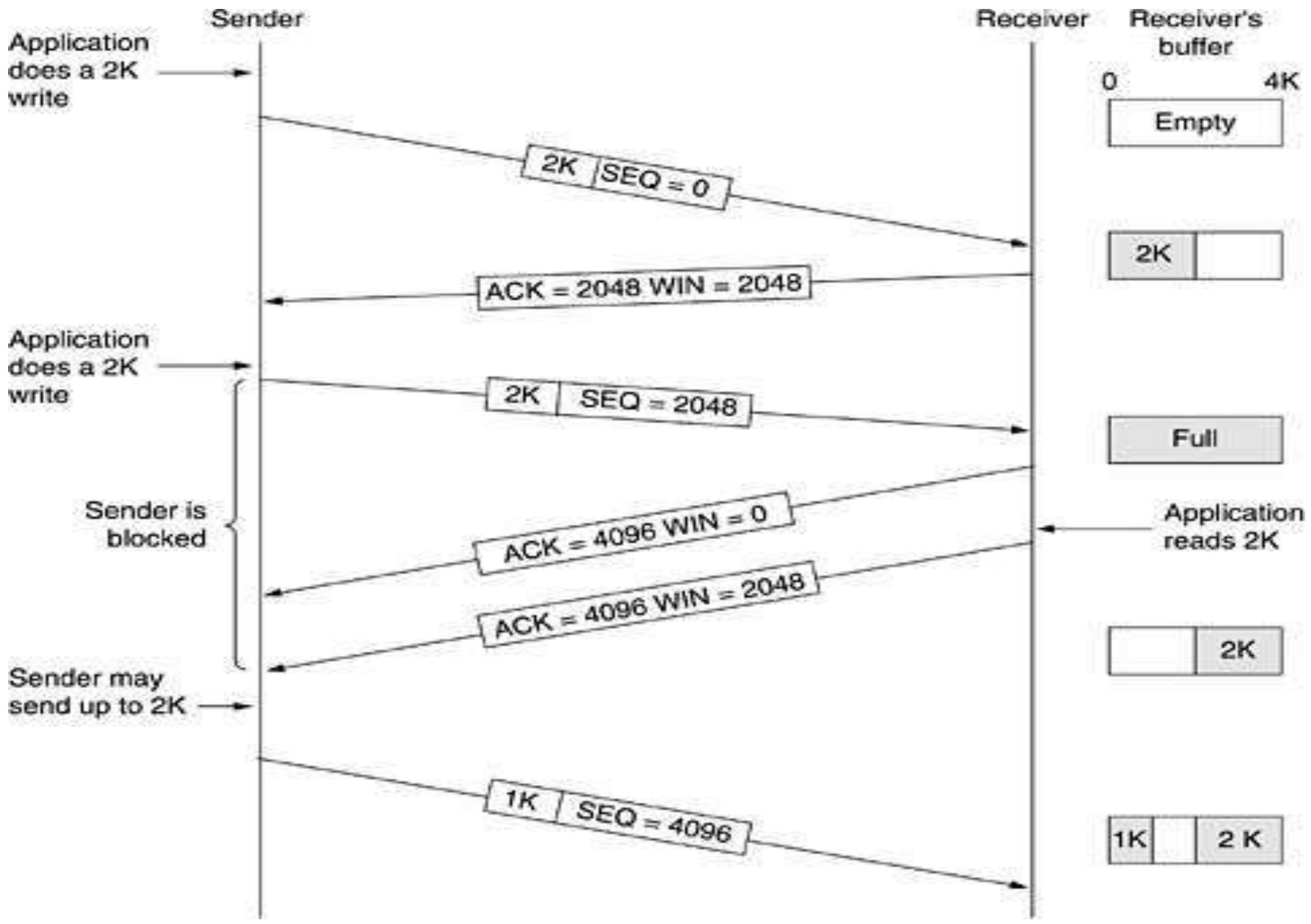
Closing the window means moving the left wall to the right. This means that some bytes have been acknowledged and the sender need not worry about them anymore.

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

Example

What is the value of the receiver window (rwnd) for host A? if the receiver, host B, has a buffer size of 5000 bytes and 1000 bytes of received and unprocessed data?

Solution

The value of rwnd =5000 - 1000 =4000. Host B can receive only 4000 bytes of data before overflowing its buffer. Host B advertises this value in its next segment to A.

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 a corrupted 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

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.

Retransmission

The heart of the error control mechanism is the retransmission of segments. When a segment is corrupted, lost, or delayed, it is retransmitted. In modern implementations, a segment is retransmitted on two occasions: when a retransmission timer expires or when the sender receives three duplicate ACKs.

Retransmission time out(RTO)

After RTO A recent implementation of TCP maintains one retransmission time-out (RTO) timer for all outstanding (sent, but not acknowledged) segments.

When the timer matures, the earliest outstanding segment is retransmitted even though lack of a received ACK can be due to a delayed segment, a delayed ACK, or a lost acknowledgment.

Out-of-Order Segments

- When a segment is delayed, lost, or discarded, the segments following that segment arrive out of order. Originally, TCP was designed to discard all out-of-order segments

- Most implementations today do not discard the out-of-order segments. They store them temporarily and flag them as out-of-order segments until the missing segment arrives. Note, however, that the out-of-order segments are not delivered to the process. TCP guarantees that data are delivered to the process in order

TCP Congestion control

•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. <u>It is up to the transport</u> <u>layer to receive congestion feedback from the network layer and slow down</u> <u>the rate of traffic that it is sending into the network.</u>

•For Congestion control, transport protocol uses an <u>AIMD</u> (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 sender has two pieces of information: the receiver-advertised window size and the congestion window size. The actual size of the window is the minimum of these two.

Actual window size = minimum (rwnd, cwnd)

Congestion Policy

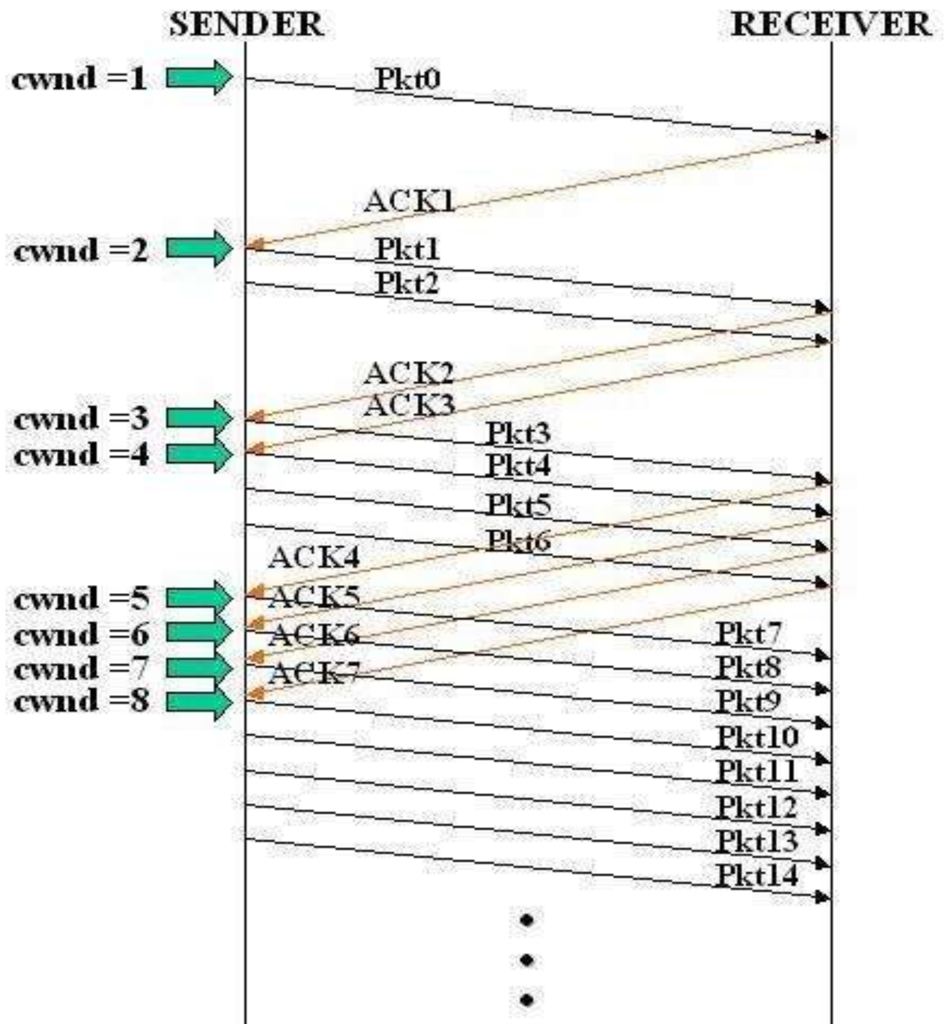
TCP's general policy for handling congestion is based on three phases: <u>slow start</u>, <u>congestion avoidance</u>, and <u>congestion detection</u>. In the slow-start phase, the sender starts with a very slow rate of transmission, but increases the rate rapidly to reach a threshold. When the threshold is reached, the data rate is reduced to avoid congestion. Finally if congestion is detected, the sender goes back to the slow-start or congestion avoidance phase based on how the congestion is detected.

slow start, exponential increase

- One of the algorithms used in TCP congestion control is called slow start. This algorithm is based on the idea that the size of the congestion window*(cwnd) starts with one maximum segment size (MSS)*.

- As the name implies, the window starts slowly, but grows exponentially.

slow start, exponential increase



In the slow-start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.

Congestion Avoidance

- In congestion avoidance phase exponential growth of slow start algorithm is stopped, and linear increment is started.
- That means for every set of acknowledgements it adds only one extra segment in cwnd
- E.g if 4 acknowledgements are received slow start with increment cwnd by 4,(i.e. cwnd = cwnd + 4)
- but in congestion avoidance phase cwnd is increment by 1 (i.e. cwnd = cwnd + 1) irrespective of how many acknowledgements are received

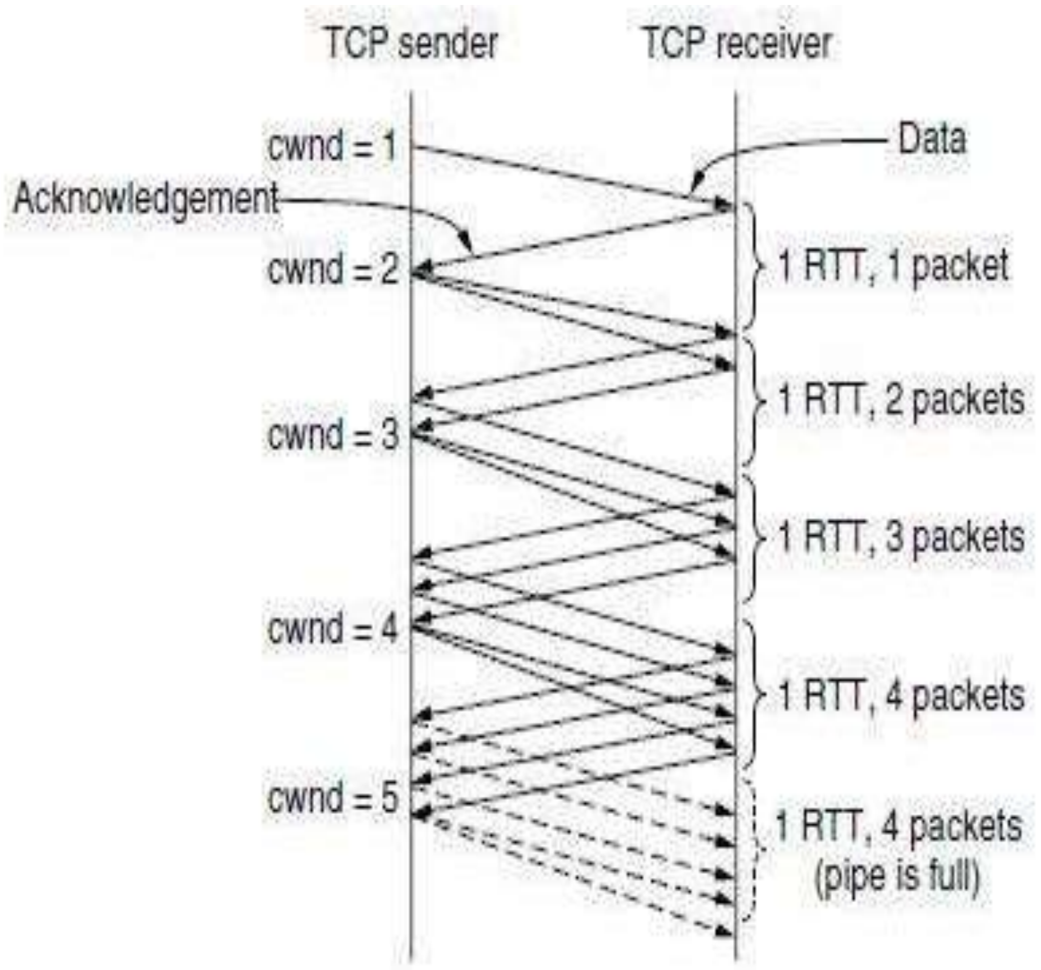


Figure 6-45. Additive increase from an initial congestion window of one segment.

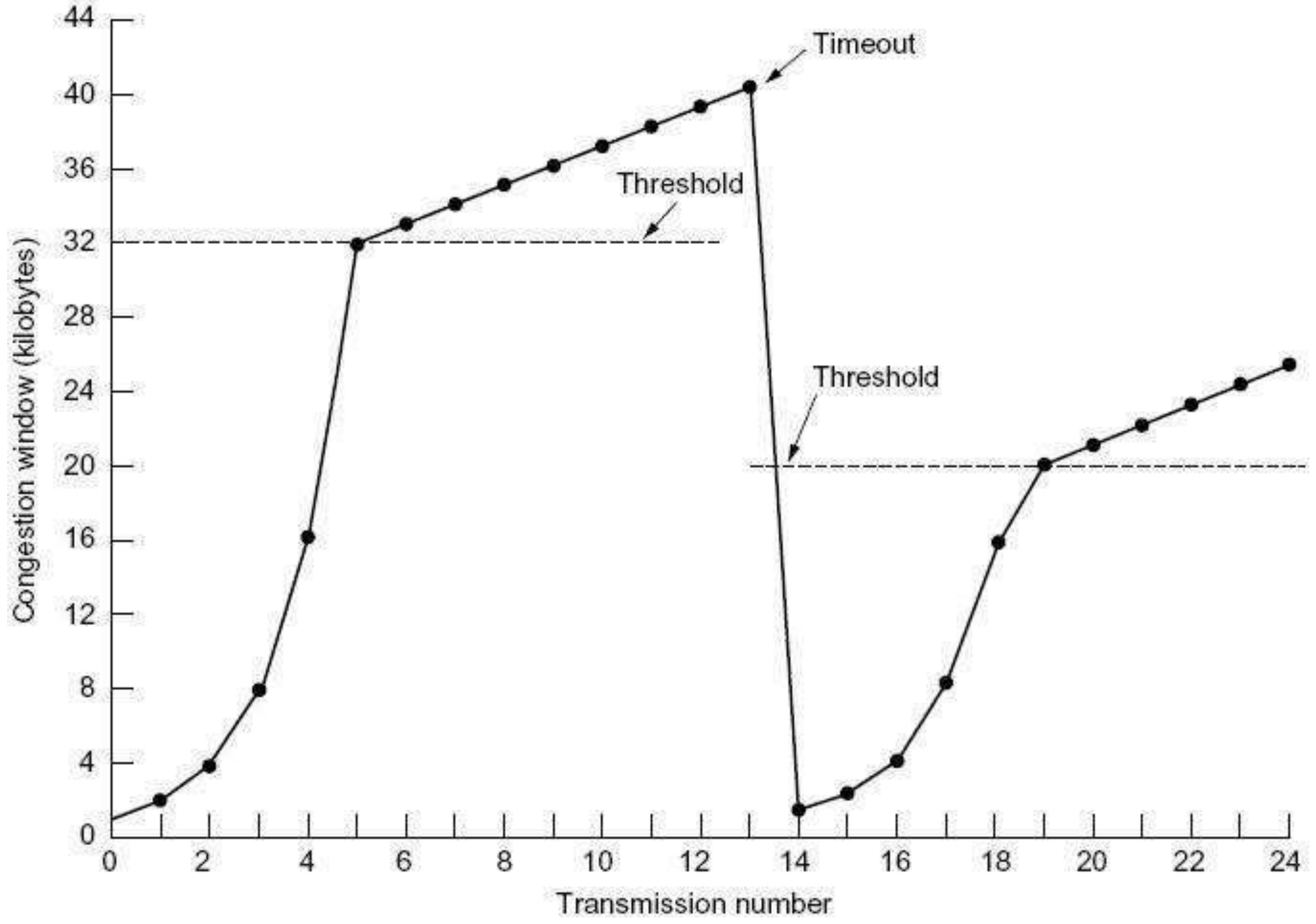


Fig. 6-37. An example of the Internet congestion algorithm.

Congestion Detection: Multiplicative Decrease.

- If congestion occurs, the congestion window size must be decreased. The only way the sender can guess that congestion has occurred is by the need to retransmit a segment. However, retransmission can occur in one of two cases: when a timer times out or when three ACKs are received. In both cases, the size of the threshold is dropped to one-half, a multiplicative decrease

An implementations reacts to congestion detection in one of the following ways:

o If detection is by time-out, a new *slow-start phase starts*.
o If detection is by three ACKs, a new *congestion avoidance phase starts*.

1. If a time-out occurs, there is a stronger possibility of congestion; a segment has probably been dropped in the network, and there is no news about the sent segments.

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

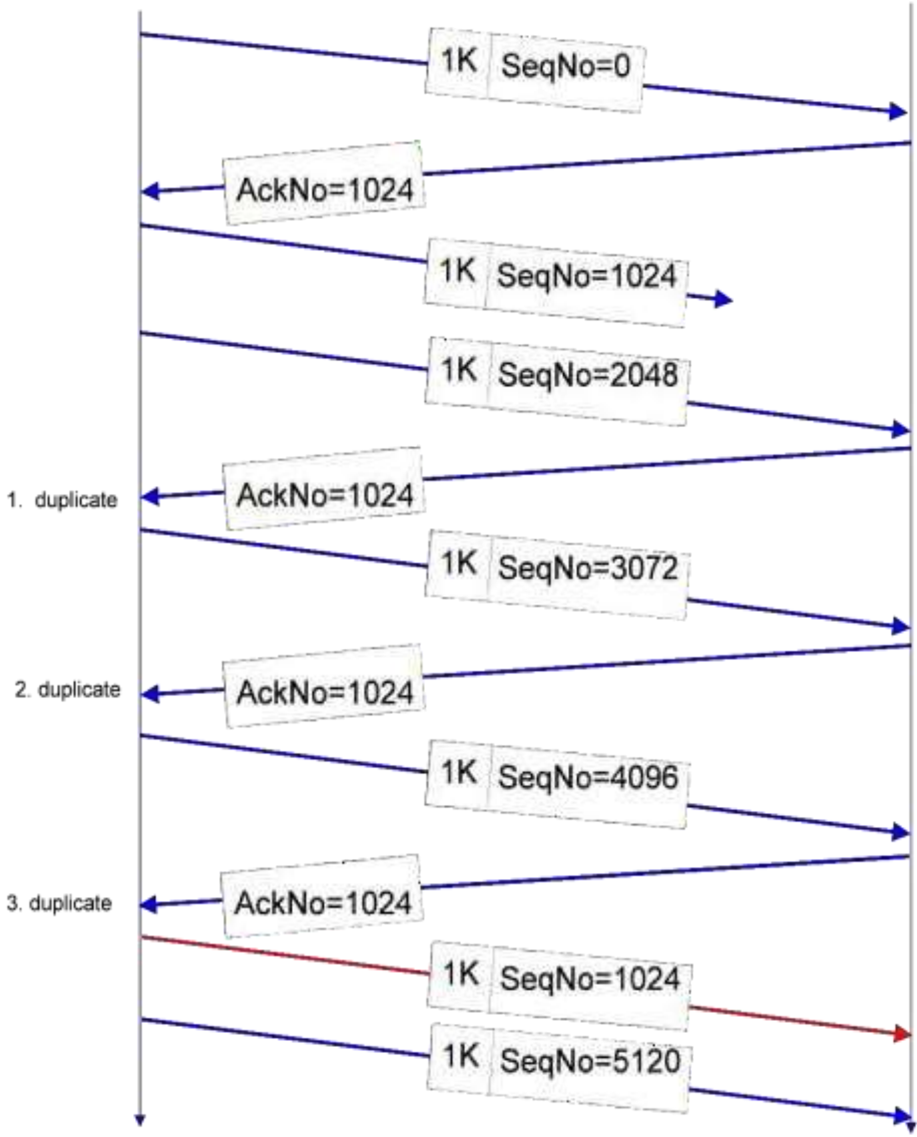
– and slow-start is entered

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 waiting for a timeout to happen.

•Enter slow start: ssthresh = cwnd/2
cwnd = 1


1K SeqNo=0
AckNo=1024
1K SeqNo=1024
1K SeqNo=2048
1. duplicate
AckNo=1024
1K SeqNo=3072
2. duplicate
AckNo=1024
1K SeqNo=4096
3. duplicate
AckNo=1024
1K SeqNo=1024
1K SeqNo=5120