Ch 9: Transport Layer

Computer Networks Course

BY

Dr. Essam Halim Houssein

Cisco | Networking Academy®
Mind Wide Open™

Version 5.0

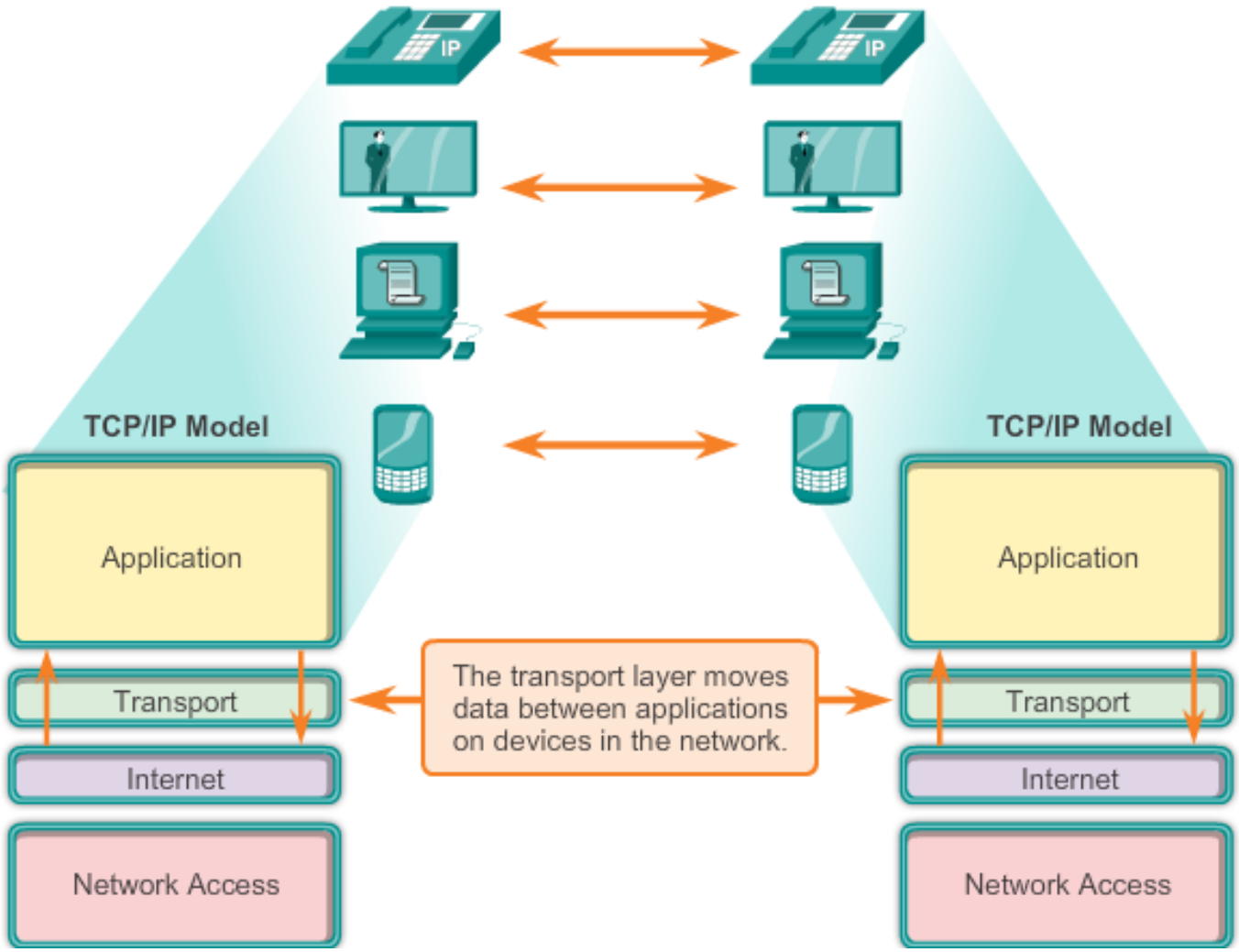
On a single device, people can use multiple applications and services such as e-mail, the web, and instant messaging to send messages or retrieve information. Data from each of these applications is packaged, transported and delivered to the appropriate application on the destination device.

The processes described in the transport layer accept data from the application layer and prepare it for addressing at the network layer. A source computer communicates with a receiving computer **to decide how to break up data into segments, how to make sure none of the segments get lost, and how to verify all the segments arrived.**

Role of the Transport Layer

The transport layer is responsible for **establishing a temporary communication session between two applications and delivering data between them.** An application generates data that is sent from an application on a source host to an application on a destination host. As shown in the figure, the transport layer is the link between the application layer and the lower layers that are responsible for network transmission.

Enabling Applications on Devices to Communicate



Transport Layer Responsibilities

Tracking Individual Conversations

At the transport layer, each 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. It is the responsibility of the transport layer to maintain and track these multiple conversations.

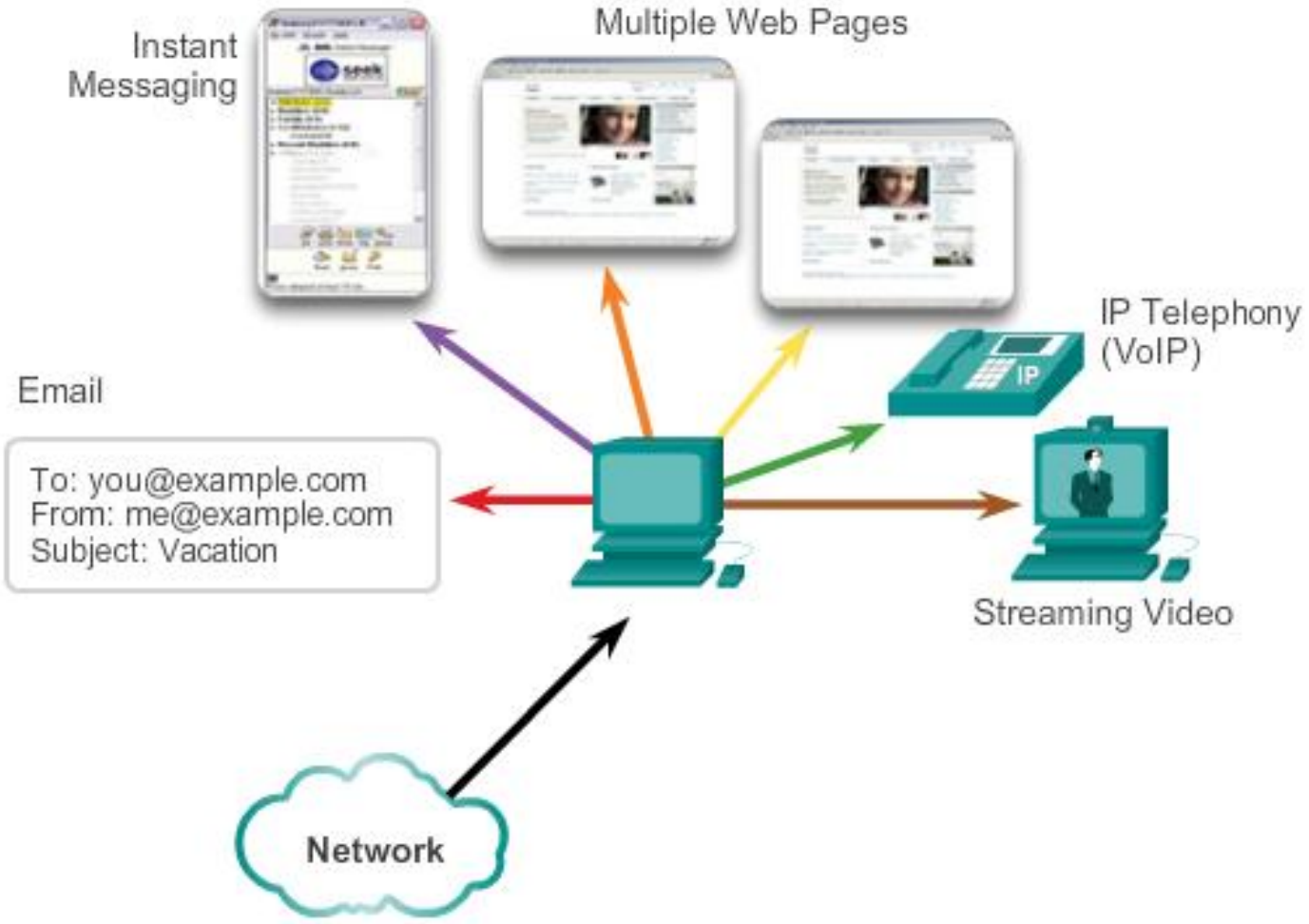
Segmenting Data and Reassembling Segments

Transport layer protocols have services that segment the application data into blocks that are an appropriate size. At the destination, the transport layer must be able to reconstruct the pieces of data into a complete data stream that is useful to the application layer.

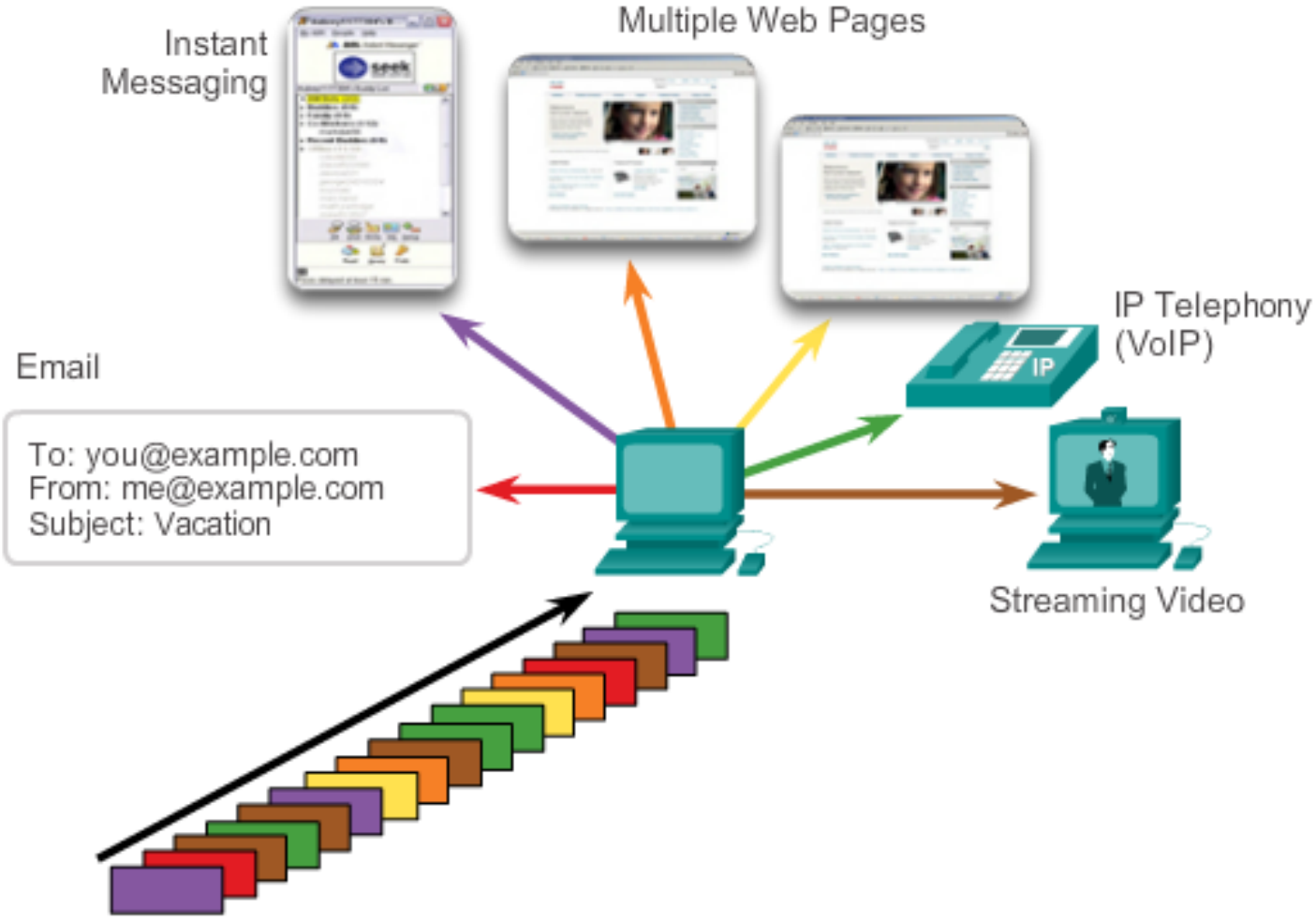
Identifying the Applications

To pass data streams to the proper applications, the transport layer must identify the target application. To accomplish this, the transport layer assigns each application an identifier called a **port number**.

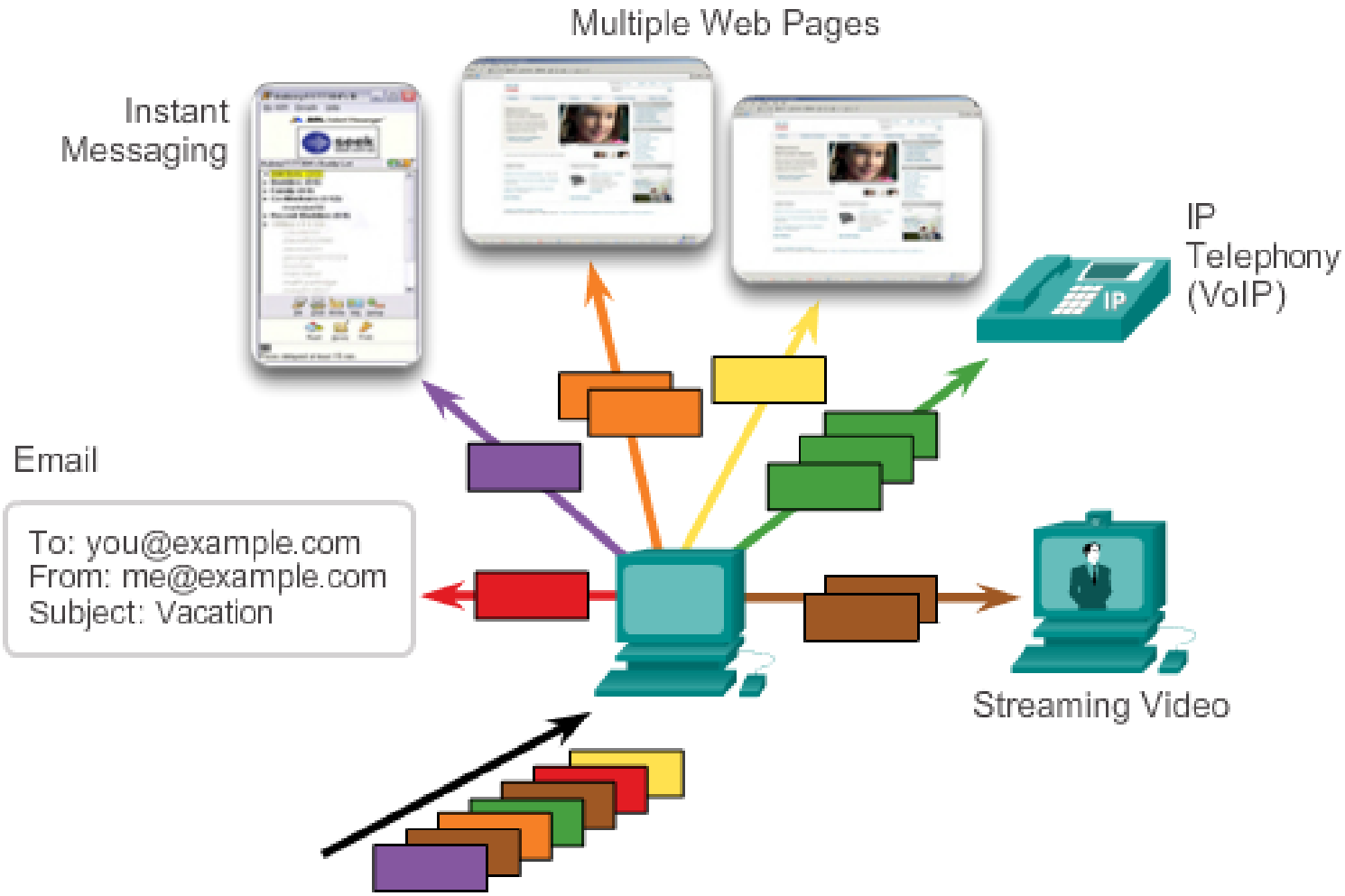
Tracking the Conversations



Segmentation



Identifying the Application



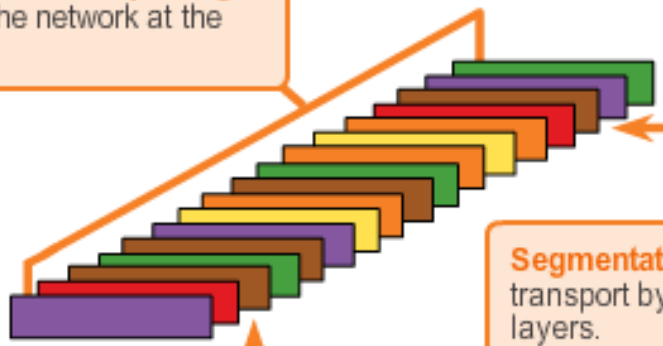
Conversation Multiplexing

Segmenting the data into smaller chunks enables many different communications, from many different users, to be interleaved (multiplexed) on the same network.

To identify each segment of data, the transport layer adds a header containing binary data organized into several fields. It is the values in these fields that enable various transport layer protocols to perform different functions in managing data communication.

Transport Layer Services

Segmentation allows conversation **multiplexing** - multiple applications can use the network at the same time.



Segmentation facilitates data transport by the lower network layers.

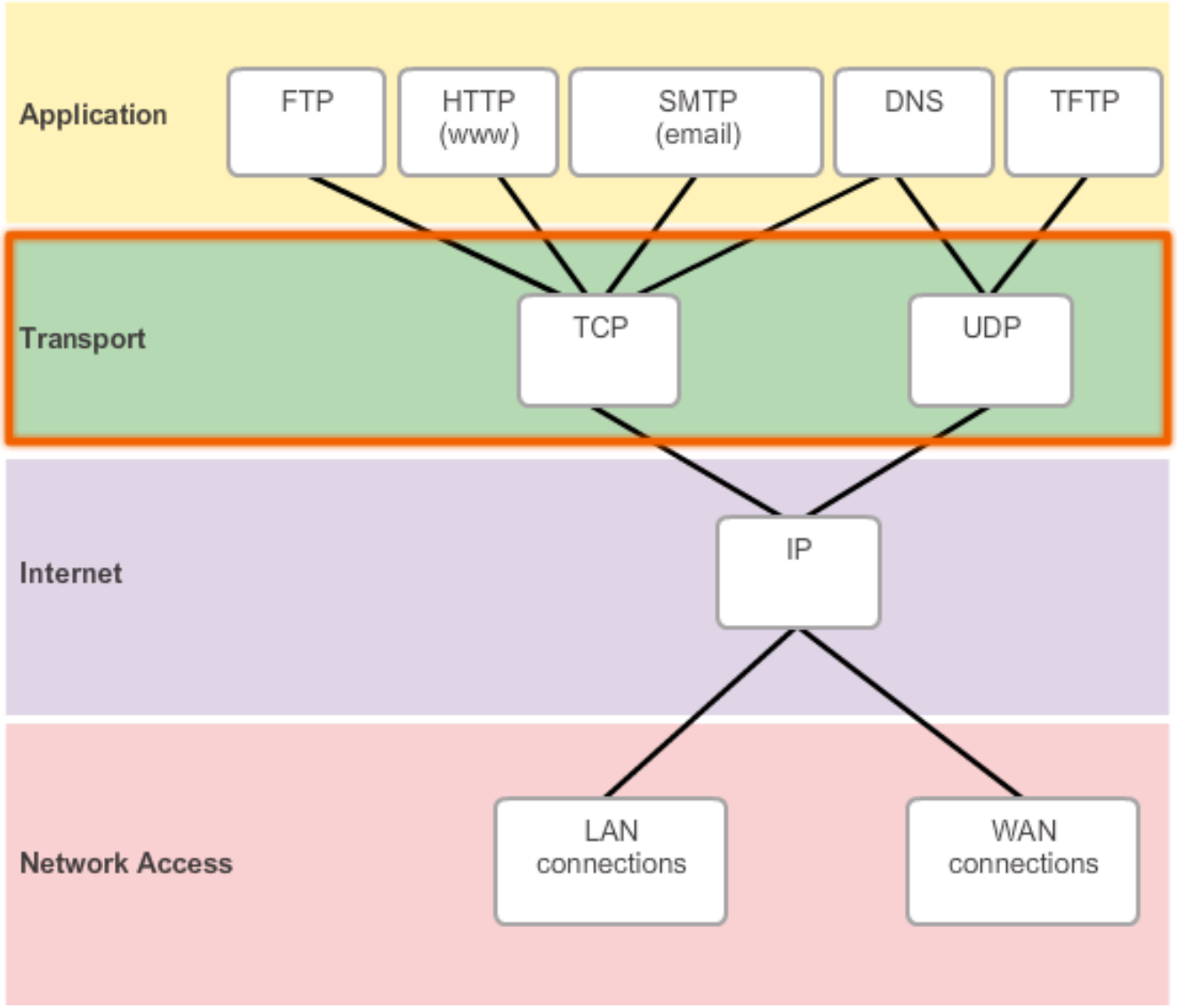
Error checking can be performed on the data in the segment to check if the segment was changed during transmission.

Transport Layer Reliability

The transport layer is also responsible for managing reliability requirements of a conversation.

Transport protocols specify how to transfer messages between hosts. TCP/IP provides two transport layer protocols, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

TCP is considered a reliable, full-featured transport layer protocol, which ensures that all of the data arrives at the destination. In contrast, UDP is a very simple transport layer protocol that does not provide for any reliability.



TCP

With TCP, there are three basic operations of reliability:

1. Numbering and tracking data segments transmitted to a specific host from a specific application
2. Acknowledging received data
3. 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. **UDP is known as a best-effort delivery protocol.** In the context of networking, best-effort delivery is referred to as **unreliable** because there is no acknowledgment that the data is received at the destination. With UDP, there are no transport layer processes that inform the sender of a successful delivery.

The Right Transport Layer Protocol for the Right Application

Application developers must choose which transport protocol type is appropriate based on the requirements of the applications. For example, applications such as databases, web browsers, and email clients, require that all data that is sent arrives at the destination in its original condition. Any missing data could cause a corrupt communication that is either incomplete or unreadable. **These applications are designed to use TCP.**

In other cases, an application can tolerate some data loss during transmission over the network, but delays in transmission are unacceptable. **UDP is the better choice for these applications** because less network overhead is required. UDP is preferable for applications such as streaming live audio, live video, and Voice over IP (VoIP). *Acknowledgments and retransmission would slow down delivery.*

Transport Layer Protocols

Application developers choose the appropriate transport layer protocol based on the nature of the application.

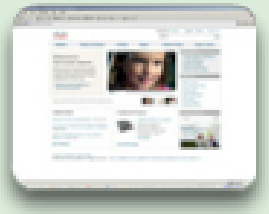
UDP

IP Telephony

Streaming Live Video

TCP

SMTP/POP (Email)



HTTP

Required protocol properties:
- Fast
- Low overhead
- Does not require acknowledgements
- Does not resend lost data
- Delivers data as it arrives

Required protocol properties:
- Reliable
- Acknowledge data
- Resends lost data
- Delivers data in sequenced order

TCP Features

Establishing a Session

TCP is a connection-oriented protocol. A connection-oriented protocol is one that negotiates and establishes a permanent session between source and destination devices prior to forwarding any traffic. Through session establishment, the devices negotiate the amount of traffic that can be forwarded at a given time, and the communication data between the two can be closely managed.

Reliable Delivery

Reliability means ensuring that each segment that the source sends arrives at the destination.

Same-Order Delivery

Numbering and sequencing the segments, TCP can ensure that these segments are reassembled into the proper order.

Flow Control

Request that the sending application reduce the rate of data flow.

For more information on TCP, read the RFC.

TCP Services

Establishing a session ensures the application is ready to receive the data.

Same order delivery ensures that the segments are reassembled into the proper order.

Reliable delivery means lost segments are resent so the data is received complete.

Flow control ensures that the receiver is able to process the data received.

TCP Header

TCP is a stateful protocol. A stateful protocol is a protocol that keeps track of the state of the communication session. To track the state of a session, TCP records which information it has sent and which information has been acknowledged.

TCP segment has 20 bytes of overhead in the header encapsulating the application layer data:

Source Port (16 bits) and Destination Port (16 bits) - Used to identify the application.

Sequence number (32 bits) - Used for data reassembly purposes.

Acknowledgment number (32 bits) - Indicates the data that has been received.

Header length (4 bits) - Known as ”data offset”. Indicates the length of the TCP segment header.

Control bits (6 bits) - Includes bit codes, or flags, which indicate the purpose and function of the TCP segment.

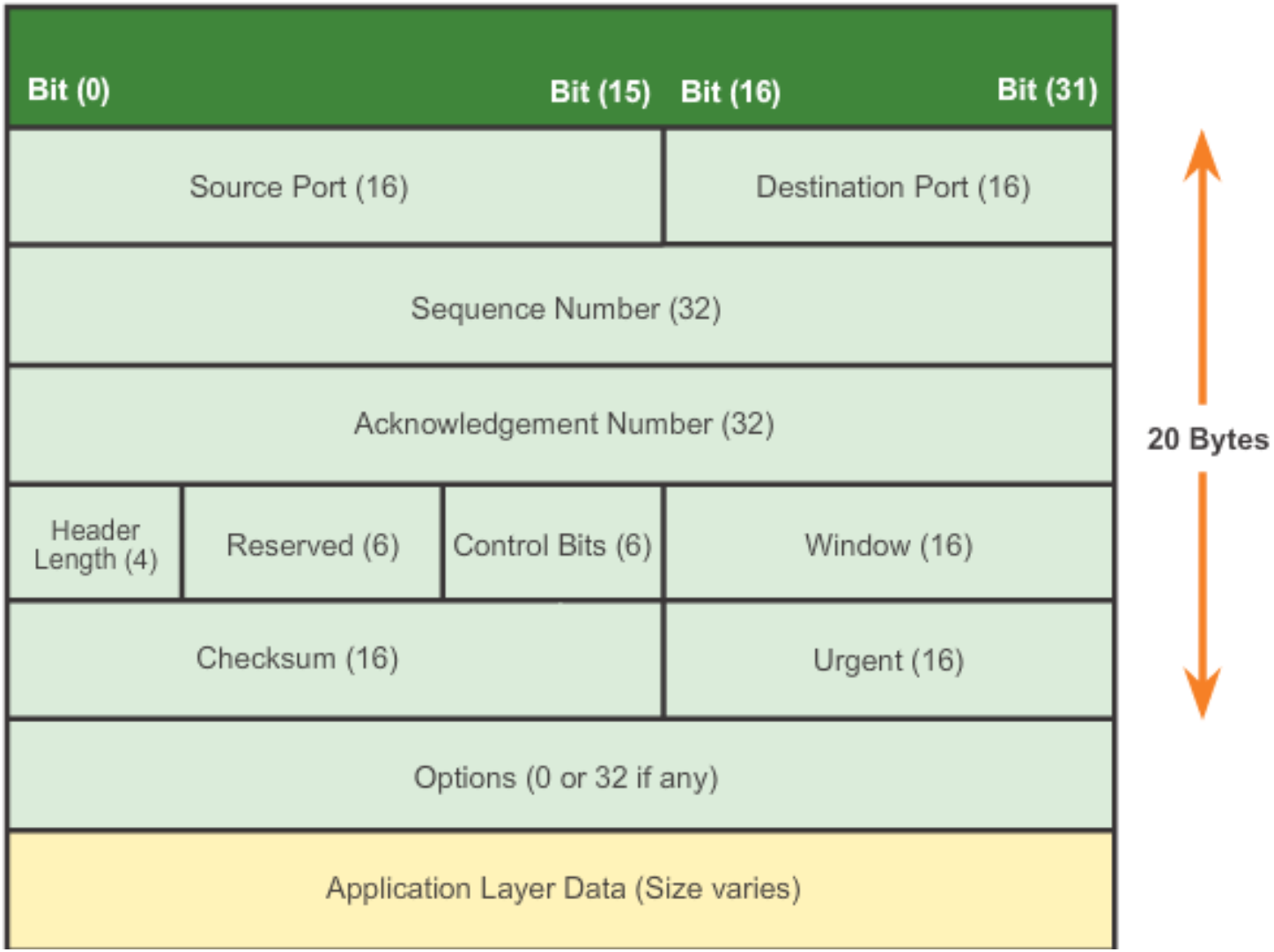
Window size (16 bits) - Indicates the number of bytes that can be accepted at one time.

Checksum (16 bits) - Used for error checking of the segment header and data.

Urgent (16 bits) - Indicates if data is urgent.

Reserved (6 bits) - This field is reserved for the future.

TCP Segment

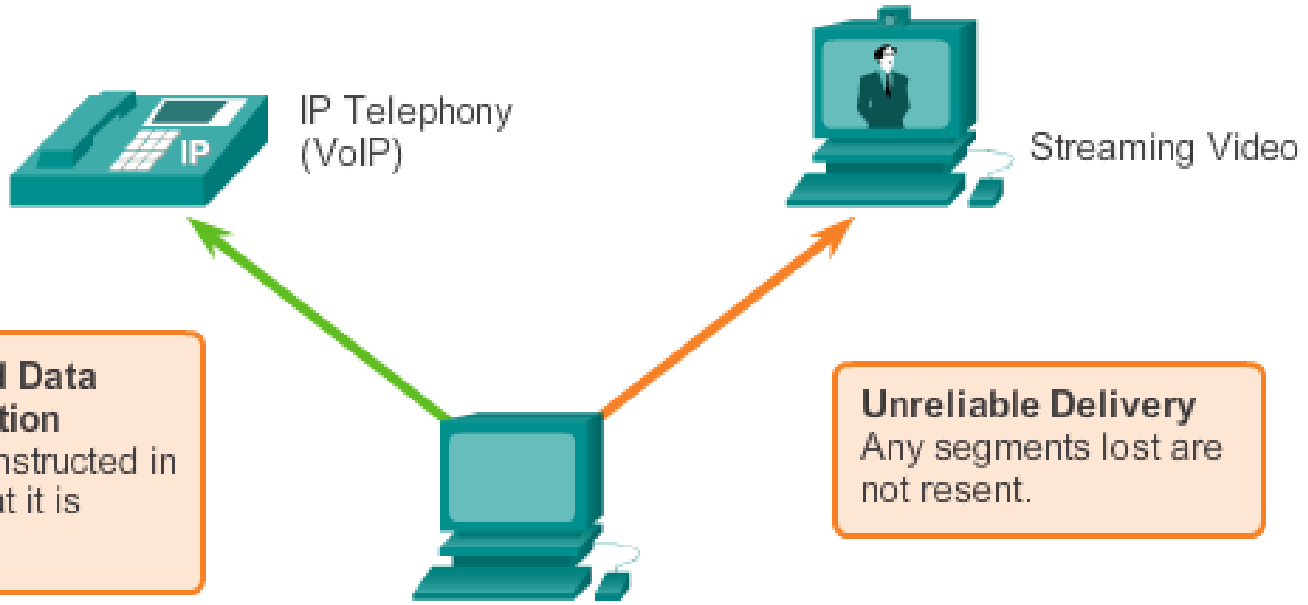


UDP Features

UDP is considered a best-effort transport protocol. **UDP is a lightweight transport protocol that offers the same data segmentation and reassembly as TCP, but without TCP reliability and flow control.**

For more information on UDP, read the RFC.

UDP



No Ordered Data Reconstruction
Data is reconstructed in the order that it is received.

Unreliable Delivery
Any segments lost are not resent.

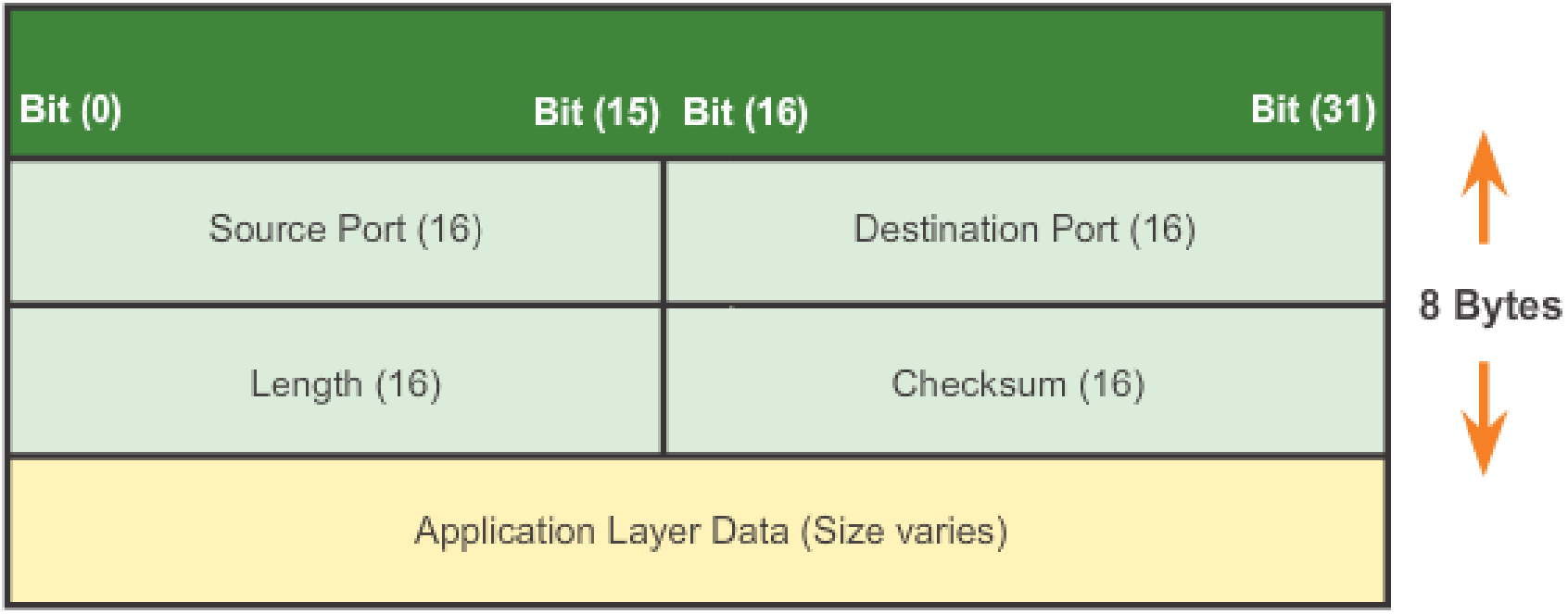
Connectionless
No session establishment.

No Flow Control
Does not inform the sender about resource availability.

UDP Header

UDP is a stateless protocol, meaning neither the client, nor the server, is obligated to keep track of the state of the communication session. If reliability is required when using UDP as the transport protocol, it must be handled by the application. The pieces of communication in UDP are called **datagrams**. These datagrams are sent as best-effort by the transport layer protocol. **UDP has a low overhead of 8 bytes.**

UDP Datagram



Multiple Separate Conversations

The transport layer must be able to separate and manage multiple communications with different transport requirement needs. Users expect to be able to simultaneously receive and send email and instant messages, view websites, and conduct a VoIP phone call. Each of these applications is sending and receiving data over the network at the same time, despite different reliability requirements. Additionally, data from the phone call is not directed to the web browser, and text from an instant message does not appear in an email.

TCP and UDP manage these multiple simultaneous conversations by using header fields that can uniquely identify these applications. **These unique identifiers are the port numbers.**

Port Addressing

Port Numbers

The source port number is associated with the originating application on the local host. The destination port number is associated with the destination application on the remote host.

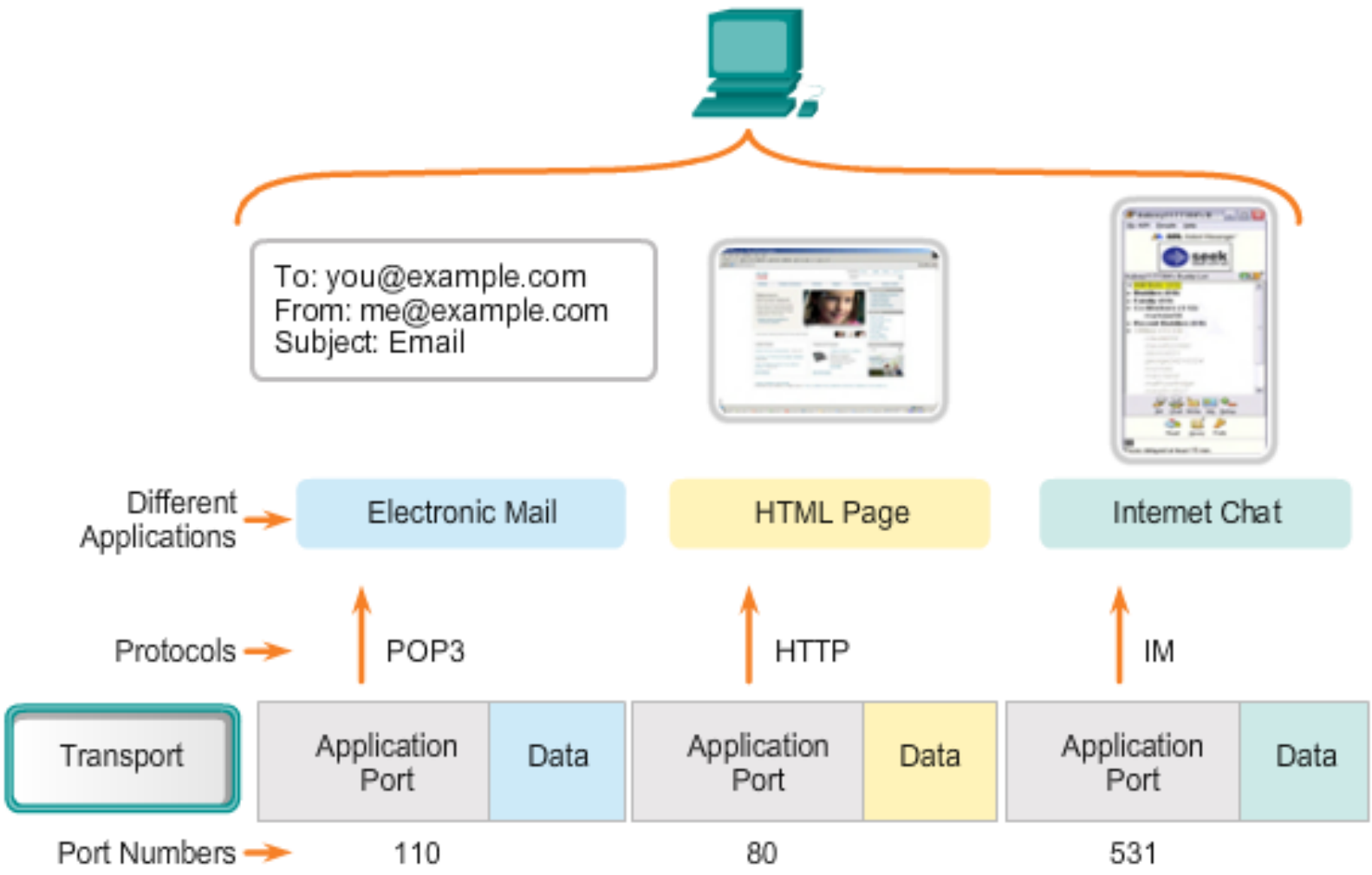
Source Port

The source port number is dynamically generated by the sending device to identify a conversation between two devices. This process allows multiple conversations to occur simultaneously.

Destination Port

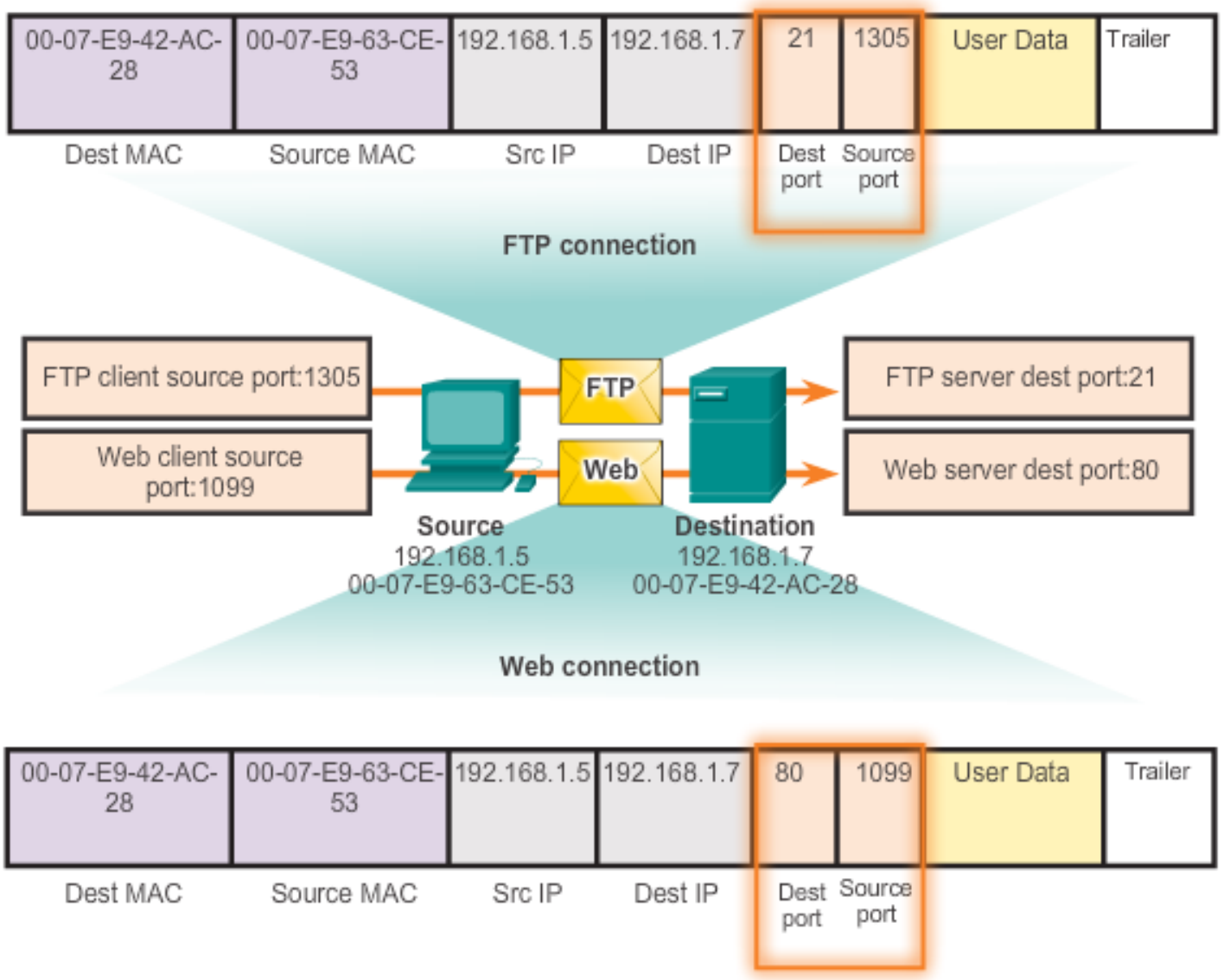
The client places a destination port number in the segment to tell the destination server what service is being requested.

Port Addressing



Socket Pairs

The source and destination ports are placed within the segment. The segments are then encapsulated within an IP packet. The IP packet contains the IP address of the source and destination. **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. **A client socket might look like this, with 1099 representing the source port number: 192.168.1.5:1099, The socket on a web server might be: 192.168.1.7:80, Together, these two sockets combine to form a socket pair: 192.168.1.5:1099, 192.168.1.7:80**



Port Number Groups

The Internet Assigned Numbers Authority (IANA) is the standards body responsible for assigning various addressing standards, including port numbers. There are different types of port numbers, as shown in Figure 1:

Well-known Ports (Numbers 0 to 1023) - These numbers are reserved for services and applications.

Registered Ports (Numbers 1024 to 49151) - These port numbers are assigned by IANA to a requesting entity to use with specific processes or applications. **For example, Cisco has registered port 1985 for its Hot Standby Routing Protocol (HSRP) process.**

Dynamic or Private Ports (Numbers 49152 to 65535) - are usually assigned dynamically by the client's OS when a connection to a service is initiated.

Port Number Range	Port Group
0 to 1023	Well-known Ports
1024 to 49151	Registered Ports
49152 to 65535	Private and/or Dynamic Ports

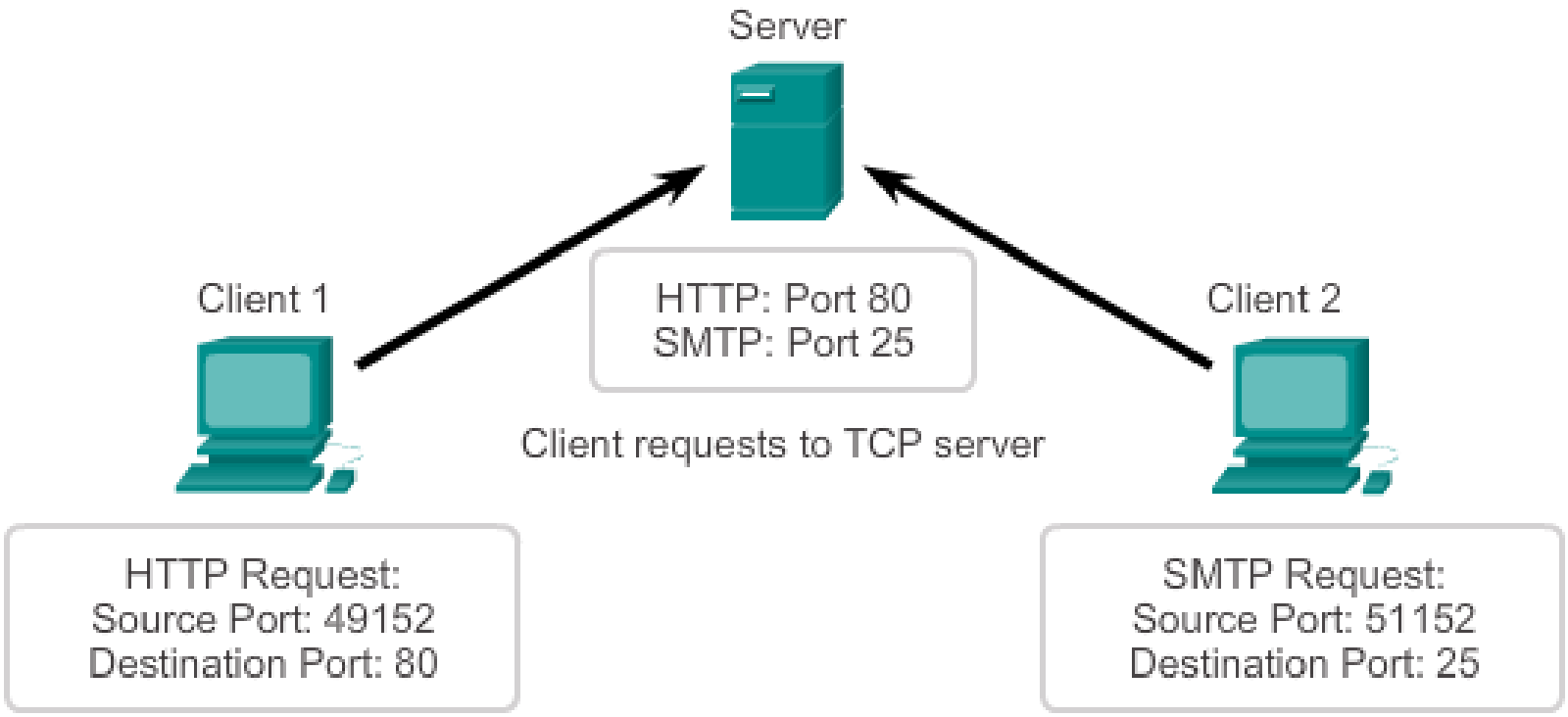
Port Number	Protocol	Application	Acronym
20	TCP	File Transfer Protocol (data)	FTP
21	TCP	File Transfer Protocol (control)	FTP
22	TCP	Secure Shell	SSH
23	TCP	Telnet	—
25	TCP	Simple Mail Transfer Protocol	SMTP
53	UDP, TCP	Domain Name Service	DNS
67	UDP	Dynamic Host Configuration Protocol (server)	DHCP
68	UDP	Dynamic Host Configuration Protocol (client)	DHCP
69	UDP	Trivial File Transfer Protocol	TFTP
80	TCP	Hypertext Transfer Protocol	HTTP
110	TCP	Post Office Protocol version 3	POP3
143	TCP	Internet Message Access Protocol	IMAP
161	UDP	Simple Network Management Protocol	SNMP
443	TCP	Hypertext Transfer Protocol Secure	HTTPS

TCP Server Processes

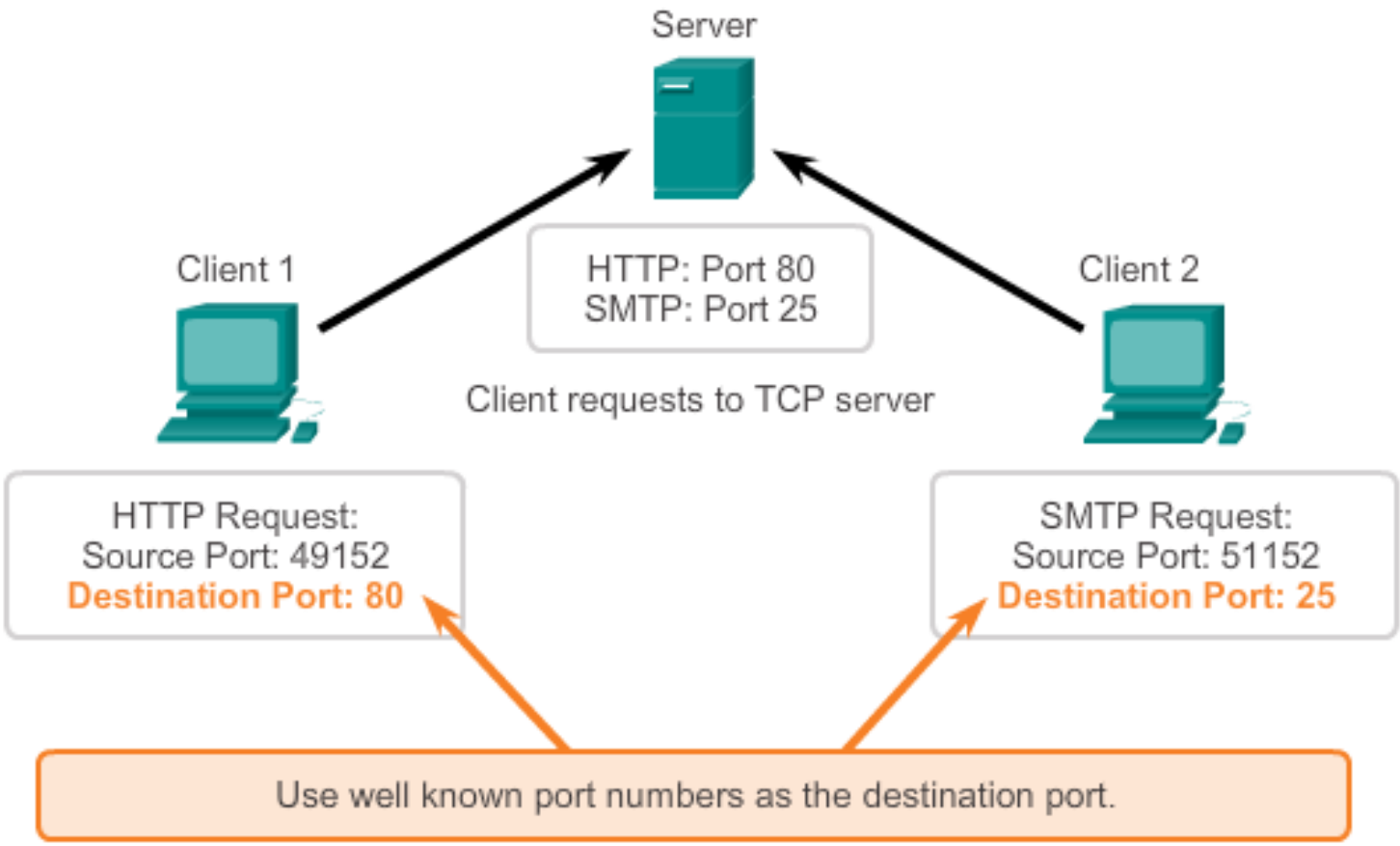
Each application process running on the server is configured to use a port number, either by default or manually, by a system administrator. An individual server cannot have two services assigned to the same port number within the same transport layer services.

Any incoming client request addressed to the correct socket is accepted, and the data is passed to the server application. There can be many ports open simultaneously on a server, one for each active server application.

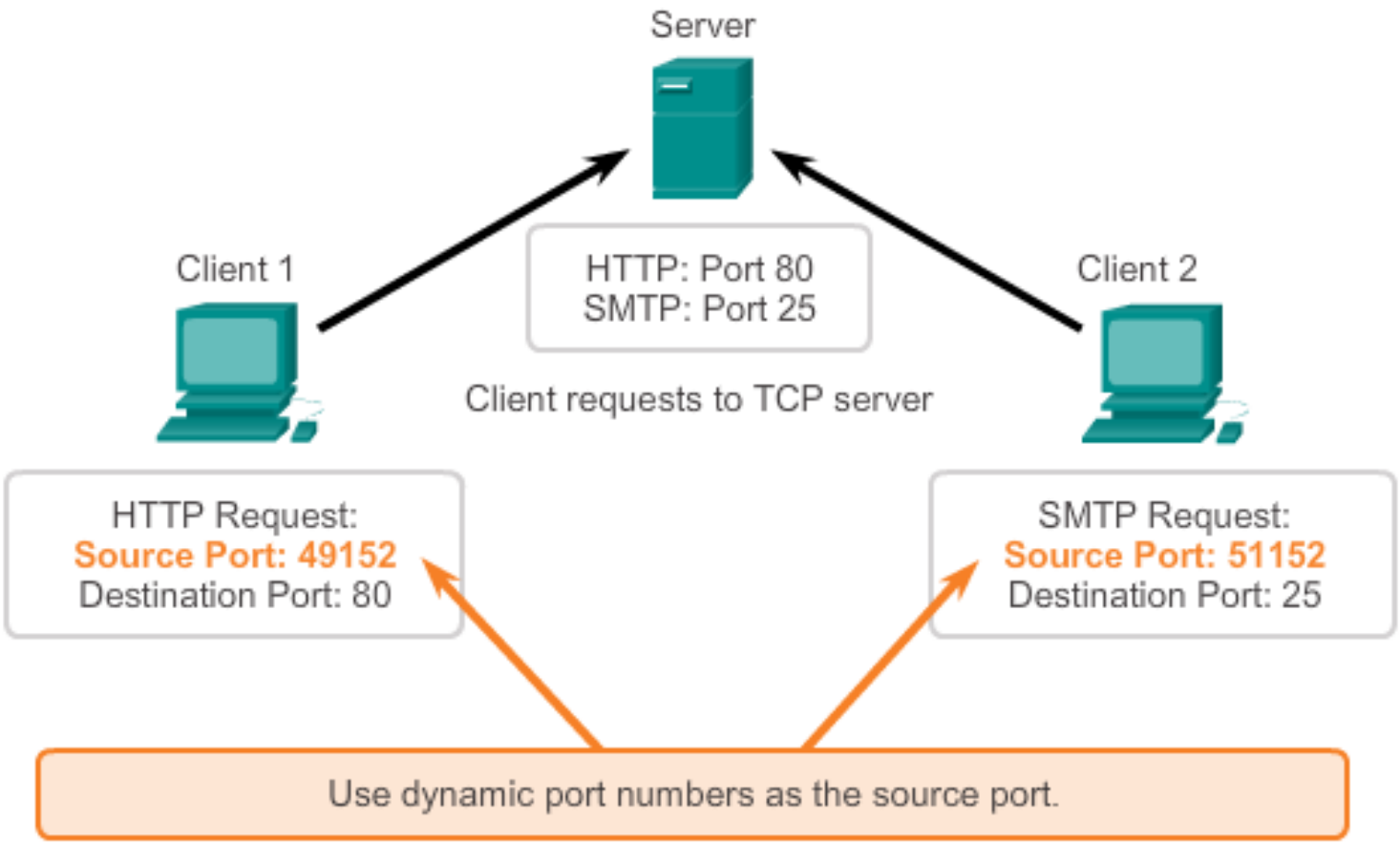
Clients Sending TCP Requests



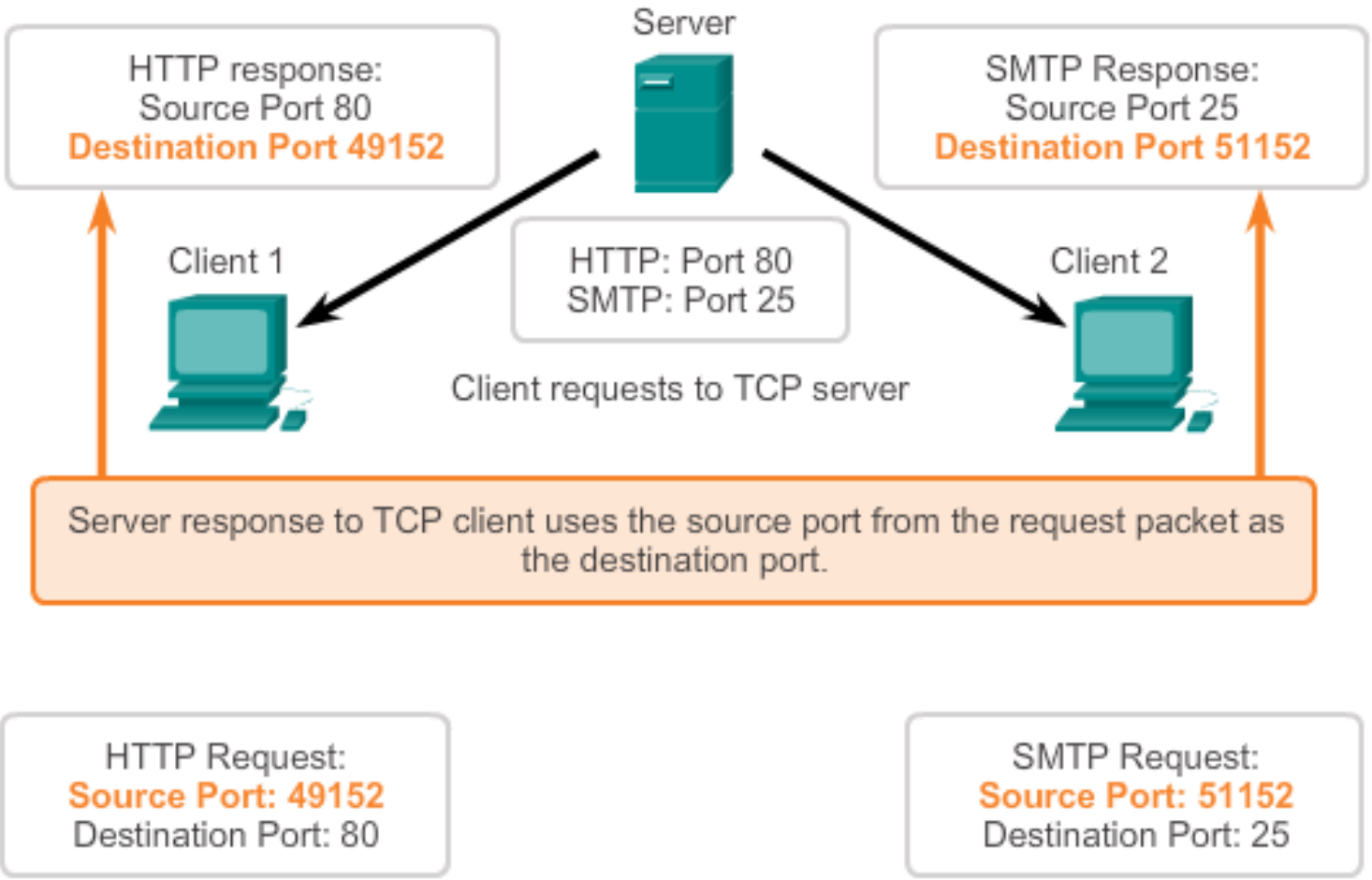
Request Destination Ports



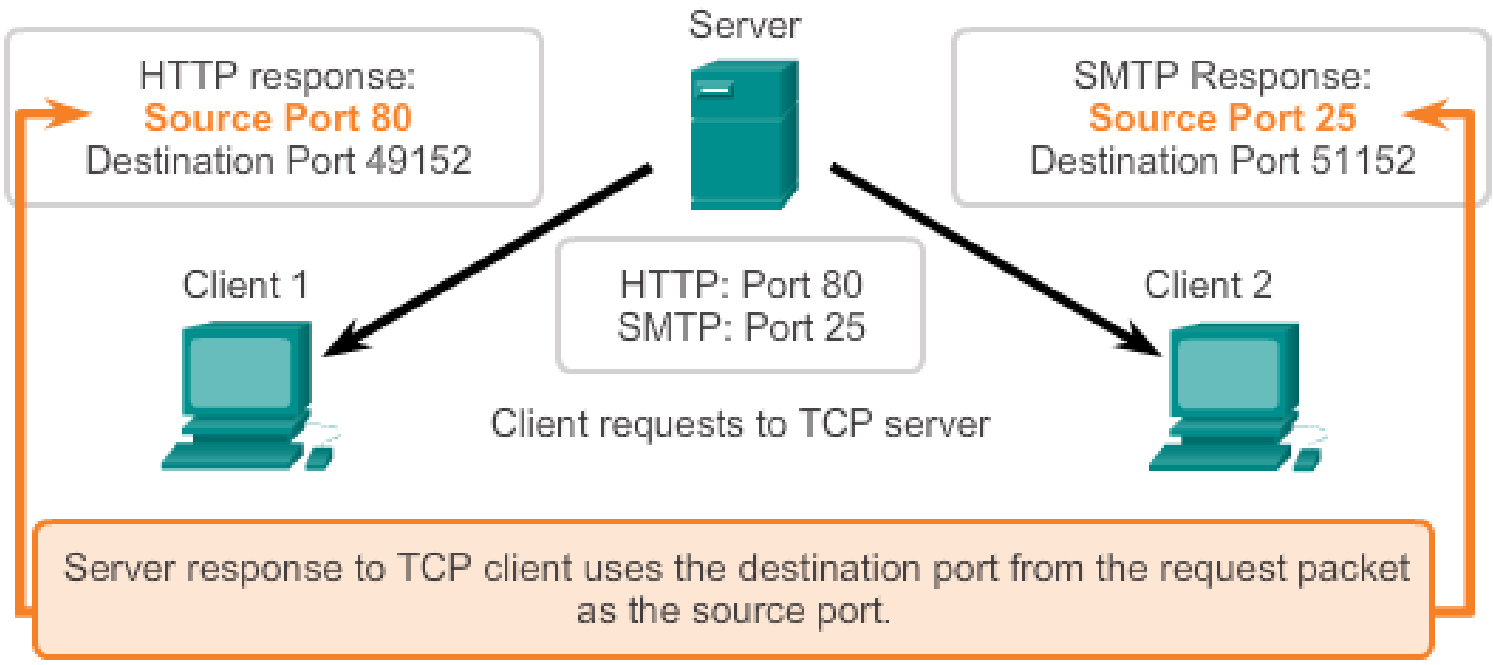
Request Source Ports



Response Destination Ports



Response Source Ports



HTTP Request:
Source Port: 49152
Destination Port: 80

SMTP Request:
Source Port: 51152
Destination Port: 25

TCP Connection Establishment

A TCP connection is established in three steps:

Step 1 - The initiating client requests a client-to-server communication session with the server.

Step 2 - The server acknowledges the client-to-server communication session and requests a server-to-client communication session.

Step 3 - The initiating client acknowledges the server-to-client communication session.

TCP Three-way Handshake Analysis

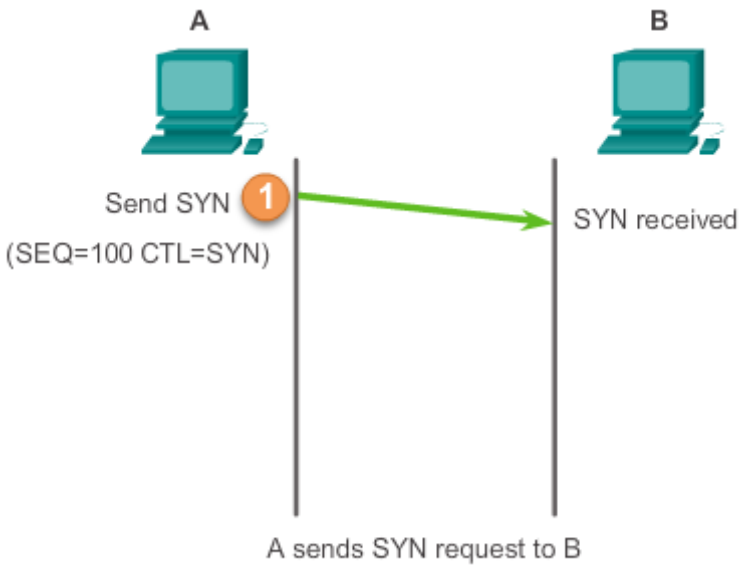
TCP is a full-duplex protocol, where each connection represents two one-way communication streams or sessions. **To establish the connection**, the hosts perform a three-way handshake. Control bits in the TCP header indicate the progress and status of the connection.

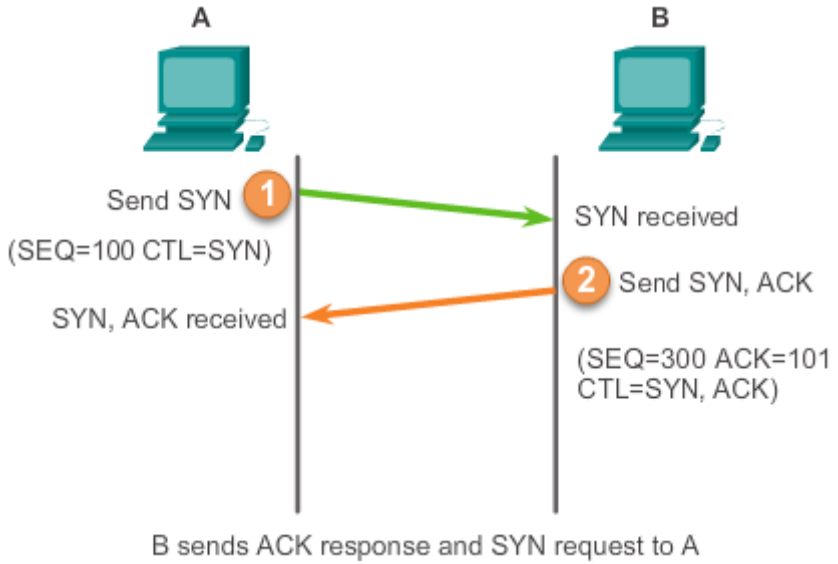
The three-way handshake:

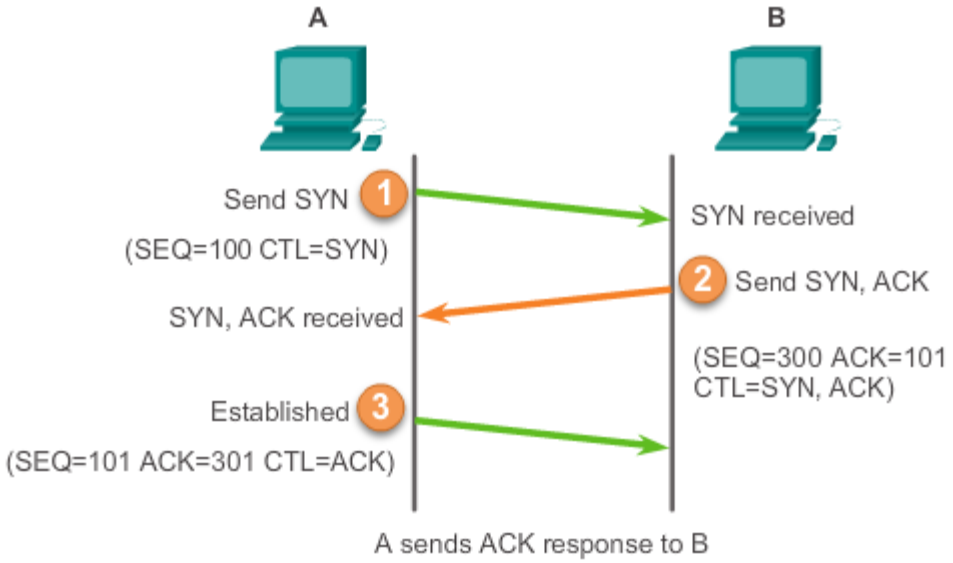
1. Establishes that the destination device is present on the network.
2. Verifies that the destination device has an active service and is accepting requests on the destination port number that the initiating client intends to use.
3. Informs the destination device that the source client intends to establish a communication session on that port number.

After the communication is completed, the sessions are closed, and the connection is terminated.

The six bits in the Control Bits field of the TCP segment header are also known as flags. A flag is a bit that is either set to on or off.







TCP Session Termination

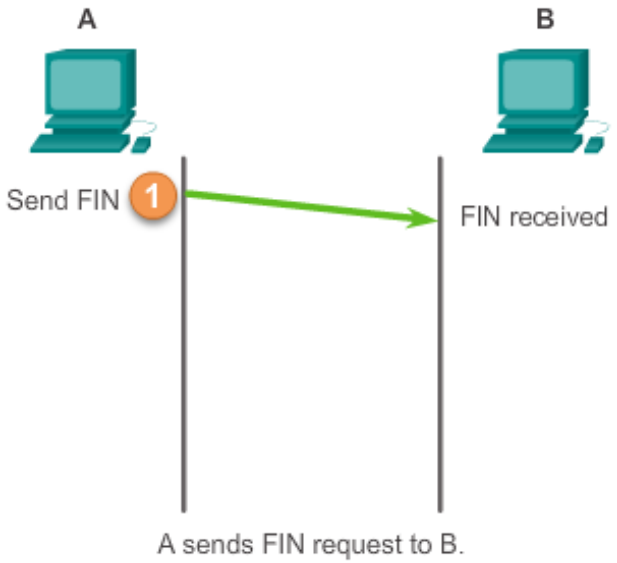
To close a connection, the Finish (FIN) control flag must be set in the segment header. To end each one-way TCP session, a two-way handshake, consisting of a FIN segment and an Acknowledgment (ACK) segment, is used. Therefore, to terminate a single conversation supported by TCP, four exchanges are needed to end both sessions.

Step 1 - When the client has no more data to send in the stream, it sends a segment with the FIN flag set.

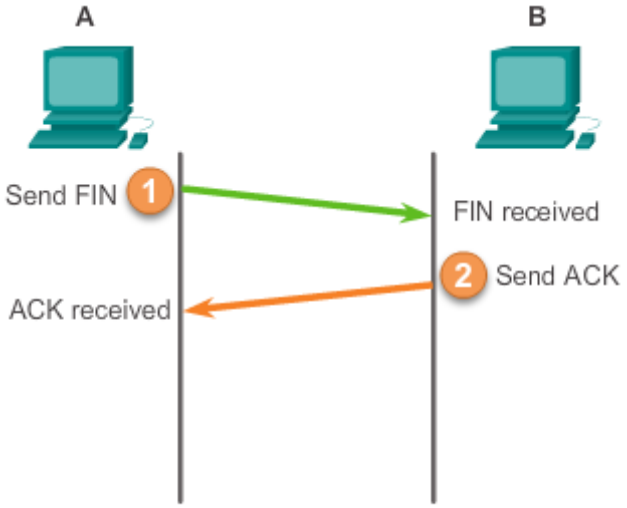
Step 2 - The server sends an ACK to acknowledge the receipt of the FIN to terminate the session from client to server.

Step 3 - The server sends a FIN to the client to terminate the server-to-client session.

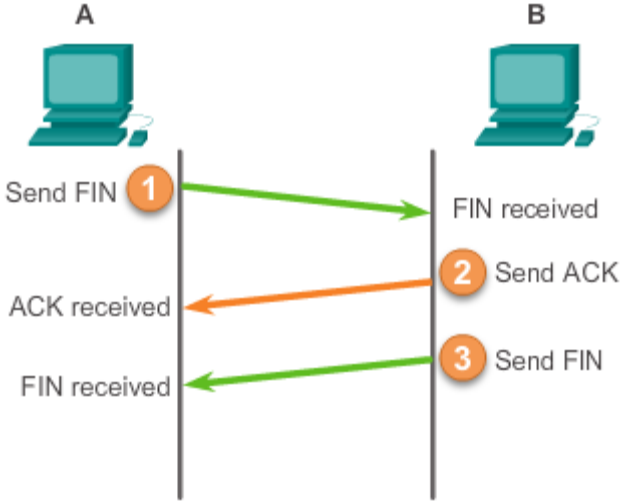
Step 4 - The client responds with an ACK to acknowledge the FIN from the server. When all segments have been acknowledged, the session is closed.



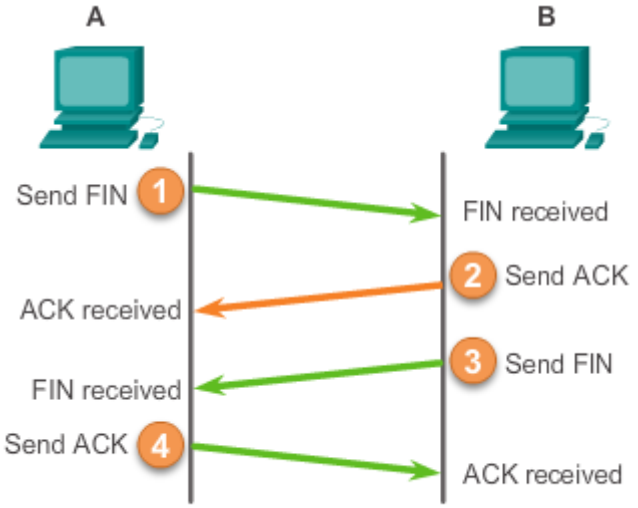
A sends FIN request to B.



B sends ACK response to A.



B sends FIN request to A



A sends ACK response to B.

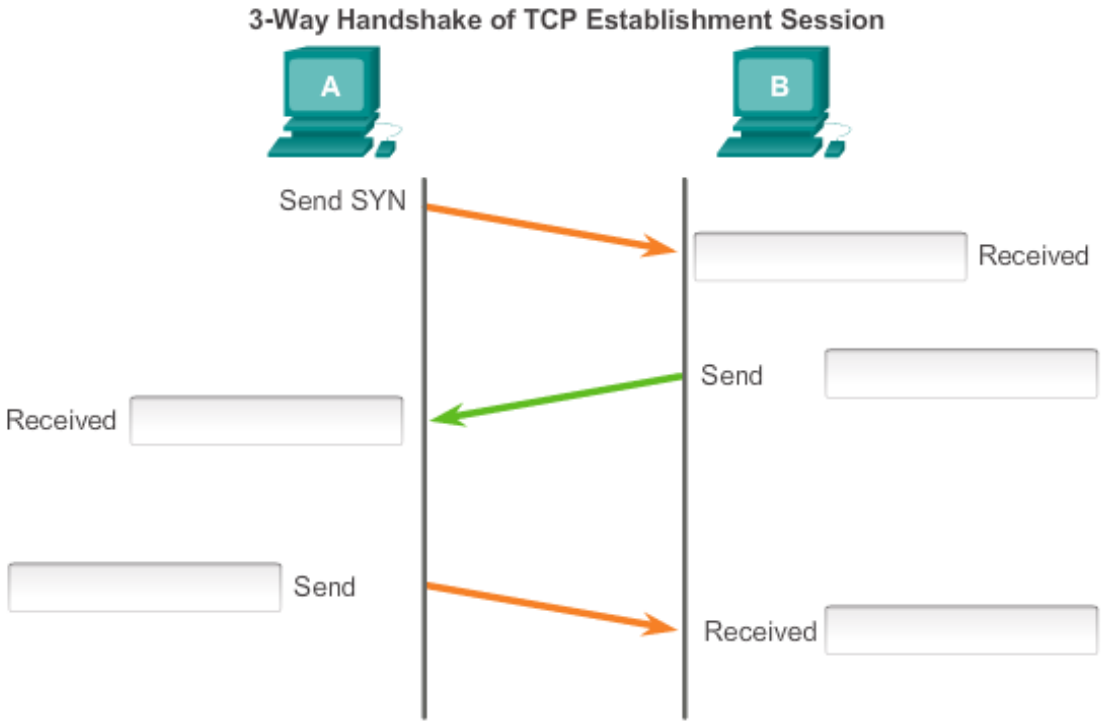
Activity (1)

Activity - Part 1: TCP Connection and Termination Process

Drag each descriptor to its appropriate place in the image to illustrate the 3-way handshake of a TCP **establishment** session.

Descriptors

- ACK
- SYN, ACK
- SYN

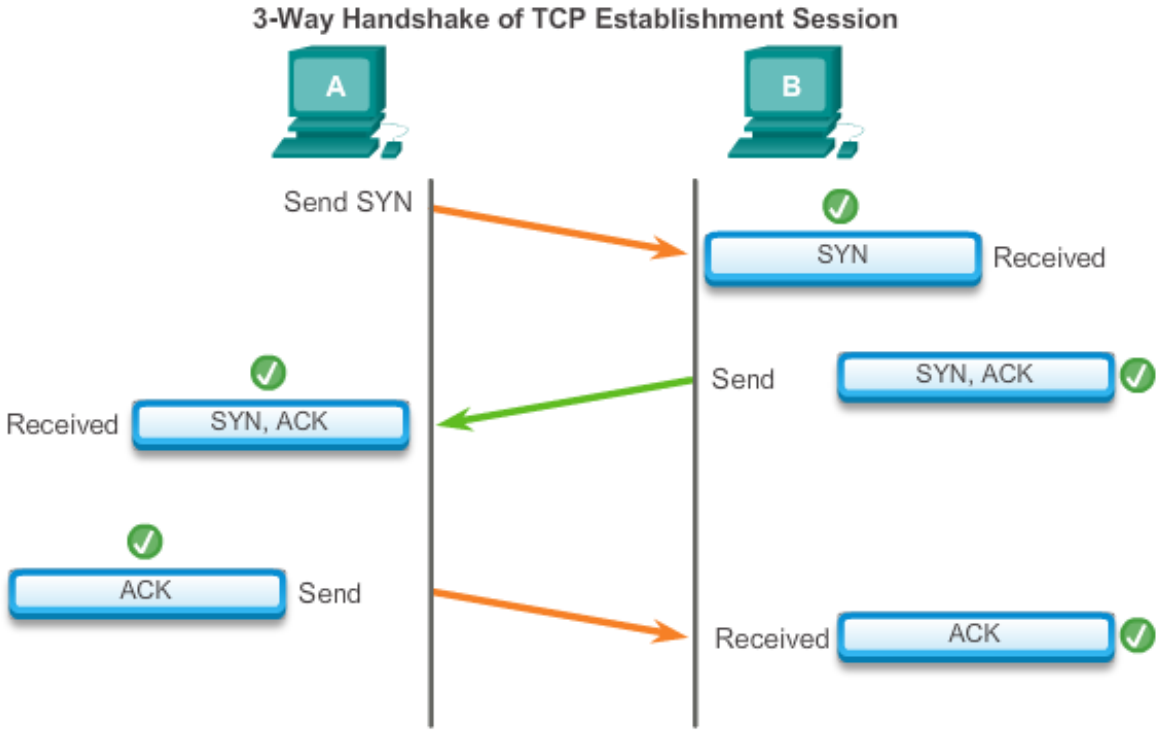


Activity - Part 1: TCP Connection and Termination Process

Drag each descriptor to its appropriate place in the image to illustrate the 3-way handshake of a TCP **establishment** session.

Descriptors

- ACK
- SYN, ACK
- SYN



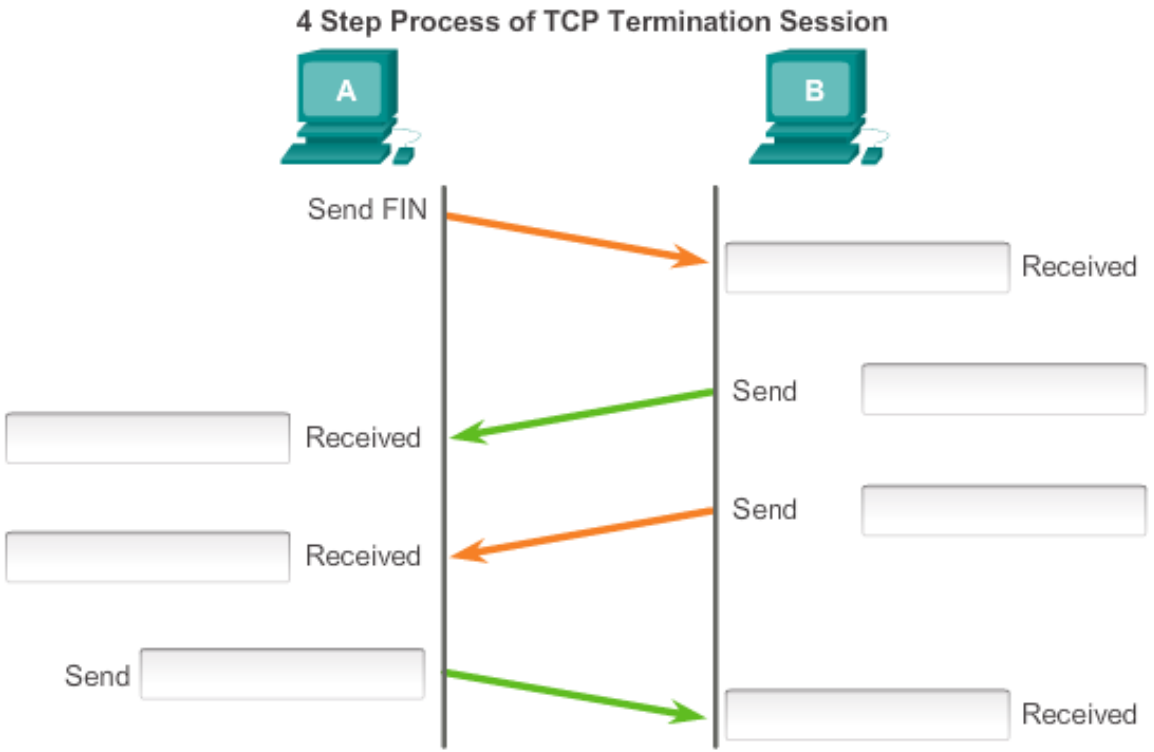
Activity (2)

Activity - Part 2: TCP Connection and Termination Process

Drag each descriptor to its appropriate place in the image to illustrate the 4 step process of a TCP **termination** session.

Descriptors

- FIN
- ACK

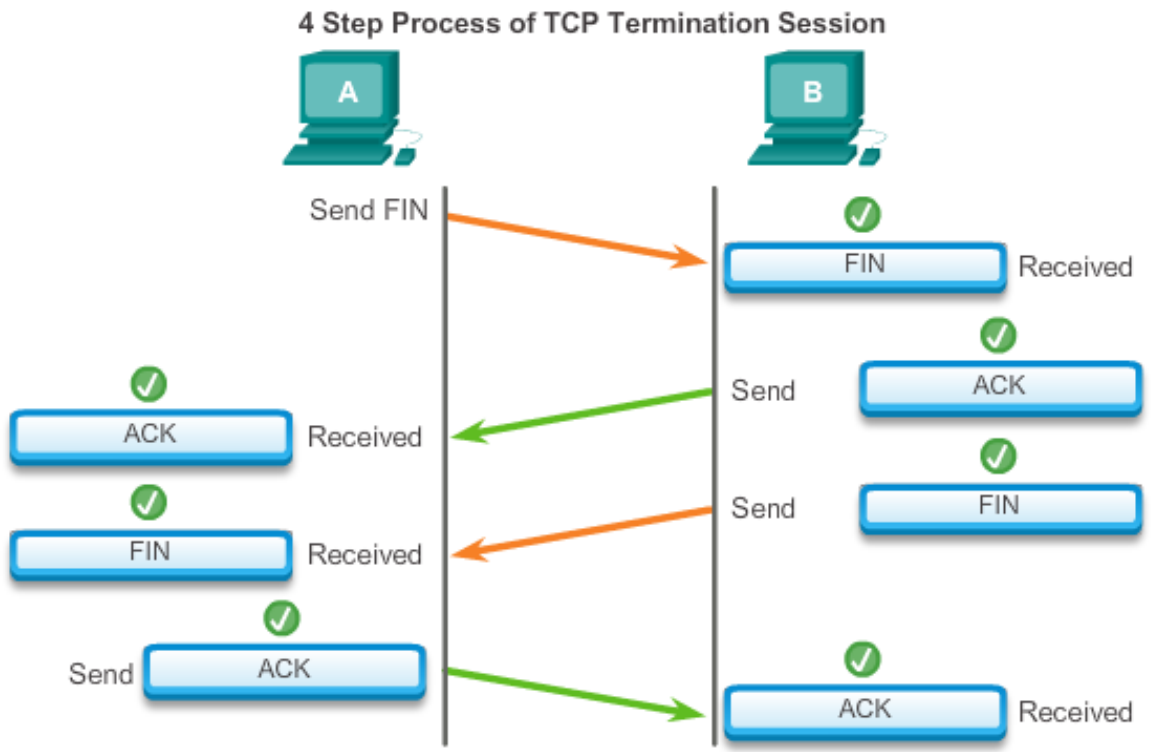


Activity - Part 2: TCP Connection and Termination Process

Drag each descriptor to its appropriate place in the image to illustrate the 4 step process of a TCP **termination** session.

Descriptors

- FIN
- ACK



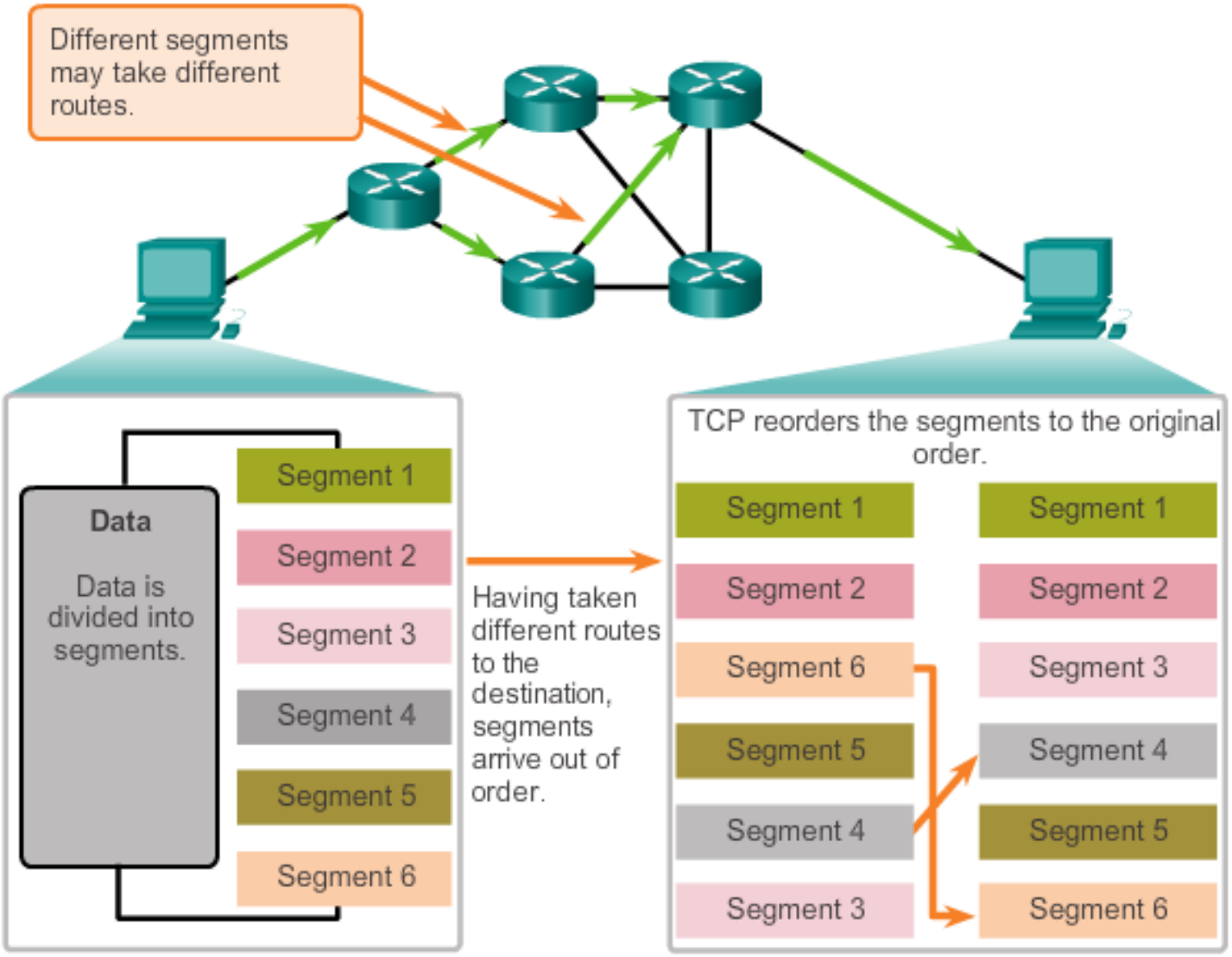
TCP Reliability - Ordered Delivery

TCP segments may arrive at their destination out of order. Sequence numbers are assigned in the header of each packet to achieve this goal. The sequence number represents the first data byte of the TCP segment.

During session setup, an initial sequence number (**ISN**) is set. This ISN represents the starting value of the bytes for this session that is transmitted to the receiving application. As data is transmitted during the session, the sequence number is incremented by the number of bytes that have been transmitted. This data byte tracking enables each segment to be uniquely identified and acknowledged. Missing segments can then be identified.

Segment sequence numbers indicate how to reassemble and reorder received segments. The receiving TCP process places the data from a segment into a **receiving buffer**.

TCP Segments Are Reordered at the Destination



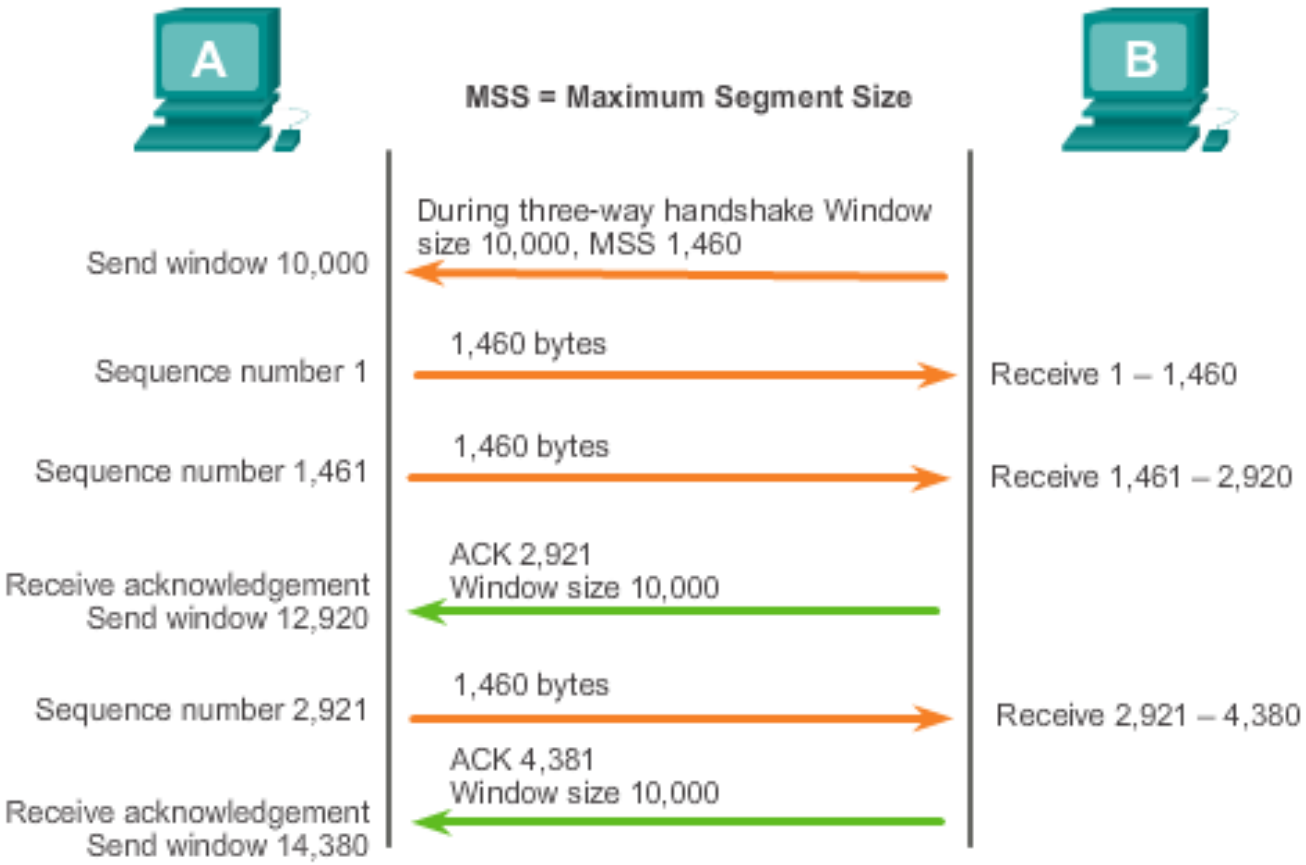
TCP Flow Control - Window Size and Acknowledgments

TCP also provides mechanisms for **flow control**, the amount of data that the destination can receive and process reliably.

The **window size** is the number of bytes that the destination device of a TCP session can accept and process at one time. In this example, **PC B's initial window size for the TCP session is 10,000 bytes.** Starting with the first byte, byte number 1, the last byte **PC A can send without receiving an acknowledgment is byte 10,000.** This is known as PC A's send window. The window size is included in every TCP segment so the destination can modify the window size at any time depending on buffer availability.

Note: In the figure, the source is transmitting 1,460 bytes of data within each TCP segment. This is known as the **MSS (Maximum Segment Size).**

TCP Window Size Example



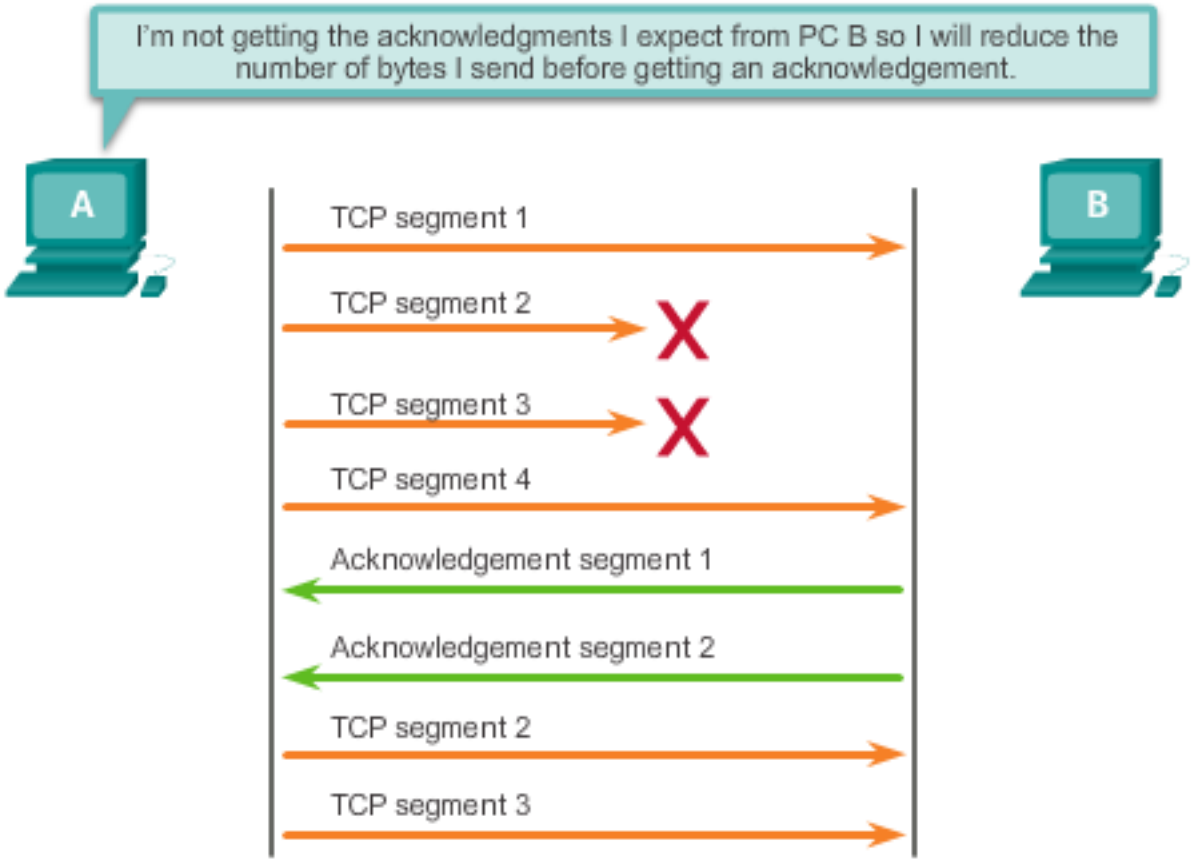
The **window size** determines the number of bytes that can be sent before expecting an acknowledgment.
The **acknowledgement** number is the number of the next expected byte.

TCP Flow Control - Congestion Avoidance

When congestion occurs on a network, it results in packets being discarded by the overloaded router. When packets containing TCP segments don't reach their destination, they are left unacknowledged. By determining the rate at which TCP segments are sent but not acknowledged, the source can assume a certain level of network congestion.

Whenever there is congestion, retransmission of lost TCP segments from the source will occur. If the retransmission is not properly controlled, the additional retransmission of the TCP segments can make the congestion even worse. To avoid and control congestion, TCP employs several congestion handling mechanisms, timers, and algorithms.

TCP Congestion Control



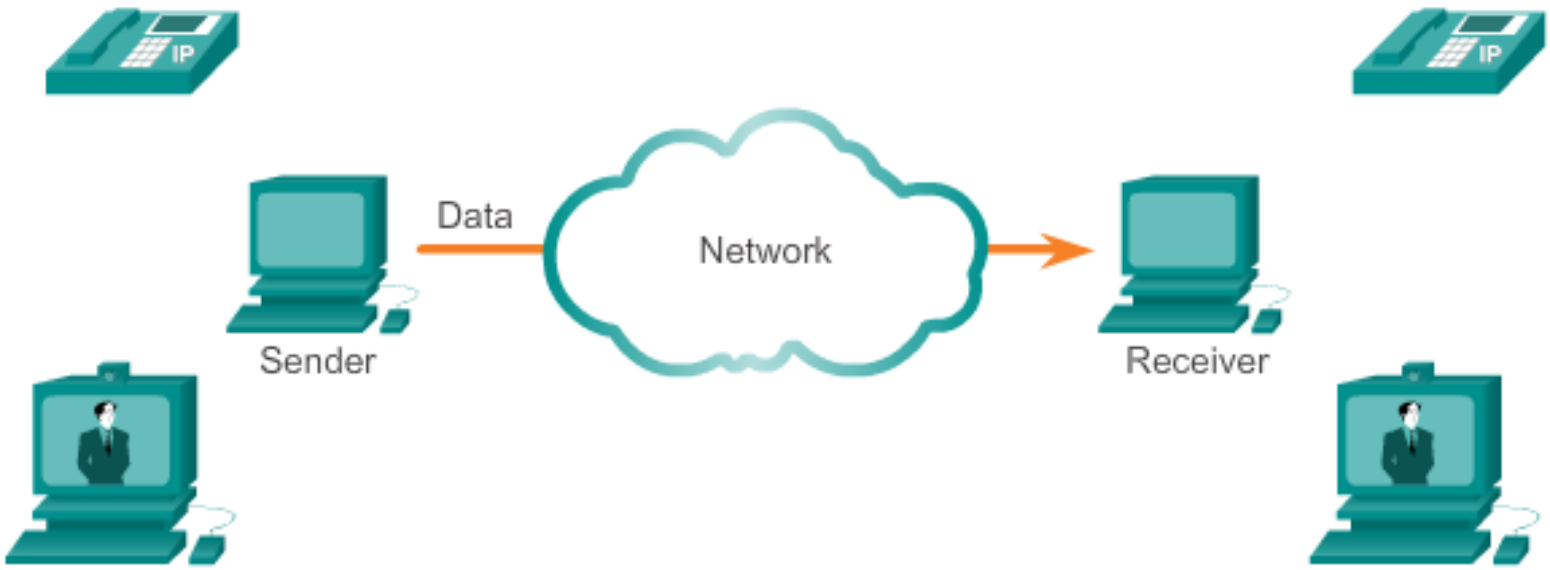
Acknowledgement numbers are for the next expected byte and not for a segment. The segment numbers used are simplified for illustration purposes.

UDP Low Overhead versus Reliability

UDP is a simple protocol that provides the basic transport layer functions. It has much lower overhead than TCP because it is **not connection-oriented** and does not offer the sophisticated **retransmission**, **sequencing**, and **flow control** mechanisms that provide reliability.

The low overhead of UDP makes it very desirable for protocols that make simple request and reply transactions.

UDP Low Overhead Data Transport



UDP does not establish a connection before sending data.

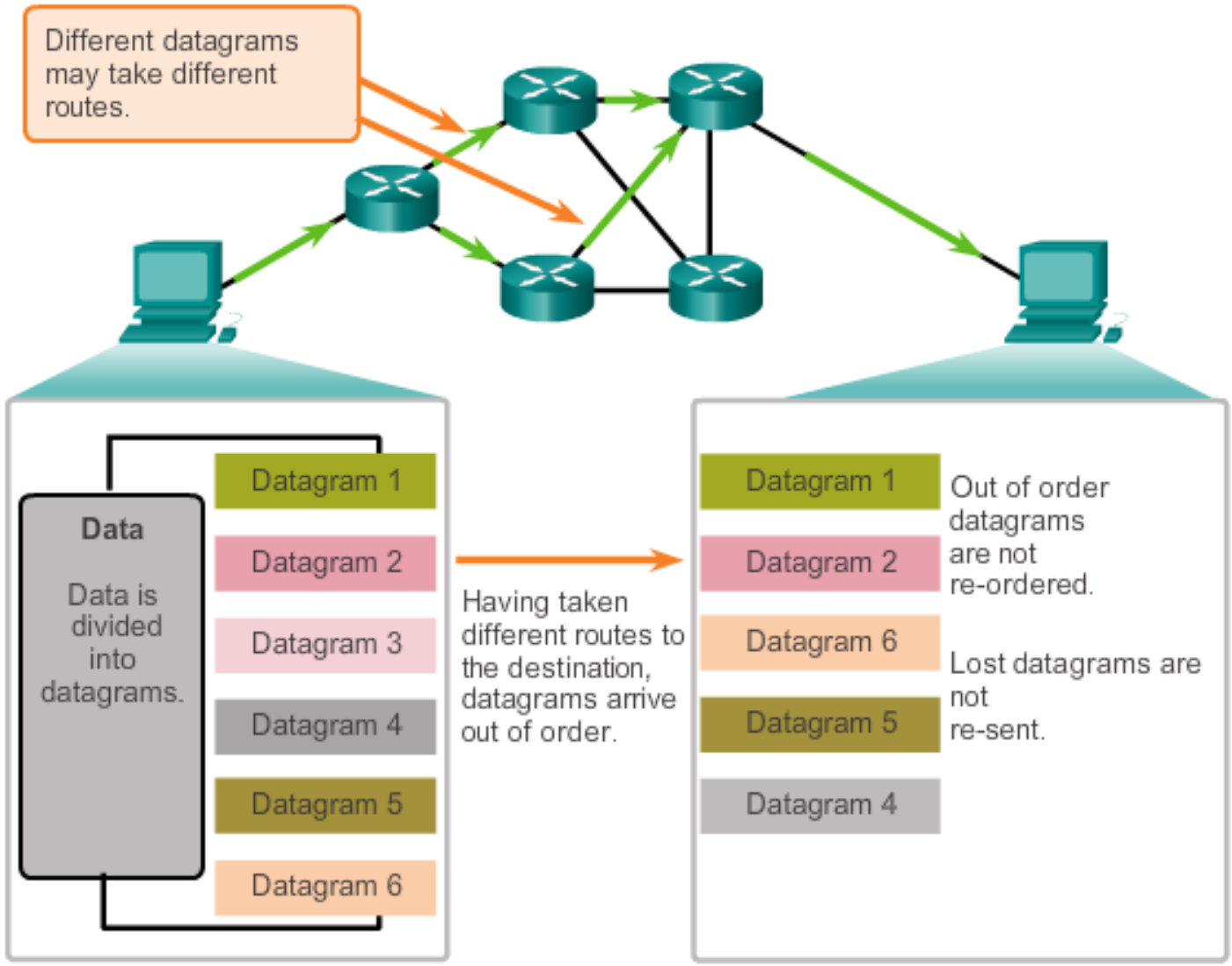
UDP provides low overhead data transport because it has a small datagram header and no network management traffic.

UDP Datagram Reassembly

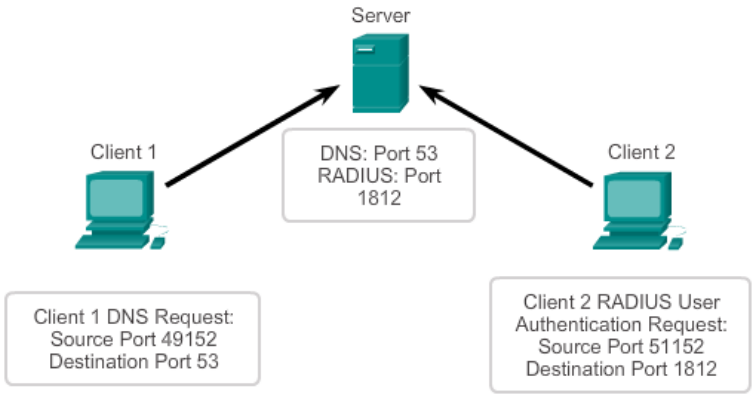
When UDP datagrams are sent to a destination, they often take different paths and arrive in the wrong order. UDP does not track sequence numbers the way TCP does. **UDP has no way to reorder the datagrams into their transmission order.**

Therefore, UDP simply reassembles the data in the order that it was received and forwards it to the application.

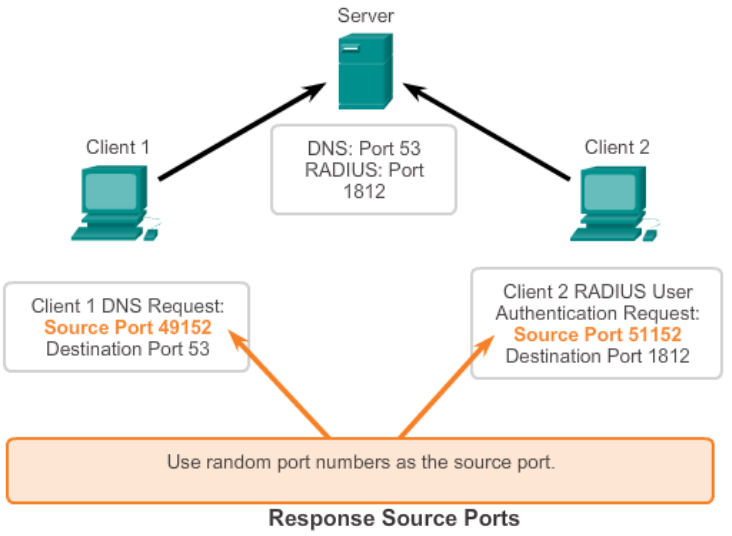
UDP: Connectionless and Unreliable



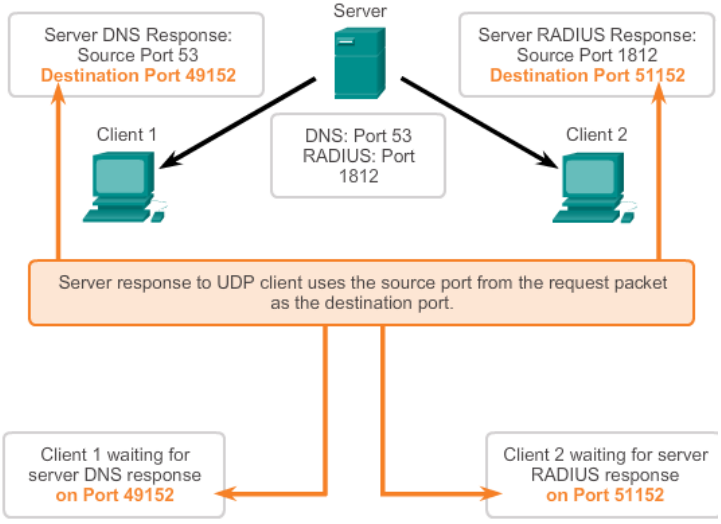
Clients Sending UDP Requests



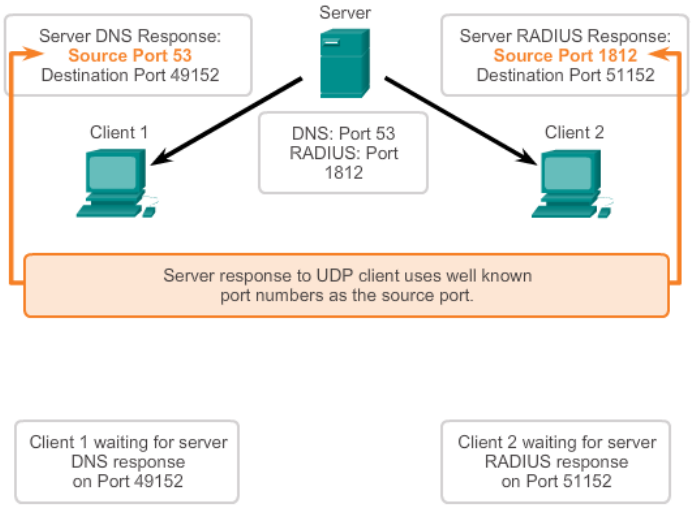
Request Source Ports



Response Destination Ports



Response Source Ports



Applications that use TCP

TCP is a great example of how the different layers of the TCP/IP protocol suite have specific roles. TCP handles all tasks associated with dividing the data stream into segments, providing **reliability**, **controlling data flow**, and the **reordering** of segments. TCP frees the application from having to manage any of these tasks.

Applications that use UDP

There are three types of applications that are best suited for UDP:

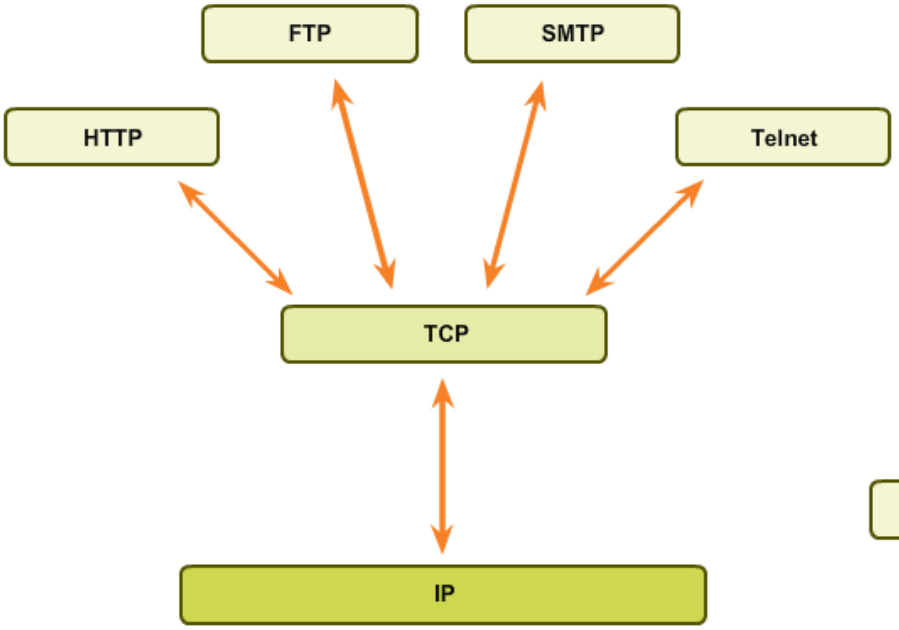
Live video and multimedia applications - Can tolerate some data loss, but require little or no delay. Examples include VoIP and live streaming video.

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.

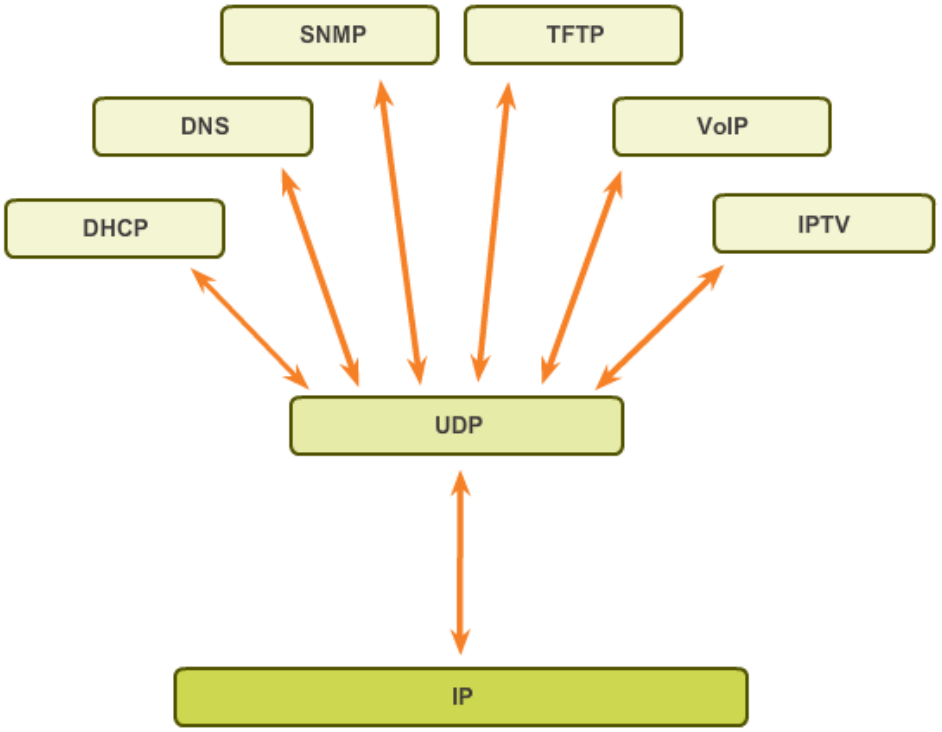
Applications that handle reliability themselves - Unidirectional communications where flow control, error detection, acknowledgments, and error recovery is not required or can be handled by the application. Examples include SNMP and TFTP.

Although DNS and SNMP use UDP by default, both can also use TCP.

Applications that use TCP



Applications that use UDP



Summary

The transport layer provides transport-related services by:
Dividing data received from an application into segments
Adding a header to identify and manage each segment
Using the header information to reassemble the segments back into application data
Passing the assembled data to the correct application
UDP and TCP are common transport layer protocols.
UDP datagrams and TCP segments have headers added in front of the data that include a source port number and destination port number. These port numbers enable data to be directed to the correct application running on the destination computer.
TCP does not pass any data to the network until it knows that the destination is ready to receive it. TCP then manages the flow of the data and resends any data segments that are not acknowledged as being received at the destination. TCP uses mechanisms of handshaking, timers, acknowledgment messages, and dynamic windowing to achieve reliability. The reliability process, however, imposes overhead on the network in terms of much larger segment headers and more network traffic between the source and destination.
If the application data needs to be delivered across the network quickly, or if network bandwidth cannot support the overhead of control messages being exchanged between the source and the destination systems, UDP would be the developer's preferred transport layer protocol. UDP provides none of the TCP reliability features. However, this does not necessarily mean that the communication itself is unreliable; there may be mechanisms in the application layer protocols and services that process lost or delayed datagrams if the application has these requirements.
The application developer decides the transport layer protocol that best meets the requirements for the application. It is important to remember that the other layers all play a part in data network communications and influences its performance.