http://ftmk.utem.edu.my

/ myftmk

Chapter 4

by

Nazrulazhar Bahaman

nazrulazhar@utem.edu.my

THE INTERNET TRANSPORT PROTOCOL

TCP & UDP

BITS 2343 | Computer Network

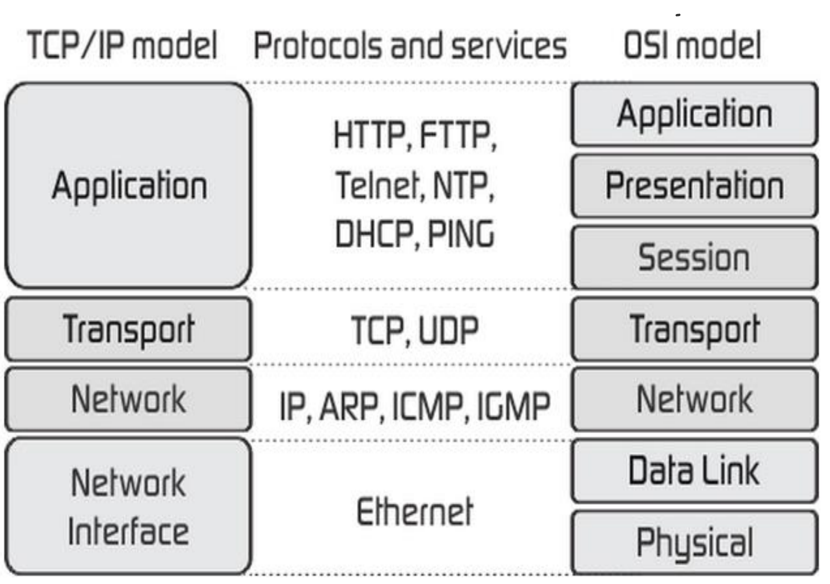
Learning Outcome

- Explain the role of Transport Layer protocols and services in supporting communications across data networks.
- Analyze the application and operation of TCP mechanisms that support reliability, reassembly and manage data loss.
- Analyze the operation of UDP to support communication between two processes on end device.

Introduction

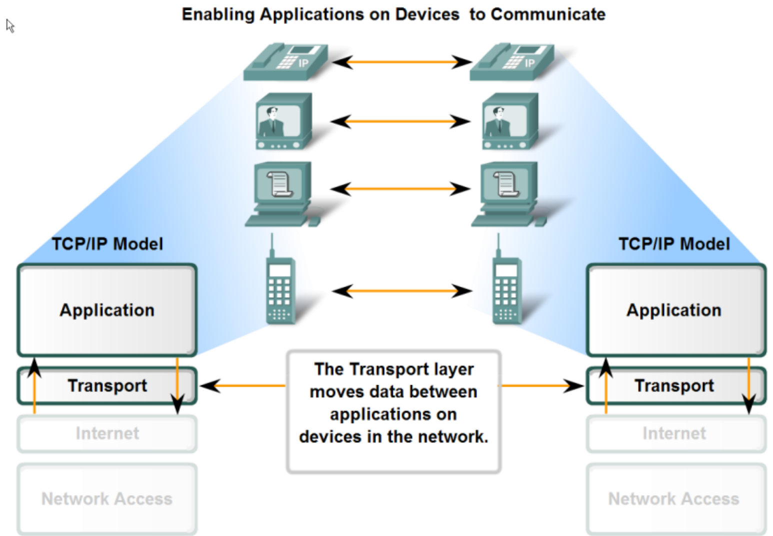
- In OSI Reference Model, Transport Layer is often referred to as **Layer 4**, or **L4**

- numbered layers are not used in TCP/IP Model



Introduction

- The Transport layer is responsible for the overall end-to-end transfer of application data.

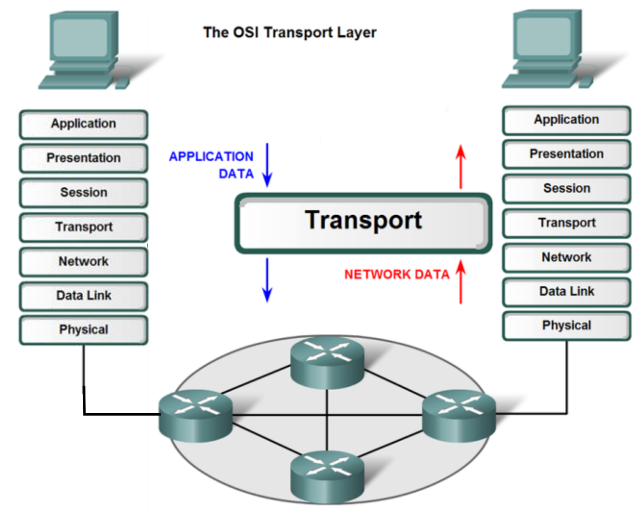


Sender

- The Transport layer accepts data from the application layer →
  - Prepares Application data for transport over the network

Receiver

- The Transport layer accepts data from the Network layer →
  - Processes Network data for use by Application



Introduction

- The transport layer performs 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.

Role of the Transport Layer

- Purpose of the transport Layer
- Supporting reliable communication
- TCP and UDP
- Port addressing
- Segmentation and reassembly: divide conquer

Purpose of the Transport Layer

Primary responsibilities of Transport layer:

1. Tracking the individual communication between applications on the source and destination hosts
2. Splitting data into small pieces and managing each piece
3. Reassembling the pieces into streams of application data
4. Identifying the different applications.

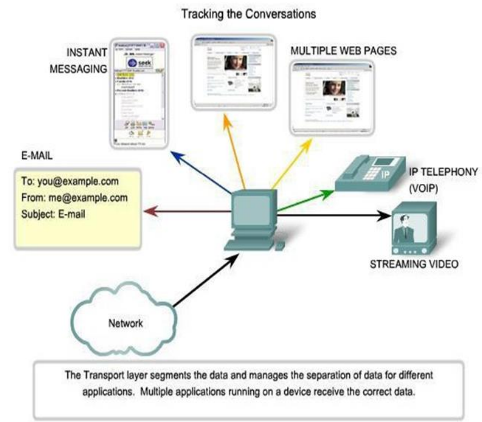
1. Tracking Individual Conversations

- It is the responsibility of the transport layer to maintain and track these multiple conversations
  - Each particular set of data flowing between a source application and a destination application is known as a conversation.

→

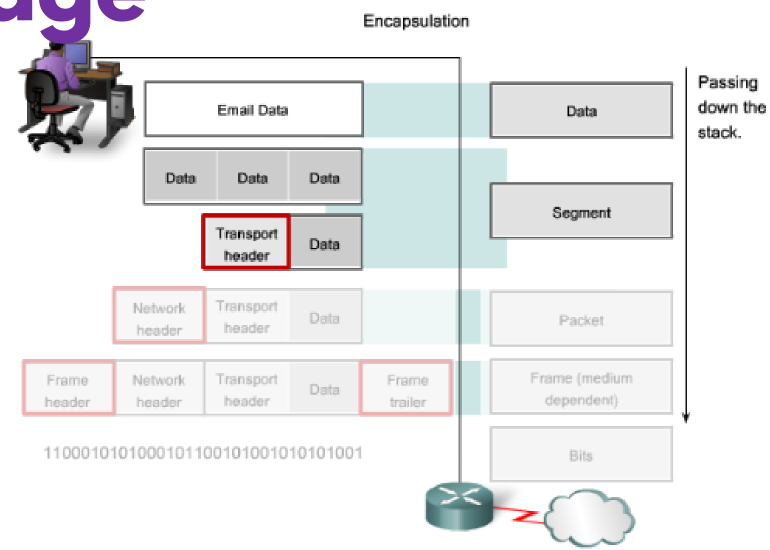
- A host may have multiple applications that are communicating across the network simultaneously

- Each of these applications communicates with one or more applications on one or more remote hosts

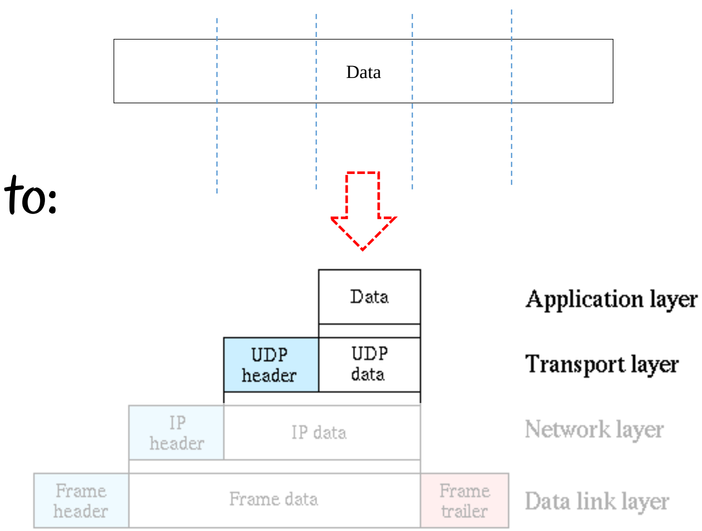


2. Split Data and Manage

- The Application layer passes large amount of data to the transport layer.
- The transport layer will split the data into smaller pieces for transmission.
  - These smaller pieces of data is called Segment.



- A header is then added to each piece of data.
  - This process is called Encapsulation.
- Among others, this header contains information to:
  - Identify the segment belongs to which application.
  - Identify the location of the segment in the full data.

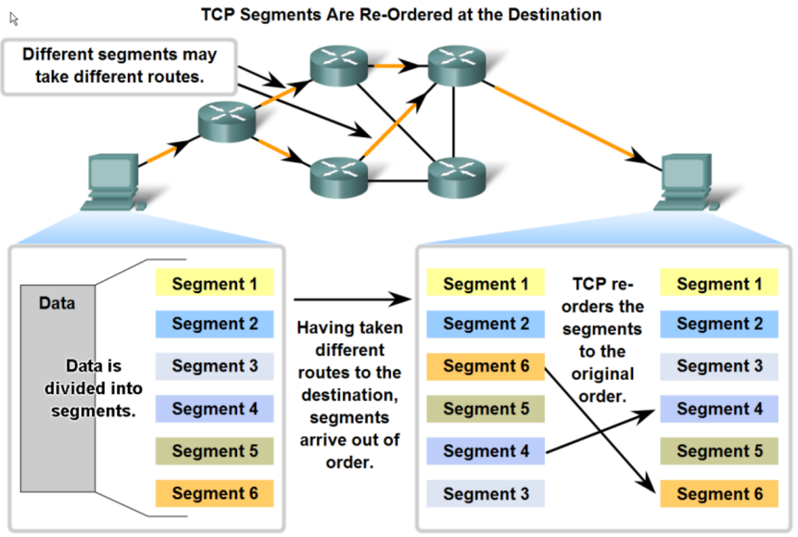


3. Reassembling Segments

- Segments are re-ordered before reassemble at the destination

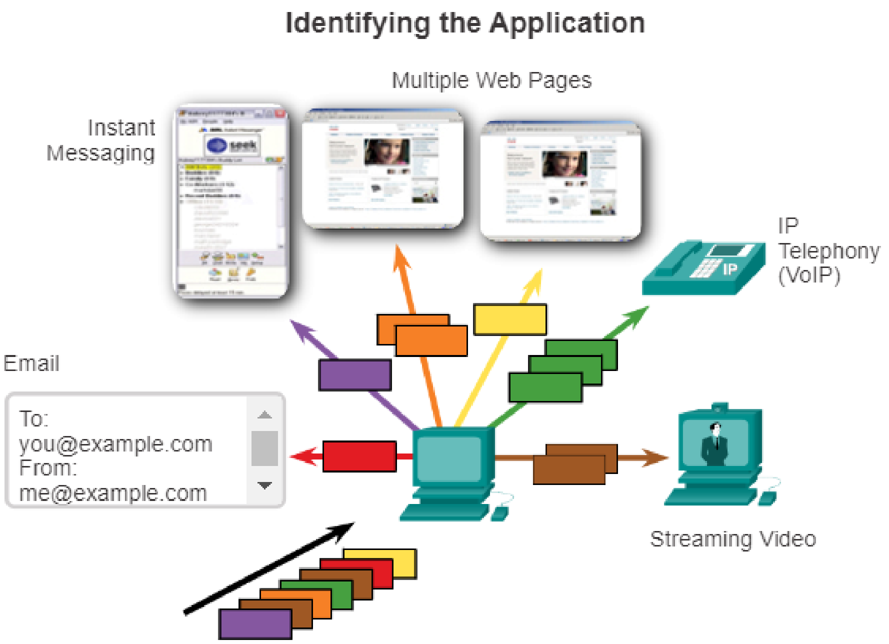
- Network normally provides multiple routes to go to the same destination.
  - Packets belonging to the same data may travel through different routes.
  - Different route may have different latency (delay).
  - Packets can arrive at the destination in the wrong order.

- The header in the segments contain sequence number to enable the receiver to reassemble the received data in the right order.
  - Only data that is fully reassembled will be passed to the receiving application.



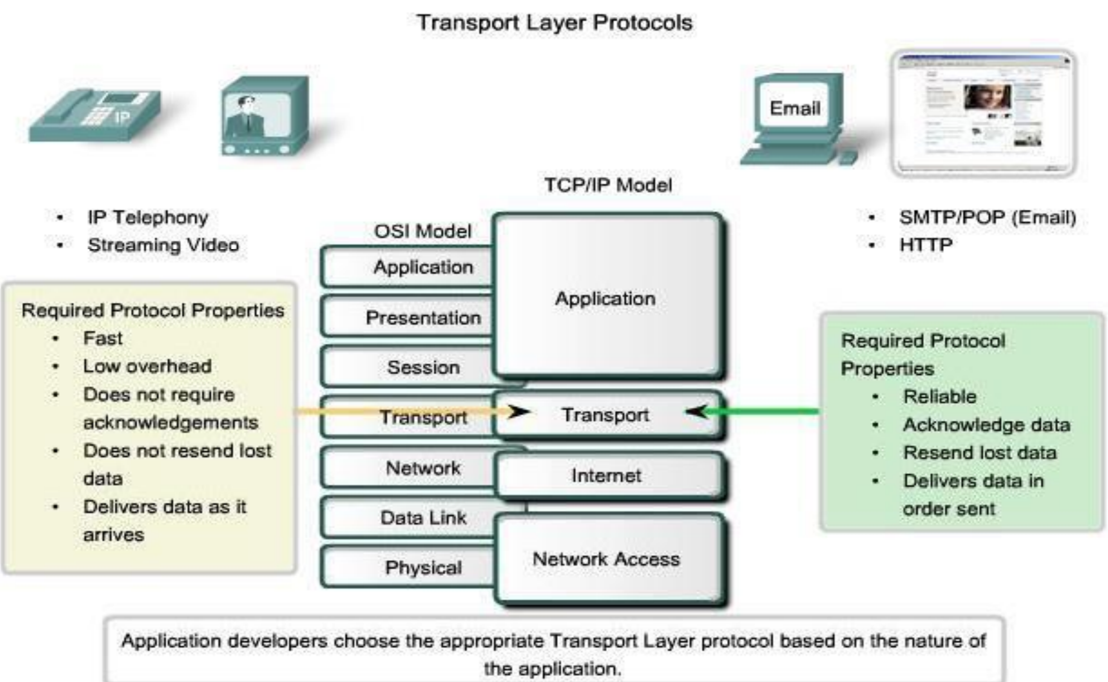
4. Identifying Applications

- Transport layer must identify the target application in order to pass the data streams to the proper application.
- Each application is assigned an identifier called the port number.
  - Port numbers must be unique within a host.
- The sender's and receiver's port number is included in the transport-layer header so that the sending and receiving processes can be identified correctly.



Supporting Various Communication

- Different applications have different requirements their data.
- Different applications may be more suitable for different protocols.
- There exists multiple different transport layer protocols.
- A transport protocol can support reliable delivery.
    - Each piece of data sent by the source must arrive at the destination.
- However, depending on the application, reliable delivery may or may not be required.
- TCP & UDP



TCP and UDP

- Two most common transport layer protocols of the TCP/IP protocol suite are:
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)
- Both protocols can manage the communication of multiple applications.
  - They can make sure that data are sent to the correct application.
- The differences between the two are the specific functions that each protocol implements.

User Datagram Protocol (UDP)

- UDP is a simple, connectionless protocol, described in RFC 768.
- It has the advantage of providing for low overhead data delivery.
- The pieces of communication in UDP are called datagrams.
- These datagrams are sent as "best effort".
  - Best effort means the sender will send the datagrams and "hopefully" they will arrive safely at the receiver.
  - No guarantee is provided.
- Applications that use UDP include: DNS, Video Streaming, Voice over IP (VoIP).

Transmission Control Protocol (TCP)

- TCP is a connection-oriented protocol, described RFC 793.

- TCP can provide more functions as compared to UDP.
  - Additional functions specified by TCP are the same order delivery, reliable delivery, and flow control.
  - This is achieved by using additional overhead.

- Each TCP segment has 20 bytes of overhead in the header encapsulating the application layer data
  - For comparison, each UDP segment only has 8 bytes of overhead.

- The pieces of communication in TCP are called packet

- Applications that use TCP are: Web browsers, emails, file transfer applications.

TCP and UDP Headers

TCP and UDP Headers

TCP Segment

Bit (0) … Bit (15)	Bit (16) … Bit (31)
Source Port (16)	Destination Port (16)
Sequence Number (32)	
Acknowledgement Number (32)	
Header Length (4) Reserved (6) Code Bits (6)	Window (16)
Checksum (16)	Urgent (16)
Options (0 or 32 if any)	
APPLICATION LAYER DATA (Size varies)	

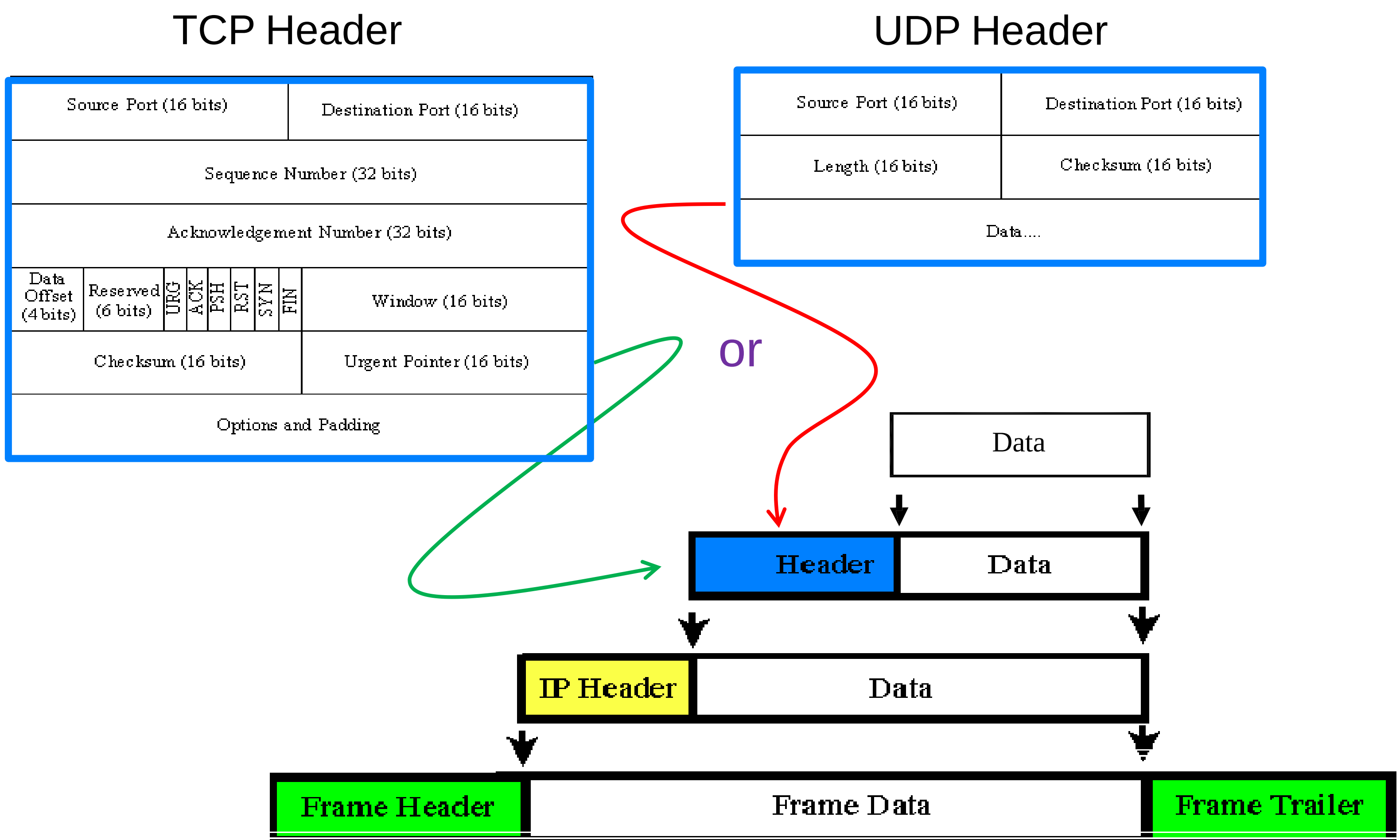
20 Bytes

UDP Datagram

Bit (0) … Bit (15)	Bit (16) … Bit (31)
Source Port (16)	Destination Port (16)
Length (16)	Checksum (16)
APPLICATION LAYER DATA (Size varies)	

8 Bytes

TCP and UDP Headers



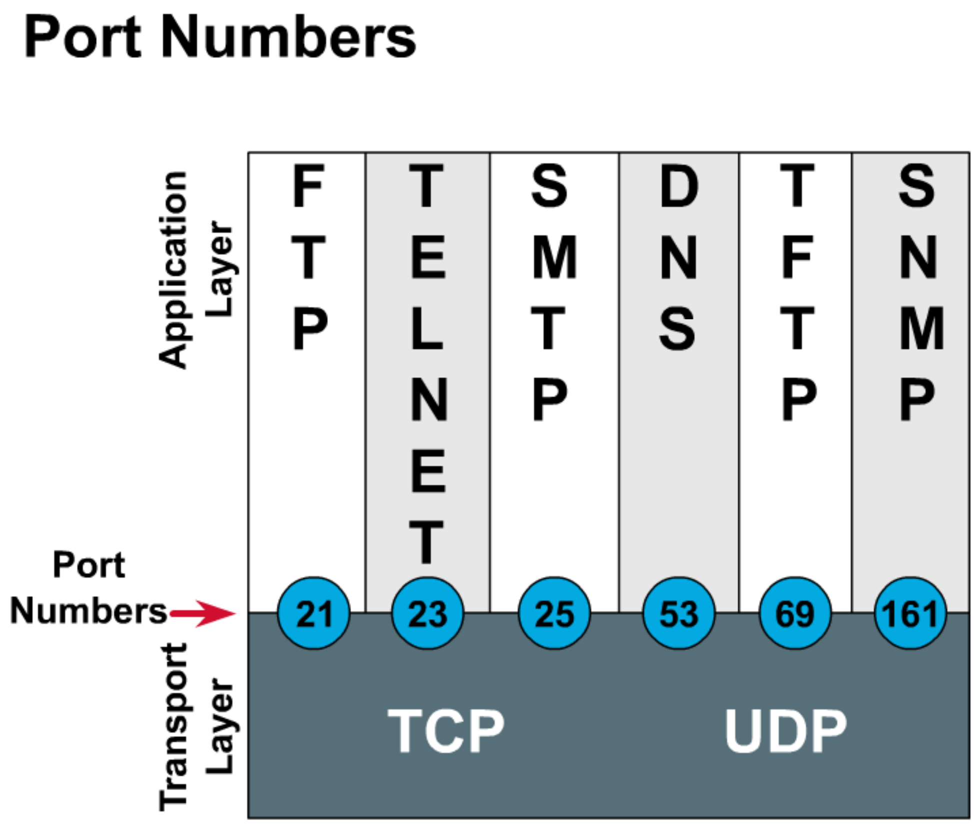
Port Addressing

- Both TCP and UDP headers contain fields for the source and destination port numbers.
  - Source port number identifies the sending application on the local host.
  - Destination port number identifies the receiving application on the remote host.

- Server processes commonly used a static, pre-defined port numbers.

- Client processes use random port numbers assigned by the operating system.

Port Addressing



Port Addressing

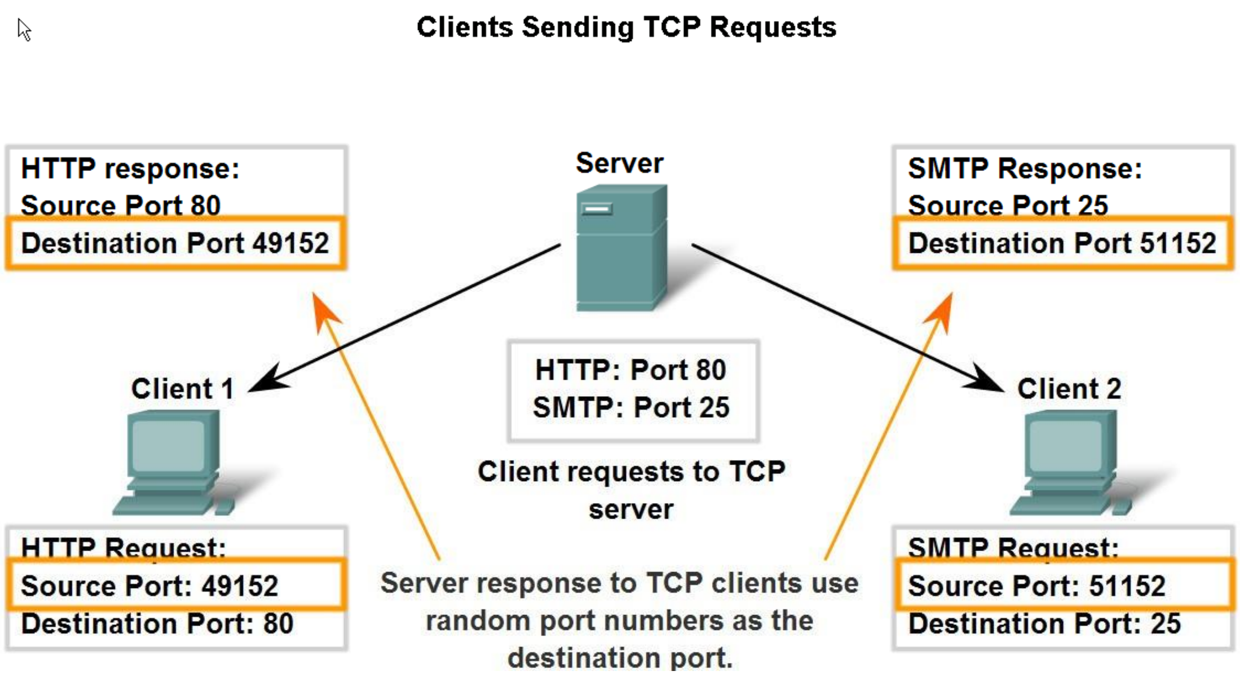
- The receiver's port number allows the data to be sent to the correct receiving application.
- The sender's port number acts like a return address for the requesting application.
  - It enables the receiver to send a reply message to the correct application.
  - In this reply message, the source port number now becomes the destination port number (and vice versa).

Port Addressing

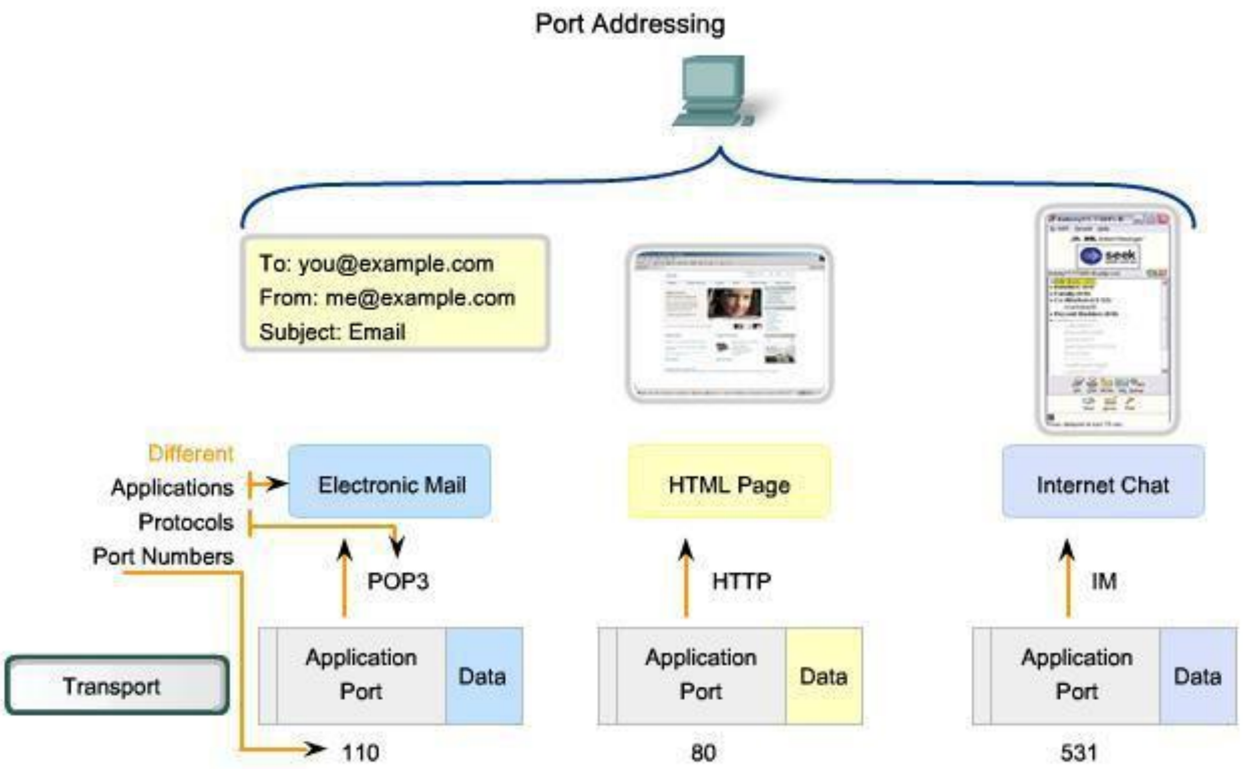
- The combination of the transport layer port number and the network layer IP address uniquely identifies a particular process running on a specific host device.
  - This combination is called a socket.

- Examples:
  - If a Web server (port 80) runs on a host with IP address 192.168.100.20, then the socket for the Web server is 192.168.100.20:80.
  - If a Web browser runs on a host with IP address 192.168.100.48 and the dynamic port number assigned is 49152, then the socket for the Web browser is 192.168.100.48:49152.

Port Addressing



Port Addressing



Data for different applications is directed to the correct application because each application has a unique port number.

Port Addressing

- The standard port numbers used by the servers are specified by the Internet Assigned Numbers Authority (IANA).
  - IANA is a standard body that is responsible for assigning various addressing standards.

- Port numbers can be categorized into three types:
  1. Well-known ports (0 to 1023)
  2. Registered ports (1024 to 49151)
  3. Dynamic or private ports (49152 to 65535)

1. Well-known Ports

- Well-known port numbers are reserved for common Internet services / protocols.
  - HTTP (Web) – port 80
  - SMTP (E-mail) – port 25
  - Telnet – port 23

- By defining the port number to be used by the server application, the client application can be programmed to connect to that specific port number.
  - Makes it easier for the user (they don't need to specify the port number).

1. Well-known Ports

Well-known Port	Application	Protocol
20	FTP Data	TCP
21	FTP Control	TCP
23	Telnet	TCP
25	SMTP	TCP
69	Trivial FTP (TFTP)	UDP
80	HTTP	TCP
110	POP3	TCP
194	IRC	TCP
443	Secure HTTP (HTTPS)	TCP
520	Routing Information Protocol (RIP)	UDP

2. Registered Ports

- Registered port numbers are port numbers that have been registered by certain companies (or organizations) to be used for their applications or protocols.
  - Registration is done with IANA.

- When not used for a server process, port numbers within this range can also be assigned to client applications.

2. Registered Ports

Registered Port	Application	Protocol
1812	RADIUS Authentication Protocol	UDP
1863	MSN Messenger	TCP
2000	CISCO Skinny Client Control Protocol (SCCP, used in VoIP)	UDP
5004	Real-Time Transport Protocol (RTP, a voice and video transport protocol)	UDP
5060	Session Initiation Protocol (SIP, used in VoIP)	UDP
8008	Alternate HTTP	TCP
8080	Alternate HTTP	TCP

3. Dynamic or Private Ports

- Port numbers within this range are dynamically assigned to client applications when they initiate a connection.

- It is not very common for a client to connect to a service using a dynamic or private port number.

- However, in practice, you can program an application and give it any port number.
  - There are non-standard applications (such as P2P file- sharing) that use port numbers within this range for them to connect with each other.

TCP and UDP Ports

- TCP and UDP port numbers are two different ports.
  - Example: Port number 80 for TCP and 80 for UDP are two different ports.

- Most applications use either TCP or UDP only.

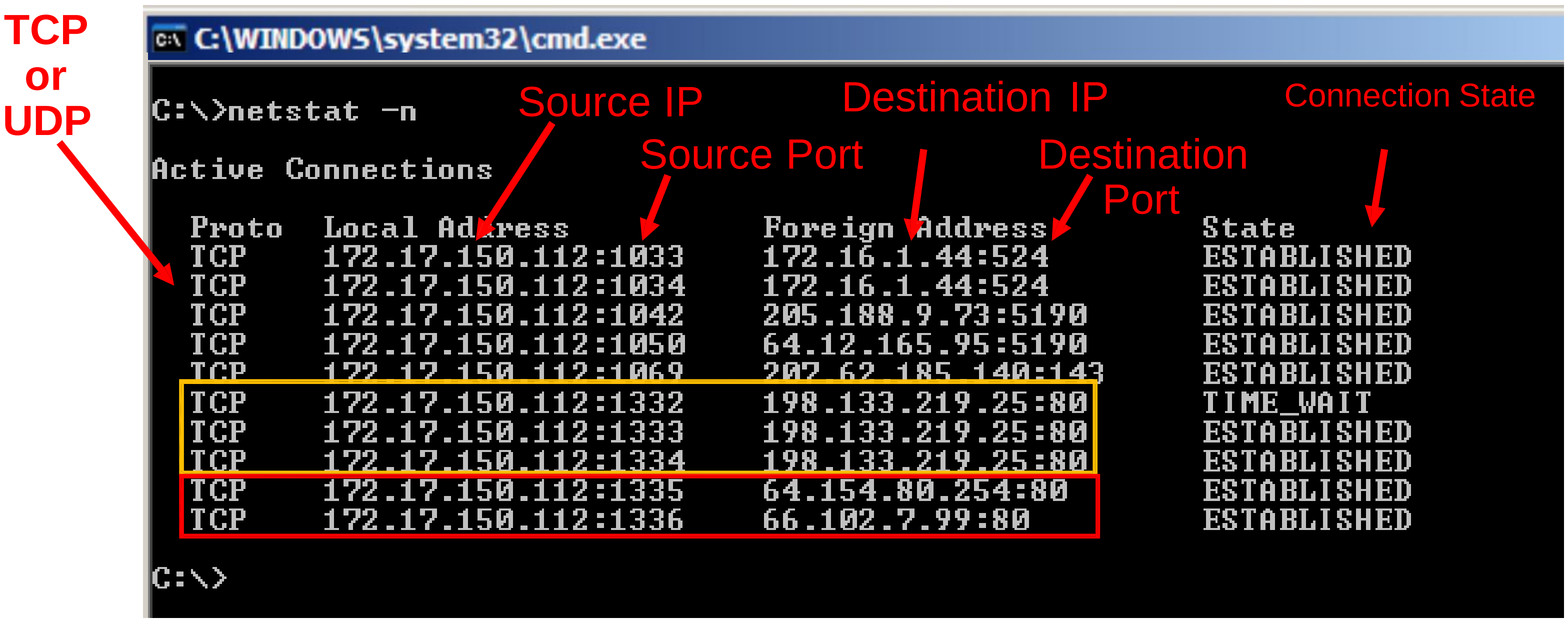
- However, there are also applications that use both TCP and UDP.
- An example would be DNS.
  - UDP is used to serve multiple client requests very quickly.
  - But sometimes, sending the requested information may require the reliability of TCP.
  - Therefore, port number 53 for both TCP and UDP are assigned to DNS.

TCP and UDP Ports

Common Port	Application	Protocol
53	DNS	Well-known TCP/UDP common port
161	SNMP	Well-known TCP/UDP common port
531	AOL Instant Messenger, IRC	Well-known TCP/UDP common port
1433	MS SQL	Registered TCP/UDP common port
2948	WAP (MMS)	Registered TCP/UDP common port

The Netstat Command

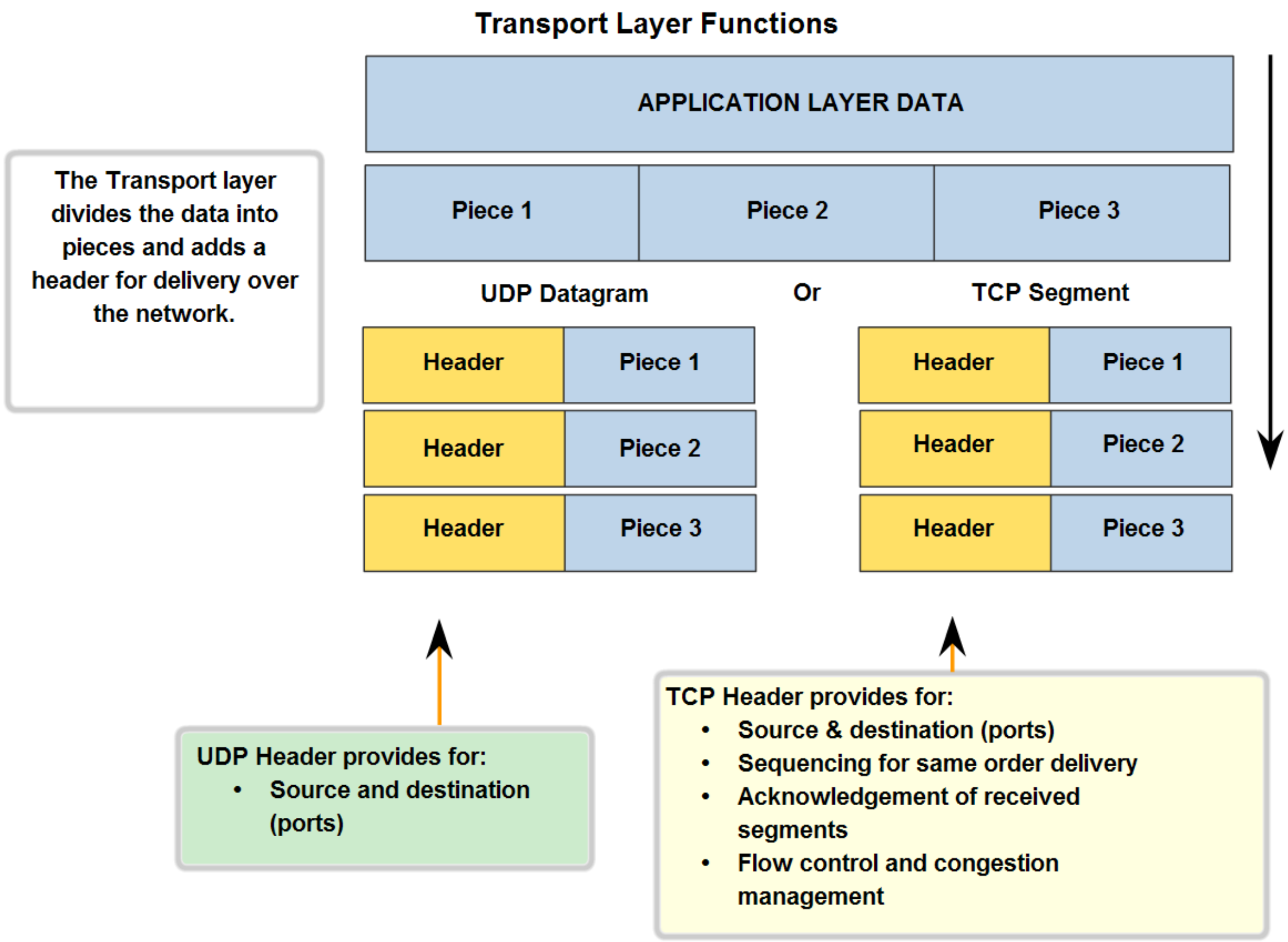
- The list of opened ports on a host can be obtained using the *netstat* command.



Segmentation and Reassembly: Divide and Conquer

- Chapter 2 explains how data is passed down through the various protocols and finally transmitted on the medium.
- At the transport layer, data is segmented into pieces.
  - In TCP, these pieces are called packets.
  - In UDP, these pieces are called datagrams.
- Dividing application into segments ensures that:
  - Data is transmitted within the limits of the media and data.
  - Data from different applications can be multiplexed onto the media.

Segmentation and Reassembly: Divide and Conquer



Segmentation and Reassembly: Divide and Conquer

- At the destination, this process is reversed until the data is passed up to the application.
- A UDP header contains:
  - Source and destination port numbers
- A TCP header contains:
  - Source and destination port numbers
  - Sequence number
  - Acknowledgements number
  - Information required for flow / congestion control

Segmentation and Reassembly: Divide and Conquer

- Sequence numbers in TCP header allows the transport layer functions on the destination host to reassemble segments in the order in which they were transmitted.
- UDP, on the other hand, is not concerned with the order in which the information was transmitted.
  - UDP also does not perform many other functions that TCP performs (i.e. reliability, flow / congestion control).
- As a result, UDP has a simpler design and generates less overhead than TCP.
  - Resulting in a faster data transfer.

TCP: Communicating with Reliability

• Making conversation reliable • TCP three-way handshake • TCP session termination • TCP acknowledgement with windowing • TCP retransmission

TCP: Communicating with Reliability

- TCP is a connection-oriented protocol.
  - A TCP connection needs to be established before data transfer.
  - This ensures that each host is aware and prepared for the data transfer.

- This connection establishment enables TCP to implement reliability mechanism.

- The main idea behind reliability is to have the receiver to acknowledge each data received from the sender.

TCP: Communicating with Reliability

- If the sender does not receive an acknowledgement within a certain period of time, it will retransmit the data.

- In order to implement all these, TCP requires more overhead as compared to UDP.
  - More bandwidth is required to transmit acknowledgement and retransmission.
  - There is some delay before data transfer due to the need to establish TCP connection at the beginning.
  - More system resources is required to keep track of the acknowledgement and retransmission process.

TCP Three-way Handshake

- When two hosts communicate using TCP, TCP connection needs to be established before data can be exchanged.
- After the data transfer is completed, the connection is terminated.
- The process of establishing the TCP connection is commonly referred to as the three-way handshake.
  - Three special segments need to be exchanged for the TCP connection to be established.

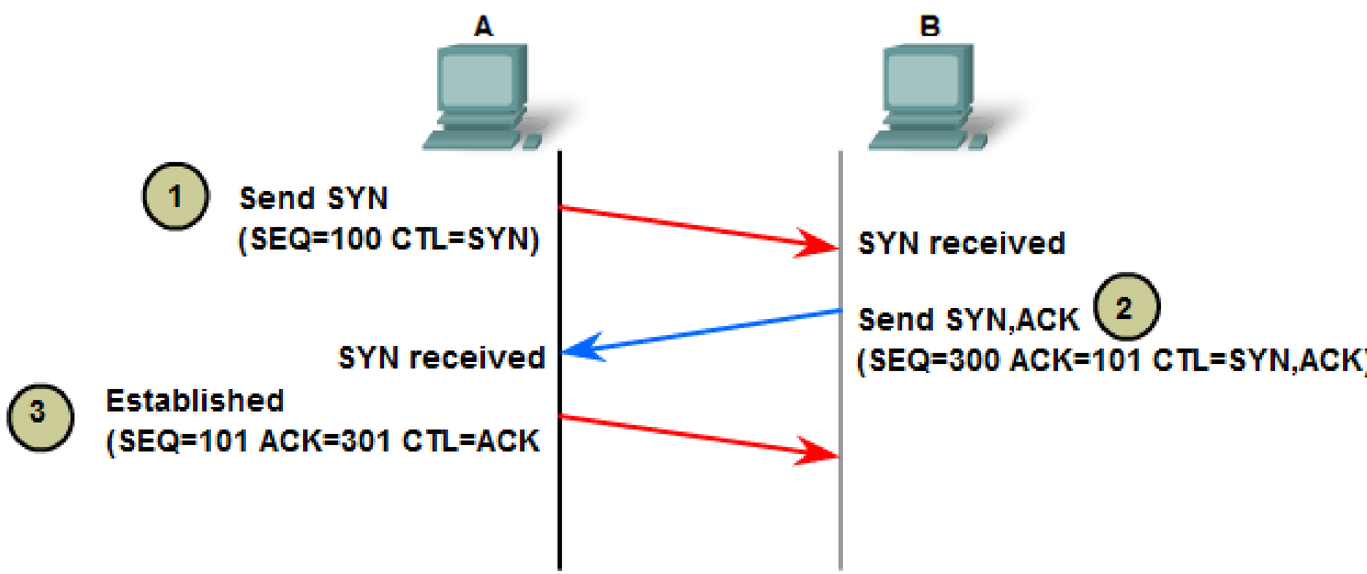
TCP Three-way Handshake

- The three-way handshake is done for the following purposes:

1. Establishes that the destination device is present on the network.
2. 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 for the session.
3. Informs the destination device that the source client intends to establish a communication session on that port number.

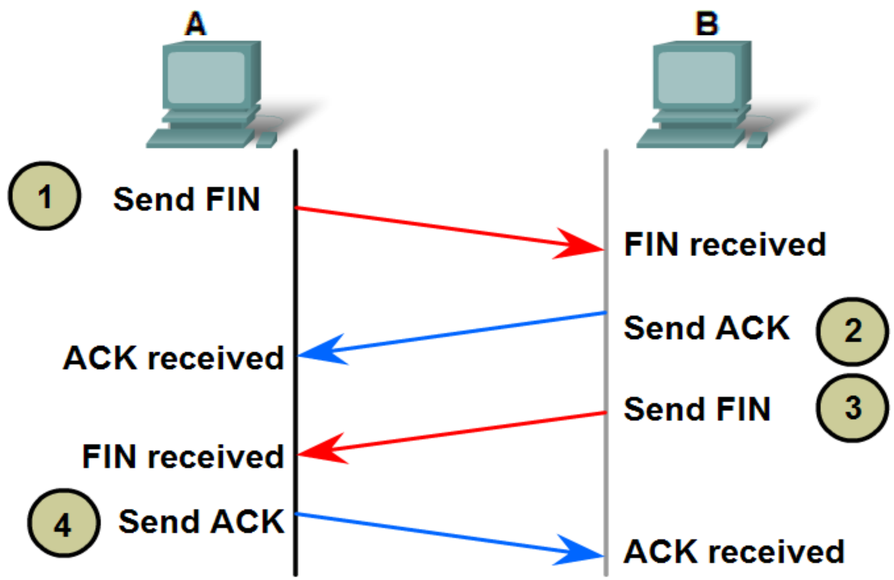
TCP Three-way Handshake

- The TCP connection is initiated by the client.
- There are three steps in TCP connection establishment:



TCP Session Termination

- TCP connection is terminated when there is no more data to send.



TCP Acknowledgement with Windowing

- TCP segments are numbered using the byte number.
  - The byte number indicates the relative number of bytes that have been transmitted in this session.
  - This number is put in the "sequence number" field of the TCP header.

- Example:
  - Say that the first segment is given a sequence number of 1, and it contains 10 bytes of data.
  - Then the second segment will be given sequence number 11.

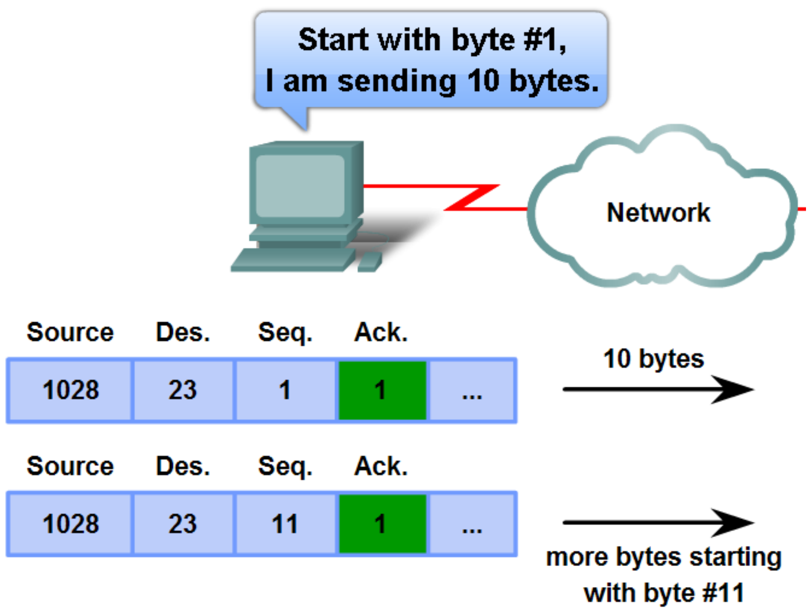
TCP Acknowledgement with Windowing

Acknowledgement of TCP Segments

Source Port	Destination Port	Sequence Number	Acknowledgement Numbers	...

Example:

- Say that the first segment is given a sequence number of 1, and it contains 10 bytes of data.
- Then the second segment will be given sequence number 11.



TCP Acknowledgement with Windowing

- Each segment sent must be acknowledged by the receiver.
  - This is done by inserting the right value in the "acknowledgement number" field in the TCP header.
  - This value must correspond to the expected sequence number to be received for the next segment.

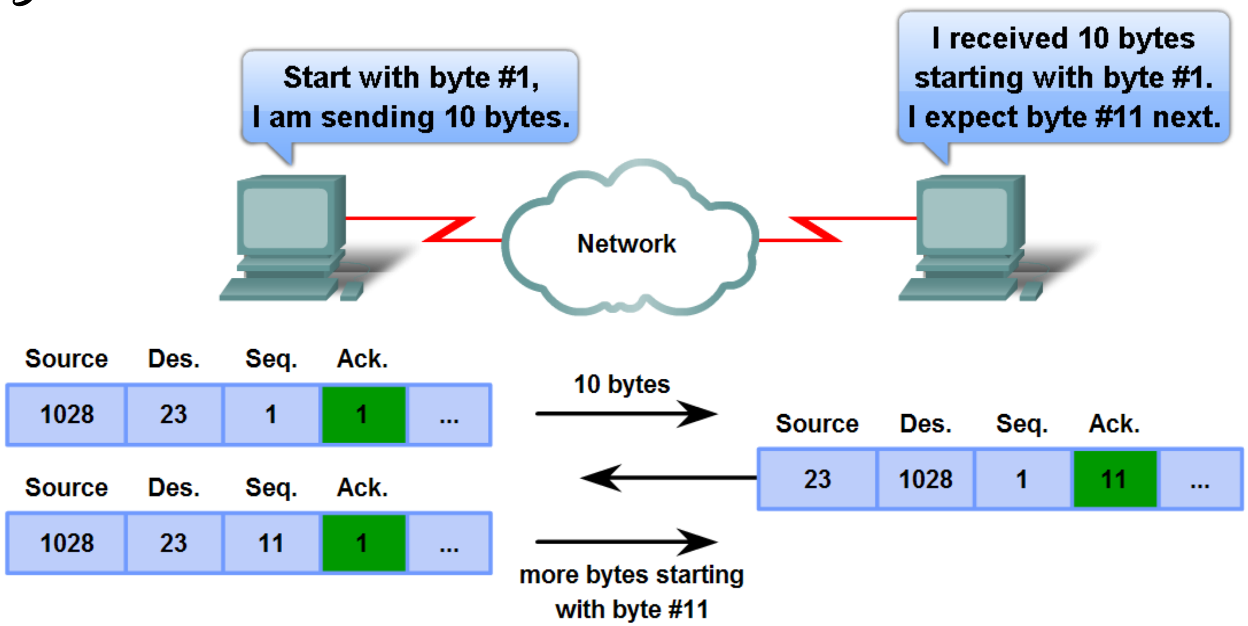
- Example:
  - If A sends a segment with sequence number of 1 and 10 bytes of data to B, then B should expect the next segment from A to have sequence number of 11.
  - Therefore, B acknowledge the first segment by sending a segment to A with acknowledgement number of 11.

TCP Acknowledgement with Windowing

Acknowledgement of TCP Segments

Source Port	Destination Port	Sequence Number	Acknowledgement Numbers	...

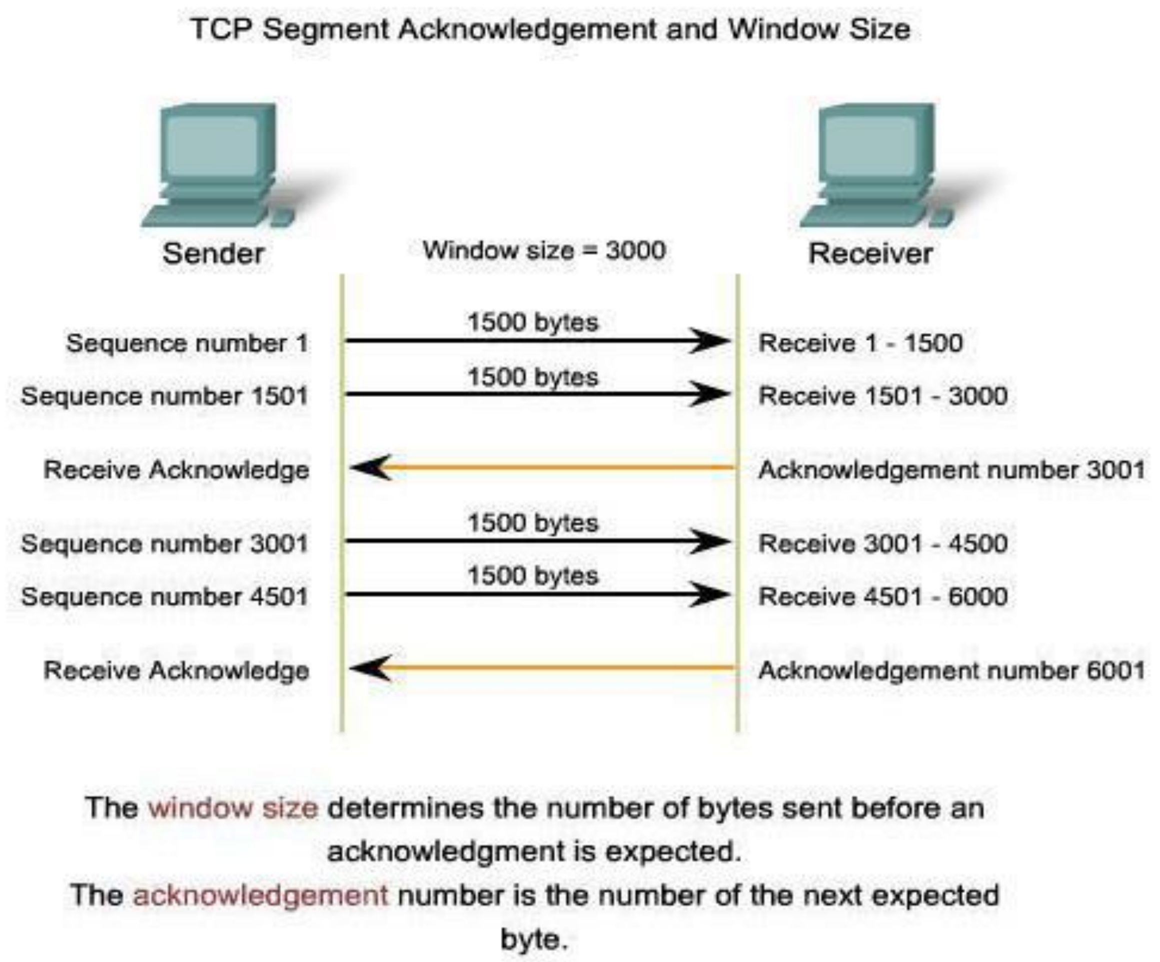
- Example:
  - If A sends a segment with sequence number of 1 and 10 bytes of data to B, then B should expect the next segment from A to have sequence number of 11.
  - Therefore, B acknowledge the first segment by sending a segment to A with acknowledgement number of 11.



TCP Acknowledgement with Windowing

- To reduce acknowledgement overhead, TCP allows multiple segments to be acknowledged with a single segment in the opposite direction.
- Example:
  - Say that A sends two segments to B:
    - Segment 1: Sequence number = 1, data = 1500 bytes.
    - Segment 2: Sequence number = 1501, data = 1500 bytes.
  - To acknowledge both segments, B can send a single segment to A.
  - In this segment the value for the acknowledgement number field would be 3001.

TCP Acknowledgement with Windowing



TCP Retransmission

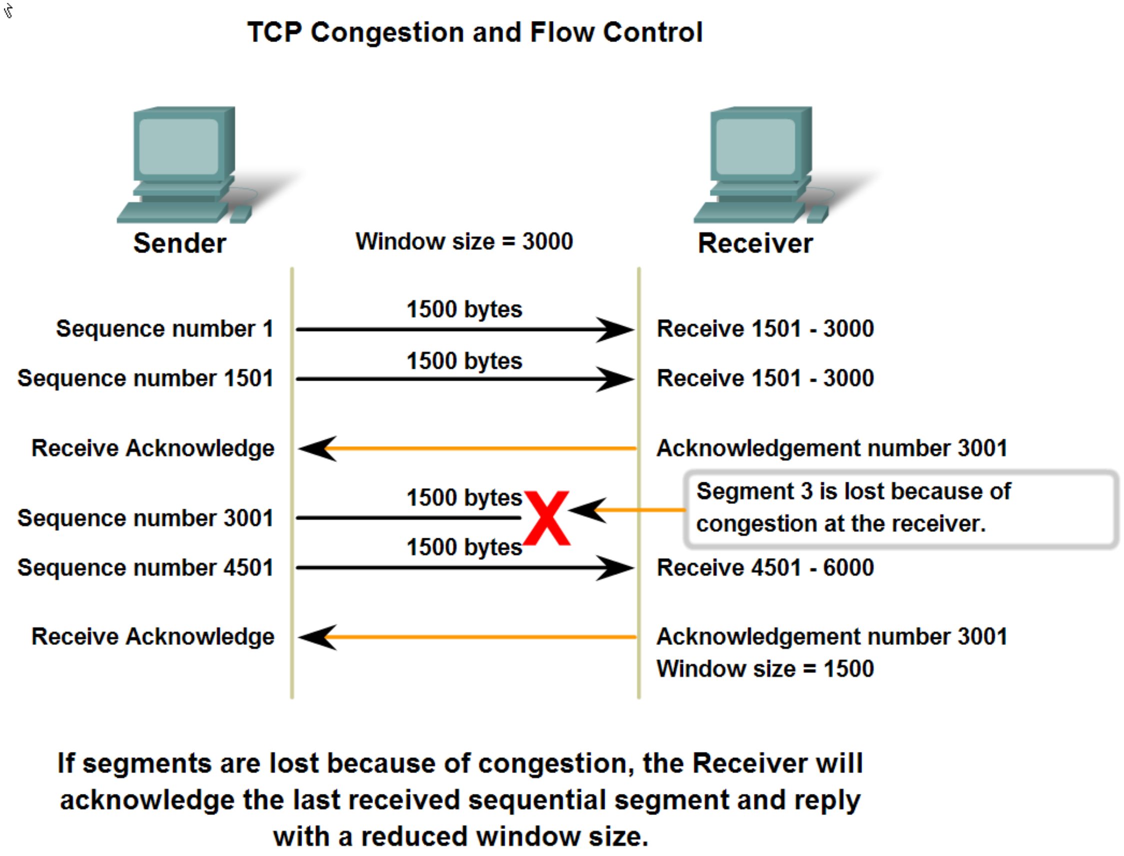
- No matter how well designed a network is, data loss will occasionally occur.
- TCP recovers from data losses using a retransmission mechanism.
  - When a TCP segment is sent, a timer is set.
  - The sender would then wait for the segment to be acknowledged.
  - If acknowledgement is not received before the timer expires, the segment is retransmitted.

TCP Retransmission

- A TCP sender can transmit multiple segments before waiting for acknowledgement.
- TCP receiver only accept segments that are received in order.
  - If the sender transmits three segments, and only segment 1 and 3 are received, then segment 3 is considered to be received out of order.
  - In this case, only segment 1 will be acknowledged.
  - When the timer for segment 2 expires, the sender will retransmit segment 2.
  - Depending on whether the receiver decides to keep or discard segment 3 (which was received out of order), segment 3 may or may not be retransmitted again.

TCP Retransmission

•



TCP Flow Control

- Flow control is the mechanism to prevent the sender from sending more data than the receiver can handle.
  - Each host has a TCP receive buffer.
  - Data that has been received will be temporarily put in this buffer while waiting for the application to read the data.
  - Flow control is used to make sure that the sender does not receive buffer.
    overflow the receiver's
- Flow control is implemented using the "window size" field in the TCP header.

TCP Flow Control

- Say that Host A sends a segment to Host B.
  - The value that Host A puts in the window size field indicates the maximum amount of data (in bytes) that Host A can send to Host B.
  - This value corresponds to the size of the empty space in A's receive buffer.

- The size of the receive buffer is determined during the TCP connection establishment.
  - The maximum window size would be equal to the size of the TCP receive buffer.
- Window size is included in every TCP segment sent from client or server.

TCP Congestion Control

- Congestion control is the mechanism to slow down the data rate when the level of congestion in the network is high.
  - Congestion happens when there are too many packets in the network.
  - During congestion, packets will move very slowly across the network (similar to traffic jam).

- One way to relieve the network of congestion is to get all the TCP senders to slow down their data transfer.
  - This would reduce the number of packets in the network.

TCP Congestion Control

- Congestion control is implemented using a TCP variable called "congestion window".

- Similar to receive window, the congestion window puts a limit on how much data a TCP sender can send into the network.

- The congestion window is initially set to be low.
  - It will slowly increase if the network seems to be "okay" (not congested).
  - But if there is any indication that the network is congested (i.e. packet loss), the congestion window will be decreased.

TCP Congestion Control

- The dynamic increasing and decreasing of congestion window is a continuous process in TCP.
  - Most of the time, the congestion window will maintain within a certain range.
  - This represents the average data rate of that particular TCP session.
- In highly efficient networks, congestion window can become very large because there is no data loss.
- In networks where the network infrastructure is not able to cope with the number of packets in the network, the congestion window is likely to remain small.

UDP: Communicating with Low Overhead

- UDP: low overhead vs. reliability
- UDP datagram reassembly

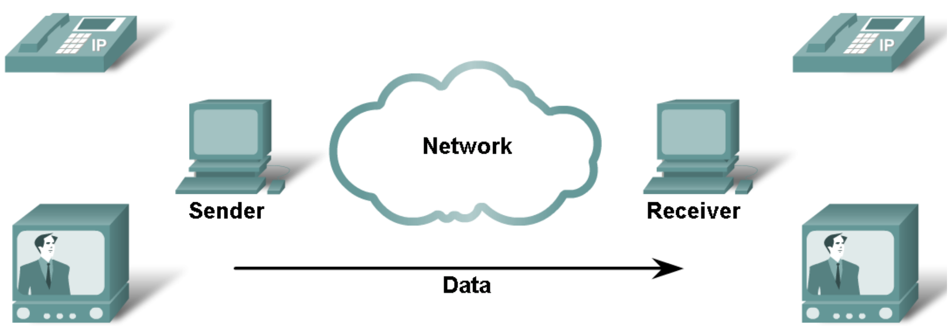
UDP: Communicating with Low Overhead

- UDP is a simple protocol that provides only the basic transport layer functions.
  - Make sure the data can be sent to the correct receiving application.

- UDP does not provide:
  - Reliable delivery
  - Ordered delivery
  - Flow control
  - Congestion control

- UDP is also connectionless.

UDP Low Overhead Data Transport



UDP does not establish a connection before sending data.

UDP: Communicating with Low Overhead

- Due to its simple functions, UDP has much lower overhead as compared to TCP.
  - The header size is smaller.
  - No connection needs to be established.
  - Use less system resource (i.e. memory).
  - No complicated retransmission, sequencing flow control and mechanisms.
- As such, UDP is commonly used for applications where low overhead is required.

UDP: Low Overhead vs. Reliability

- UDP has the advantage of having low overhead, but has the disadvantage of not being reliable.
- Therefore, UDP is suitable for applications that:
  - Can tolerate some data loss.
    - Examples: VoIP, online games.
  - Can simply send another message if the previous one is not received.
    - Examples: DNS, RIP (routing protocol).
  - Can be greatly affected if data transfer is delayed by TCP's reliability or congestion control mechanisms.
    - Examples: VoIP, online games.

UDP: Low Overhead vs. Reliability

- Among application-layer protocols that use UDP:
  - Domain Name System (DNS)
  - Simple Network Management Protocol (SNMP)
  - Dynamic Host Configuration Protocol (DHCP)
  - Routing Information Protocol (RIP)
  - Trivial File Transfer Protocol (TFTP)
- Applications that require reliability can still use UDP.
  - However, the reliability mechanism must be implemented in the application itself.
  - With this, the application can have both low overhead and reliability at the same time.

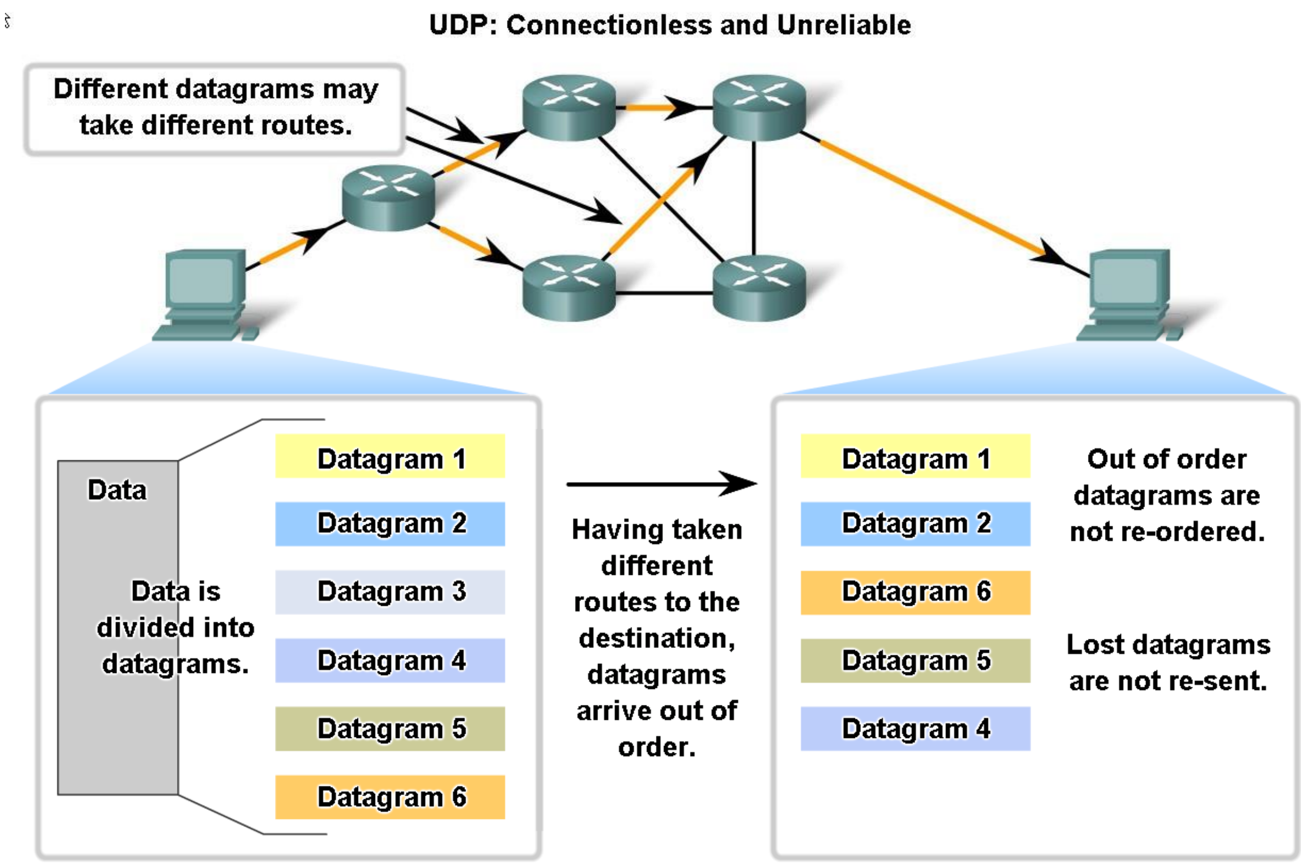
UDP Datagram Reassembly

- Most applications that use UDP send only a small amount of data that can fit into a single datagram.
- However, if the application sends a large amount of data, the data must be split into multiple pieces.
- When multiple datagrams are sent to a destination, they can take different paths and arrive in the wrong order.

UDP Datagram Reassembly

- UDP cannot keep track of datagram order like TCP.
  - There is no field in the header for sequence number.
  - Therefore, there is no way to reorder the datagrams into the order in which they were transmitted.
- UDP just reassemble the data in the order in which it was received and forward it to the application.
  - If the sequence of data is important to the application, the application itself must implement the mechanism to identify the proper sequence of the data and determine how it should be processed.

UDP Datagram Reassembly



KEMENTERIAN PENDIDIKAN MALAYSIA

UTeM

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

FTMK

FAKULTI TEKNOLOGI MAKLUMAT DAN KOMUNIKASI

/ myftmk

http://ftmk.utem.edu.my

Outline

- Roles of the transport layer
  - Purpose of the transport Layer
  - Supporting reliable communication
  - TCP and UDP
  - Port addressing
  - Segmentation and reassembly: divide conquer and
- TCP: Communicating with reliability
  - Making conversation reliable

Outline

- TCP three-way handshake
- TCP session termination
- TCP acknowledgement with windowing
- TCP retransmission
- TCP flow control
- TCP congestion control

- UDP: Communicating with low overhead
  - UDP: low overhead vs. reliability
  - UDP datagram reassembly