TRANSMISIÓN CONFIABLE DE DATOS

REDES DE DATOS

TRANSPORT LAYER PROTOCOLS

TRANSPORTATION OF DATA

Role of the Transport Layer

- Responsible for establishing a **temporary communication session** between two applications and delivering data between them.
- Provides **Connection-oriented data stream support, Reliability, Flow control, Multiplexing**

Transport Layer Responsibilities

- Track individual conversations.
- Segment Data and Reassemble Segments.
- Identify the Applications.

Conversation Multiplexing

- Segments data into small chunks.
- Label data chunks according to the conversation.

Transport Layer Reliability

- Two protocols provided: TCP and UDP.
- TCP supports reliability while UDP doesn't.

TRANSPORT LAYER PROTOCOLS

TRANSPORTATION OF DATA

TCP

- **Supports packet delivery confirmation.**
- There are three basic operations that enable reliability with TCP:
  - Numbering and tracking data segments transmitted to a specific host from a specific application
  - Acknowledging received data
  - Retransmitting any unacknowledged data after a certain period of time

UDP

- UDP provides the basic functions for delivering data segments between the appropriate applications, with very **little overhead and data checking**.
- Perfect for applications that don't require reliability.

The Right Transport Layer Protocol for the Right Application

- TCP is better for databases, web browsers, email clients, etc.
- UDP is better for live audio or video streaming, VoIP, etc.

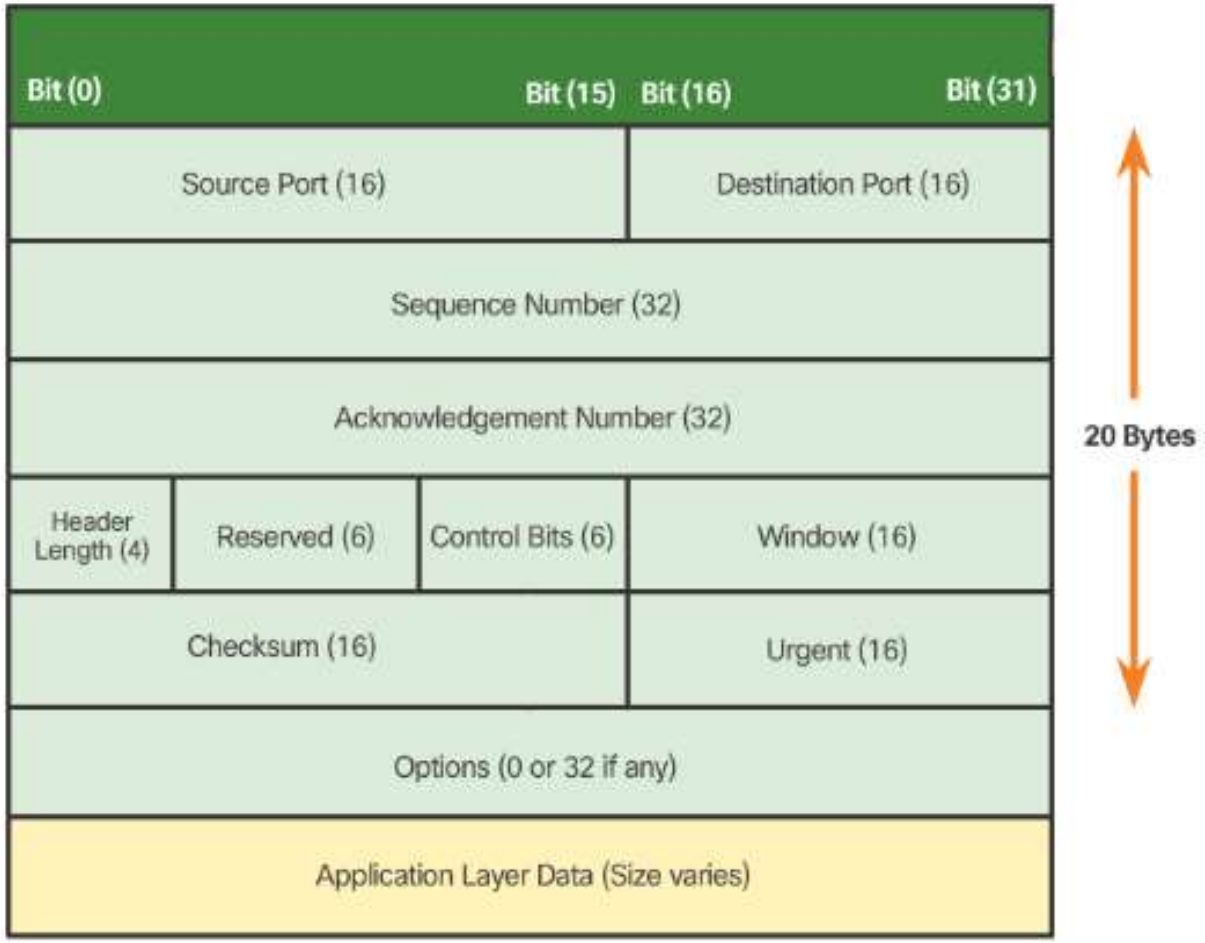
TCP OVERVIEW

TCP Features

- Establishing a session
- Reliable delivery
- Same-Order delivery
- Flow control

TCP Header

- TCP is a stateful protocol.
- TCP adds 20 bytes of overhead in the segment header.



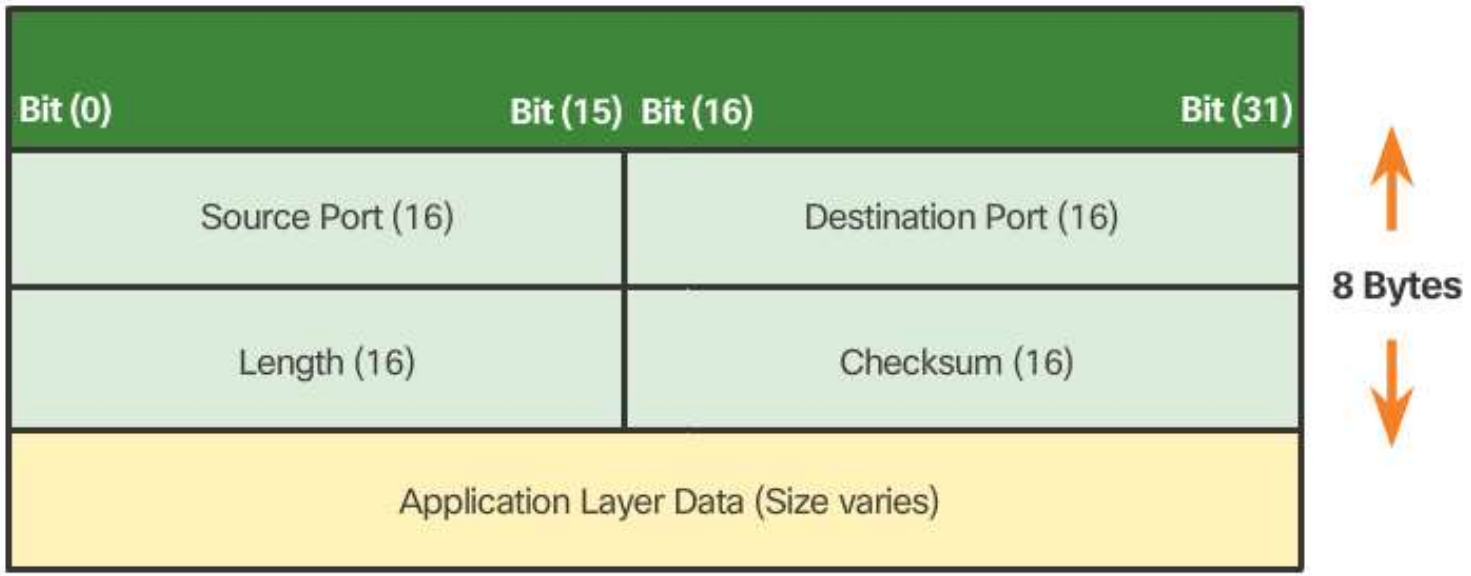
UDP OVERVIEW

UDP Features

- Simple and fast.

UDP Header

- UDP is a stateless protocol.
- Reliability must be handled by the application.
- The pieces of communication in UDP are called Datagrams.
- UDP adds only 8 bytes of overhead.



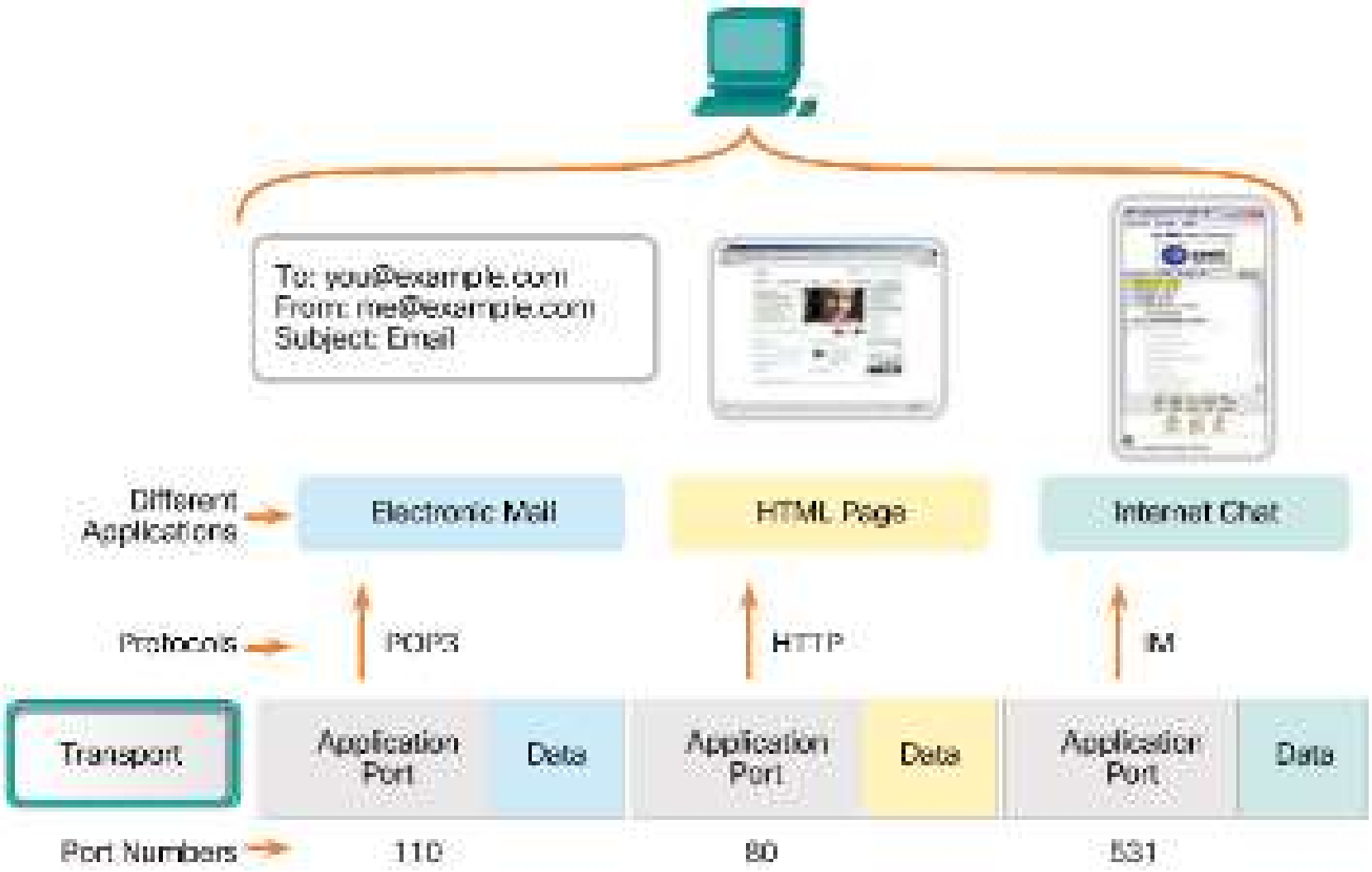
PORT NUMBERS

Multiple Separate Conversations

- The transport layer separate sand manages multiple communications with different transport requirements.
- Different applications are sending and receiving data over the network simultaneously.
- Unique header values allow TCP and UDP to manage these multiple and simultaneous conversations by identifying these applications.
- These unique identifiers are the port numbers.

Port Numbers

- Usually seen in pairs: source port and destination port.
- The source port is dynamically chosen by the sender.
- The destination port is used to identify an application on the server (destination).


To: you@example.com
From: me@example.com
Subject: Email
Different Applications
Electronic Mail
HTML Page
Internet Chat
Protocols
POP3
HTTP
IM
Transport
Application Port
Data
Application Port
Data
Application Port
Data
Port Numbers
110
80
531

TCP AND UDP

Socket Pairs

- The combination of the source IP address and source port number, or the destination IP address and destination port number, is known as a socket.
- The socket is used to identify the server and service being requested by the client.
- Two sockets combine to form a socket pair: (192.168.1.5:1099, 192.168.1.7:80).
- Sockets enable multiple processes running on a client and multiple connections to a server process to be distinguished from each other.

Port Number Groups

- The IANA has created three port number groups:
- Well-known ports (0 to 1023)
- Registered Ports (1024 to 49151)
- Private and/or Dynamic Ports (49152 to 65535)

TCP COMMUNICATION PROCESS

TCP Server Processes

- Each application process running on the server uses a port number.
- An individual server cannot have two services assigned to the same port number within the same transport layer service.
- An active server application assigned to a specific port is considered to be open.
- Any incoming client request addressed to an open port is accepted and processed by the server application bound to that port.
- There can be many ports open simultaneously on a server, one for each active server application.

TCP Connection Establishment

- A TCP connection is established in three steps:
    - The initiating client requests a client-to-server communication session with the server.
    - The server acknowledges the client-to-server communication session and requests a server-to-client communication session.
    - The initiating client acknowledges the server-to-client communication session.

TCP COMMUNICATION PROCESS

TCP Session Termination

- The FIN TCP flag is used to terminate a TCP connection.
  - When the client has no more data to send in the stream, it sends a segment with the FIN flag set.
  - The server sends an ACK to acknowledge the receipt of the FIN to terminate the session from client to server.
  - The server sends a FIN to the client to terminate the server-to-client session.
  - The client responds with an ACK to acknowledge the FIN from the server.
  - When all segments have been acknowledged, the session is closed.

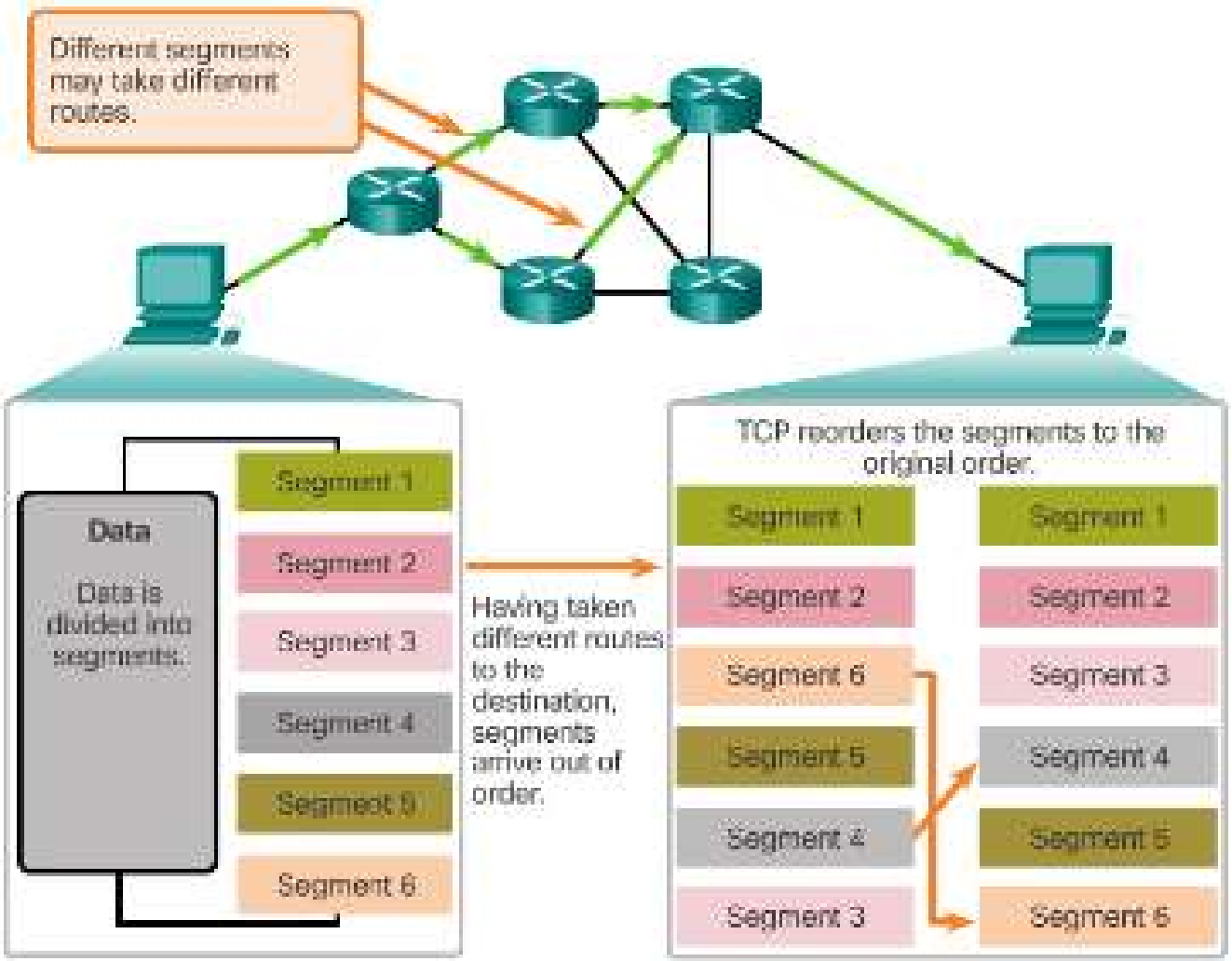
TCP Three-way Handshake Analysis

- The three-way handshake:
  - Establishes that the destination device is present on the network.
  - Verifies that the destination device has an active service and is accepting requests on the destination port number that the initiating client intends to use
  - Informs the destination device that the source client intends to establish a communication session on that port number.

RELIABILITY AND FLOW CONTROL

TCP Reliability – Ordered Delivery

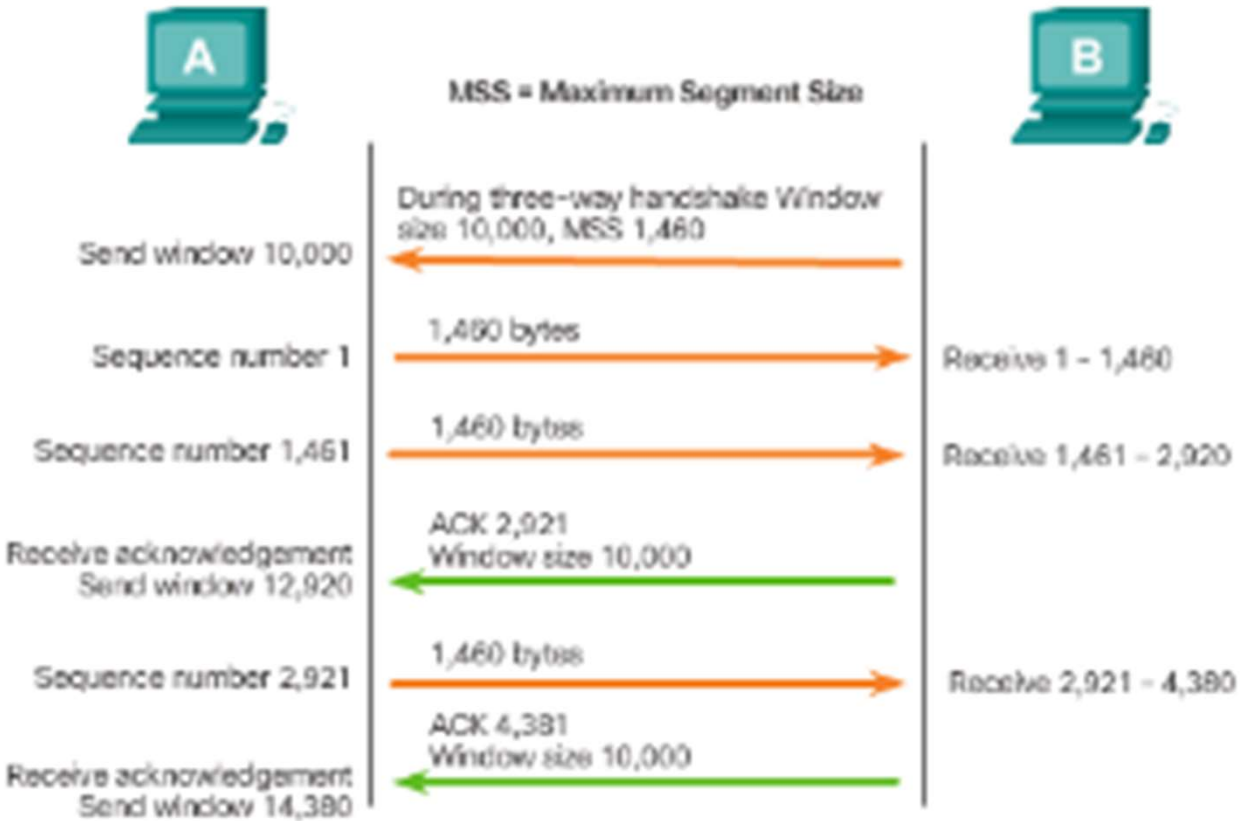
- TCP segments use sequence numbers to uniquely identify and acknowledge each segment, keep track of segment order, and indicate how to reassemble and reorder received segments.
- An initial sequence number (ISN) is randomly chosen during the TCP session setup. The ISN is then incremented by the number of transmitted bytes.
- The receiving TCP process buffers the segment data until all data is received and reassembled.
- Segments received out of order are held for later processing.
- The data is delivered to the application layer only when it has been completely received and reassembled.


Different segments may take different routes.
Data
Data is divided into segments.
Segment 1
Segment 2
Segment 3
Segment 4
Segment 5
Segment 6
Having taken different routes to the destination, segments arrive out of order.
TCP reorders the segments to the original order.
Segment 1
Segment 2
Segment 6
Segment 5
Segment 4
Segment 3
Segment 1
Segment 2
Segment 3
Segment 4
Segment 5
Segment 6

RELIABILITY AND FLOW CONTROL

TCP Flow Control – Window Size and Acknowledgments

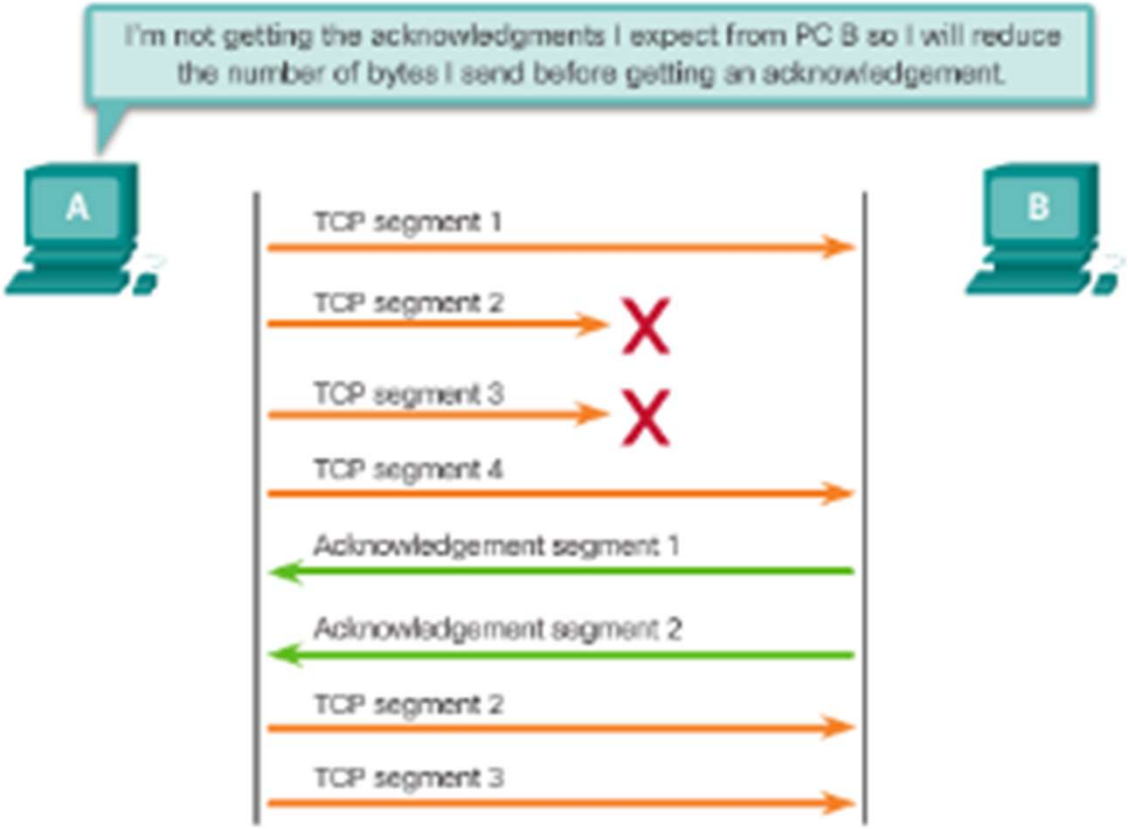
- TCP provides mechanisms for flow control.
- Flow control ensures the TCP endpoints can receive and process data reliably.
- TCP handles flow control by adjusting the rate of data flow between source and destination for a given session.
- TCP flow control function relies on a 16-bit TCP header field called the Window size. The window size is the number of bytes that the destination device of a TCP session can accept and process at one time.
- TCP source and destination agree on the initial window size when the TCP session is established
- TCP endpoints can adjust the window size during a session if necessary.



The **window size** determines the number of bytes that can be sent before expecting an acknowledgment.
The **acknowledgement** number is the number of the next expected byte.

TCP Flow Control – Congestion Avoidance

- Network congestion usually results in discarded packets.
- Undelivered TCP segments trigger re-transmission. TCP segment retransmission can make the congestion even worse.
- The source can estimate a certain level of network congestion by looking at the rate at which TCP segments are sent but not acknowledged.
- The source can reduce the number of bytes it sends before receiving an
  acknowledgement upon congestion detection.
- The source reduces the number of unacknowledged bytes it sends and not
  the window size, which is determined by the destination.
- The destination is usually unaware of the network congestion and sees no need to suggest a new window size.



Acknowledgement numbers are for the next expected byte and not for a segment. Segment number are only used here for simplicity.

UDP Low Overhead Vs. Reliability

- UDP has much lower overhead than TCP.
- UDP is not connection-oriented and does not offer the sophisticated retransmission, sequencing, and flow control mechanisms.
- Applications running UDP can still use reliability, but it must be implemented in the application layer.
- However, UDP is not inferior.

UDP Datagram Reassembly

- UDP simply reassembles the data in the order in which it was received.
- The application must identify the proper sequence, if necessary.

UDP Server Processes and Requests

- UDP-based server applications are also assigned well-known or registered port numbers.
- Requests received on a specific port are forwarded to the proper application based on port numbers.

UDP Client Processes

- UDP client-server communication is also initiated by a client application.
- The UDP client process dynamically selects a port number and uses this as the source port.
- The destination port is usually the well-known or registered port number assigned to the server process.
- The same source-destination pair of ports is used in the header of all datagrams used in the transaction.
- Data returning to the client from the server uses a flipped source and destination port numbers in the datagram header.

Applications that Use TCP

- TCP handles all transport layer related tasks.
- This frees the application from having to manage any of these tasks.
- Applications can simply send the data stream to the transport layer and use the services of TCP.

Applications that Use UDP

- Live video and multimedia applications - Can tolerate some data loss, but require little or no delay. Examples include VoIP and live streaming video.
- Simple request and reply applications - Applications with simple transactions where a host sends a request and may or may not receive a reply. Examples include DNS and DHCP.
- Applications that handle reliability themselves – Unidirectional communications where flow control, error detection, acknowledgements, and error recovery is not required or can be handled by the application.
  Examples include SNMP and TFTP.