Transport Layer - Introduction

- In a protocol architecture, the transport protocol lies above network or internetwork layer, and just below application and other upper layer protocols.
- 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.

Processes Communication

- It is important to understand how the programs, running in multiple end systems, communicate with each other. They are processes that communicate.
- A process is simply a program that is running within an end system.
- When processes are running on the same end system, they can communicate with each other with inter process communication, using rules that are governed by the end system's operating system.
- We are not particularly interested in how processes in the same host communicate, but instead in how processes running on different hosts communicate.
- Processes on two different end systems communicate with each other by exchanging messages across the computer network.
- A sending process creates and sends messages into the network; a receiving process receives these messages and possibly responds by sending messages back.
- The transport layer is responsible for process to process delivery the delivery of a packet from one process to another.

Services Provided by Transport Layer

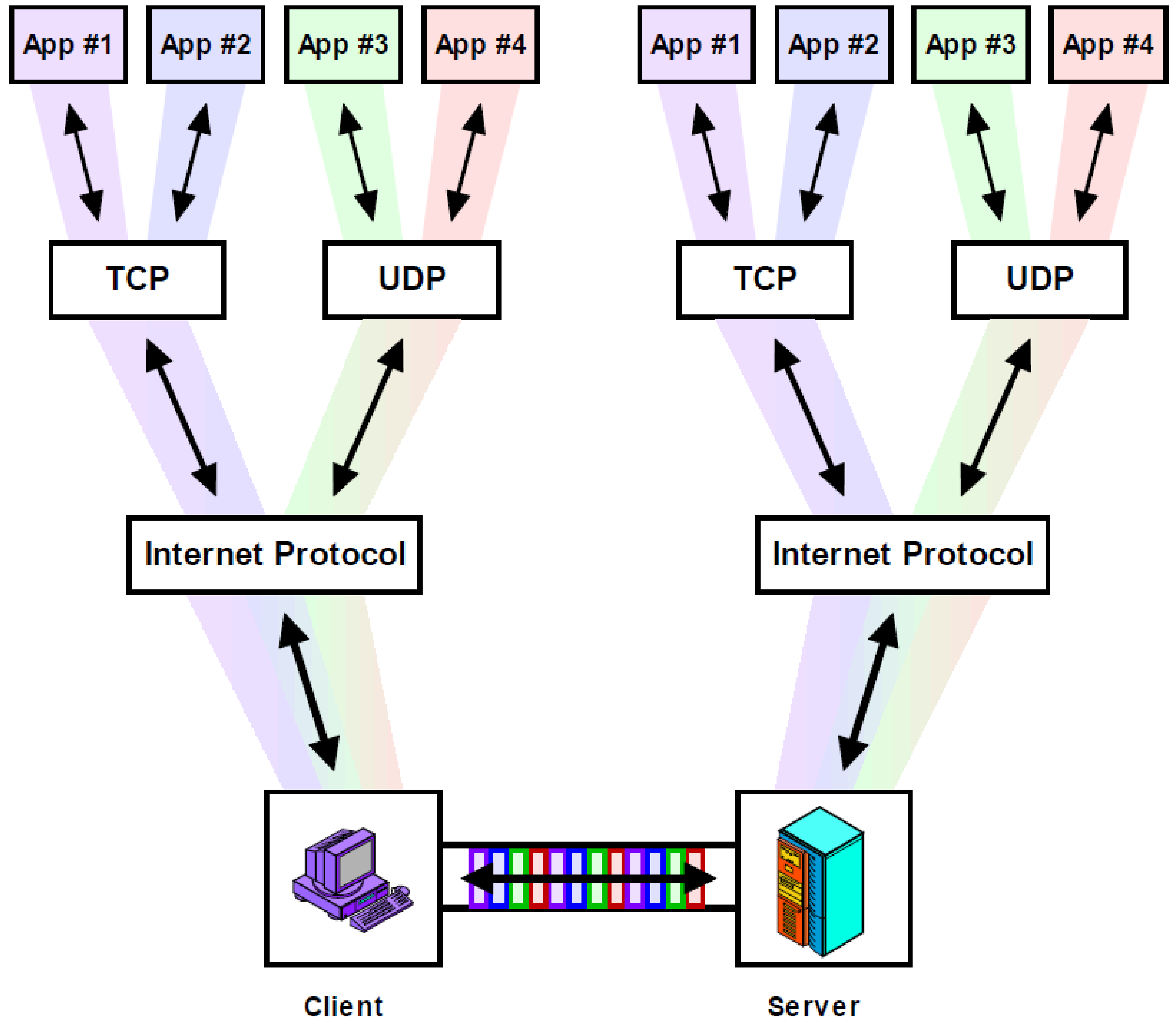
- A transport layer protocol provides logical communication between application processes running on different hosts.
- Logical communication from an application’s perspective is as if the hosts running the processes were directly connected; in reality, the hosts may be on geographically distant connected via numerous routers and a wide range of link types.
- On the sending side, the transport layer converts the application layer messages it receives from a sending application process into transport layer packets, known as transport layer segments.
- This is done by breaking the application messages into smaller chunks and adding a transport layer header to each chunk to create the transport layer segment.
- The transport layer then passes the segment to the network layer at the sending end system, where the segment is encapsulated within a network layer packet (a datagram) and sent to the destination.
- On the receiving side, the network layer extracts the transport layer segment from the datagram and passes the segment up to the transport layer. The transport layer then processes the received segment, making the data in the segment available to the receiving application.

Relationship between Transport & Network Layer

- Any message sent from one process to another must go through the underlying network.
- A process sends messages into, and receives messages from, the network through a software interface called a socket.
- A socket is the interface between the application layer and the transport layer within a host.
- It is also referred to as the Application Programming Interface (API) between the application and the network, since the socket is the programming interface with which network applications are built.
- In order for a process running on one host to send packets to a process running on another host, the receiving process needs to have an address.
- To identify the receiving process, two pieces of information need to be specified:
  - the address of the host, and
  - an identifier that specifies the receiving process in the destination host.
- In the Internet, the host is identified by its IP address.
- In addition to knowing the address of the host to which a message is destined, the sending process must also identify the receiving process running in the host.
- This information is needed because in general a host could be running many network applications. A destination port number is used for this purpose.

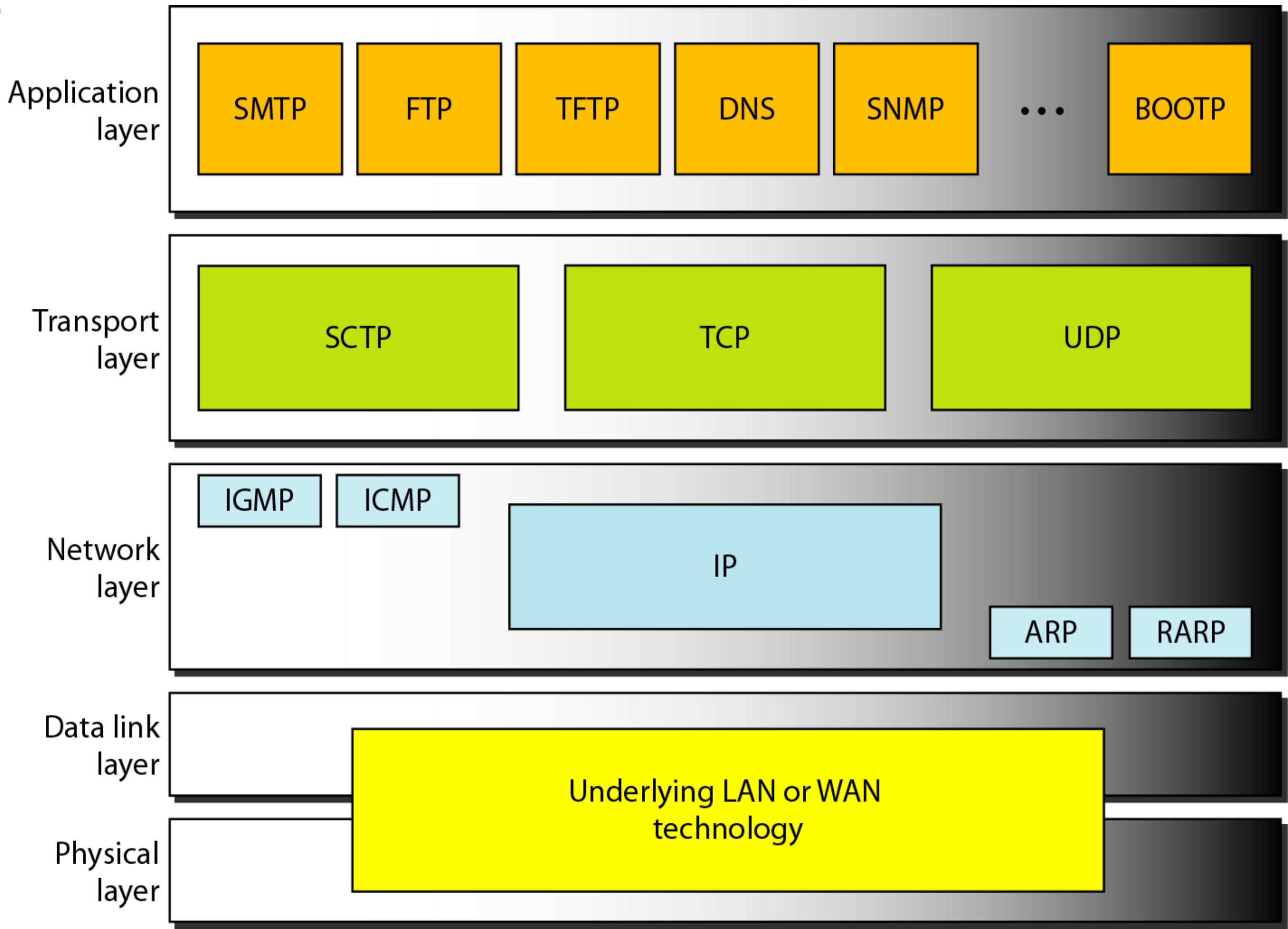
Multiplexing and Demultiplexing

- A sending host has multiple processes each needing to send and receive datagrams.
- All of them, however, must be sent using the same interface to the internetwork, using the IP layer.
- TCP/IP is designed to allow many different applications to send and receive data simultaneously using the same Internet Protocol.
- Transmitted data is multiplexed from many sources as it is passed down to the IP layer.
- As a stream of IP datagrams is received, it is demultiplexed and the appropriate data passed to each application software instance on the receiving host.
- Transport layer multiplexing requires:
    - Sockets to have unique identifiers, and
    - Each segment to have special fields that indicate the socket to which the segment is to be delivered.
- These special fields, are the source port number field and the destination port number field.
- Each port number is a 16 bit number, ranging from 0 to 65535.
- Common application protocols such as HTTP uses port number 80 and FTP uses port number 21.


App #1
App #2
App #3
App #4
App #1
App #2
App #3
App #4
TCP
UDP
TCP
UDP
Internet Protocol
Internet Protocol
Client
Server

Transport Layer Protocols

- The TCP/IP protocol suite specifies two protocols for the transport layer:
- UDP – connectionless and unreliable service.
- TCP – connection oriented and reliable service



User Datagram Protocol (UDP) - Operation

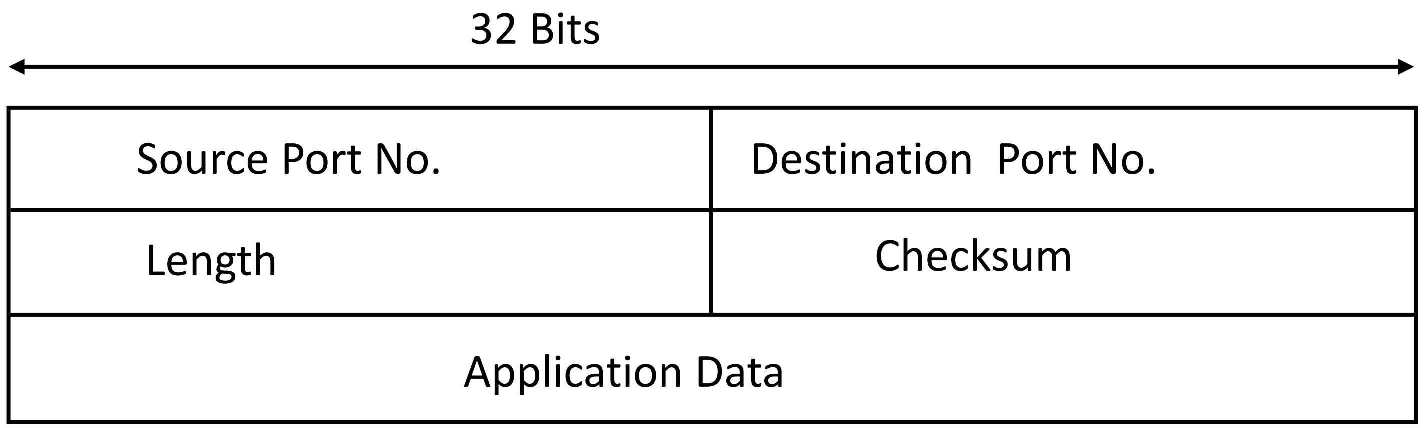
- UDP's task is to take data from higher layer protocols and place it in UDP messages, which are then passed down to the Internet Protocol for transmission. The basic steps for transmission using UDP are:
- Higher Layer Data Transfer - An application sends a message to the UDP software.
- UDP Message Encapsulation - The higher layer message is encapsulated into the Data field of a UDP message. The headers of the UDP message are filled in, including the Source Port of the application that sent the data to UDP, and the Destination Port of the intended recipient. The checksum value may also be calculated.
- Transfer Message To IP - The UDP message is passed to IP for transmission.

User Datagram Protocol (UDP) - Operation

- UDP does not do include the following:
- UDP does not establish connections before sending data. – connectionless service.
- UDP does not provide acknowledgments to show that data was received. – unreliable service.
- UDP does not provide any guarantees that its messages will arrive.
- UDP does not detect lost messages and retransmit them.
- UDP does not ensure that data is received in the same order that they were sent.
- UDP does not provide any mechanism to manage the flow of data between devices, or handle congestion.

UDP – Segment Structure

- The UDP header has four fields, each consisting of two bytes.
- Source Port number - The 16 bit port number of the process that originated the UDP message on the source devices.
- Destination port number - The 16 bit port number of the process that is the ultimate intended recipient of the message on the destination device.
- Length – This field specifies the number of bytes in the UDP segment (header plus application data).
- Checksum - is used by the receiving host to check whether errors have been introduced into the segment.
- Application Data - The encapsulated higher-layer message to be sent.



UDP - Checksum

- The UDP checksum provides for error detection. That is, the checksum is used to determine whether bits within the UDP segment have been altered as it moved from source to destination.
- UDP at the sender side performs the 1s complement of the sum of all the 16 bit words in the segment, with any overflow encountered during the sum being wrapped around. This result is put in the checksum field of the UDP segment.
- At the receiver, all four 16 bit words are added, including the checksum. If no errors are introduced into the packet, then clearly the sum at the receiver will be all 1's. If one of the bits is a 0, then we know that errors have been introduced into the packet.
- Although UDP provides error checking, it does not do anything to recover from an error. Sometimes UDP simply discard the damaged segment or pass the damaged segment to the application with a warning.
- The use of the Checksum field is optional in UDP. If it is not used, it is set to a value of all zeroes. This could potentially create confusion, however, since when the checksum is used, the calculation can sometimes result in a value of zero. To avoid having the destination think the checksum was not used in this case, this zero value is instead represented as a value of all ones.

Why Some TCP/IP Applications Use UDP

- The following lists some uses of the UDP protocol:
- Data Where Performance Is More Important Than Completeness - multimedia application. If you are streaming a video clip over the Internet, the most important thing is that the stream starts flowing quickly and keeps flowing. We only notice significant disruptions in the flow of this type of information, so a few bytes of data missing due to a lost datagram is not a big problem.
- Data Exchanges That Are Short - There are many TCP/IP applications where the underlying protocol consists of only a very simple request/reply exchange. A short request message is sent from a client to a server, and a short reply message goes back from the server to the client. In this situation, there is no real need to set up a connection like TCP does.
- If an application needs to multicast or broadcast data, it must use UDP, because TCP is only supported for unicast communication between two devices.

Transmission Control Protocol(TCP) - Characteristics

- TCP is connection-oriented service - TCP requires that devices first establish a connection with each other before they send data.
- Suppose a process(client process) running in one host wants to initiate a connection with another process(server process) in another host.
- The client application process first informs the client transport layer that it wants to establish a connection to a process in the server.
- TCP in the client then proceeds to establish a TCP connection with TCP in the server. The client first sends a special TCP segment; the server responds with a second special TCP segment; and finally the client responds again with a third special segment.
- The first two segments carry no payload, that is, no application layer data; the third of these segments may carry a payload.
- Because three segments are sent between the two hosts, this connection establishment procedure is often referred to as a three way handshake.

Transmission Control Protocol(TCP) - Characteristics

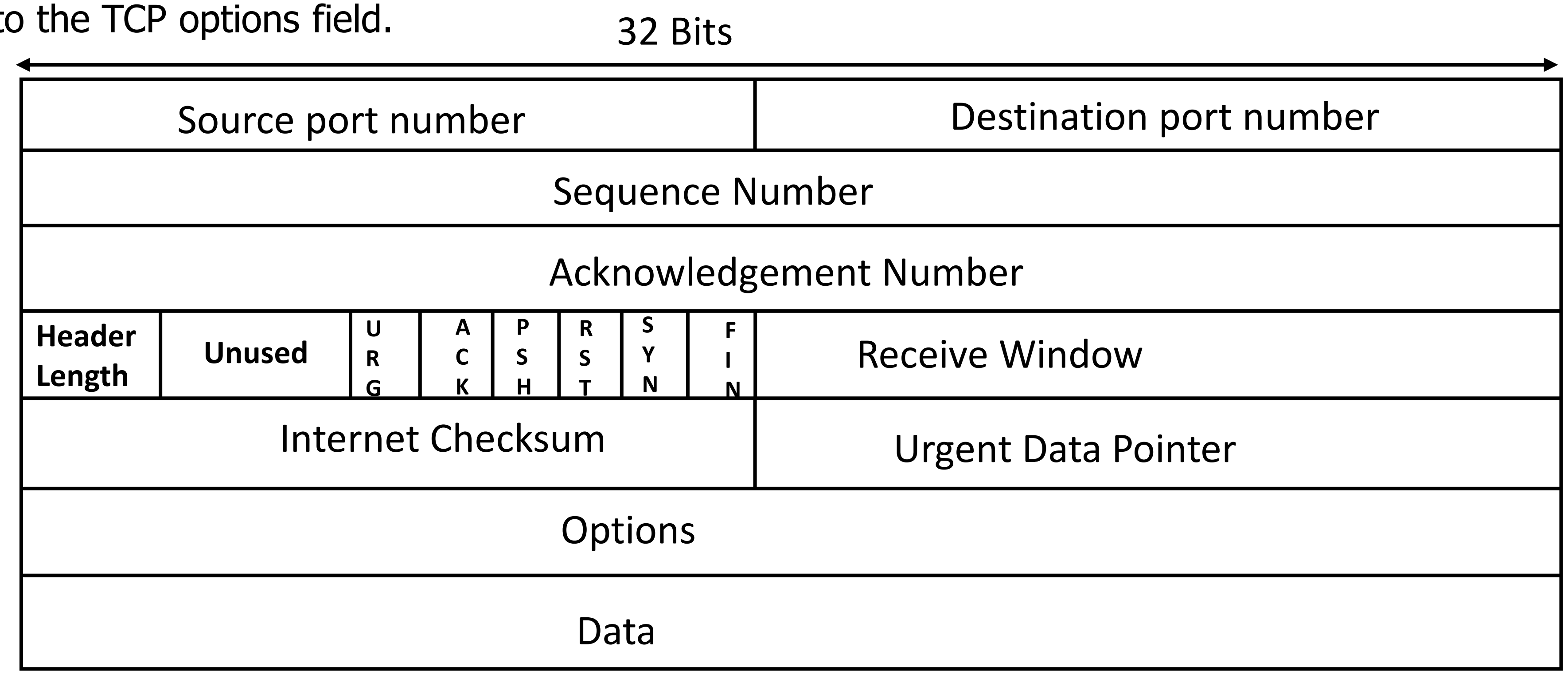
- Once a TCP connection is established, the two application processes can send data to each other.
- The client process passes a stream of data through the socket . Once the data passes through the socket , the data is in the hands of TCP running in the client. TCP directs this data to the connection's send buffer, which is one of the buffers that is set aside during the initial three-way handshake.
- From time to time, TCP will grab chunks of data from the send buffer and pass the data to the network layer. The maximum amount of data that can be grabbed and placed in a segment is limited by the maximum segment size (MSS).
- TCP pairs each chunk of client data with a TCP header, thereby forming TCP segments. The segments are passed down to the network layer, where they are separately encapsulated within network layer IP datagrams.
- The IP datagrams are then sent into the network. When TCP receives a segment at the other end, the segment's data is placed in the TCP connection's receive buffer. The application reads the stream of data from this buffer.
- Each side of the connection has its own send buffer and its own receive buffer.

Transmission Control Protocol(TCP) - Characteristics

- A TCP connection provides a full duplex service - if there is a TCP connection between Process A on one host and Process B on another host, then application layer data can flow from Process A to Process B at the same time as application layer data flows from Process B to Process A.
- A TCP connection is also always point to point - between a single sender and a single receiver.
- TCP is reliable - TCP keeps track of data that has been sent and received to ensure it all gets to its destination. A key to providing reliability is that all transmissions in TCP are acknowledged.
- TCP provides flow control and congestion avoidance - allows the flow of data between two devices to be controlled and managed. It also includes features to deal with congestion that may be experienced during communication between devices.

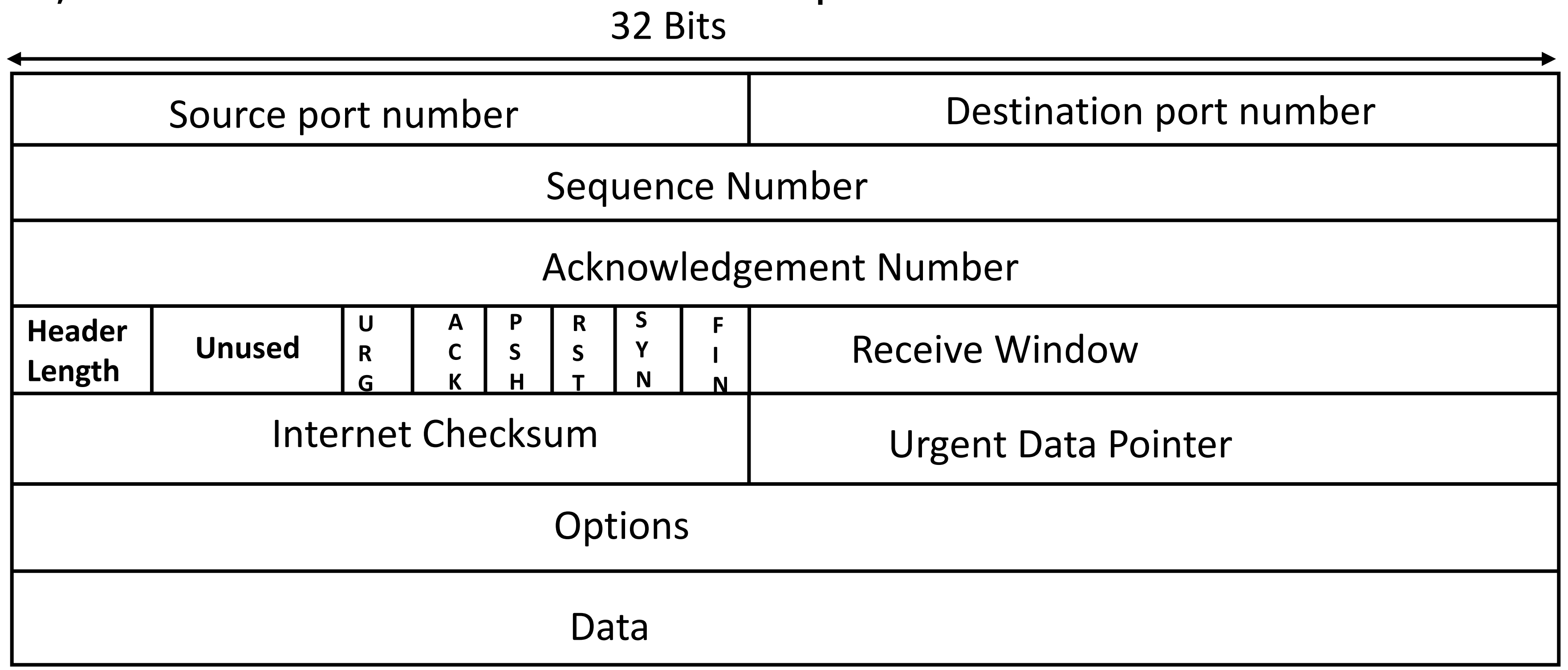
TCP Segment Structure

- Source & Destination Port Number - specifies the 16 bit port number of the process that originated the TCP segment on the source device and the 16 bit port number of the process that is the ultimate intended recipient of the message on the destination device respectively.
- Sequence number and acknowledgment number field - are used by the TCP sender and receiver in implementing a reliable data transfer service.
- Receive window field - used for flow control.
- Header length field - specifies the length of the TCP header in 32 bit words. The TCP header can be of variable length due to the TCP options field.



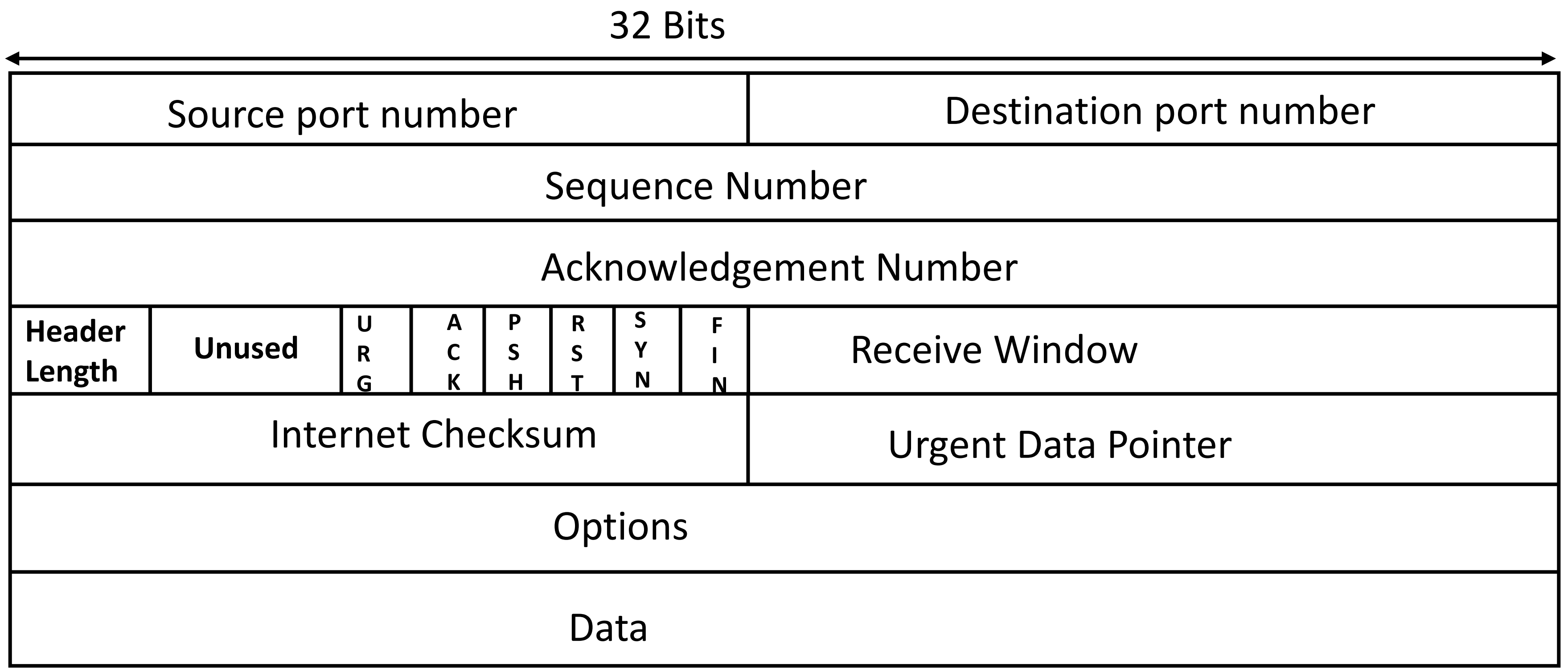
TCP Segment Structure

- Flags - For each flag, if set to 1, the meaning is:
  - URG – used to indicate that there is data in this segment that the sending side upper layer entity has marked as urgent.
  - ACK - used to indicate that the value carried in the acknowledgment field is valid.
  - PSH - indicates that the receiver should pass the data to the upper layer immediately.
  - RST, SYN, and FIN - used for connection setup and teardown.



TCP Segment Structure

- Internet Checksum – used for data integrity protection, computed over the entire TCP datagram.
- Urgent Pointer - Used in conjunction with the URG control bit for priority data transfer. This field contains the sequence number of the last byte of urgent data. This allows the receiver to know how much urgent data is coming.
- Options field - is used when a sender and receiver negotiate options such as the maximum segment size (MSS).



TCP Flow Control

- TCP provides a flow control service to its applications to eliminate the possibility of the sender overflowing the receiver’s buffer.
- Flow control is thus a speed matching service—matching the rate at which the sender is sending against the rate at which the receiving application is reading.
- TCP provides flow control by having the sender maintain a variable called the receive window. Informally, the receive window is used to give the sender an idea of how much free buffer space is available at the receiver.
- TCP is full duplex, the sender at each side of the connection maintains a distinct receive window.

TCP Congestion Control

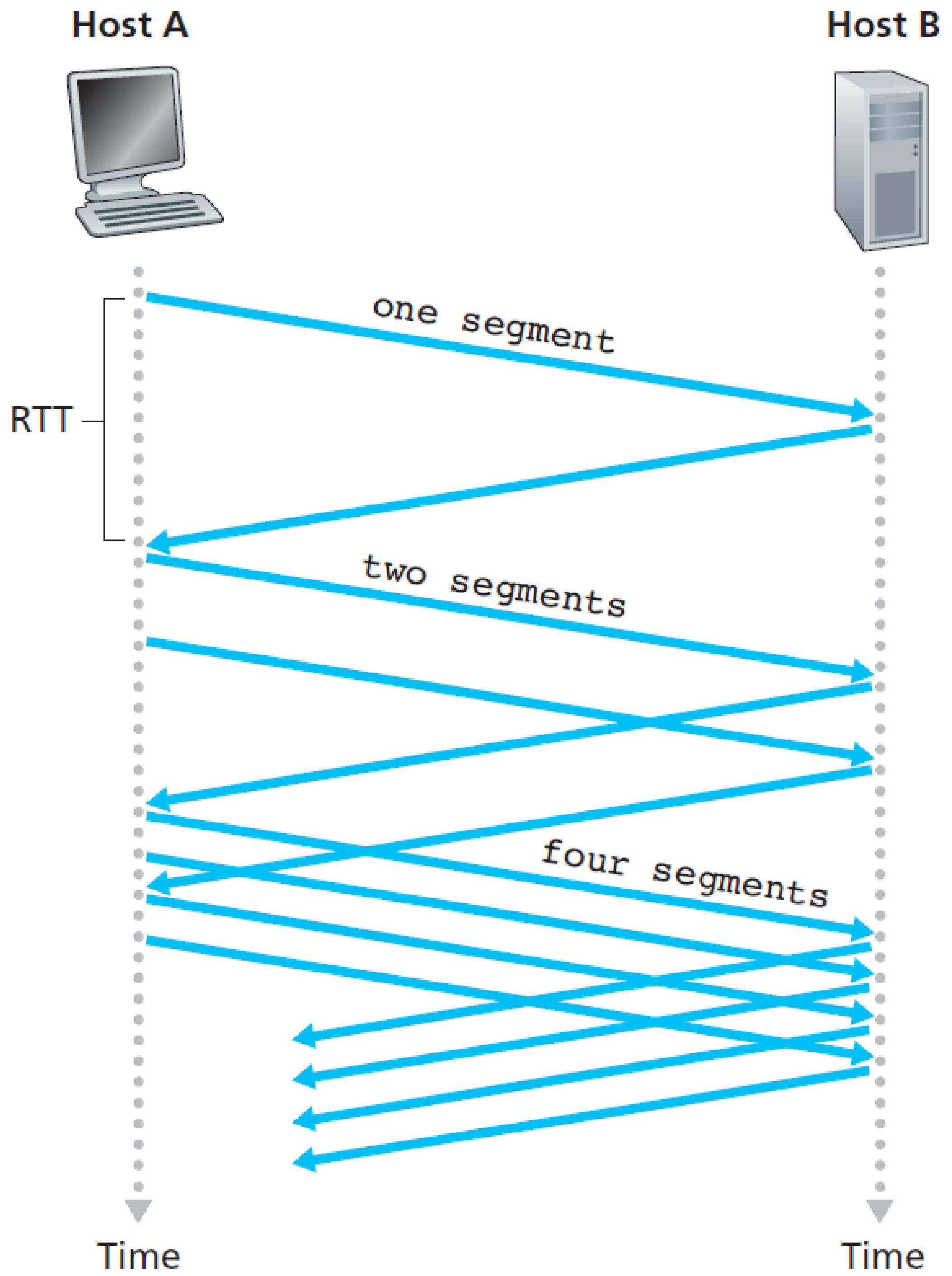
- TCP provides an end to end approach to congestion control, in which the network layer provides no explicit support to the transport layer for congestion control purposes. Even the presence of congestion in the network must be inferred by the end systems based only on observed network behavior (for example, packet loss and delay).
- The approach taken by TCP is to have each sender limit the rate at which it sends traffic into its connection as a function of perceived network congestion.
- If a TCP sender perceives that there is little congestion on the path between itself and the destination, then the TCP sender increases its send rate; if the sender perceives that there is congestion along the path, then the sender reduces its send rate.
- Three questions must be answered here. They are:
- How does a TCP sender limit the rate at which it sends traffic into its connection?
- How does a TCP sender perceive that there is congestion on the path between itself and the destination?
- What algorithm should the sender use to change its send rate as a function of perceived end-to-end congestion?

TCP Congestion Control

- How does a TCP sender limit the rate at which it sends traffic into its connection?
- Each side of a TCP connection consists of a receive buffer, a send buffer. The TCP congestion control mechanism operating at the sender keeps track of an additional variable, the congestion window.
- The congestion window imposes a constraint on the rate at which a TCP sender can send traffic into the network.
- Thus the sender's send rate is $congestion\ window/RTT\ bytes/sec$. By adjusting the value of congestion window , the sender can therefore adjust the rate at which it sends data into its connection.
- How a TCP sender perceives that there is congestion on the path between itself and the destination?
- When there is excessive congestion, then one (or more) router buffers along the path overflows, causing a datagram (containing a TCP segment) to be dropped.
- The dropped datagram may result in timeout, which is taken by the sender to be an indication of congestion on the sender to receiver path.
- What algorithm should the sender use to change its send rate as a function of perceived end-to-end congestion?
- When the network is congestion free and acknowledgments for previously unacknowledged segments are received at the TCP sender, TCP will take the arrival of these acknowledgments as an indication that all the segments being transmitted into the network are being successfully delivered to the destination.
- TCP will use acknowledgments to increase its congestion window size and hence its transmission rate.

TCP Congestion Control Algorithm

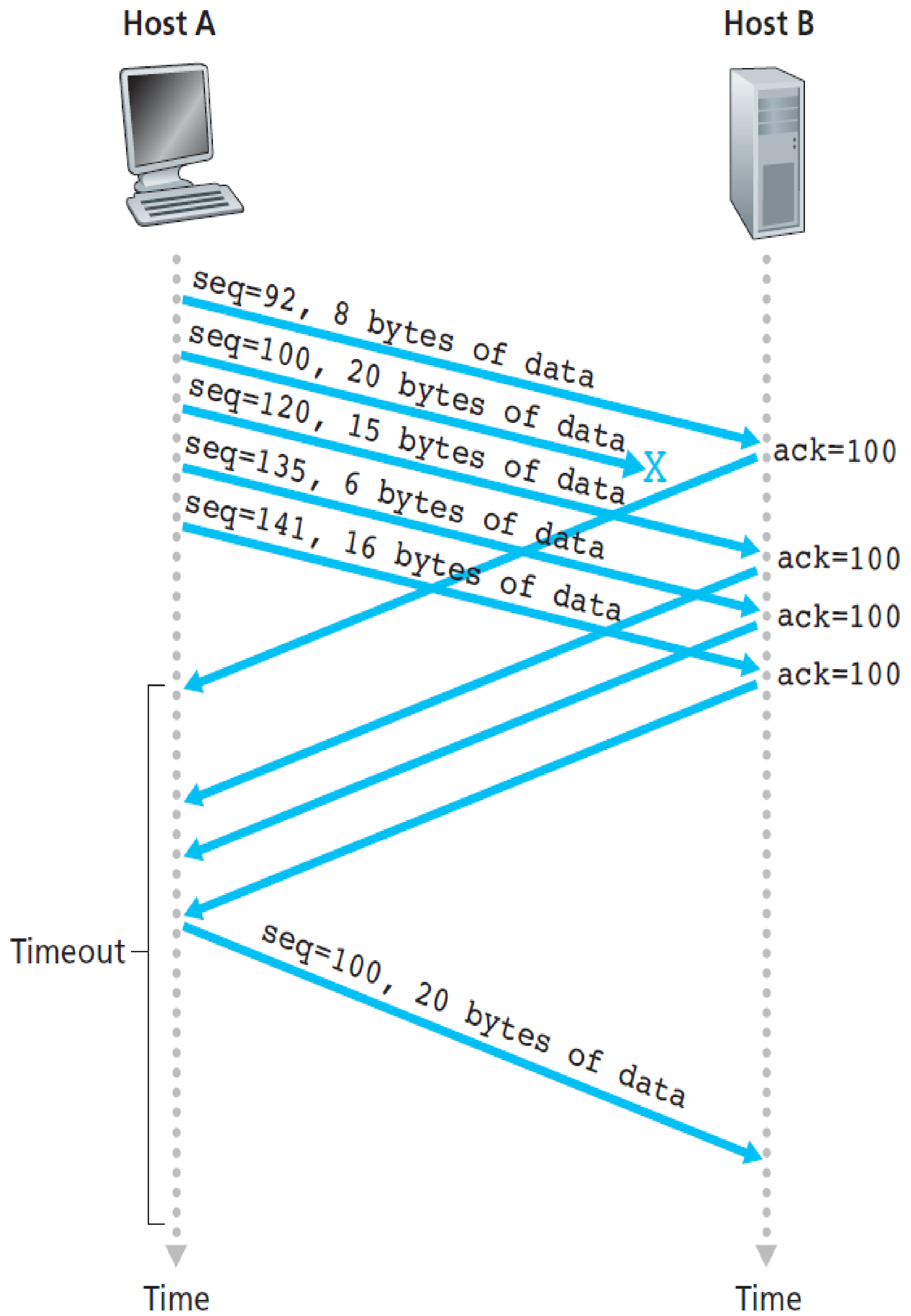
- The algorithm has three major components.
- Slow Start - When a TCP connection begins, the value of congestion window is typically initialized to a small value of 1 MSS, resulting in an initial sending rate of roughly MSS/RTT.
- The value of congestion window begins at 1 MSS and increases by 1 MSS every time a transmitted segment is first acknowledged.



- TCP sends the first segment into the network and waits for an acknowledgment.
- When this acknowledgment arrives, the TCP sender increases the congestion window by one MSS and sends out two maximum sized segments.
- These segments are then acknowledged, with the sender increasing the congestion window by 1 MSS for each of the acknowledged segments, giving a congestion window of 4 MSS, and so on.
- Thus, the TCP send rate starts slow but grows exponentially during the slow start phase.

TCP Congestion Control Algorithm

- Slow start cannot continue indefinitely. There must be a threshold to stop this phase. The sender keeps track of a variable named ssthresh (slow start threshold).
- When the size of window in bytes reaches this threshold, slow start stops and the next phase starts.
- In most implementations the value of ssthresh is 65,535 bytes.
- Congestion Avoidance – When the value of congestion window reaches the slow start threshold , TCP adopts a more conservative approach and increases the value of congestion window by just a single MSS every RTT.
- Fast Retransmit – One of the problems with timeout triggered retransmissions is that the timeout period can be relatively long. When a segment is lost, this long timeout period forces the sender to delay resending the lost packet, thereby increasing the end to end delay. The sender can often detect packet loss well before the timeout event occurs by duplicate ACKs. A duplicate ACK is an ACK that re-acknowledges a segment for which the sender has already received an earlier acknowledgment.



- When a TCP receiver receives a segment with a sequence number that is larger than the next, expected, in-order sequence number, it detects a gap in the data stream—that is, a missing segment.
- Since TCP does not use negative acknowledgments, the receiver cannot send an explicit negative acknowledgment back to the sender.
- Instead, it simply re –acknowledges (that is, generates a duplicate ACK for) the last in-order byte of data it has received.
- If the TCP sender receives three duplicate ACKs for the same data, it takes this as an indication that the segment following the segment that has been ACKed three times has been lost.
- In the case that three duplicate ACKs are received, the TCP sender performs a fast retransmit , retransmitting the missing segment before that segment's timer expires.