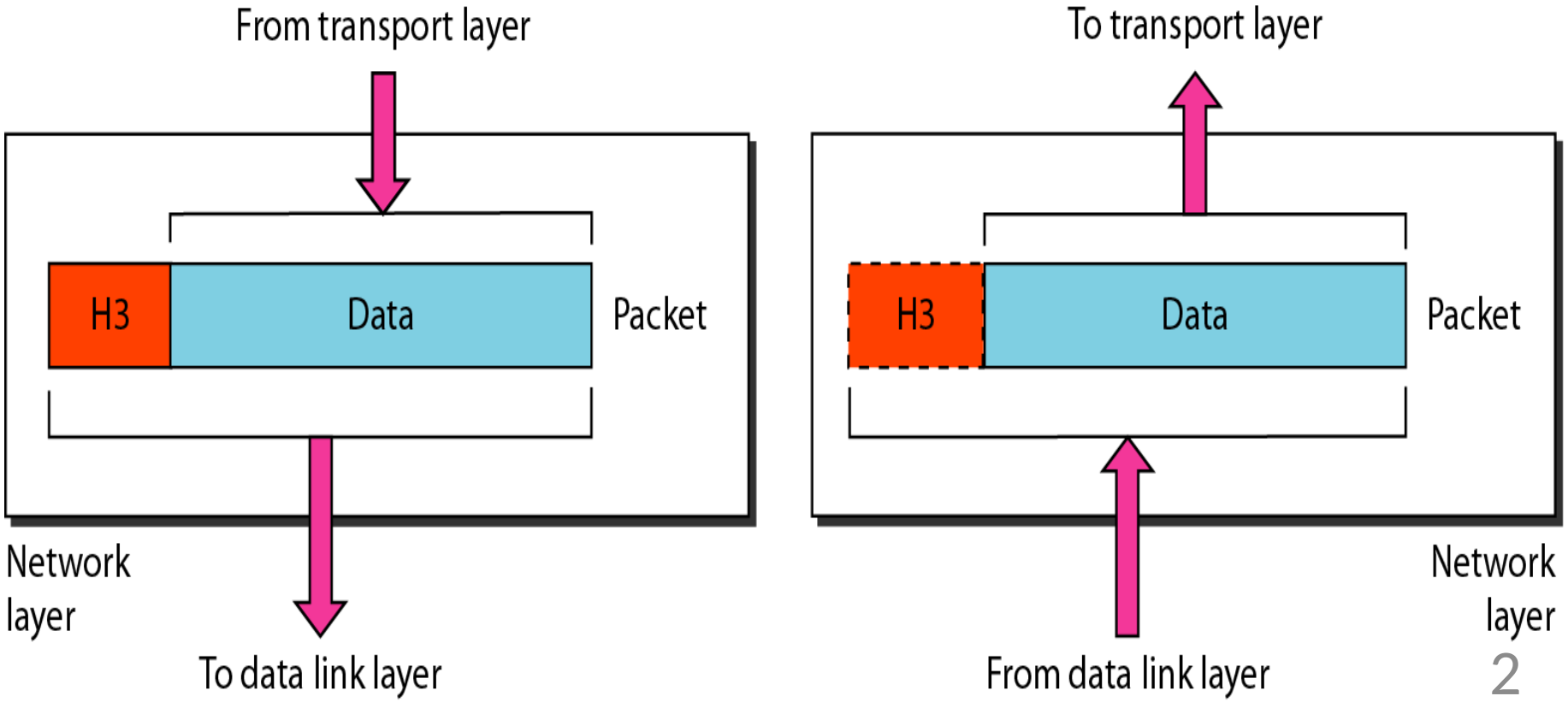
Network Layer (3rd OSI Layer)

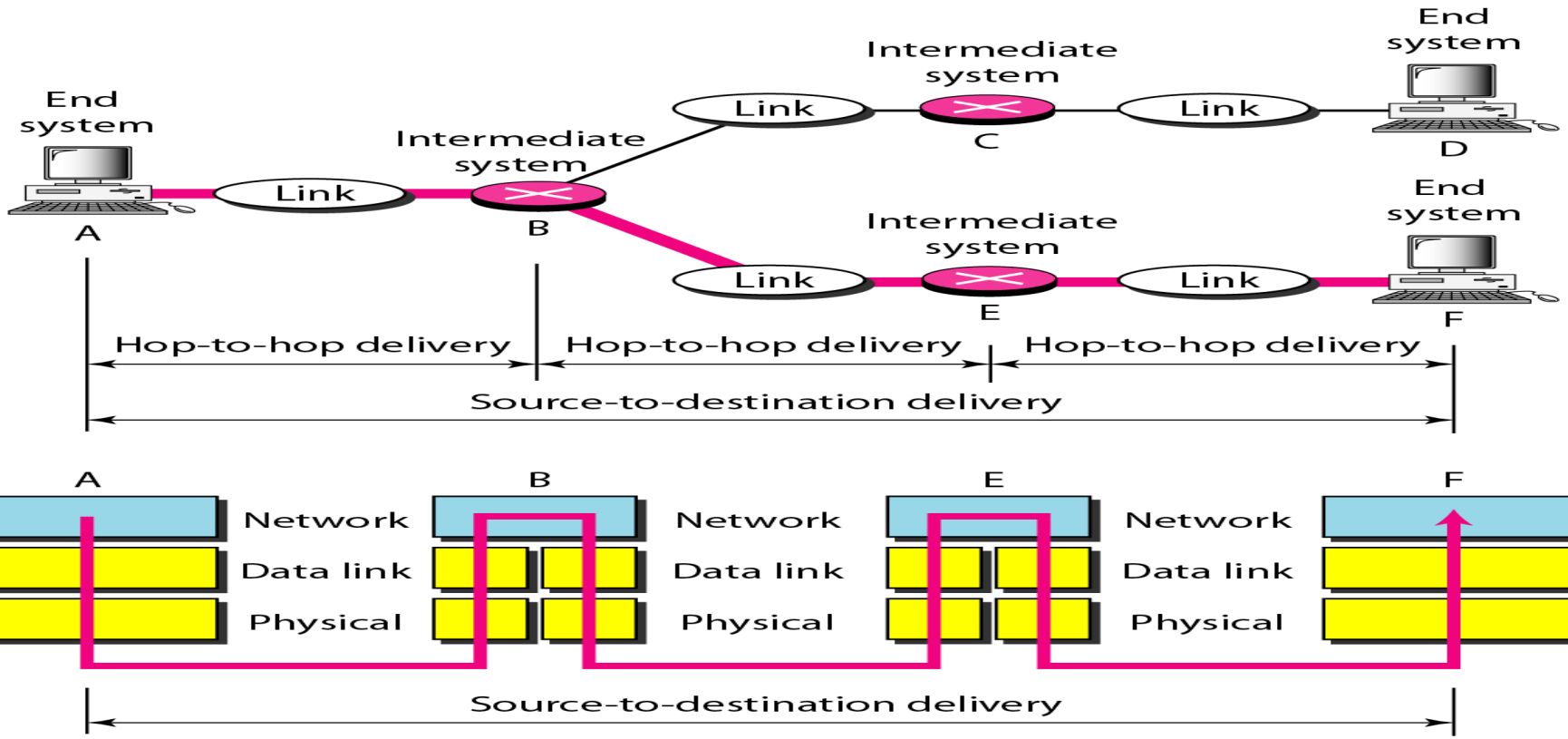
- Concerned with getting packets from source to destination.
- The network layer must know the topology of the subnet and choose appropriate paths through it.
- When source and destination are in *different networks*, the network layer must deal with these differences.
- The network layer is responsible for the source-to-destination delivery of a packet, possibly across multiple networks. Whereas the data link layer oversees the delivery of the packet between two systems on the same network, the network layer ensures that each packet gets from its point of origin to its final destination.
- If two systems are connected to the same local network, there is usually no need for a network layer. However, if the two systems are attached to different networks with connecting devices between the networks, there is often a need for the network layer to accomplish source-to-destination delivery.

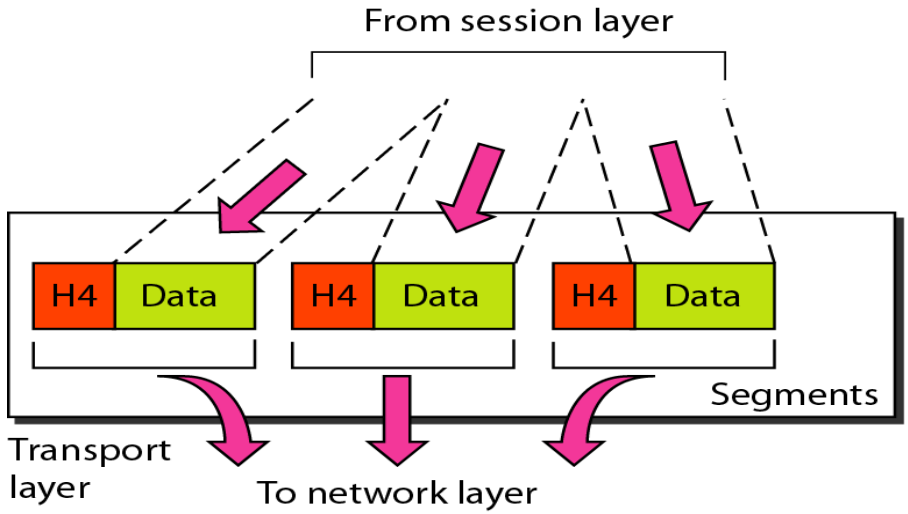
The network layer is responsible for the delivery of individual packets from the source host to the destination host.

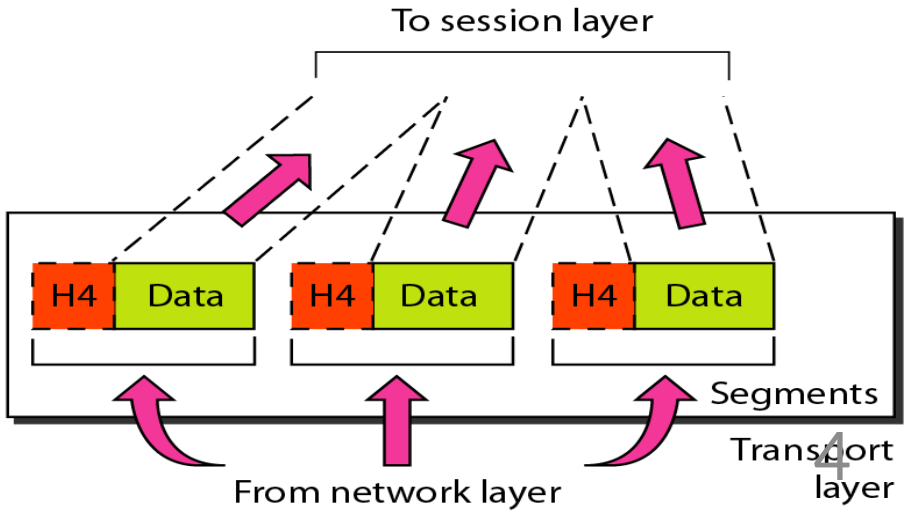


Other responsibilities of the network layer include the following:

- Logical addressing. The physical addressing implemented by the data link layer handles the addressing problem locally. If a packet passes the network boundary, we need another addressing system to help distinguish the source and destination systems. The network layer adds a header to the packet coming from the upper layer that, among other things, includes the logical addresses of the sender and receiver. We discuss logical addresses later in chapter Seven.
- Routing. When independent networks or links are connected to create *internetworks* (network of networks) or a large network, the connecting devices (called *routers* or *switches*) *route or switch the packets to their final destination*. *One of the functions* of the network layer is to provide this mechanism.


End system
A
Link
Intermediate system
B
Link
Intermediate system
C
Link
End system
D
Intermediate system
E
Link
Link
End system
F
Hop-to-hop delivery
Hop-to-hop delivery
Hop-to-hop delivery
Source-to-destination delivery
A
B
E
F
Network
Data link
Physical
Network
Data link
Physical
Network
Data link
Physical
Source-to-destination delivery


From session layer
H4
Data
H4
Data
H4
Data
Segments
Transport layer
To network layer

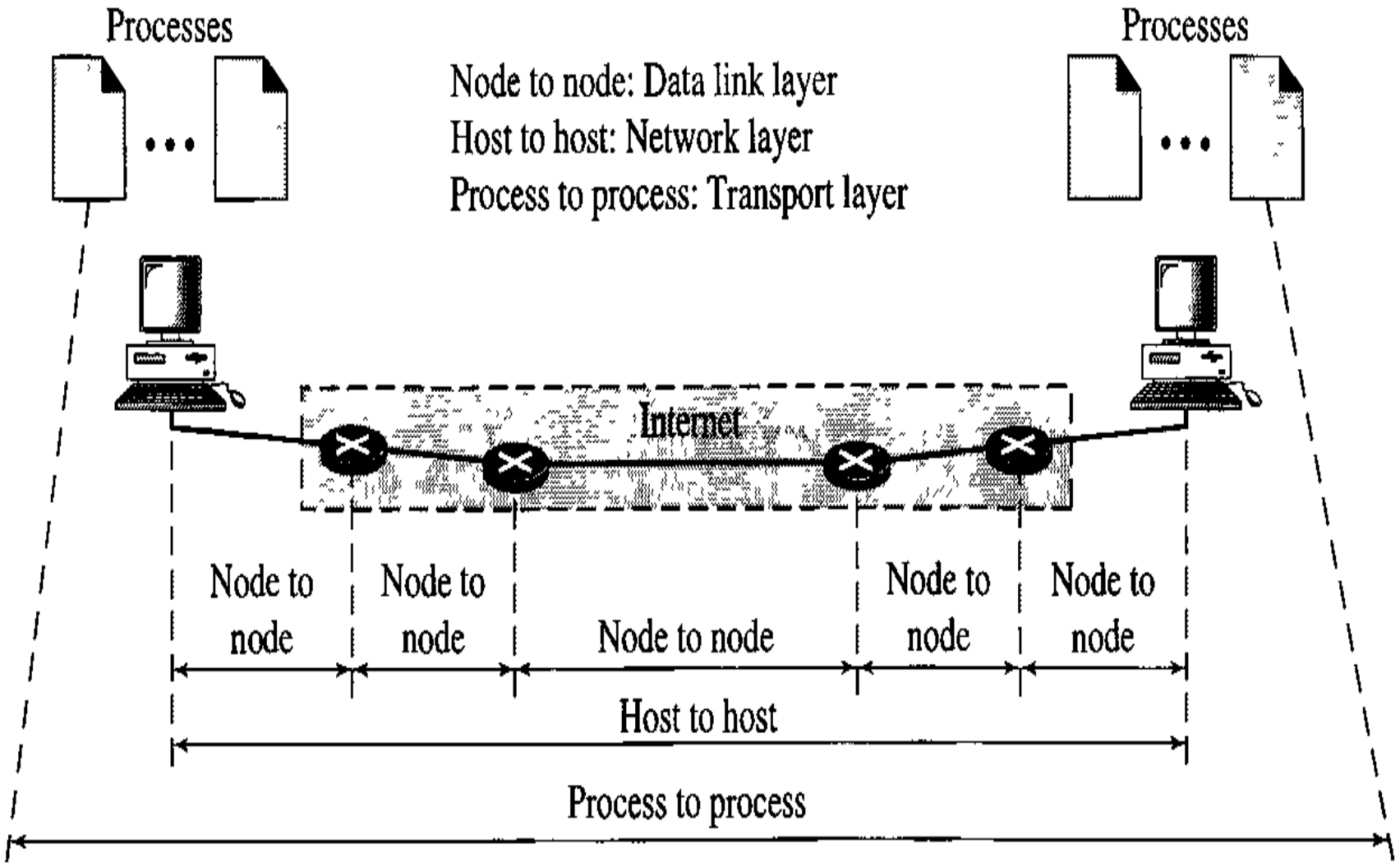

To session layer
H4
Data
H4
Data
H4
Data
Segments
From network layer
Transport layer

Transport layer (4th OSI layer)

- The transport layer is responsible for process-to-process delivery of the entire message.
- A process is an application program running on a host.
- Whereas the network layer oversees source-to-destination delivery of individual packets, it does not recognize any relationship between those packets.
- The network layer treats each packet independently, as though each piece belonged to a separate message, whether or not it does.
- The transport layer, on the other hand, ensures that the whole message arrives intact and in order, overseeing both error control and flow control at the source-to-destination level.

Major functions of the transport layer

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


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

Responsibilities of the transport layer include the following:

1. Service-point addressing. Computers often run several programs at the same time. For this reason, source-to-destination delivery means delivery not only from one computer to the next but also from a specific process (running program) on one computer to a specific process (running program) on the other. The transport layer header must therefore include a type of address called a *service-point address* (or *port address*). *The network layer gets each packet to the correct* computer; the transport layer gets the entire message to the correct process on that computer.
2. Segmentation and reassembly. A message is divided into transmittable segments, with each segment containing a sequence number. These numbers enable the transport layer to reassemble the message correctly upon arriving at the destination and to identify and replace packets that were lost in transmission.

3. Flow control. Like the data link layer, the transport layer is responsible for flow control. However, flow control at this layer is performed end to end rather than across a single link.
4. Error control. Like the data link layer, the transport layer is responsible for error control. However, error control at this layer is performed process-to-process rather than across a single link. The sending transport layer makes sure that the entire message arrives at the receiving transport layer without error (damage, loss, or duplication). Error correction is usually achieved through retransmission.
5. Connection control. The transport layer can be either connectionless or connection-oriented. A connectionless transport layer treats each segment as an independent packet and delivers it to the transport layer at the destination machine. A connection-oriented transport layer makes a connection with the transport layer at the destination machine first before delivering the packets. After all the data are transferred, the connection is terminated.

Process-to-Process Delivery

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

Transport layer 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 (temporal) port number. (next slide, more on port number)

Identifying Applications (Processes)

- In order to pass data streams to the proper applications, the Transport layer must identify the target application.
- To accomplish this, the Transport layer assigns an application an identifier. This identifier is called a port number.
- Each software process that needs to access the network is assigned a port number unique in that host
- This port number is used in the transport layer header to indicate to which application that piece of data is associated.

- The server process must also define itself with a port number. This port number, however, cannot be chosen randomly. If the computer at the server site runs a server process and assigns a random number as the port number, the process at the client site that wants to access that server and use its services will not know the port number. Of course, one solution would be to send a special packet and request the port number of a specific server, but this requires more overhead. The Internet has decided to use universal port numbers for servers; these are called well-known port numbers.
- Example of well known port numbers: 21 for FTP, 23 telnet,
  25 SMTP, 80 HTTP, etc
- There are some exceptions to this rule; for example, there are clients that are assigned well-known port numbers. Every client process knows the well-known port number of the corresponding server process.

Internet Assigned Number Authority (*IANA) Ranges*

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

Socket Addresses

- Process-to-process delivery needs two identifiers, IP address and the port number, at each end to make a connection.
- The combination of an IP address and a port number is called a socket address. The client socket address defines the client process uniquely just as the server socket address defines the server process uniquely.
- A transport layer protocol needs a pair of socket addresses: the client socket address and the server socket address.
- These four pieces of information are part of the IP header and the transport layer protocol header. The IP header contains the IP addresses; the UDP or TCP header contains the port numbers.



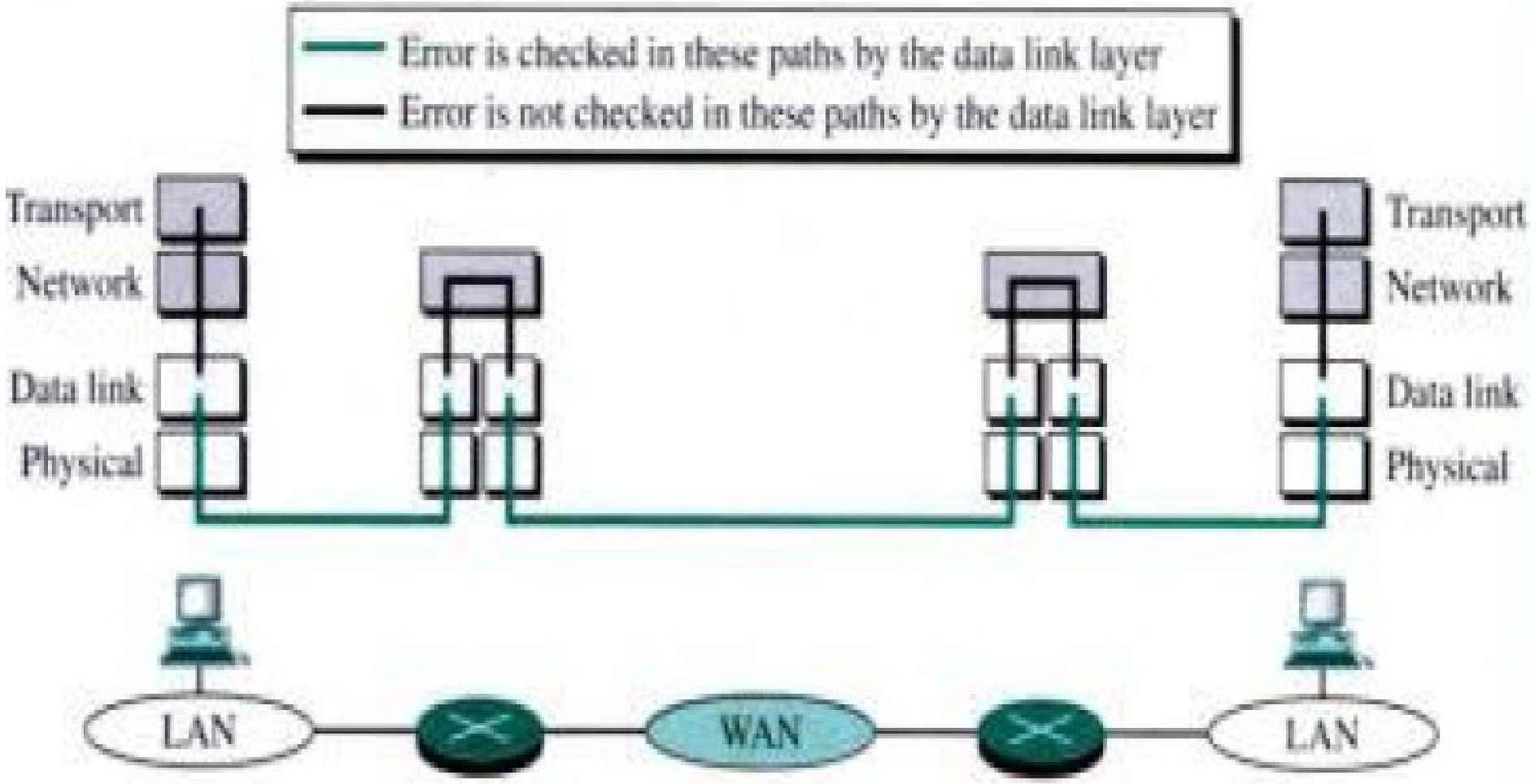
Connection Control: Connectionless Versus Connection-Oriented Service

- A transport layer protocol can either be connectionless or connection-oriented.
- *Connectionless Service*
  - In a connectionless service, the packets are sent from one party to another with no need for connection establishment or connection release. The packets are not numbered; they may be delayed or lost or may arrive out of sequence. There is no acknowledgment either. UDP is connectionless.
- Connection-Oriented *Service*
  - In a connection-oriented service, a connection is first established between the sender and the receiver. Data are transferred. At the end, the connection is released. TCP and SCTP are connection-oriented protocols.

Reliable Versus Unreliable

- The transport layer service can be reliable or unreliable. If the application layer program needs reliability, we use a reliable transport layer protocol by implementing flow and error control at the transport layer. This means a slower and more complex service.
- On the other hand, if the application program does not need reliability because it uses its own flow and error control mechanism or it needs fast service or the nature of the service does not demand flow and error control (real-time applications), then an unreliable protocol can be used.
- In the Internet, there are two common different transport layer protocols, as we have already mentioned. UDP is connectionless and unreliable; TCP and SCTP are connection-oriented and reliable protocols. These three can respond to the demands of the application layer programs.

Why flow control and error control at the transport layer as we have it at the data link layer???

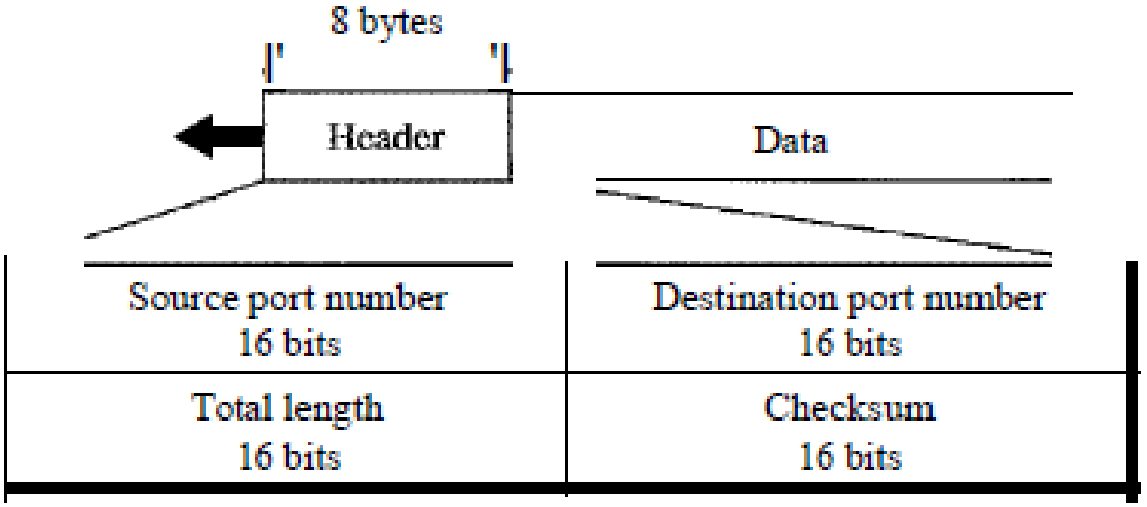


USER DATAGRAM PROTOCOL (UDP)

- The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to-process communication.
- 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
- Source port number. This is the port number used by the process running on the source host.
- Destination port number. This is the port number used by the process running on the destination host.
- Length. This is a 16-bit field that defines the total length of the user datagra



Use of UDP

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

Transmission Control Protocol (TCP)

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

- UDP does not perform:
  - Flow control
  - Error control
  - Connection control
    - All these functions are done by the processes (application layer programs) using UDP
- But UDP, like TCP, performs:
  - Service point addressing
  - Segmentation and reassembly

TCP Services

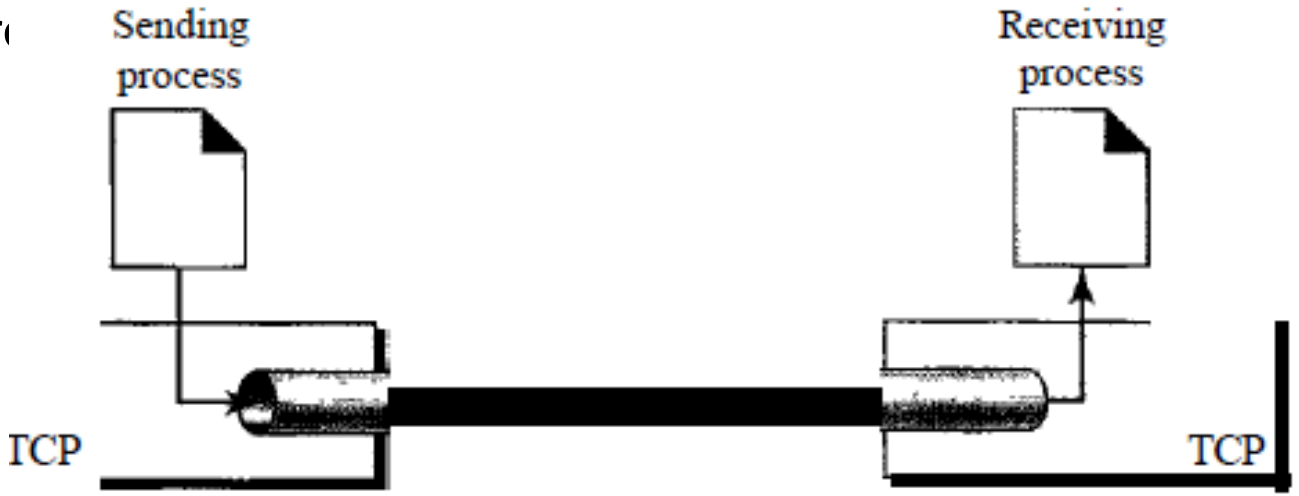
1. *Process-to-Process Communication*

   Like UDP, TCP provides process-to-process communication using port numbers.

2. *Stream Delivery Service*
3. *Full-Duplex Communication*
4. *Connection-Oriented Service*
5. *Reliable Service*

Stream Delivery Service

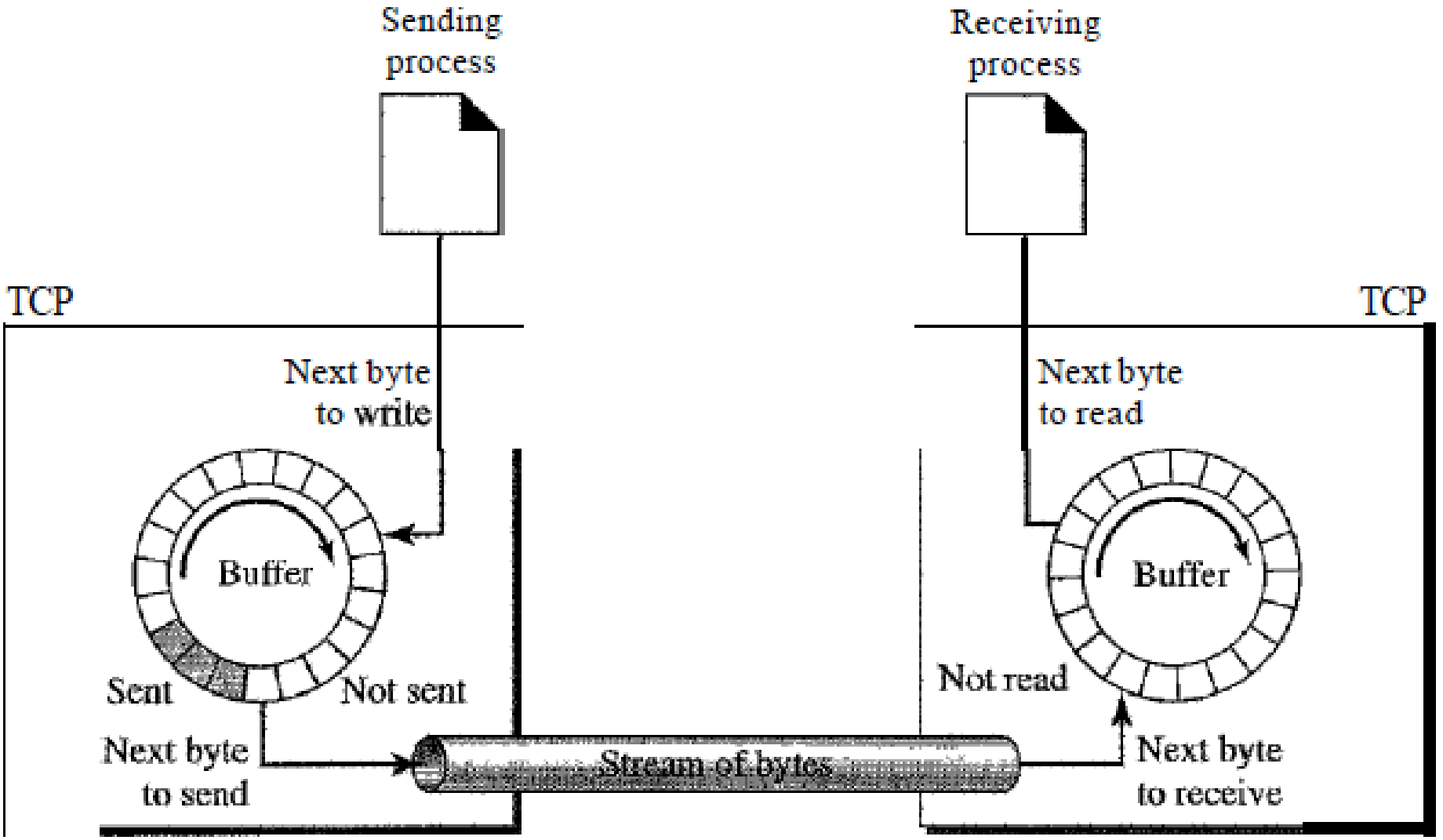
- TCP, unlike UDP, is a stream-oriented protocol. In UDP, a process (an application program) sends messages, with predefined boundaries, to UDP for delivery. UDP adds its own header to each of these messages and delivers them to IP for transmission. Each message from the process is called a user datagram and becomes, eventually, one IP datagram. Neither IP nor UDP recognizes any relationship between the datagrams.
- 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. The sending process produces (writes to) the stream of bytes, and the receiving pr



Sending and Receiving Buffers

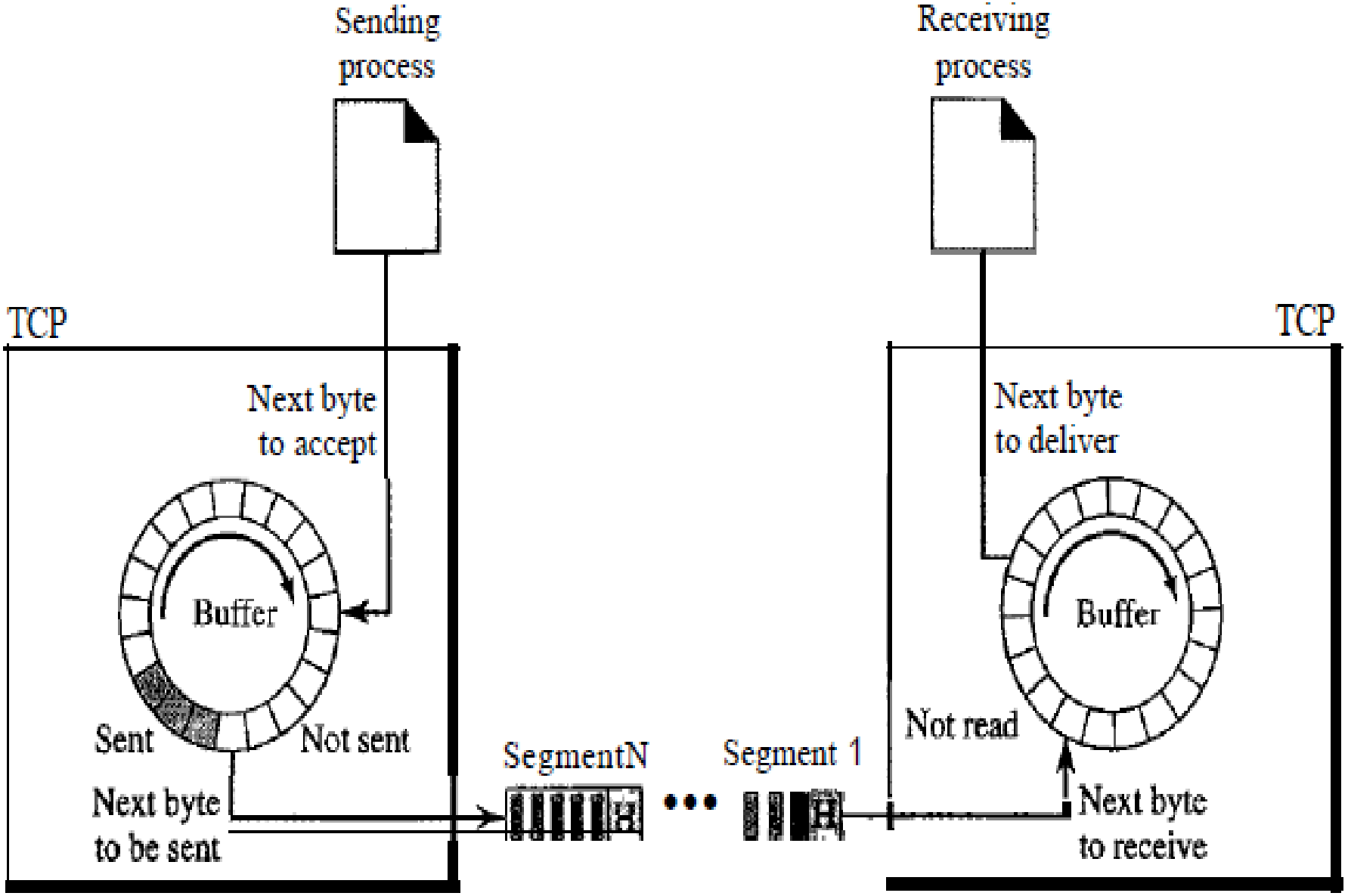
- Because the sending and the receiving processes may not write or read data at the same speed, TCP needs buffers for storage. There are two buffers, the sending buffer and the receiving buffer, one for each direction.
- One way to implement a buffer is to use a circular array of 1-byte locations. For simplicity, we have shown two buffers of 20 bytes each in example next slide; normally the buffers are hundreds or thousands of bytes, depending on the implementation. We also show the buffers as the same size, which is not always the case.

Sending and receiving buffers



Segments

- Although buffering handles the disparity between the speed of the producing and consuming processes, we need one more step before we can send data. The IP layer, as a service provider for TCP, needs to send data in packets, not as 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.


Sending process
Receiving process
TCP
TCP
Next byte to accept
Next byte to deliver
Buffer
Buffer
Sent
Not sent
Not read
SegmentN
Segment 1
Next byte to be sent
Next byte to receive

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.

Connection-Oriented Service

- TCP, unlike UDP, 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.**
- Note that this is a virtual connection, not a physical connection. The TCP segment is encapsulated in an IP datagram and can be sent out of order, or lost, or corrupted, and then resent. Each may use a different path to reach the destination. There is no physical connection. TCP creates a stream-oriented environment in which it accepts the responsibility of delivering the bytes in order to the other site.

TCP Segment *Numbering System*

- Byte Number. TCP numbers all data bytes that are transmitted in a connection. Numbering is independent in each direction. When TCP receives bytes of data from a process, it stores them in the sending buffer and numbers them. The numbering does not necessarily start from 0. Instead, TCP generates a random number between 0 and $2^{32} - 1$ for the number of the first byte. 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. Byte numbering is used for flow and error control too.
- 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**.

Quiz three

- Suppose a TCP connection is transferring a file of 8000 bytes. The first byte is numbered 10,001. What are the sequence numbers for each segment if data are sent in four segments, each carrying 2000 bytes?

Solution

The following shows the sequence number for each segment:

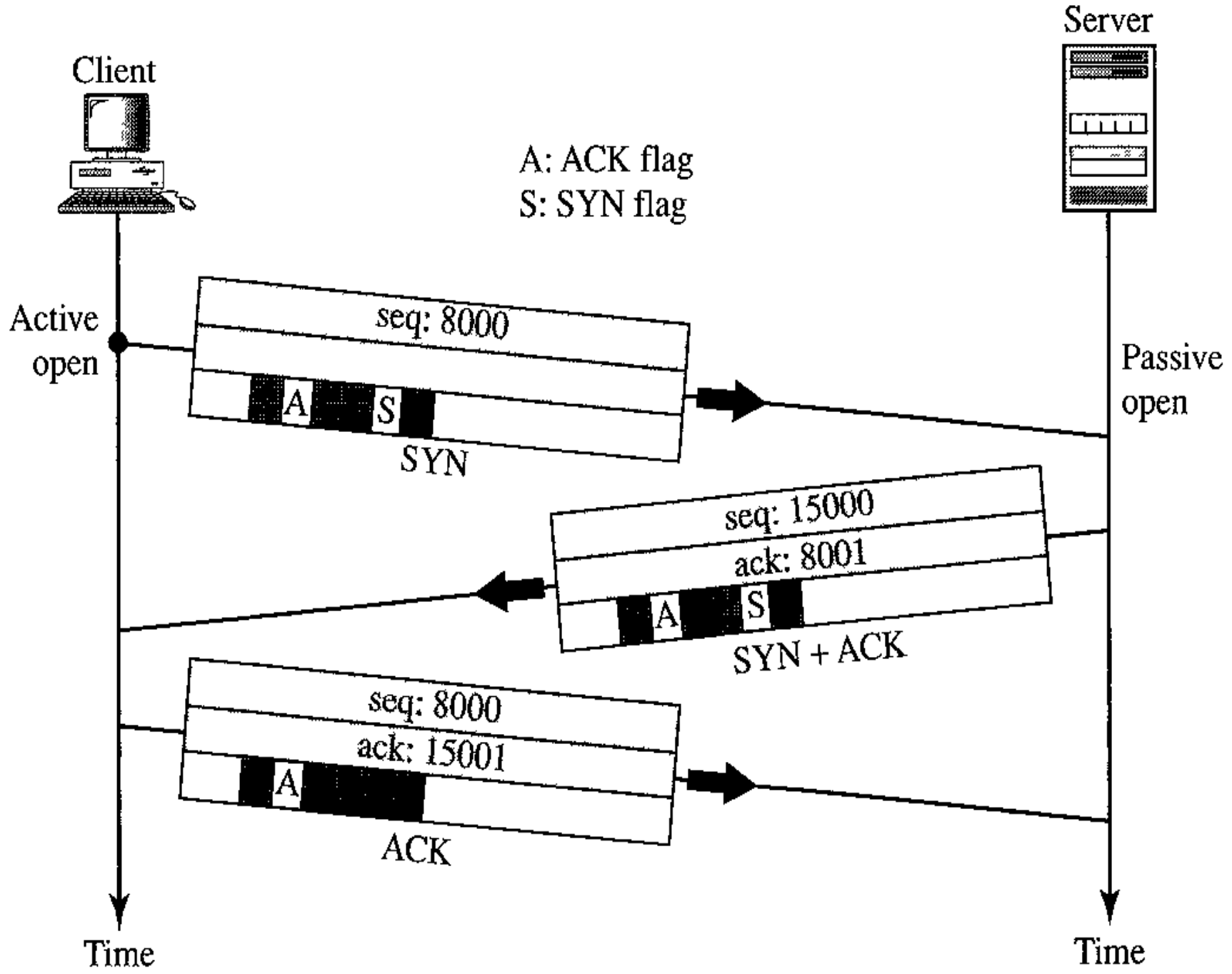
Segment 1 Sequence Number: 10,001 (range: 10,001 to 12,000)
Segment 2 Sequence Number: 12,001 (range: 12,001 to 14,000)
Segment 3 Sequence Number: 14,001 (range: 14,001 to 16,000)
Segment 4 Sequence Number: 16,001 (range: 16,001 to 18,000)

....

- Acknowledgment Number. As we discussed previously, communication in TCP is full duplex; when a connection is established, both parties can send and receive data at the same time. Each party numbers the bytes, usually with a different starting byte number.
- The sequence number in each direction shows the number of the first byte carried by the segment. Each party also uses an acknowledgment number to confirm the bytes it has received. However, the acknowledgment number defines **the number of the next byte** that the party expects to receive. In addition, the acknowledgment number is cumulative, which means that the **party takes the number of the last byte that it has received, safe and sound, adds 1 to it, and announces this sum as the acknowledgment number.**
- **The term *cumulative here means that if a party uses 5643 as an acknowledgment number*, it has received all bytes from the beginning up to 5642. Note that this does not mean that the party has received 5642 bytes because the first byte number might not have to start from 0.**

Three-Way Handshaking in connection establishment

- The connection establishment in TCP is called three-way handshaking. An application program, called the client, wants to make a connection with another application program, called the server, using TCP as the transport layer protocol.
- The process starts with the server. The server program tells its TCP that it is ready to accept a connection. This is called a request for a *passive open*. *Although the server* TCP is ready to accept any connection from any machine in the world, it cannot make the connection itself.
- The client program issues a request for an *active open*. *A client that wishes to connect* to an open server tells its TCP that it needs to be connected to that particular server. TCP can now start the three-way handshaking process as shown in the figure in next slide.
- To show the process, we use two time lines: one at each site. Each segment has values for all its header fields and perhaps for some of its option fields, too. However, we show only the few fields necessary to understand each phase. We show the sequence number, the acknowledgment number, the control flags (only those that are set), and the window size, if not empty.

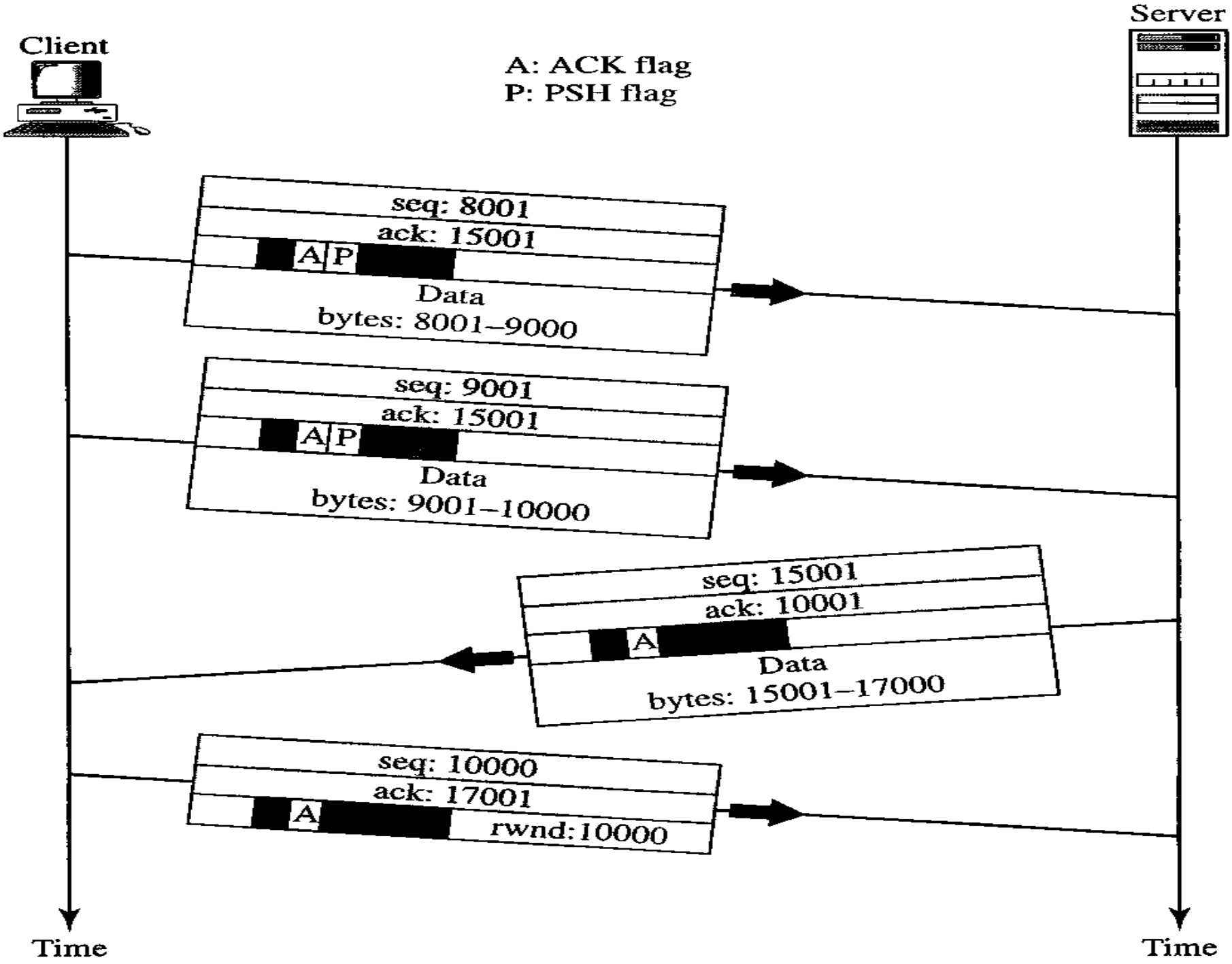

Client
Server
A: ACK flag
S: SYN flag
Active open
Passive open
seq: 8000
A
S
SYN
seq: 15000
ack: 8001
A
S
SYN + ACK
seq: 8000
ack: 15001
A
ACK
Time
Time

The three steps in this phase are as follows.

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

Data Transfer

- After connection is established, bidirectional data transfer can take place. The client and server can both send data and acknowledgments. Data and acknowledgment traveling in the same direction are carried on the same segment.
- The acknowledgment is piggybacked with the data. In the example next slide, after connection is established (not shown in the figure), the client sends 2000 bytes of data in two segments. The server then sends 2000 bytes in one segment. The client sends one more segment. The first three segments carry both data and acknowledgment, but the last segment carries only an acknowledgment because there are no more data to be sent.
- Note the values of the sequence and acknowledgment numbers. The data segments sent by the client have the PSH (push) flag set so that the server TCP knows to deliver data to the server process as soon as they are received.

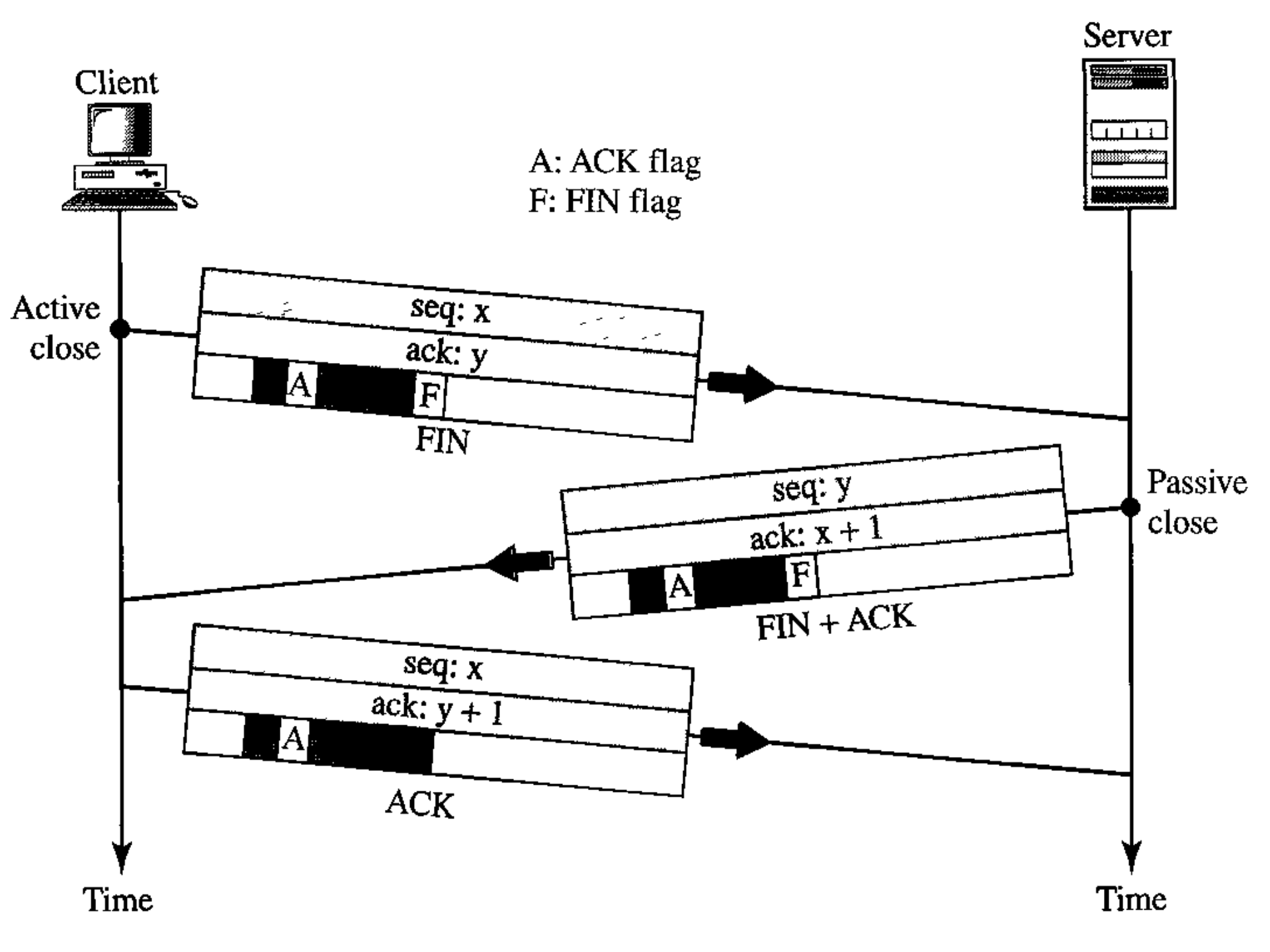

Client
Server
A: ACK flag
P: PSH flag
seq: 8001
ack: 15001
A P
Data
bytes: 8001–9000
seq: 9001
ack: 15001
A P
Data
bytes: 9001–10000
seq: 15001
ack: 10001
A
Data
bytes: 15001–17000
seq: 10000
ack: 17001
A
rwnd:10000
Time
Time

Connection Termination

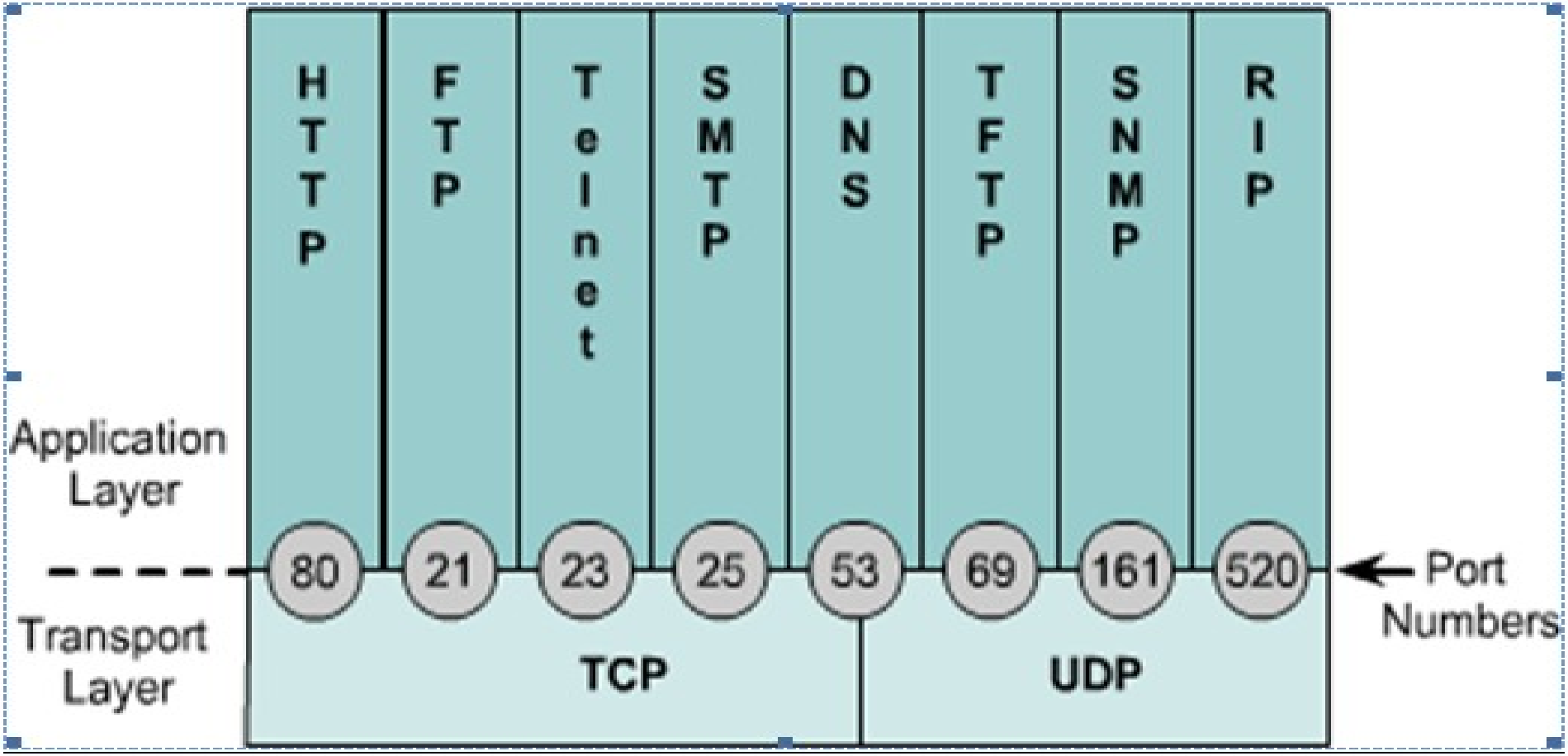
- Any of the two parties involved in exchanging data (client or server) can close the connection, although it is usually initiated by the client.

Three-Way Handshaking for connection termination

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. If it is only a control segment, it consumes only one sequence number.
2. The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN +ACK segment, to confirm the receipt of the FIN segment from the client and at the same time to announce the closing of the connection in the other direction. This segment can also contain the last chunk of data from the server. If it does not carry data, it consumes only one sequence number.
3. The client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server. This segment contains the acknowledgment number, which is 1 plus the sequence number received in the FIN segment from the server. This segment cannot carry data and consumes no sequence numbers.


Client
Server
A: ACK flag
F: FIN flag
Active close
seq: x
ack: y
A
F
FIN
seq: y
ack: x + 1
A
F
FIN + ACK
Passive close
seq: x
ack: y + 1
A
ACK
Time
Time

Some of the Application and Transport layer protocols



DNS can use either UDP or TCP. In both cases the well-known port used by the server is port 53. UDP is used when the size of the response message is less than 512 bytes because most UDP packages have a 512-byte packet size limit. If the size of the response message is more than 512 bytes, a TCP connection is used.