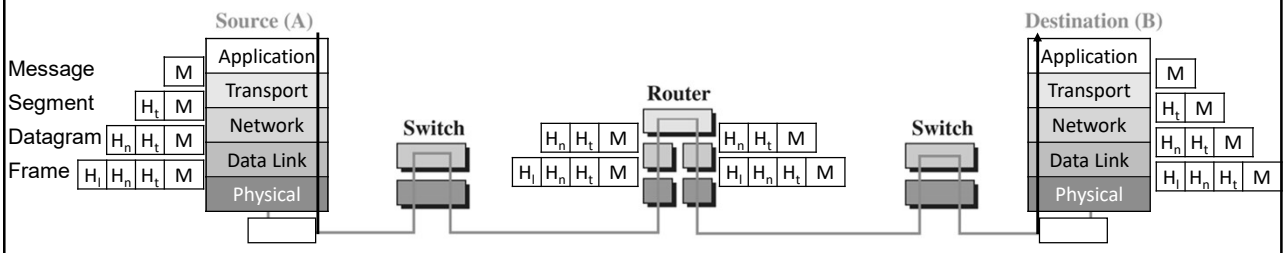
Welcome!

ELEC 8560 – Computer Networks

Transport Layer

1

Recall: End-to-End Communication via Internet



2

Outline

- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

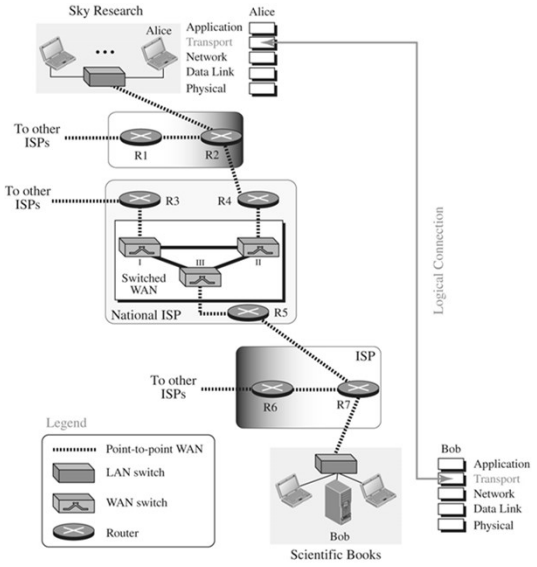
- Recommended reading: Forouzan – Chapter 9

Outline

- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

Communication at the Transport Layer

- Transport layer provides a logical connection between application processes running on different hosts
  - Transport messages from source to destination
  - Sender: encapsulates messages from application layer into segments, passes to network layer
  - Receiver: reassemble segments into messages, delivers to application layer



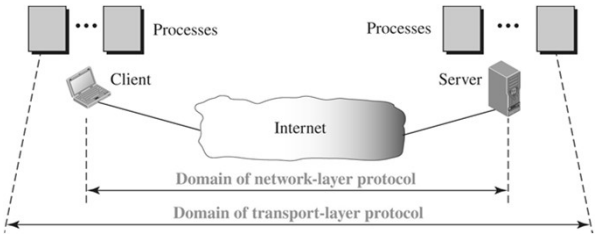
Data Flow Diagram

Outline

- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

Transport vs. Network Layer Services

- Network layer provides a logical connection between hosts
- Transport layer provides a logical connection between processes running on different hosts
  - A process is an application-layer entity (running program) that uses the services of the transport layer
  - Process-to-process communication between two application layers, one at the local host and the other at the remote host

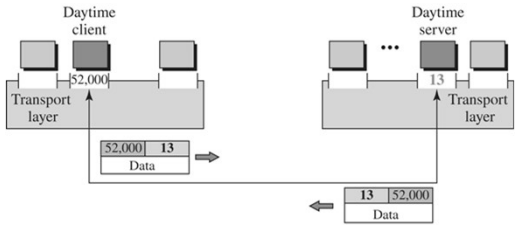


Outline

- Communication at the transport layer
- Transport-layer services
  - Addressing
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

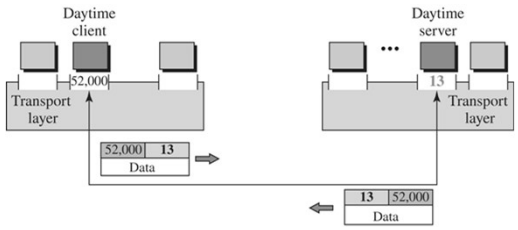
Addressing: Port Numbers

- A common way to achieve process-to-process communication is through the client-server paradigm
  - A process on the local host, called a client, needs services from a process usually on the remote host, called a server
  - Local and remote hosts are defined using IP address
  - Processes on a particular host are defined using port numbers
  - That is, transport layer uses port numbers to accomplish process-to-process communication (i.e., end-to-end addresses)



Addressing: Port Numbers (cont.)

- Operating systems today support both multiprogramming and multiuser environment
  - A remote computer can run several programs at the same time, just as several local computers can run one or more client programs at the same time
- ICANN has divided the port numbers into three ranges:
  - Well-known: 0 – 1023, every client knows them to get service
  - Registered: 1024 – 49,151
  - Dynamic (private, temporary, or ephemeral): 49,152 – 65,535

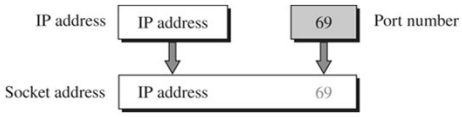


Example: Some Well-known Ports Numbers

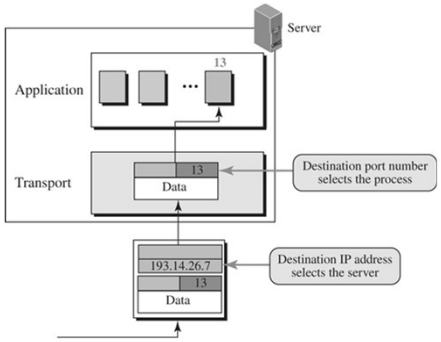
Port	*Procces*	*UDP*	*TCP*	*SCTP*	*Description*
7	Echo	√	√	√	Echoes back a received datagram
9	Discard	√	√	√	Discards any datagram that is received
11	Users	√	√	√	Active users
13	Daytime	√	√	√	Returns the date and the time
17	Quote	√	√	√	Returns a quote of the day
19	Chargen	√	√	√	Returns a string of characters
20	FTP-data		√	√	File Transfer Protocol
21	FTP-21		√	√	File Transfer Protocol
23	TELNET		√	√	Terminal Network
25	SMTP		√	√	Simple Mail Transfer Protocol
53	DNS	√	√	√	Domain Name System
67	DHCP	√	√	√	Dynamic Host Configuration Protocol
69	TFTP	√	√	√	Trivial File Transfer Protocol
80	HTTP		√	√	Hypertext Transfer Protocol
111	RPC	√	√	√	Remote Procedure Call
123	NTP	√	√	√	Network Time Protocol
161	SNMP-server	√			Simple Network Management Protocol
162	SNMP-client	√			Simple Network Management Protocol

Addressing: Socket Address

- The combination of the IP address and port number used by the transport layer is called a socket number



- In the Internet, we need two socket addresses:
  - Client socket address
    - Clients use ephemeral port numbers to identify themselves
  - Server socket address
    - Servers use well-known port numbers to provide services

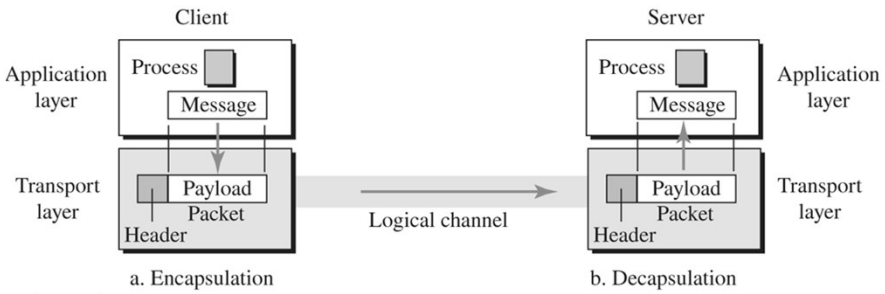


Outline

- Communication at the transport layer
- Transport-layer services
  - Encapsulation and Decapsulation
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

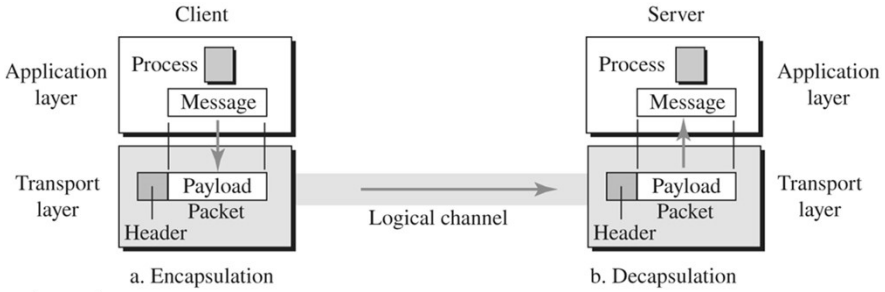
Encapsulation and Decapsulation

- To send a message from one process to another, transport-layer protocol encapsulates (sender) and decapsulates (receiver) messages
- When a process has a message to send:
  - Passes the message to the transport layer plus a pair of socket addresses and some other information which depend on the transport-layer protocol
  - Transport layer receives the data and adds the transport-layer header



Encapsulation and Decapsulation (cont.)

- To send a message from one process to another, transport-layer protocol encapsulates (sender) and decapsulates (receiver) messages
- When a message arrives at destination transport layer:
  - Header is dropped and message is delivered to the process running at the application layer
  - Sender socket address may be passed to the process if needed



Outline

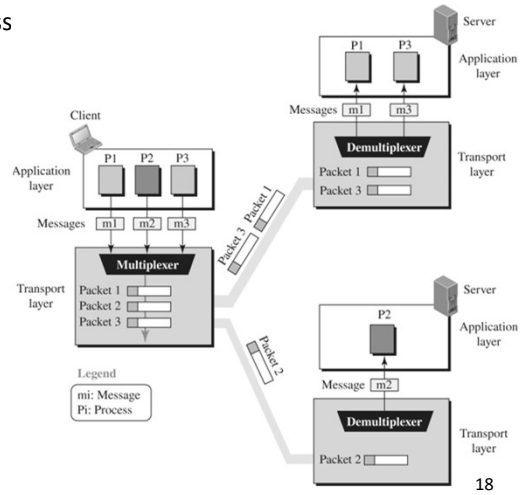
- Communication at the transport layer
- Transport-layer services
  - Multiplexing and Demultiplexing
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

Multiplexing and Demultiplexing

- When transport layer accepts items from more than one socket, this is referred to as multiplexing (many-to-one)
- When transport layer delivers items to more than one socket, this is referred to as demultiplexing (one-to-many)
- The transport layer performs multiplexing at the source and demultiplexing at the destination
  - A host can be both source and destination at the same time
- Port numbers allow multiplexing and de-multiplexing at this layer

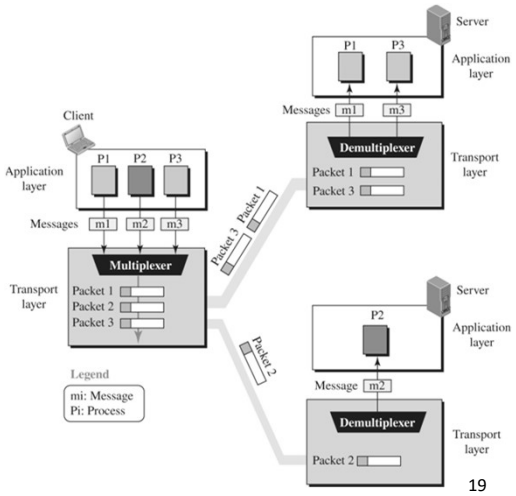
Example: Multiplexing and Demultiplexing

- Scenario: Client communicating with two servers
  - Three client processes running at the client site
    - P1 and P3 need to send requests to server process running in top server
    - P2 needs to send requests to server process running in bottom server
  - Transport layer at client accepts three messages and creates three packets, multiplex and send them



Example: Multiplexing and Demultiplexing (cont.)

- Scenario: Client communicating with two servers
  - Transport layer at top server demultiplex and distribute messages to the corresponding processes using destination port address
  - Transport layer at bottom server demultiplex and distribute message to the corresponding process using destination port address



Outline

- Communication at the transport layer
- Transport-layer services
  - Flow Control
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

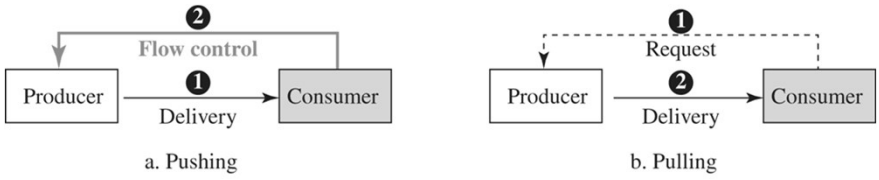
Flow Control

- Whenever an entity produces items and another entity consumes them, there should be a balance between production and consumption rates
  - If the items are produced faster than they can be consumed, the consumer can be overwhelmed and may need to discard some items
  - If the items are produced more slowly than they can be consumed, the consumer must wait, and the system becomes less efficient
- Flow control is related to the first issue to prevent losing data items at the consumer site

Flow control in Layer 4 ensures the delivery of the message end-to-end, whereas Layer 2 delivers message to next node

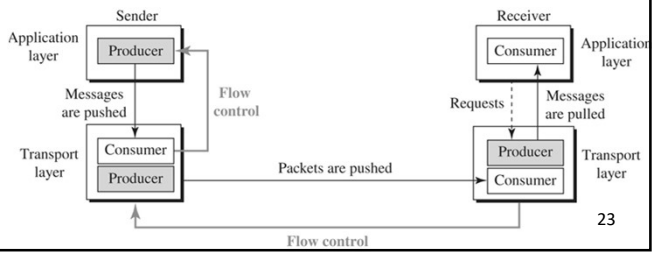
Pushing vs. Pulling

- Delivery of items from a producer to a consumer can occur in one of two ways:
  - Pushing: producer delivers items whenever they are produced without a prior request from the consumer
    - Need flow control in the opposite direction
  - Pulling: producer delivers the items after the consumer has requested them
    - No need for flow control



Handling Flow Control

- Transport layer deals with four entities:
  - Sender process: only a producer
    - Produces message chunks and pushes them to the transport layer
  - Sender transport layer: both a consumer and a producer
    - Consumes the messages pushed by the producer, encapsulates the messages in packets and pushes them to the receiving transport layer
  - Receiver transport layer: both a consumer and a producer
    - Consumes the packets received from the sender, decapsulates the messages and delivers them to the application layer
  - Receiver process: only a consumer
    - Consumes message pulled from the transport layer

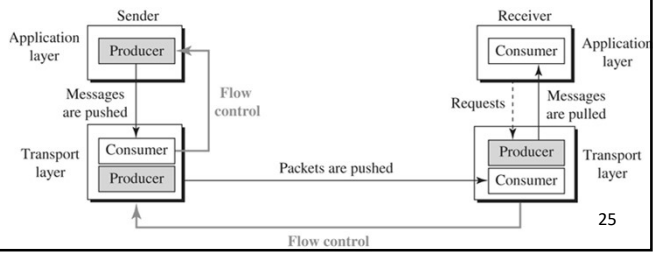


Buffers

- Flow control requires the use of two buffers to hold packets; one at the sender site and the other at the receiver site
- Consumers communicate with the producers on two occasions:
  - Buffer of sending transport layer:
    - Full: ask the application layer to stop passing chunks of messages
    - Vacancies: inform the application that it may pass messages
  - Buffer of receiving transport layer:
    - Full: ask sending transport layer to stop sending packets
    - Vacancies: inform the sending transport layer that it may send packets

Example: Buffers

- Scenario: A transport layer uses a buffer with only one slot
  - When slot in sending transport layer is empty, send a note to the application layer to send its next chunk
  - When slot in receiving transport layer is empty, send an acknowledgment to the sending transport layer to send its next packet
  - As we will see later, this type of flow control using a single-slot buffer is inefficient

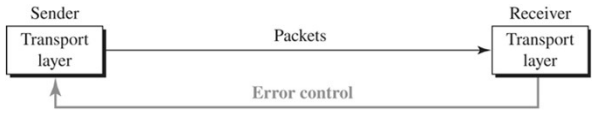


Outline

- Communication at the transport layer
- Transport-layer services
  - Error Control
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

Error Control

- In the Internet, network layer (IP) is unreliable, we need to make the transport layer reliable if the application requires reliability
- Error control services at the transport layer is responsible for:
  - Detecting and discarding corrupted packets
  - Keeping track of lost and discarded packets and resending them
  - Recognizing duplicate packets and discarding them
  - Buffering out-of-order packets until the missing packets arrive
- Error control requires the use of sequence and acknowledgment numbers by both sides



Acknowledgments

- Receiving transport layer needs to send an ACK for each packet that arrived without error
  - or a collection of packets
- Receiver silently discards corrupted packets
- Sending transport layer starts a timer when it sends a packet
- Sender knows a packet was lost or corrupted when it does not receive an ACK before the timer expires

Sequence Numbers

- In error control,
  - sending transport layer needs to know which packet to retransmit
  - receiving transport layer needs to know which packet is a duplicate or out-of-order
- Sequence number is added to transport-layer packets
  - When a packet is lost or corrupted, receiver can inform sender to resend it
  - Receiver can reorganize out-of-order packets
  - When two packets have the same sequence number, receiver knows it is a duplicate and silently drops it

Outline

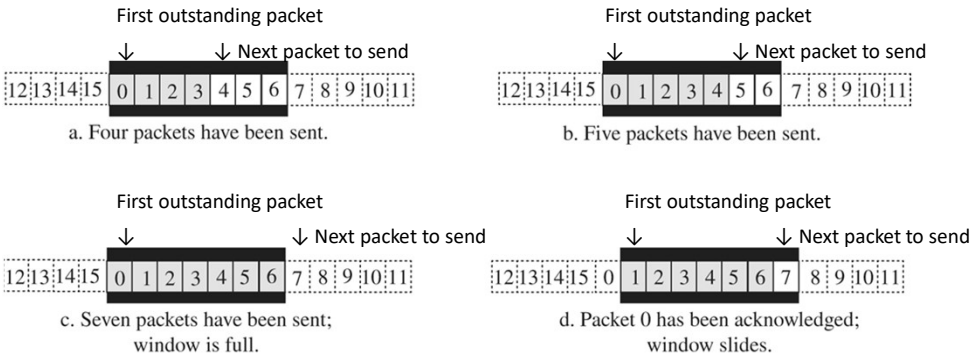
- Communication at the transport layer
- Transport-layer services
  - Combination of Flow and Error Control
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

Combination of Flow and Error Control

- Flow and error control can be combined if we use two numbered buffers, one at the sender and one at the receiver
- Sender:
  - A packet is ready to be sent, use the number of next free location in the buffer as sequence number
  - A packet is sent, store a copy in the same location waiting for an ACK
  - When an ACK related to the packet arrives, purge and free memory location
- Receiver:
  - When a packet arrives, store in a buffer at location of sequence number waiting for pull request from application layer and send an ACK

Example: Sliding Window

- Sequence numbers are usually a power of 2 and calculations are modulo 2^m
- Scenario: Sending transport layer uses a sliding window of size 7



Outline

- Communication at the transport layer
- Transport-layer services
  - Congestion Control
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

Congestion Control

- Congestion in a network occurs when the load on the network (i.e., the number of packets sent to the network) is greater than the capacity of the network (i.e., the number of packets a network can handle)
  - Long delays
  - Packet loss
- Congestion control refers to the mechanisms and techniques that control the congestion and keep the load below the capacity

Flow control: one sender, too fast for one receiver

Congestion control: too many senders, sending too fast

Outline

- Communication at the transport layer
- Transport-layer services
  - Connectionless and connection-oriented protocols
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

Connectionless and Connection-Oriented Protocol

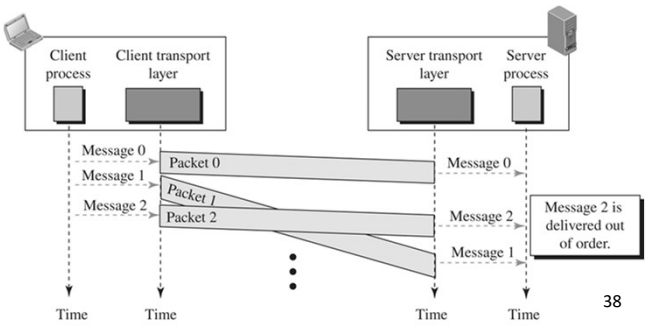
- A transport-layer protocol can provide two types of services:
  - Connectionless service: means independency between packets
  - Connection-oriented service: means dependency between packets
- Note:
  - Connectionless at network layer may mean different path for different datagrams belonging to the same message (i.e., physical path)
  - At transport layer, we are not concerned about the physical path

Connectionless Service

- Source process divides its message into chunks of data of the size acceptable by the transport layer and deliver them to the transport layer one by one
- Transport layer treats each chunk as a single unit without any relation between the chunks
- When a chunk arrives from the application layer, the transport layer encapsulates it in a packet and sends it
- Packets may arrive at transport layer out-of-order and delivered to server process as is

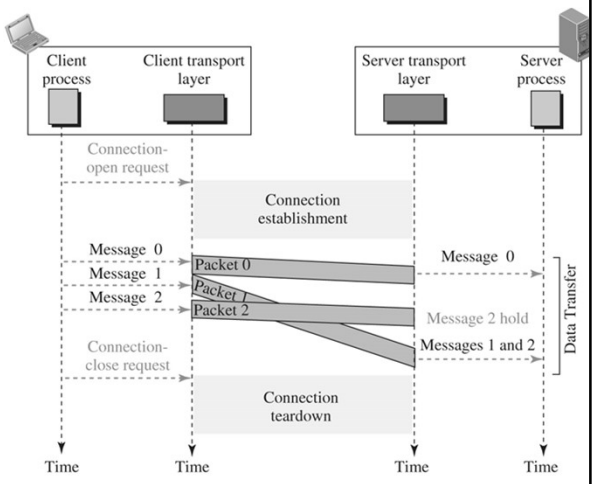
Connectionless Service (cont.)

- When packets delivered to the server out-of-order:
  - If the three packets belong to the same message, the server would not know
  - If a packet is lost, the server would not know
- Due to the lack of coordination, no flow control, error control, or congestion control can be implemented



Connection-Oriented Service

- The client and the server first need to establish a logical connection between themselves
- Data exchange can only happen after the connection establishment
- After data exchange, the connection needs to be torn down



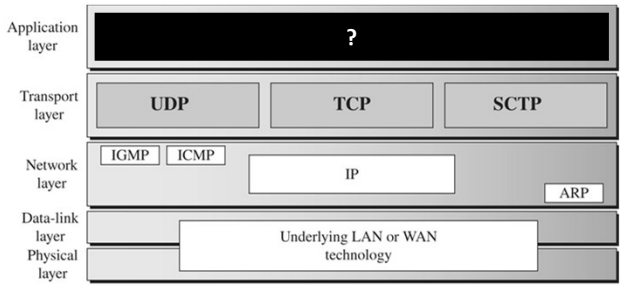
- Note:
  - Service may be connectionless or connection-oriented at the network layer

Outline

- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

Transport-layer Protocols

- Now, we will discuss three protocols in the TCP/IP protocol suite used in the Internet today:
  - User Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)



Transport-layer Protocols (cont.)

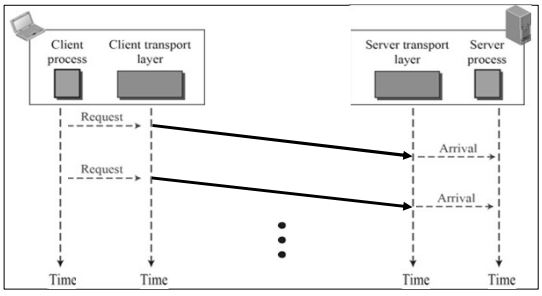
- UDP: User datagram protocol
  - Unreliable connectionless protocol
  - Simple and efficient where error control can be done in the application layer
- TCP: transmission control protocol
  - Reliable connection-oriented protocol
  - Used in any application where reliability and in-order delivery are important, i.e., congestion, error, and flow control
- SCTP: stream control transmission protocol
  - New transport-layer protocol that combines the features of UDP and TCP
- Note:
  - Services not available: delay or bandwidth guarantees

Transport-layer Protocols (cont.)

- To better understand the behavior of these protocols, we will start with a very simple one
- Then, incrementally add more complexity to develop both sender and receiver sides
- The TCP/IP protocols use a transport-layer protocol that is either a modification or a combination of some of the following protocols:
  - Simple Protocol
  - Stop-and-Wait Protocol
  - Go-Back-N Protocol
  - Selective-Repeat Protocol

Simple Protocol

- Let's start with a simple connectionless protocol with neither flow nor error control and assume no bit errors or packet loss
- Sender:
  - Sends packets one after another without thinking about the receiver
- Receiver:
  - Can never be overwhelmed with incoming packets

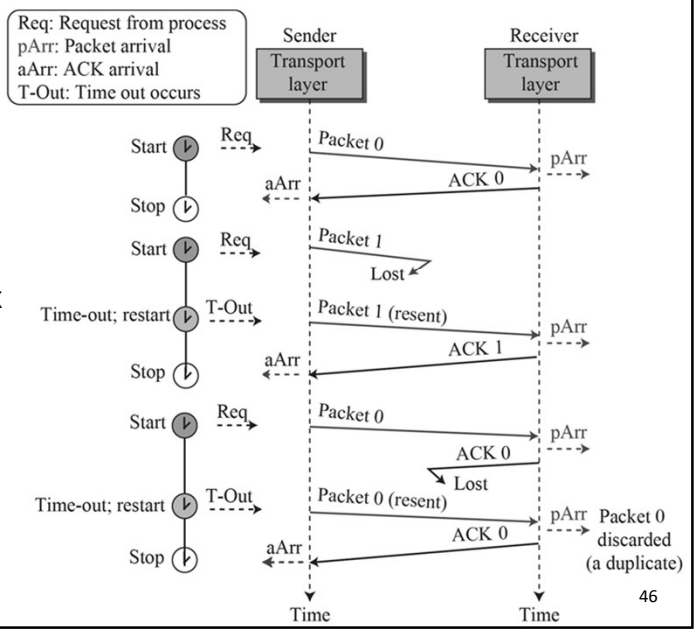


Stop-and-Wait Protocol

- What if channel is not perfectly reliable (e.g., errors and packet loss)?
- Our second protocol is a connection-oriented which uses both flow and error control
- Sender:
  - Adds a checksum to each packet to detect corrupted packets at the receiver
  - Sends one packet at a time
  - Waits for an ACK before sending the next one; if timeout, retransmit
- Receiver: Checks the packet upon reception
  - If the checksum is correct, send an ACK
  - If the checksum is incorrect, the packet is corrupted and silently discarded

Example: Stop-and-Wait Protocol

- Packet 0: sent + ACK
- Packet 1: lost and resent after the time-out
- Packet 1: resent + ACK
- Packet 0: sent, but ACK is lost, resent after the time-out
  - Sender does not know if packet or ACK is lost
- Packet 0: resent + ACK
- Flaw:
  Sender does not know what happened at receiver, possible duplicate, handled with sequence number

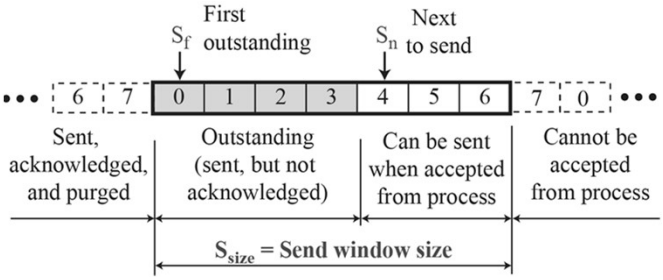


Go-Back-N Protocol

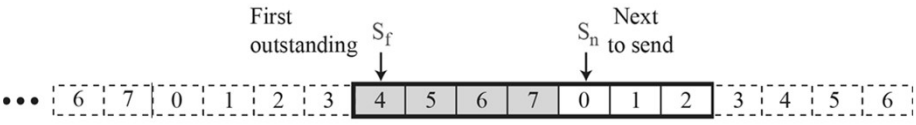
- Channel is not efficiently utilized (not busy enough!)
- To improve the efficiency of transmission, multiple packets may be in transition while the sender is waiting for acknowledgment
- Sender:
  - Sends a window of up to N consecutive packets
  - Upon receiving ACK *n*, move window forward to begin at *n+1*
  - Timeout*(n):* go back and retransmit starting from packet *n* and up
- Receiver: Checks each packet upon reception
  - Sends ACK for correctly-received packet, with highest in-order sequence number
  - That is, on receipt of out-of-order packet, resend ACK with highest in-order sequence number

Example: Sliding Window

- Before sliding:

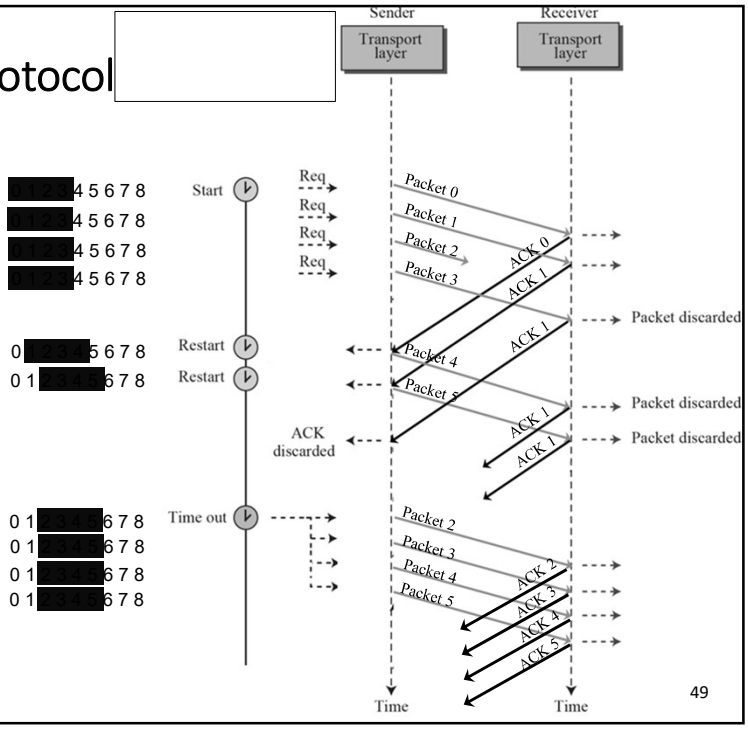


- After sliding: received an ACK with sequence number 3



Example: Go-Back-N Protocol

- Window size $N = 4$



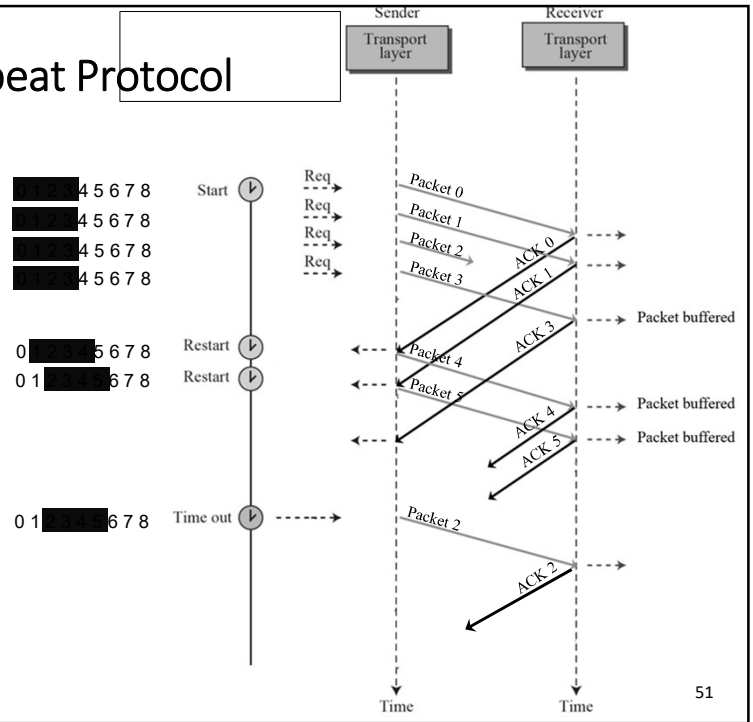
- Flaw:
  Each time a single packet is lost or corrupted, sender resends all outstanding packets, although some packets may have been received safe but out of order

Selective-Repeat Protocol

- Channel is still not efficiently utilized by Go-Back-N (many duplicates!)
- To improve the efficiency of transmission, individually acknowledge each packet
- Sender:
  - Sends a window of up to N consecutive packets
  - Upon receiving ACK *n*, mark packet *n* as received move window forward to begin at *n+1*
  - Timeout*(n):* go back and retransmit unACKed packet *n* only
- Receiver: Checks each packet upon reception
  - Sends ACK for each correctly-received packet
  - Buffer out-of-order packets, deliver upon receiving missing packet

Example: Selective-Repeat Protocol

- Window size $N = 4$
- In the last step, receiver delivers packets 2-5
- What happens after sender receives ACK 2?



Outline

- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

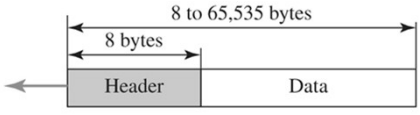
RFC 768

User Dadagram Protocol (UDP)

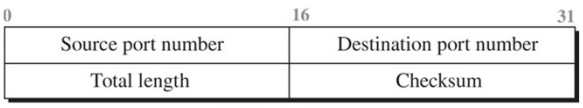
- Connectionless transport protocol:
  - no handshaking between UDP sender and receiver
  - each UDP segment handled independently of others
- Unreliable (best effort) transport protocol:
  - UDP segments may be lost or delivered out-of-order to application
- Does not add anything to the IP services, besides providing process-to-process communication
- If UDP is so powerless, why would a process want to use it?
  - very simple: use a minimum of overhead, no flow or error control
  - fast: no connection establishment, no congestion control
  - Good for apps such as streaming multimedia (less tolerant and rate sensitive)

UDP Packet Format

- UDP packets, called user datagrams, have a fixed-size header of 8 bytes made of four fields, each of 2 bytes (16 bits)
  - First two fields define the source and destination port numbers
  - Third field defines the total length of the user datagram (header plus data)
    - Total length of 0 to 65,535 bytes
    - Actual length is less because UDP user datagram is encapsulated in an IP datagram with the total length of 65,535 bytes
  - Last field can carry an optional checksum to detect errors in a segment



a. UDP user datagram



b. Header format

Example: User Datagram

The following is the contents of a UDP header in hexadecimal format:

CB84000D001C001C

a. What is the source port number?
b. What is the destination port number?
c. What is the total length of the user datagram?
d. What is the length of the data?
e. Is the packet directed from a client to a server or vice versa?
f. What is the client process?

Solution:

- The source port number is the first four hexadecimal digits $(CB84)_{16}$ or 52100
- The destination port number is the second four hexadecimal digits $(000D)_{16}$ or 13
- The third four hexadecimal digits $(001C)_{16}$ define the length of the whole UDP packet as 28 bytes

Example: User Datagram (cont.)

The following is the contents of a UDP header in hexadecimal format:

CB84000D001C001C

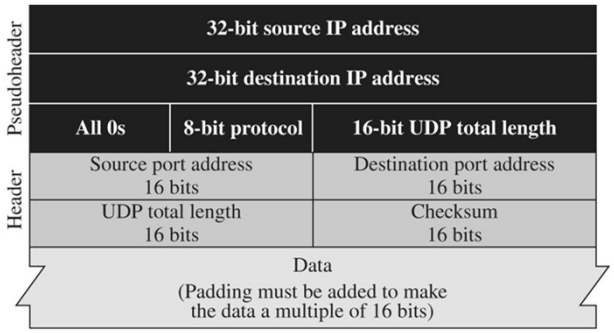
a. What is the source port number?
b. What is the destination port number?
c. What is the total length of the user datagram?
d. What is the length of the data?
e. Is the packet directed from a client to a server or vice versa?
f. What is the client process?

Solution:

- The length of the data is the length of the whole packet minus the length of the header, or 28 − 8 = 20 bytes
- Since destination port number is 13, the packet is from the client to the server
- The client process is the Daytime

Checksum

- UDP checksum calculation includes three sections (words of 16 bits):
  - A pseudo-header: the part of the header of the IP packet in which the user datagram is to be encapsulated with some fields filled with 0s
  - UDP header
  - Data coming from the application layer



UDP Services

- UDP provides process-to-process communication using socket addresses; a combination of IP addresses and port numbers
- UDP service is connectionless:
  - Each user datagram sent by UDP is an independent datagram
  - No relationship between the different user datagrams even if they are coming from the same source process and going to the same destination program
  - No connection establishment or termination
  - User datagrams are not numbered
- There is no flow control, and hence no window mechanism
  - The receiver may overflow with incoming messages

UDP Services (cont.)

- There is no error control, except for the checksum
  - The sender does not know if a message has been lost or duplicated
  - If the receiver detects an error using checksum, datagram is silently discarded
- There is no congestion control
  - UDP assumes that packets sent are small and sporadic and cannot create congestion in the network
  - This assumption may or may not be true today, when UDP is used for interactive real-time transfer of audio and video
- To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages

UDP Services (cont.)

- UDP multiplexes and demultiplexes when there are several processes that may want to use its services
- Queueing:
  - In UDP, queues are associated with ports
  - At the client site, when a process starts, it requests a port number from the operating system
  - Some implementations create both an incoming and an outgoing queue associated with each process
  - Other implementations create only an incoming queue associated with each process

UDP vs. Simple Protocol

- The only difference is that UDP provides an optional checksum to detect corrupted packets at the receiver site
- If the checksum is added to the packet, the receiving UDP can check the packet and discard the packet if it is corrupted
  - No feedback, however, is sent to the sender

Typical UDP Applications

- UDP is suitable for:
  - A process that requires simple request-response communication with little concern for flow and error control
  - A process with internal flow- and error-control mechanisms
  - Multicasting
- UDP is used for:
  - Management processes such as SNMP
  - Some route updating protocols such as Routing Information Protocol (RIP)
  - Interactive real-time applications that cannot tolerate uneven delay between sections of a received message
- We briefly discuss some features of UDP and their advantages and disadvantages in the following examples

Example 1: Domain Name System (DNS)

- A client-server application such as DNS uses the services of UDP because a client needs to send a short request to a server and to receive a quick response from it
- The request and response can each fit in one user datagram
- Since only one message is exchanged in each direction, the connectionless feature is not an issue
  - Client and server do not worry that messages are delivered out of order

Example 2: Simple Mail Transfer Protocol (SMTP)

- A client-server application such as SMTP, which is used in electronic mail, cannot use the services of UDP
  - For example, a user may send a long e-mail message, which may include multimedia (images, audio, or video)
  - If the application uses UDP and the message does not fit in one single user datagram, the message must be split by the application into multiple user datagrams
  - Here, the connectionless service may create problems
    - User datagrams may arrive and be delivered to the receiver application out of order
    - The receiver application may not be able to reorder the pieces
  - Thus, a connectionless service has a disadvantage for an application program that sends long messages

Example 3: File Downloading

- Assume we are downloading a very large text file from the Internet
- We need to use a transport layer that provides reliable service
  - We don't want part of the file to be missing or corrupted when we open the file
- The delay created between the deliveries of the parts is not an overriding concern for us
  - We can wait until the whole file is composed before looking at it
- In this case, UDP is not a suitable transport layer

Example 4: Real-time Interactive Applications

- Assume we are using a real-time MS Teams
- Audio and video are divided into frames and sent one after another
- If the transport layer is supposed to resend a corrupted or lost frame, the synchronizing of the whole transmission may be lost
  - The viewer suddenly sees a blank screen and needs to wait until the second transmission arrives
  - This is not tolerable!
- However, if each small part of the screen is sent using one single user datagram, the receiving UDP can easily ignore the corrupted or lost packet and deliver the rest to the application program
  - That part of the screen is blank for a very short period of time, which most viewers do not even notice

Outline

- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

RFCs: 793, 1122, 2018, 5681, 7323

Transmission Control Protocol (TCP)

- Connection-oriented transport protocol:
  - Handshaking: defines connection establishment, data transfer, and connection teardown phases
- Reliable transport protocol:
  - Uses checksum for error detection
  - In-order message delivery to application
  - Uses a combination of Go-Back-N and Selective-Repeat protocols to provide reliability

TCP Services

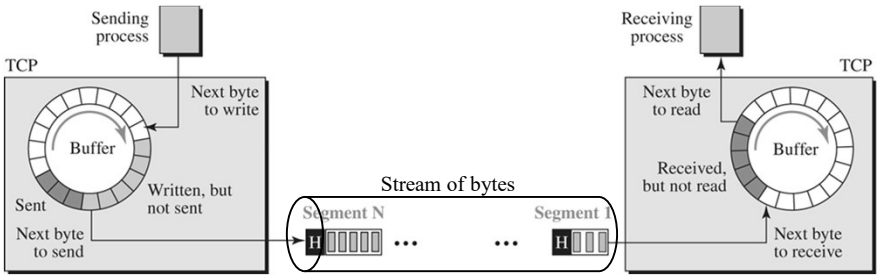
- TCP provides process-to-process communication using socket addresses, a combination of IP addresses and port numbers
- Full-duplex service: data can flow in both directions at the same time
  - Each TCP endpoint has its own sending and receiving buffer, and segments move in both directions
- Like UDP, TCP performs multiplexing at the sender and demultiplexing at the receiver
  - However, since TCP is a connection-oriented protocol, a connection needs to be established for each pair of processes
- Encapsulation and decapsulation

TCP Services (cont.)

- Unlike UDP, TCP service is connection-oriented:
  - When a process at site A wants to send to and receive data from another process at site B, the following three phases occur:
    - The two TCP's establish a logical connection
    - Data are exchanged in both directions
    - The connection is terminated
  - Note that TCP segments are encapsulated in IP datagrams that can be sent out of order, lost, corrupted, and routed over different paths to destination
- Reliable service: TCP uses an acknowledgment mechanism to check the safe and sound arrival of data
  - Flow and error control

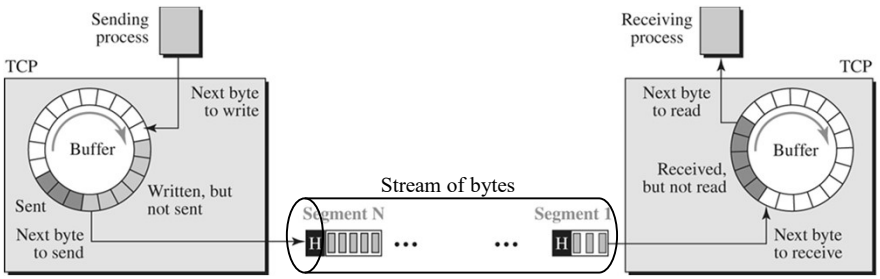
TCP Services (cont.)

- Stream delivery service: TCP allows a process to deliver (and obtain) data as a stream of bytes (byte-oriented approach)
  - TCP creates an environment in which the two processes seem to be connected by an imaginary "tube" that carries their bytes across the Internet
  - The sending process produces (writes to) the stream and the receiving process consumes (reads from) it



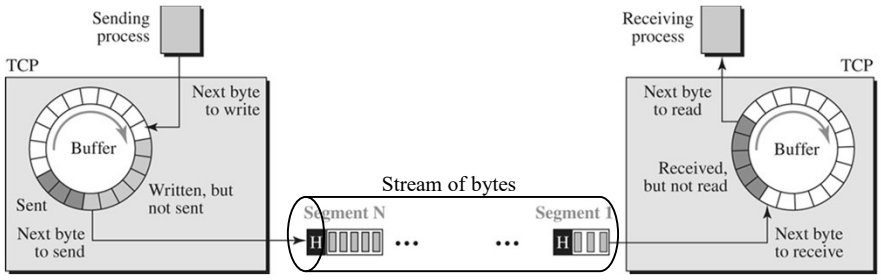
TCP Services (cont.)

- Stream delivery service: TCP allows a process to deliver (and obtain) data as a stream of bytes (byte-oriented approach)
  - Two buffers are used for sending and receiving (usually 1000s of bytes)
  - TCP groups a number of bytes into a segment, adds header, and send to network layer for transmission
  - Segments are not necessarily the same size



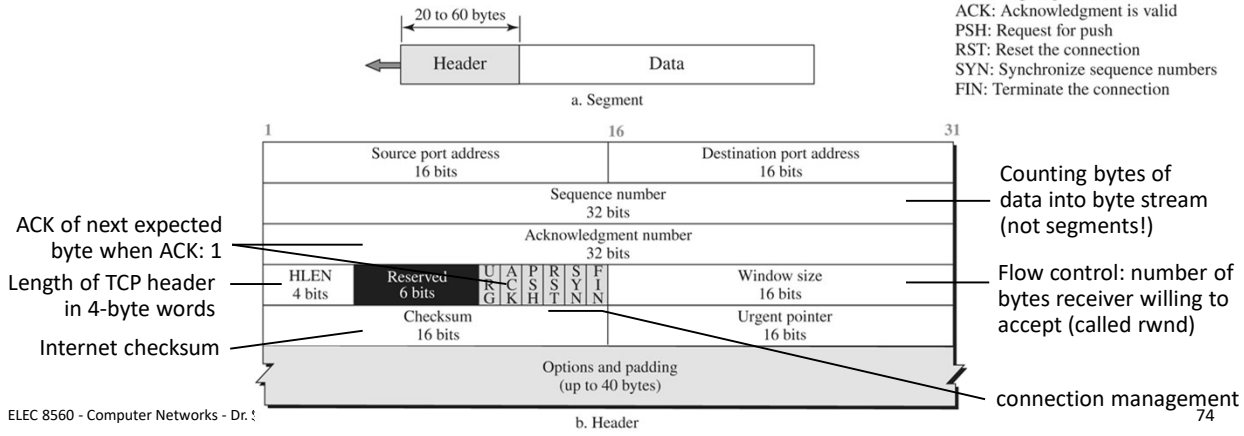
TCP Services (cont.)

- Stream delivery service: TCP allows a process to deliver (and obtain) data as a stream of bytes (byte-oriented approach)
  - This is different from the message-oriented approach of UDP where message boundaries are preserved
    - e.g., if an application sends 100-byte message using UDP, the peer application will receive all 100 bytes in a single read



TCP Packet Format

- A packet in TCP is called a segment
- The segment consists of a 20-60 byte header (based on options) followed by data from the application program



Sequence Number

- A counter used to keep track of every byte sent outward by a host
  - That is, it defines the number assigned to the first byte of data contained in the segment
- The sequence number of a segment is the sequence number of previous segment plus the number of bytes carried by the previous segment
  - For example, if a TCP segment contains 1400 bytes of data, then the sequence number will be increased by 1400 after the packet is transmitted
- The initial sequence number (ISN) is set randomly for the first segment during connection establishment

Acknowledgement number

- A counter to keep track of every byte that has been received
  - That is, it defines the number of the next byte the receiver is expecting to receive
  - For example, if 1000 bytes are received by a host, it increases the acknowledgement number by 1000
- Acknowledgment number is cumulative (all previous bytes received)
- Piggybacking:
  - When the host sends out its next packet, it will send this updated acknowledgement number
  - It will also turn on the ACK flag to indicate to the other end that it is acknowledging the receipt of data

Example: TCP Segment

Suppose a TCP connection is transferring a file of 5,000 bytes. The first byte is numbered 10,001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1,000 bytes?

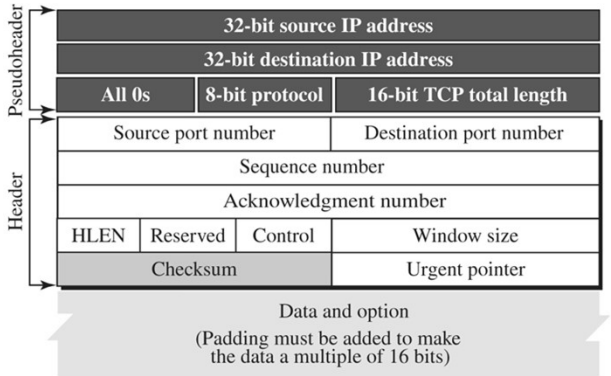
Solution:

- The following shows the sequence number for each segment:

Segment 1	→	**Sequence number:**	10,001	**Range:**	10,001	to	11,000
Segment 2	→	**Sequence number:**	11,001	**Range:**	11,001	to	12,000
Segment 3	→	**Sequence number:**	12,001	**Range:**	12,001	to	13,000
Segment 4	→	**Sequence number:**	13,001	**Range:**	13,001	to	14,000
Segment 5	→	**Sequence number:**	14,001	**Range:**	14,001	to	15,000

Checksum

- Unlike UDP, TCP checksum is mandatory
- TCP checksum calculation is similar to UDP including a pseudo-header, TCP header, and data from the application layer



Outline

- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
    - TCP connection
  - Stream Control Transmission Protocol (SCTP)

A TCP Connection

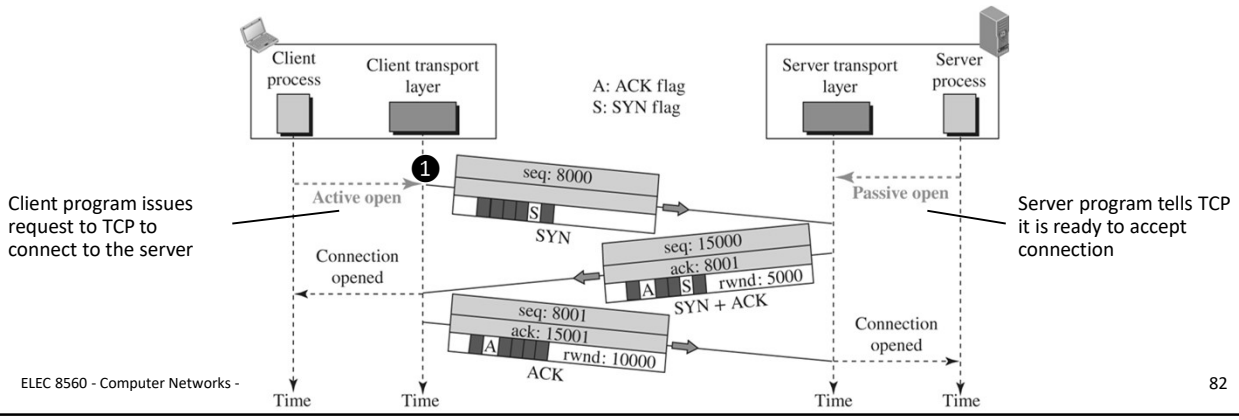
- TCP is connection-oriented: all of the segments belonging to a message are then sent over the same logical path
  - Facilitates the acknowledgment and retransmission of damaged or lost frames
- Does it make sense that TCP can be connection-oriented while using the services of IP which is a connectionless protocol?
  - The point is that a TCP connection is logical, not physical
  - TCP operates at a higher level
  - TCP uses the services of IP to deliver individual segments to the receiver, but it controls the connection itself
    - IP is unaware of TCP retransmissions
    - IP is unaware of TCP holding and reordering out-of-order segments

Connection Establishment

- TCP transmits data in full-duplex mode
  - When two TCPs in two machines are connected, they are able to send segments to each other simultaneously
  - This implies that each party must initialize communication and get approval from the other party before any data are transferred

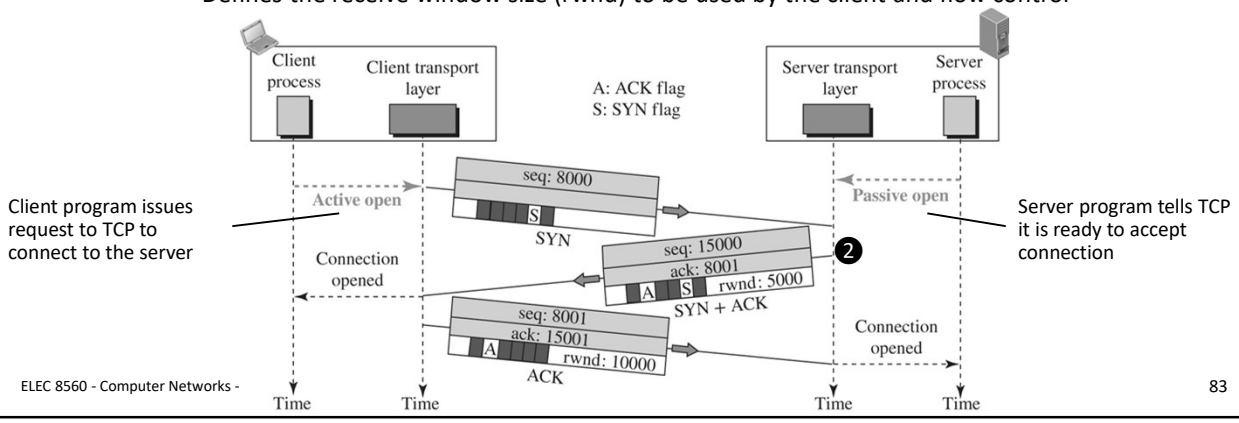
Connection Establishment (cont.)

- Three-way handshaking:
  - ❶ Synchronize sequence number
    - SYN flag is set and sequence number is initialized randomly (ISN)



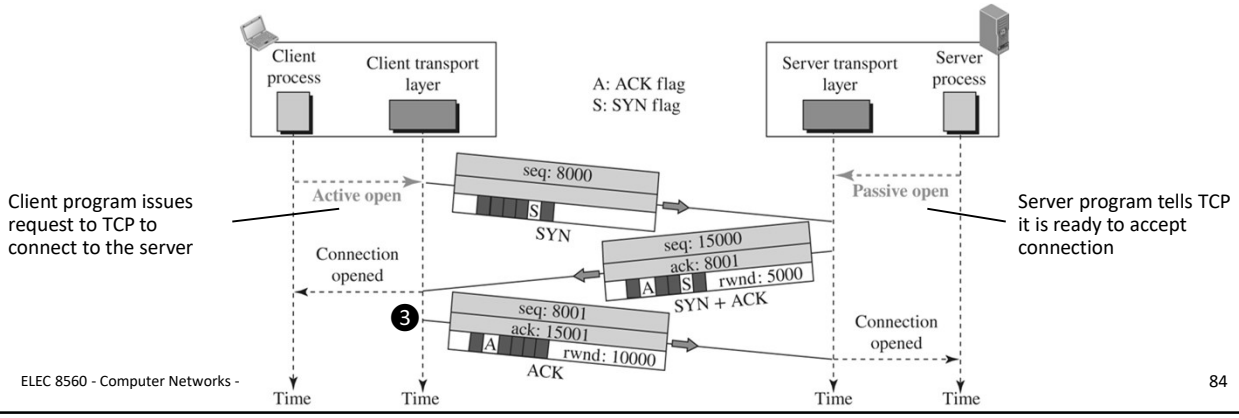
Connection Establishment (cont.)

- Three-way handshaking:
  - ❷ Synchronize sequence number in the other direction and acknowledge receiving SYN from the client
    - SYN and ACK flags are set and sequence number is initialized randomly (ISN)
    - Defines the receive window size (rwnd) to be used by the client and flow control



Connection Establishment (cont.)

- Three-way handshaking:
  - ❸ Acknowledge receiving SYN from the server
    - ACK flags is set
    - Defines the receive window size (rwnd) to be used by the server and flow control

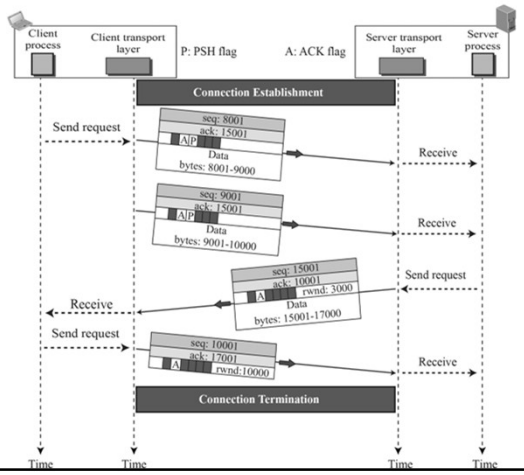


SYN Flooding Attack

- An example of a denial of service attack
- Malicious attackers may send a large number of SYN segments to a server with fake IP addresses
- Server allocate resources and sends SYN+ACK segments until eventually runs out of resources and become unable to accept connection requests from valid clients
- Some solutions:
  - Limit number of requests for a period of time
  - Postpone resource allocation until verifying connection requests

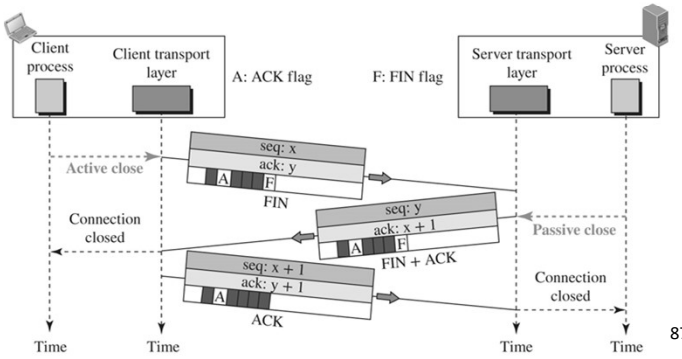
Data Transfer

- Connection established, bidirectional data transfer can take place
  - The client and server can send data and acknowledgments in both directions
  - The acknowledgment is piggybacked with the data



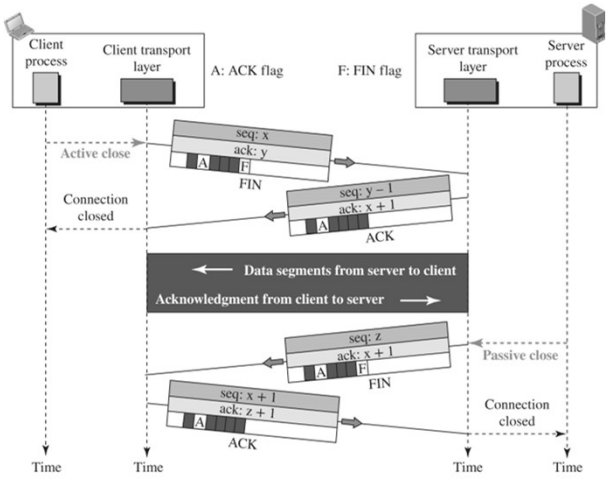
Connection Termination

- Either of the two parties involved in exchanging data (client or server) can close the connection (usually initiated by the client)
- Option 1: Three-way handshaking
  - FIN segment (may carry the last chunk of data)
  - FIN+ACK acknowledge receipt and announces closing the other direction (may carry the last chunk of data)
  - ACK last chunk and FIN



Connection Termination

- Either of the two parties involved in exchanging data (client or server) can close the connection (usually initiated by the client)
- Option 2: Half-Close
  - One end can stop sending data while still receiving data



Connection Reset

- TCP at one end may deny a connection request, may abort an existing connection, or may terminate an idle connection
- All of these are done with the RST (reset) flag

Notes on Sequence Numbers

- A SYN segment cannot carry data, but it consumes one sequence number
- An ACK segment, if carrying no data, consumes no sequence number
- A SYN + ACK segment cannot carry data, but does consume one sequence number
- The FIN segment consumes one sequence number if it does not carry data
- The FIN + ACK segment consumes one sequence number if it does not carry data

Outline

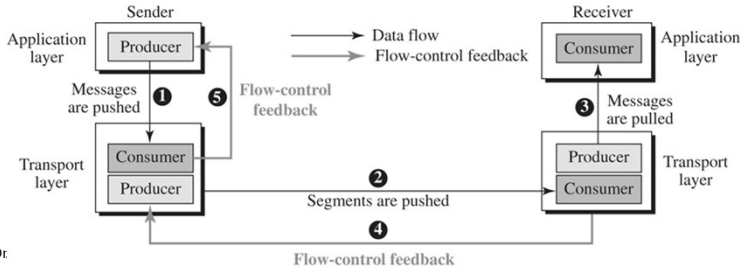
- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
    - Flow control
  - Stream Control Transmission Protocol (SCTP)

Windows in TCP

- TCP uses two windows for each direction of data transfer
- Send window size is dictated by the receiver (flow control) and traffic in the underlying network (congestion control)
- Similar to Selective-Repeat (SR) protocol windows with differences:
  - Window size is in bytes, compared to packets in SR
  - TCP allows receiving process to pull data at its own pace, i.e., part of the buffer is occupied by ACKed data waiting to be pulled
  - ACKs in TCP are cumulative, compared to selective in SR

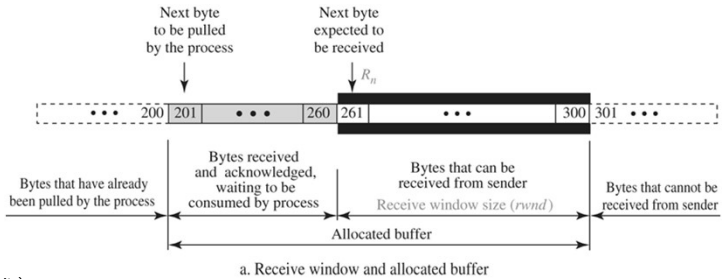
Flow Control

- Flow control balances the rate a producer creates data with the rate a consumer can use the data
- TCP does not provide flow-control feedback from the receiving process to the receiving TCP
  - Receiving process pulls data from receiving TCP whenever it is ready



Flow Control (cont.)

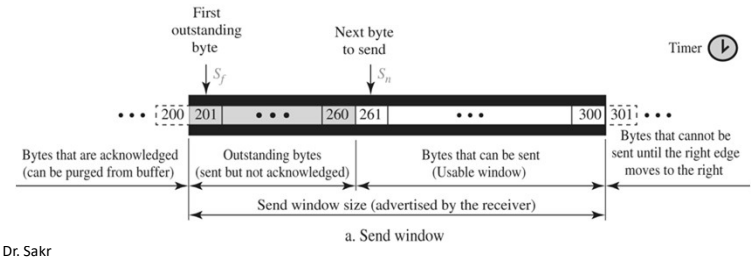
- Initially, the size of the buffers is fixed when connection is established
- Upon packets reception
  - Receive window closes (moves its left wall to the right) when more bytes arrive from the sender
  - Receive window opens (moves its right wall to the right) when more bytes are pulled by the process



a. Receive window and allocated buffer

Flow Control (cont.)

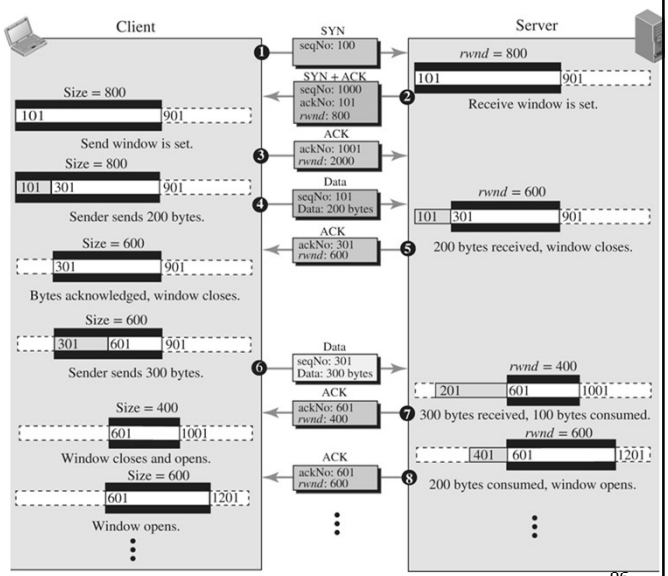
- To achieve flow control, TCP forces the sender to adjust its window size, controlled by the receiver
  - Send window closes (moves its left wall to the right) when bytes are acknowledged by the receiver
  - Send window opens (moves its right wall to the right) when window size (rwnd) advertised by the receiver allows
  - Send window can shrink (moves its right wall to the left) otherwise



a. Send window

Example: Flow Control

- Scenario: Simple unidirectional data transfer (client to server)
  - Only one window at each side



Outline

- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
    - Error control
  - Stream Control Transmission Protocol (SCTP)

Error Control

- TCP is a reliable transport-layer protocol
  - This means application programs can rely on TCP to deliver the entire stream of data to the application program on the other end in order, without error, and without any part lost or duplicated
- To achieve reliability, TCP uses three tools:
  - Checksum
  - Acknowledgment
  - Time-out

Checksum

- 16-bit checksum that is mandatory in every segment
- Used to check for corrupted segments
- If a segment is corrupted, segment is discarded by receiver and considered lost

Acknowledgments

- Confirm the receipt of data segments
- Acknowledgments are cumulative (similar to Go-Back-N protocol)
  - Receiver advertises next byte it expects to receive ignoring all segments received and stored out-of-order
- Some implementations use selective acknowledgment (SACK)
  - Does not replace an ACK
  - Reports a block of bytes that is out of order or duplicated
  - Implemented in the Options field
- ACK segments are never acknowledged

Acknowledgments (cont.)

- Rules to generate acknowledgments:
  1. When A sends segment to B, it must include (piggyback) an ACK with the next sequence number it expects to receive
  2. When a receiver receives an in-order segment but has not data to send and previous segment is already ACKed, it delays sending an ACK until another segment arrives or a preset ACK timer (usually 500 ms) has passed
  3. When a receiver receives an in-order segment but has no data to send and previous segment is not ACKed, send immediately
  4. When a receiver receives an out-of-order segment with higher sequence number, send an ACK with the next expected segment
  5. When a receiver receives a missing segment, send an ACK with the next expected segment
  6. When a receiver receives a duplicate segment, discard and send an ACK with the next expected segment

Retransmissions

- TCP uses one retransmission time-out (RTO) timer for each connection
- When a segment is sent, it is stored in a queue until it is ACKed
- When the retransmission timer expires, TCP resends the segment in the front of the queue and restarts the timer
- When the sender receives three duplicate ACKs for a segment, TCP retransmits the first segment in the queue without waiting for the time-out
  - This called fast retransmission

Out-of-order Segments

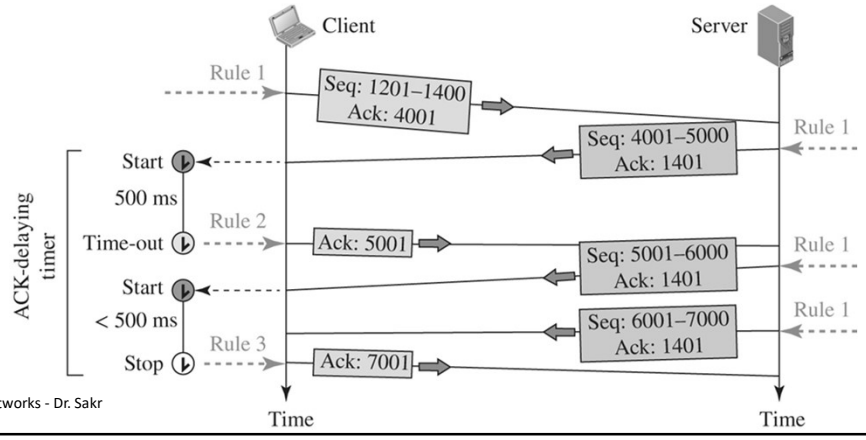
- TCP guarantees that data are delivered to the process in order
- TCP implementations today do not discard out-of-order segments
  - Similar to Selective-Repeat protocol
- Stored temporarily, flagged as out-of-order, until missing segments arrive
- Out-of-order segments are never delivered to the process

Some Scenarios

- Let's see some examples of scenarios that occur during the operation of TCP, considering only error control issues
- If the segment carries data, we show the range of byte numbers and the value of the acknowledgment field
- If it carries only an acknowledgment, we show only the acknowledgment number

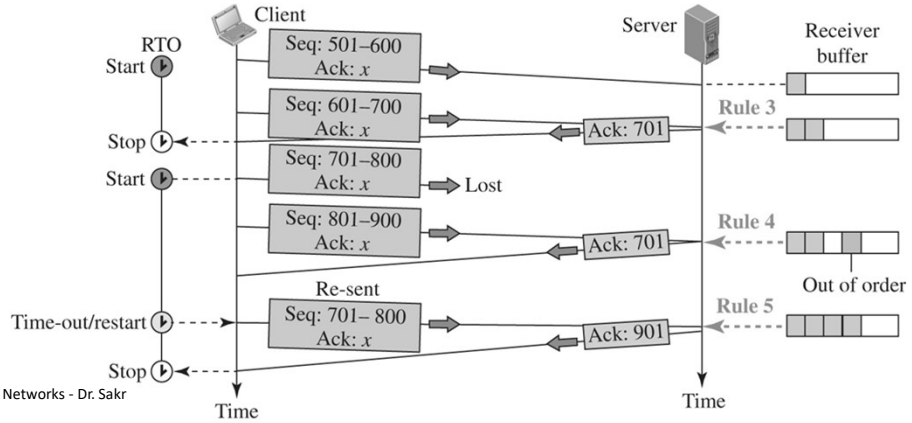
Scenario 1: Normal Operation

- Note:
  - server piggyback ACK with data
  - client TCP waits up to 500ms for next segment, if no next segment, send ACK
  - client sends an ACK immediately after receiving second in-order segment



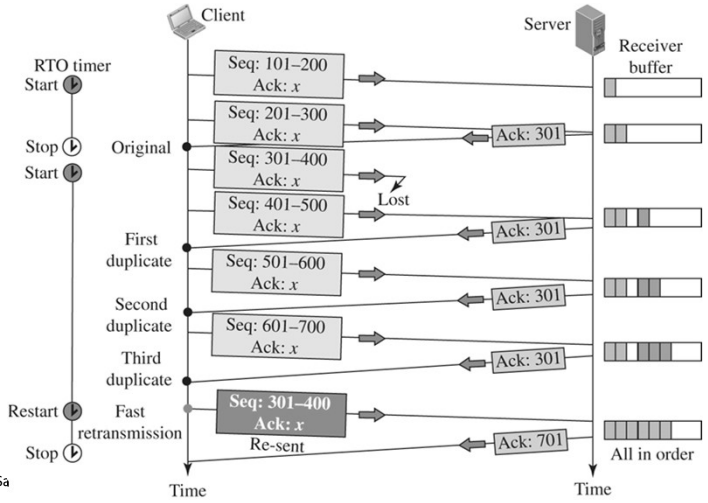
Scenario 2: Lost Segment

- Note:
  - sender retransmits first segment in the queue when timer expires
  - receiver gets an out-of-order segment, send an ACK with expected segment
  - receiver gets a missing segment, send an ACK with expected segment



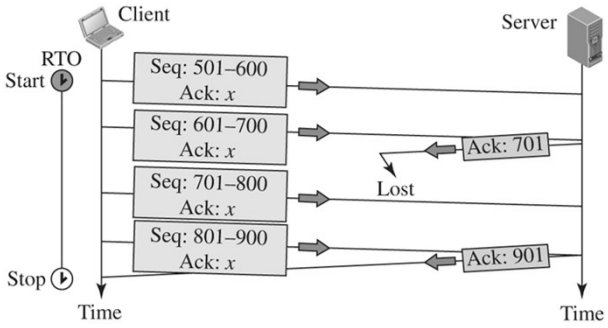
Scenario 3: Fast Retransmission

- Note:
  - sender retransmits first segment in queue when receives three duplicate ACKs



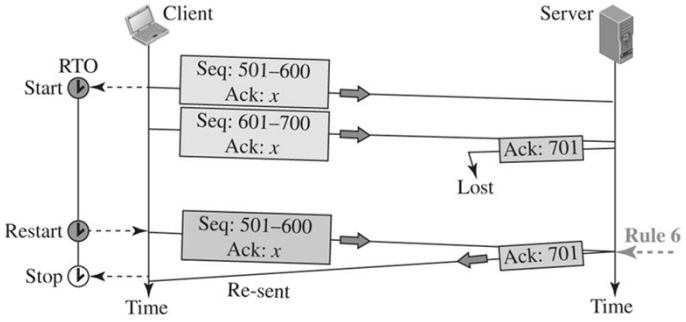
Scenario 4: Lost Acknowledgment

- Note:
  - sender does not need to retransmit as ACKs are cumulative



Scenario 5: Lost Acknowledgment

- Note:
  - sender retransmits first segment in the queue when timer expires before receiving an ACK
  - receiver receives a duplicate segment, discard and send an ACK with the next expected segment



Outline

- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
    - Congestion control
  - Stream Control Transmission Protocol (SCTP)

Congestion

- IP has no congestion control, so TCP needs to take care of it
- How can a TCP sender detect congestion?
- TCP sender uses the occurrence of two events as signs of congestion in the network:
  - Time-out:
    - If a TCP sender does not receive an ACK for a segment or a group of segments before the time-out occurs, it assumes that the loss of segments is due to congestion
  - Receiving three duplicate ACKs
    - If a TCP sender receives three duplicate ACKs for a segment, it assumes that the loss of segment is due to congestion

Congestion Window

- To control the number of segments a sender can transmit, TCP defines another window called a congestion window (*cwnd)*
- Maintained by the sender as a mean of stopping a link between the sender and receiver from becoming overloaded with too much traffic
- This is different from the sliding window (*rwnd*) maintained by the sender to prevent the receiver from becoming overloaded
  - Dictating the size of *rwnd* by the receiver guarantees that the receive window is never overflowed (no end congestion)
  - Does not mean that intermediate buffers in the routers do not become congested since a router may receive data from more than one sender

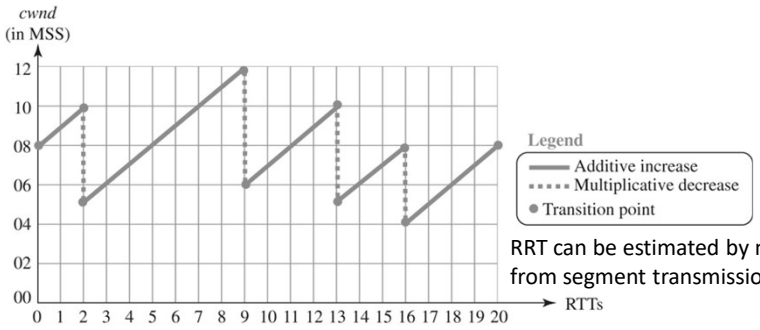
Congestion Window (cont.)

- *cwnd* is a multiple of the Maximum Segment Size (MSS), a parameter in TCP header specifies the largest amount of data (in bytes) in a single segment
- A sender can send data less than its own congestion window and the receive window

Actual window size = min(*rwnd*, *cwnd*)

Congestion Control

- Key idea: senders can increase sending rate until a packet loss (congestion) occurs, then decrease sending rate on loss event
- Additive Increase, Multiplicative Decrease (AIMD):
  - *cwnd* is dynamically adjusted in response to observed network congestion
  - Every RTT until loss detected: increase *cwnd* (e.g., by 1 MMS)
  - At each loss event: cut *cwnd* (e.g., in half or to 1 MSS)



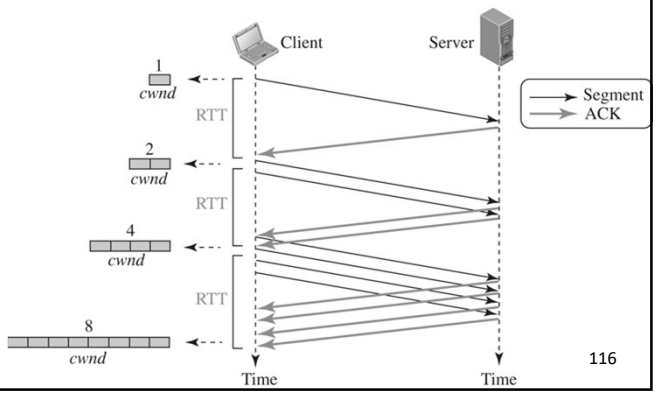
RRT can be estimated by measuring time from segment transmission until ACK receipt

Congestion Policies

- TCP uses three policies to handle congestion:
  - Slow start
  - Congestion avoidance
  - Fast recovery

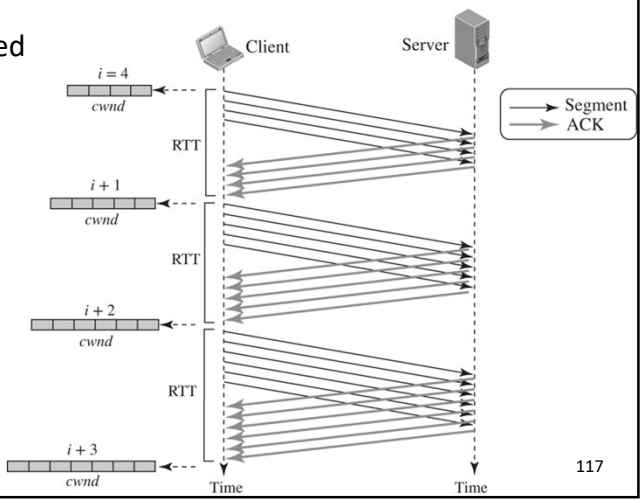
Slow Start, Exponential Increase

- Increase *cwnd* exponentially until first loss event:
  - initially *cwnd* = 1 MSS (Maximum Segment Size)
  - double *cwnd* every RTT (or increase by 1 for each ACK received)
  - increase until a packet loss is detected or reaches a threshold
    - If a loss event occurs, TCP assumes that it is due to network congestion and takes steps to reduce the load on the network
    - With exponential increase, rate ramps up fast, causing congestion, so increase until slow start threshold (*ssthresh*) is reached



Congestion Avoidance, Additive Increase

- To avoid congestion, switch to linear increase when *cwnd* is higher than threshold *ssthresh* to slow down transmissions
  - increase *cwnd* by 1 every RTT
  - increase until a packet loss is detected
    - If a loss event occurs, TCP assumes that it is due to network congestion and takes steps to reduce the load on the network



Fast Recovery

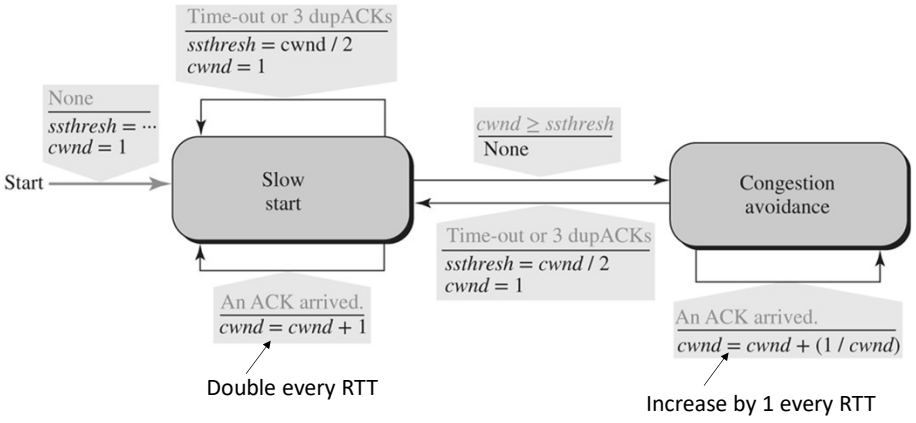
- To recover quickly from losses, if three duplicate ACKs are received
  - perform a fast retransmission (i.e., do not wait for the retransmission timer)
  - skip the slow start phase and go back to collision avoidance as soon as possible

Policy Transition

- Now the question is when each of these policies are used and when TCP moves from one policy to another
- To answer these questions, there are multiple versions of TCP:
  - TCP Tahoe
  - TCP Reno
  - TCP New Reno
  - TCP Vegas
  - and more

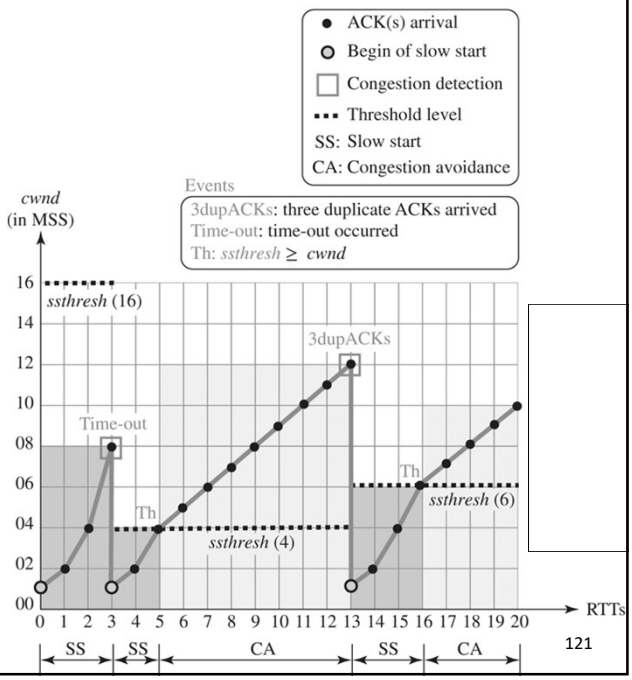
TCP Tahoe

- When a loss occurs, retransmit is sent, half of the current *cwnd* is saved as *ssthresh* and slow start begins again from its initial *cwnd*



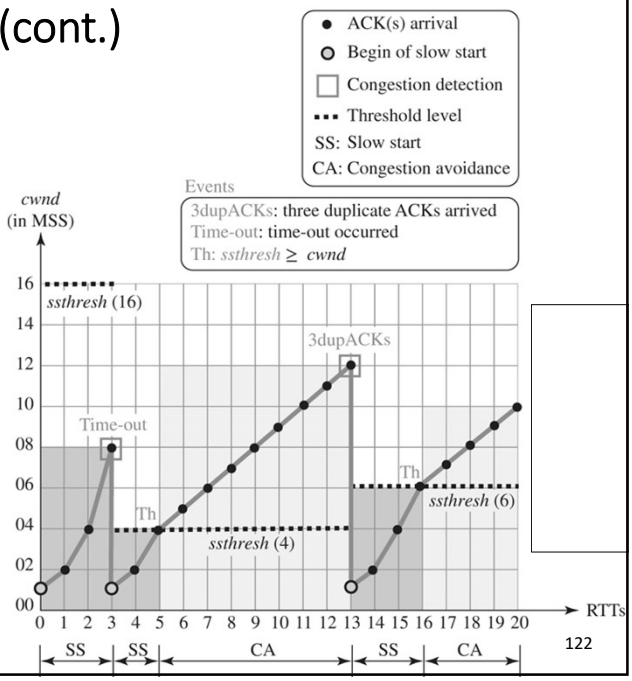
Example: AIMD in TCP Tahoe

- *ssthresh* variable is set to 16 MSS
- TCP begins at the slow-start (SS) state with the *cwnd* = 1, and *cwnd* grows exponentially
- A time-out occurs (before reaching the threshold)
  - TCP assumes a congestion in network
  - sets the *ssthresh* = 4 MSS (half of the current *cwnd*)
  - begins a new SS with *cwnd* = 1 MSS
- *rwnd* grows exponentially until it reaches the newly set threshold
  - TCP moves to congestion avoidance (CA) state, and *cwnd* grows additively until it reaches *cwnd* = 12 MSS



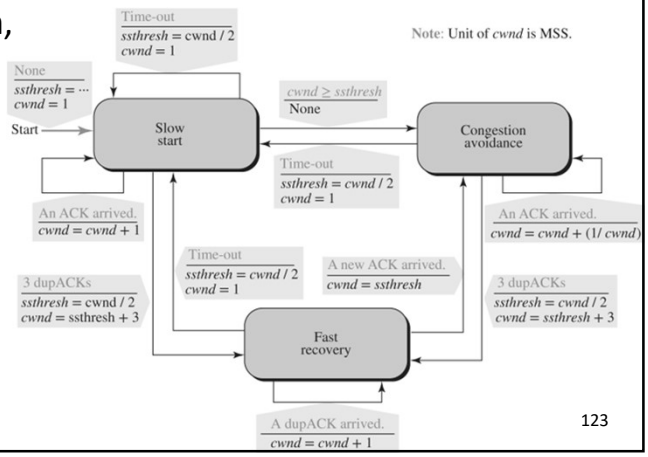
Example: AIMD in TCP Tahoe (cont.)

- Three duplicate ACKs arrive
  - TCP assumes congestion
  - sets the *ssthresh* = 6 MSS (half of the current cwnd)
  - begins a new SS with *cwnd* = 1 MSS
- *rwnd* grows exponentially until it reaches the newly set threshold
  - TCP moves to congestion avoidance (CA) state, and *cwnd* grows additively until it reaches *cwnd* = 12 MSS
- *rwnd* grows exponentially until it reaches the newly set threshold
  - TCP moves to congestion avoidance (CA) state, and *cwnd* grows additively until the connection is terminated after RTT 20



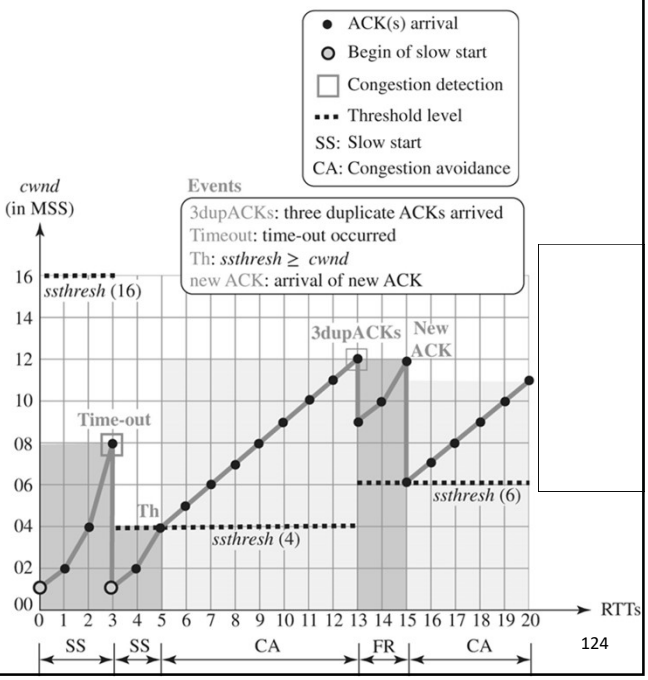
TCP Reno

- Most common version today
- Goes into fast recovery mode upon receiving three duplicate ACKs, half of the current *cwnd* is saved as *ssthresh*
- For successive duplicate ACKs (4th, 5th, 6th), *cwnd* increases by 1
- Once the receiver finally receives missing packet, TCP will move to congestion avoidance or slow state upon a timeout



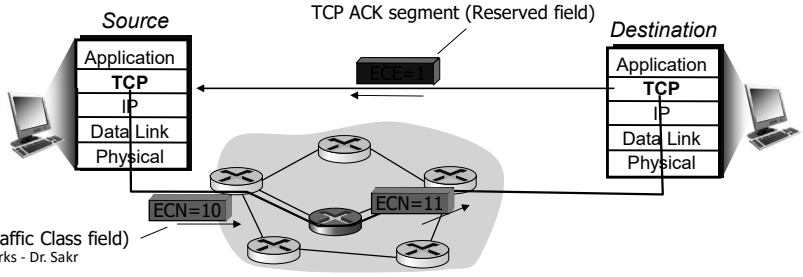
Example: AIMD in TCP Reno

- Same as Taho TCP until RTT 13
- Three duplicate ACKs arrive
  - TCP assumes congestion
  - sets the *ssthresh* = 6 MSS
  - sets *cwnd* to a much higher value (*ssthresh* + 3 = 9 MSS) instead of 1 MSS
  - moves to the fast recovery state
  - two more duplicate ACKs arrive until RTT 15, where *cwnd* grows exponentially
- A new ACK (not duplicate) arrives that announces the receipt of the lost segment
  - moves to CA state, but first deflates *rwnd* to 6 MSS as though ignoring the whole fast-recovery state and moving back to the previous track



Explicit Congestion Notification (ECN)

- Congestion is handled only by the transmitter, but it is known only after a packet was sent
- TCP/IP supports network-assisted congestion control to allow destination (and routers) to inform the sender about the congestion
  1. Congestion indication carried from routers to destination
  2. Destination echoes indication in the ACK to notify sender of congestion
  - Note that this operation involves both IP and TCP layers



Outline

- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
    - Timers
  - Stream Control Transmission Protocol (SCTP)

TCP Timers

- Most TCP implementations use at least four timers:
  - Retransmission
  - Persistence
  - Keepalive
  - TIME-WAIT

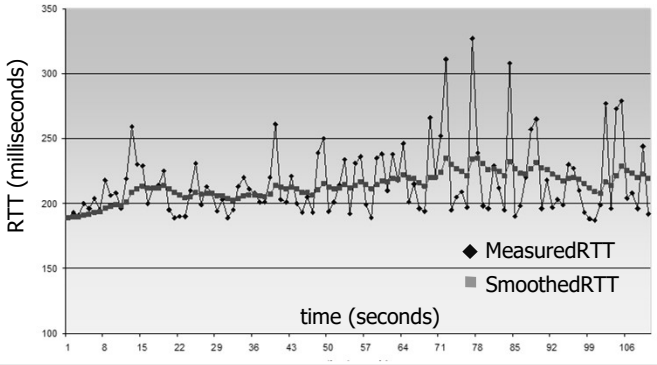
Retransmission Timer

- To retransmit lost segments, TCP employs one retransmission timer (RTO) for the whole connection period
  - Tracks the waiting time for an acknowledgment of a segment
  - Starts when the segment in front of the queue is sent
  - First segment in front of the queue is resent when timer expires
  - When one or more segments are cumulatively ACKed, they are purged from the queue

RTO Calculation

- RTT can be measured from segment transmission until ACK receipt
- Problem: highly varies for each round trip and needs to be smoothed
- Solution: smooth using exponential weighted moving average

$$SmoothedRTT = (1-\alpha) \times SmoothedRTT + \alpha \times MeasuredRTT$$



RTO Calculation (cont.)

- To add some safety margin, DevRTT is used to account for large deviations

$$DevRTT = (1-\beta) \times DevRTT + \beta \times |MeasuredRTT - SmoothedRTT|$$

- Finally, RTO can be calculated as

$$RTO = SmoothedRTT + 4 \times DevRTT$$

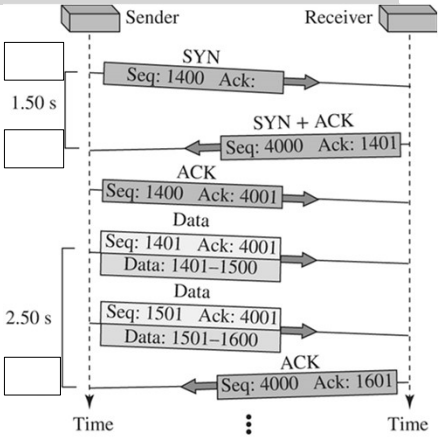
- Typical value of α is 1/8 and β is 1/4
- Exponential backoff: if a retransmission occurs
  - Do not update RTO for that segment (no way to tell the ACK was for the original or retransmitted segment)
  - Double RTO

Example 1: RTO Calculation

Consider the scenario below with initial RTO=6 sec. Calculate RTO after each measurement.

Solution:

- Initially, there is no value for MeasuredRTT, SmoothedRTT, or DevRTT
- After first ACK: MeasuredRTT = 1.5 sec
  - SmoothedRTT = 1.5 sec (initially set to first MeasuredRTT)
  - DevRTT = 1.5/2 = 0.75 (initially set to half MeasuredRTT)
  - RTO = 1.5 + 4 × 0.75 = 4.5 sec
- After second ACK: MeasuredRTT = 2.5 sec
  - SmoothedRTT = 7/8 × 1.5 + 1/8 × 2.5 = 1.625 sec
  - DevRTT = 3/4 × 0.75 + 1/4 × |1.625-2.5| = 0.78 sec
  - RTO = 1.625 + 4 × 0.78 = 4.74 sec



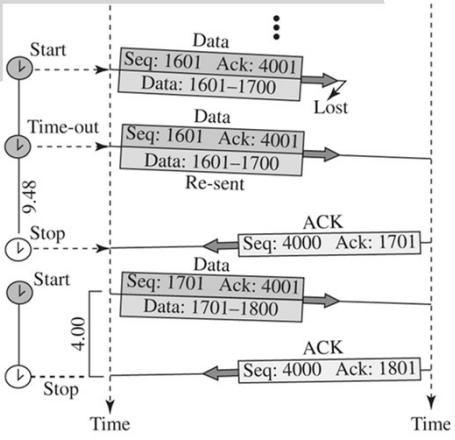
Example 2: RTO Calculation

Calculate RTO after the scenario. Consider the following state:

RTO = 4.74 sec
MeasuredRTT = 2.5 sec
SmoothedRTT = 1.625 sec
DevRTT = 0.78 sec

Solution:

- First segment is sent and lost (i.e., timer expired):
  - RTO = 2 × 4.74 = 9.48 sec
- ACK received before the time-out, do not change RTO, and wait to send a new segment to recalculate
- After second ACK: MeasuredRTT = 4 sec
  - SmoothedRTT = 7/8 × 1.625 + 1/8 × 4 = 1.92 sec
  - DevRTT = 3/4 × 0.78 + 1/4 × |1.92-4| = 1.105 sec
  - RTO = 1.92 + 4 × 1.105 = 6.34 sec



Persistence Timer

- In scenarios with a zero-window-size advertisement, the sending TCP stops transmitting segments and waits for an ACK with nonzero receiving window update
- If the ACK is lost, receiver will not resend it (ACKs are not acknowledged) and both sender and receiver will continue to wait
- To correct this deadlock, TCP uses a persistence timer for each connection
  - Starts when sender receives and ACK with window size of zero
  - When it expires, sending TCP sends a probe segment to cause receiving TCP to resend the ACK
  - If no response, resend a probe segment and double the timer (max. 60s)

Keepalive Timer

- A keepalive timer is used in some implementations to prevent a long idle connection between two TCP's
  - e.g., client opens a TCP connection to a server, transfers some data, and becomes silent
  - In this case, the connection remains open forever
- TCP uses a keepalive timer to avoid this situation
  - Each time the server hears from a client, it resets this timer
  - If the server does not hear from the client after 2 hours, it sends a probe segment
  - If there is no response after 10 probes, each of which is 75s apart, it assumes that the client is down and terminates the connection

TIME-WAIT Timer

- TIME-WAIT timer is used during connection termination
- When TCP performs an active close and sends the final ACK, the connection must stay for some time to allow TCP resend the final ACK in case the ACK is lost
- This requires that the RTO timer at the other end times out and new FIN and ACK segments are resent
- The timer is set to twice the maximum segment life time (MSL); the amount of time any segment can exist in a network before being discarded
- The implementation needs to choose a value for MSL
  - Common values are 30 seconds, 1 minute, or even 2 minutes

Outline

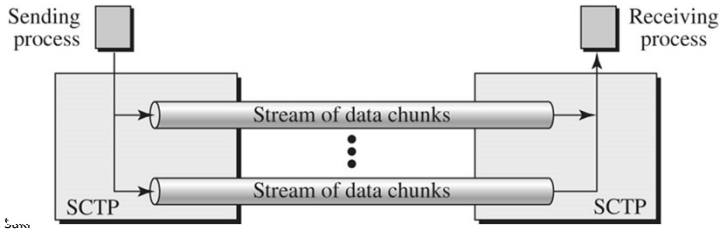
- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)

Stream Control Transmission Protocol (SCTP)

- A new transport-layer protocol designed to combine some features of UDP and TCP to create a protocol for multimedia communication
  - provides the message-oriented feature of UDP, while ensuring reliable, in-sequence transport of messages with congestion control like TCP

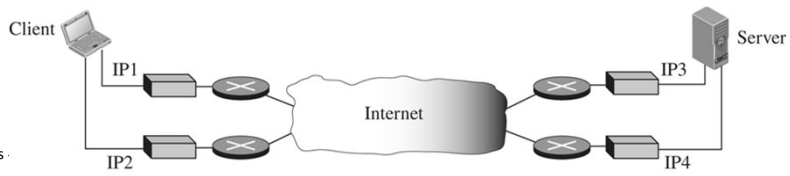
SCTP Services

- SCTP, like UDP or TCP, provides process-to-process communication
- Multiple-stream delivery: SCTP allows multistream service in each connection, which is called association in SCTP terminology
  - TCP connection involves one single stream between TCP client and TCP server
  - A loss at any point in the stream blocks the delivery of the rest of the data
  - This is not acceptable when sending real-time data such as audio or video
  - In SCTP, if one of the streams is blocked, other streams can still deliver their data



SCTP Services (cont.)

- Multihoming: sending host and receiving host can define multiple IP addresses in each end for an association
  - TCP connection involves one source and one destination IP address
  - Even if the sender or receiver is a multihomed host, only one of these IP addresses per end can be used during the connection
    - Multihomed: connected to more than one physical address with multiple IP addresses
  - In SCTP, when one path fails, another interface can be used for data delivery without interruption
  - This fault-tolerant feature is very helpful when we are sending and receiving a real-time payload such as Voice over IP (VoIP)



SCTP Services (cont.)

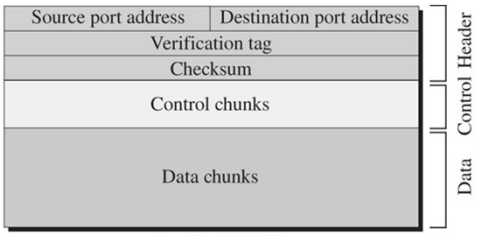
- Like TCP, SCTP offers full-duplex service, where data can flow in both directions at the same time
  - Each SCTP endpoint has its own sending and receiving buffer, and segments move in both directions
- Like TCP, SCTP is a connection-oriented protocol
  - In SCTP, a connection is called an association
- SCTP, like TCP, is a reliable transport protocol
  - It uses an acknowledgment mechanism to check the safe and sound arrival of data

Data Chunk vs. Stream vs. Packets

- Data chunk is unit of data in SCTP
  - May or may not have a one-to-one relationship with the message coming from the process because of fragmentation
  - A transmission sequence number (TSN) is used to number data chunks
  - Each data chunk must carry the corresponding 32-bit TSN in its header
- Stream:
  - SCTP allows several streams in each association
  - A stream identifier (SI) is used to identify each stream
  - A stream sequence number (SSN) is used to number data chunks in a stream
  - Each data chunk must carry both the 16-bit SI and 16-bit SSN in its header so that when it arrives at the destination, it can be properly placed in its stream and in the proper order

Data Chunk vs. Stream vs. Packets (cont.)

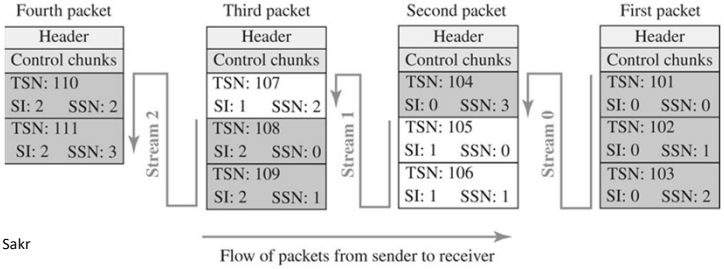
- In SCTP packets, data are carried as data chunks, control information as control chunks
  - Several control chunks and data chunks can be packed together in a packet
- That is, an association my send many packets, a packet may contain several chucks, and chunks may belong to different streams!



A packet in SCTP

Example: Packets, Data Chunks, and Streams

- Scenario: Process A sends 11 messages to process B in 3 streams
  - Assumption: each message fits into one data chunk, a packet carries only 3 data chunks, and process delivers data stream by stream
- Note:
  - TSN is cumulative (initialized randomly), SI defines the stream, and SSN defines the order the chunk in a particular stream (starts from 0)
  - If a message has been fragmented, SSN must be the same for all fragments and TSN is used as a fragment sequence number

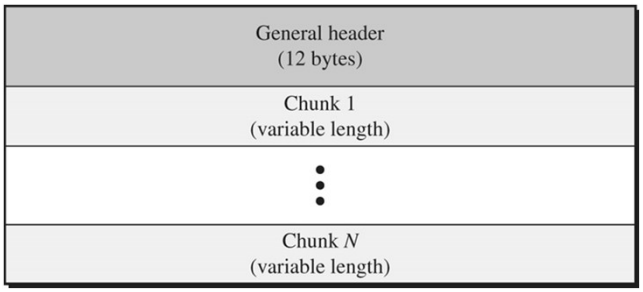


Acknowledgment Number

- SCTP acknowledgment numbers are chunk-oriented, refer to the TSN
- Acknowledgment numbers are used to acknowledge only data chunks
- Control chunks are acknowledged by other control chunks if necessary

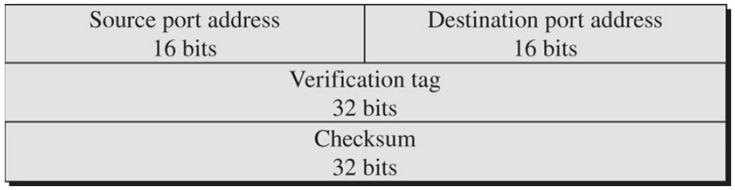
SCTP Packet Format

- An SCTP packet has a mandatory general header and a set of chunks
- There are two types of chunks:
  - A control chunk controls and maintains the association
  - A data chunk carries user data
- In a packet, control chunks come before the data chunks



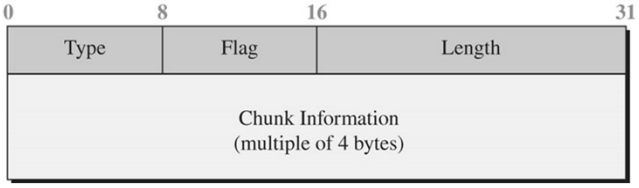
Header

- Includes four fields:
  - Source and destination port numbers
  - Verification tag: identifier for the association to match packets to an association
    - Repeated in every packet during an association to define the end points of association
  - Checksum: 32 bits, compared to 16 bits in TCP
    - Allow STCP to use CRC-32
    - Preserves the integrity of the contents of the packet including the header itself



Chunks

- Carry control information or user data and have a common layout
  - First three fields are common to all chunks
  - Type field can define up to 256 types of chunks; only a few have been defined
  - Flag field defines special flags depending on the Type field
  - Length field defines the total size of chunk (in bytes) including type, flag, and length fields plus the information without padding
  - Information field depends on the type of chunk

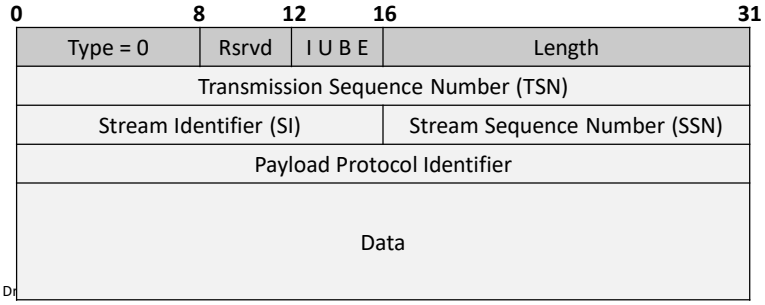


Example: Some Defined Chunk Types

Type	*Chunk*	*Description*
0	DATA	User data
1	INIT	Sets up an association
2	INIT ACK	Acknowledges INIT chunk
3	SACK	Selective acknowledgment
4	HEARTBEAT	Probes the peer for liveliness
5	HEARTBEAT ACK	Acknowledges HEARTBEAT chunk
6	ABORT	Aborts an association
7	SHUTDOWN	Terminates an association
8	SHUTDOWN ACK	Acknowledges SHUTDOWN chunk
9	ERROR	Reports errors without shutting down
10	COOKIE ECHO	Third packet in association establishment
11	COOKIE ACK	Acknowledges COOKIE ECHO chunk
14	SHUTDOWN COMPLETE	Third packet in association termination
192	FORWARD TSN	For adjusting cumulating TSN

Example: DATA Chunk

- Note:
  - Chunk type: 0 for payload data (DATA)
  - Chunk flags: only 4 flags are used
    - I is related to acknowledgments and U, B, and E are related to fragmentation and order
  - Chunk length: minimum value of 17 bytes (i.e., at least 1 byte of data)
  - Fixed parameters: TSN, SI, SSN, and protocol identifier
  - Data



Outline

- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)
    - SCTP Association

An SCTP Association

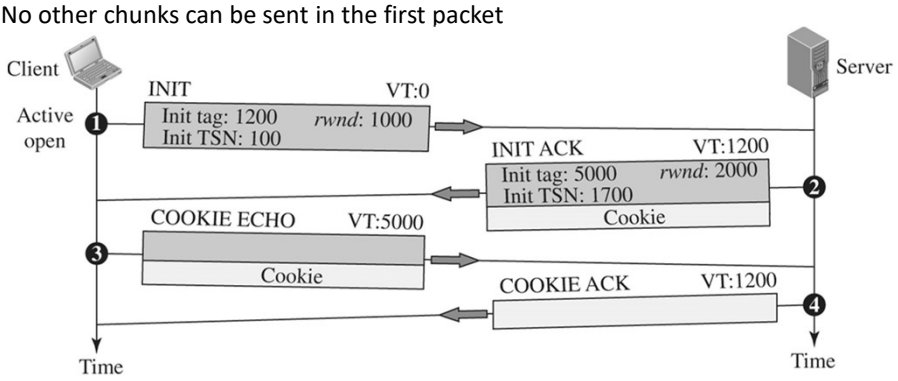
- Like TCP, SCTP is a connection-oriented protocol: all of the segments belonging to a message are then sent over the same logical path
- A connection between two SCTP endpoints is called an association to emphasize multihoming

Association Establishment

- Association establishment in SCTP requires a four-way handshake
- In this procedure, a process, normally a client, wants to establish an association with another process, normally a server, using SCTP as the transport-layer protocol
- Similar to TCP,
  - SCTP server needs to be prepared to receive any association (passive open)
  - Association establishment is initiated by the client (active open)

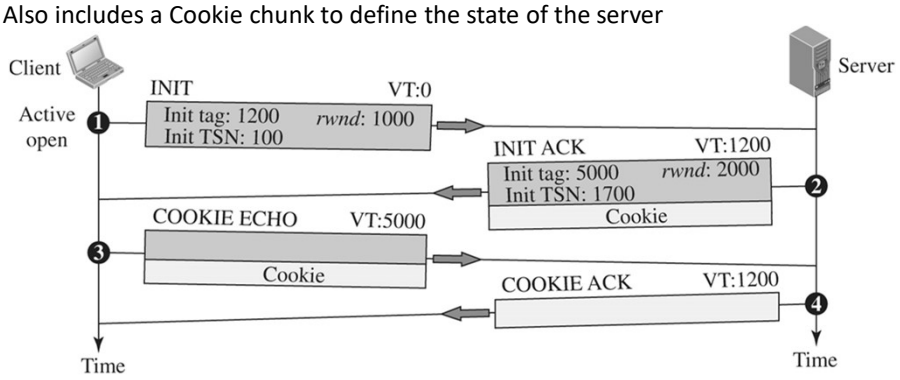
Association Establishment

- Four-way handshaking:
  - ❶ Client sends an INIT chunk in a packet to the server to initiate an association
    - Client-to-server direction: verification tag not known yet (set to 0), initialize TSN
    - Server-to-client direction: use verification tag 1200, rwnd = 1000
    - No other chunks can be sent in the first packet



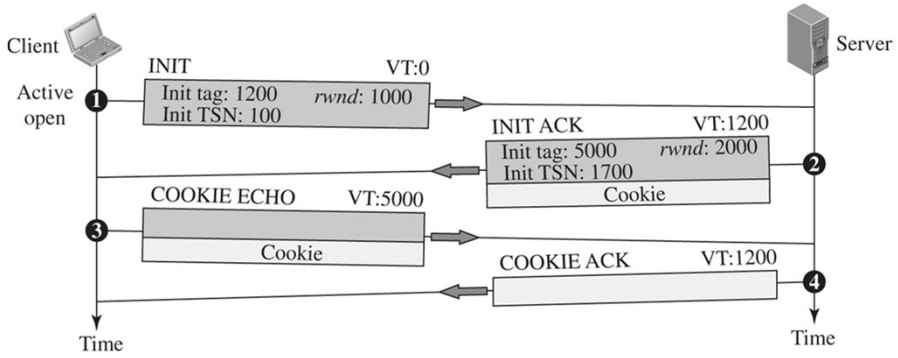
Association Establishment (cont.)

- Four-way handshaking:
  - ❷ Server sends an INIT ACK chunk in response to the client
    - Client-to-server direction: use verification tag 5000, rwnd = 2000
    - Server-to-client direction: use verification tag 1200, initialize TSN
    - Also includes a Cookie chunk to define the state of the server



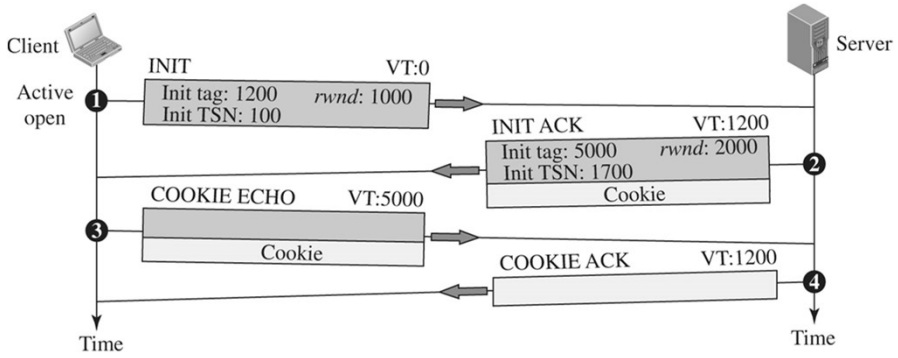
Association Establishment (cont.)

- Four-way handshaking:
  - ❸ Client sends a COOKIE ECHO response
    - Just echoes the state cookie without change
    - Data may be included in this packet



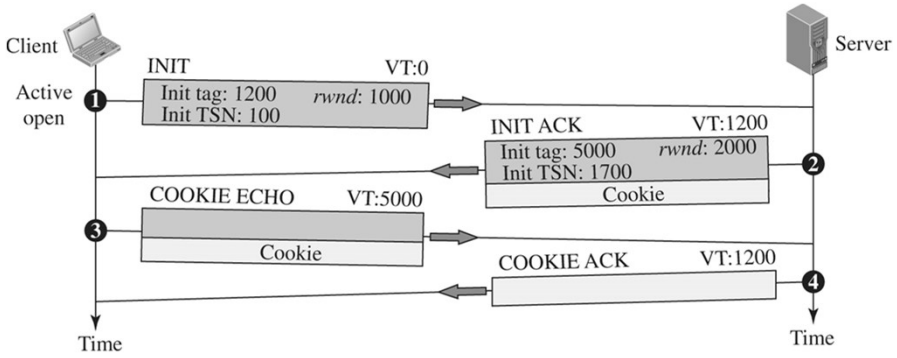
Association Establishment (cont.)

- Four-way handshaking:
  - ❹ Server sends a packet including the COOKIE ACK chunk
    - Acknowledges the receipt of COOKIE ECHO chunk
    - Data may be included in this packet



Security Against SYN Flooding Attacks

- Note:
  - After ❸, the server verifies the authenticity of the state cookie using a secret key was included in the Cookie chunk
  - The server then allocates the resources for the association, and start ❹

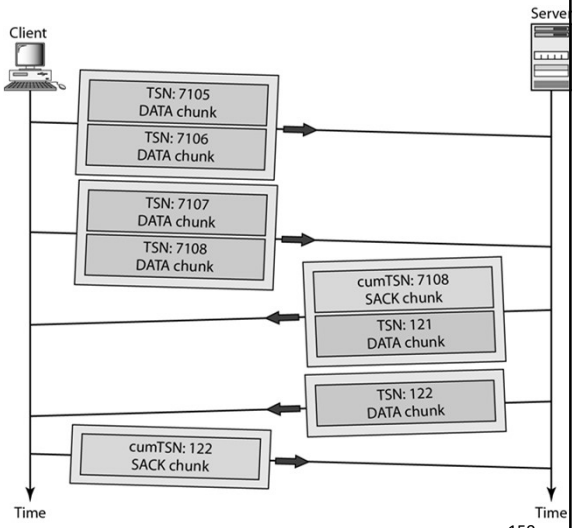


Data Transfer

- After the association is established, bidirectional data transfer can take place (i.e., client and the server can both send data)
- Recall that:
  - TCP receives messages from a process as a stream of bytes without recognizing any boundary between them
  - TCP takes each message and appends it to its buffer
  - A segment can carry parts of two different messages
  - The only ordering system imposed by TCP is the byte numbers
- Unlike TCP,
  - SCTP recognizes and maintains message boundaries, i.e., treated as one unit
  - STCP takes each message and inserts it into a DATA chunk (unless fragmented)
  - Unlike UDP, data chunks are related to each other

Example: Simple Data Transfer

- When a message is received from a process:
  - Insert message in a DATA chunk (or chunks if fragmented) and add a DATA chunk header
  - Each DATA chunk gets a TSN
- Note:
  - Like TCP, SCTP supports piggybacking
  - Acknowledgment in SCTP defines the cumulative TSN; the TSN of the last data chunk received in order
  - Only DATA chunks use TSNs
  - Only Data chunks are acknowledged by SACK chunks



Multihomed Data Transfer

- Multihoming allows both ends to define multiple IP addresses for communication
- However, only one of these addresses can be defined as the primary address; the rest are alternative addresses
- Data transfer uses the primary address of the destination
  - If not available, one of the alternative address is used
  - A process can explicitly request a message to be sent an alternative address

Mulistream Data Delivery

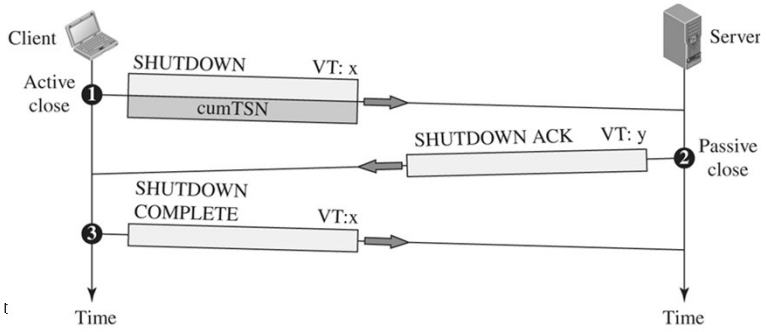
- There is a distinction between data transfer and data delivery in SCTP
  - SCTP uses TSNs to handle data transfer
  - The delivery of data chunks to the receiving process is controlled by SIs and SSNs
- SCTP supports two types of data delivery:
  - Ordered data delivery: default
    - Data chunks in a stream use SSNs to define their order in the stream
    - Destination SCTP may delay delivery if chunks arrive out of order
  - Unordered data delivery:
    - SSN field value is ignored and the U flag is set
    - Destination SCTP delivers chunks to the application without waiting

Association Termination

- In SCTP, like TCP, either of the two parties involved in exchanging data (client or server) can close the connection
- Unlike TCP, SCTP does not allow a “half-closed” association
  - If one end closes the association, the other end must stop sending new data
  - If any data are left over in the queue of the recipient of the termination request, they are sent and the association is closed

Association Termination (cont.)

- Association termination uses three packets:
  1. Client sends a SHUTDOWN signal to the server
  2. Server responds by sending a SHUTDOWN ACK acknowledgement
  3. Client sends a SHUTDOWN COMPLETE signal back to the server
- In this example termination is initiated by the client, however, it can also be initiated by the server



Outline

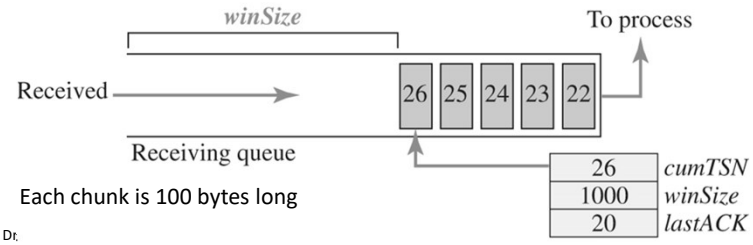
- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)
    - Flow Control

Flow Control

- Flow control in SCTP is similar to that in TCP
- In SCTP, we need to handle two units of data, the byte and the chunk
  - The values of *rwnd* and *cwnd* are expressed in bytes
  - The values of TSN and acknowledgments are expressed in chunks
- To show the concept, we make some unrealistic assumptions:
  - No congestion in the network: *cwnd* is infinite
  - Network is error-free: no packet is lost, delayed, or out-of-order
  - Data transfer is unidirectional

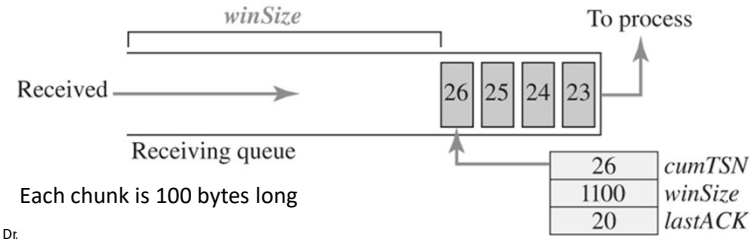
Flow Control: Receiver Site

- The receiver has one buffer (queue) and three variables:
  - *cumTSN*: holds the last TSN received
  - *winSize*: holds the available buffer size
  - *lastACK*: holds the last cumulative acknowledgement
- The buffer holds the received data chunks that have not yet been read by the process



Flow Control: Receiver Site (cont.)

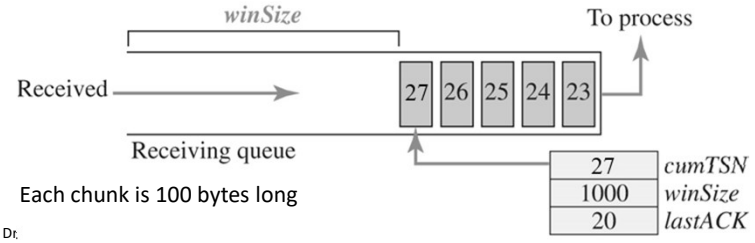
- When the process reads a chunk
  - Remove from the buffer
  - Add the size of removed chunk to *winSize*



Each chunk is 100 bytes long

Flow Control: Receiver Site (cont.)

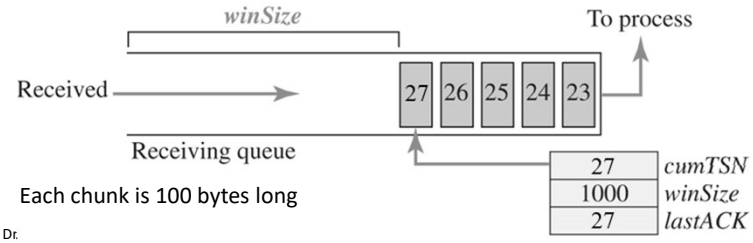
- When the site receives a data chunk
  - Store at the end of the buffer
  - Subtract the size of the chunk from *winSize*
  - Store TSN number of the chunk in the *cumTSN* variable



Each chunk is 100 bytes long

Flow Control: Receiver Site (cont.)

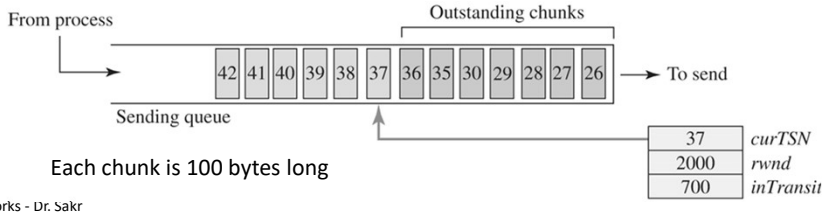
- When the receiver decides to send a SACK
  - Check the value of *lastAck*
  - If less than *cumTSN*
    - Send a SACK with cumulative TSN number equal to the *cumTSN*
    - The advertised window size is set to *winSize*
    - Update the value of *lastACK* to hold the value of *cumTSN*



Each chunk is 100 bytes long

Flow Control: Sender Site

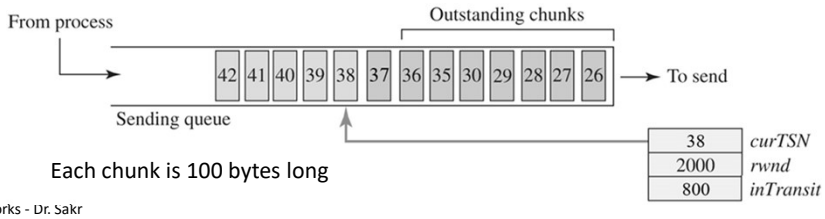
- The sender has one buffer (queue) and three variables:
  - *curTSN*: refers to the next chunk to be sent, TSNs before that were sent but not acknowledged yet
  - *rwnd*: holds the last value advertised by the receiver in bytes
  - *inTransit*: holds the number of bytes in transit, bytes sent but not yet acknowledged
- The buffer holds the data chunks produced by the process that have either been sent or are ready to send



Each chunk is 100 bytes long

Flow Control: Sender Site (cont.)

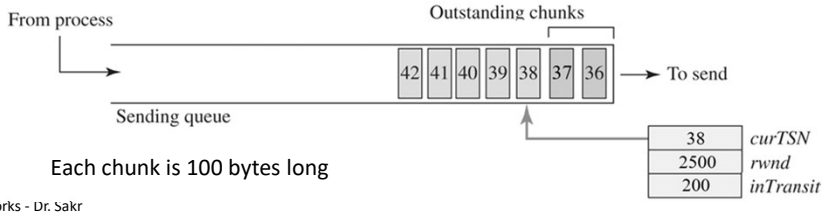
- A chunk pointed by *curTSN* can be sent if its size is less than or equal to (*rwnd – inTransit*)
- After sending a chunk
  - Increment the value of *curTSN* by 1 and point to the next chunk to be sent
  - Increment the value of *inTransit* by the size of chunk



Each chunk is 100 bytes long

Flow Control: Sender Site (cont.)

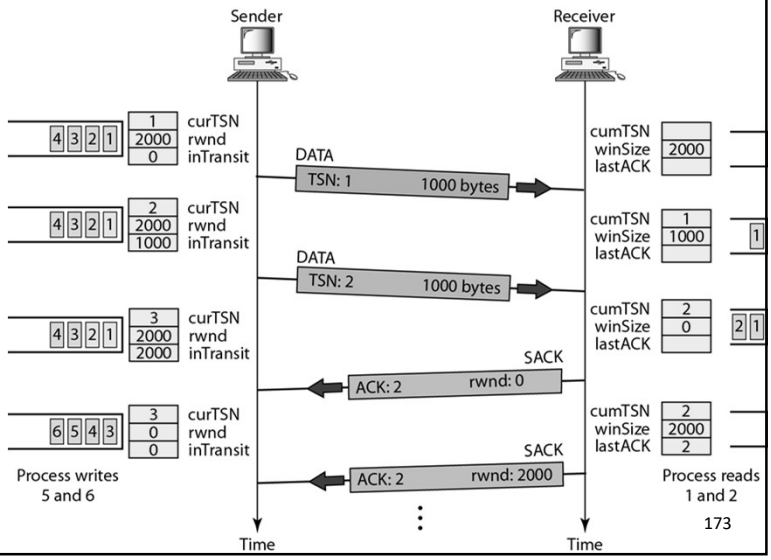
- When a SACK is received
  - Remove all chunks with TSN less than or equal to the cumulative TSN in SACK
  - Reduce the value of *inTransit* by the total size of discarded chunks
  - Update the value of *rwnd* with the value of advertised window in SACK



Each chunk is 100 bytes long

Example

- Scenario: connection is initiated with *rwnd* at sender is 2000 bytes and *winSize* at receiver is 2000 bytes
- Each chunk is 1000 bytes



Outline

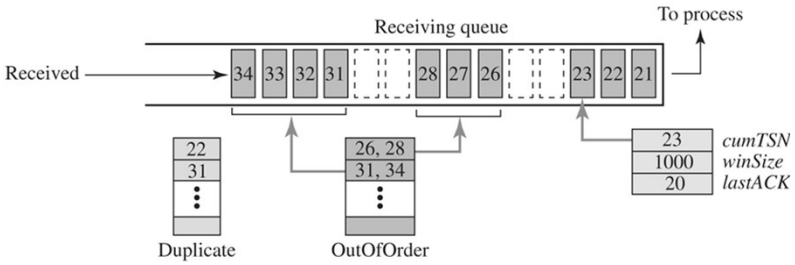
- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)
    - Error Control

Error Control

- SCTP, like TCP, is a reliable transport-layer protocol
- It uses a SACK chunk to report the state of the receiver buffer to the sender
- Each implementation uses a different set of entities and timers for the receiver and sender sites
- We will consider a very simple design to convey the concept with 100-byte chunks

Error Control: Receiver Site

- Receiver:
  - Stores all chunks received in its queue including the out-of-order ones
  - Leaves spaces for any missing chunks
  - Discards duplicate messages, but keeps track of them to reports to the sender
  - Sends SACKs



Error Control: Receiver Site

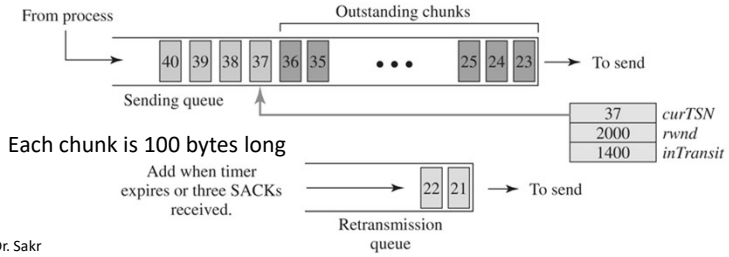
- SACKs include:
  - The TSN numbers for out-of-order chunks relative to the cumulative TSN
  - The TSN number for duplicate data chunks

SACK chunk

Type: 3	Flag: 0	Length: 32
Cumulative TSN: 23		
Advertised receiver window credit: 1000		
Number of gap ACK blocks: 2		Number of duplicates: 2
Gap ACK block #1 start: 3		Gap ACK block #1 end: 5
Gap ACK block #2 start: 8		Gap ACK block #2 end: 11
Duplicate TSN: 22		
Duplicate TSN: 31		

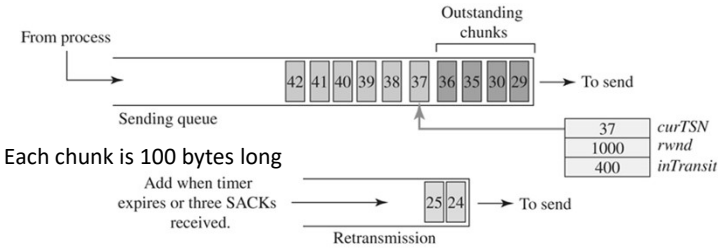
Error Control: Sender Site

- Sender uses two buffers: a sending queue and a retransmission queue, and three variables: *rwnd*, *inTransit*, and *curTSN*
- Sender:
  - When a packet is sent, a retransmission timer starts for that packet (all data chunks in that packet)
  - When (i) the retransmission timer for a packet expires or (ii) 4 SACKs arrive that declare a packet missing, all chunks in packet are moved to the retransmission buffer to be resent



Error Control: Sender Site (cont.)

- The buffers below are after receiving a SACK with *cumTSN*=23 and acknowledge out-of-order chunks 26-28 and 31-34
  - Chunks with TSN less than or equal to 23 are removed (i.e., 21, 22, and 23)
  - All acknowledged out-of-order chunks are removed (i.e., 26-28 and 31-34)
  - Assume timer of chunks 24 and 25 has expired, move to retransmission buffer



Sending Data Chunks

- An end can send a data packet whenever
  - There are data chunks in the sending queue with a TSN greater than or equal to *curTSN* or
  - There are data chunks in the retransmission queue
- Chunks in the retransmission queue have priority
  - The next time the sender sends a chunk, it would be first one in the queue
- The total size of the data chunk or chunks included in the packet must not exceed:
  - The (*rwnd – inTransit*) value and
  - The total size of the frame must not exceed the MTU size

Retransmissions

- SCTP, like TCP, employs two strategies for retransmissions:
  - Timeout of the retransmission timer (i.e., RTO)
    - Handles waiting time for an ACK
    - RTO is calculated similar to TCP using RTT measurements
  - Receiving four SACKs whose Gap ACK indicates one or some chunks missing
    - Do not wait for RTO to expire
    - Move chunks to the retransmission queue upon receiving four SACKs
    - This called fast retransmission

SACK Generation Rules

- The rules for generating SCTP SACK chunks are similar to TCP ACKs
  1. When an end sends a DATA chunk to the other end, it must include a SACK chunk advertising the receipt of unacknowledged DATA chunks
  2. When an end receives a packet containing data, but has no data to send, it needs to acknowledge receiving the packet within a specified time (500 ms)
  3. An end must send at least one SACK for every other packet it receives
  4. When a packet arrives with out-of-order data chunks, it needs to immediately send a SACK chunk
  5. When an end receive a packet with duplicate DATA chunk and no new DATA chunk, the duplicate data chunks must be reported immediately with a SACK chunk

Outline

- Communication at the transport layer
- Transport-layer services
- Transport-layer protocols
  - User-Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
  - Stream Control Transmission Protocol (SCTP)
    - Congestion Control

Congestion Control

- SCTP use the same strategies for congestion control used in TCP
- Since SCTP supports multihoming, each IP address must be maintained with a different value of *rwnd*

Summary: TCP vs. UDP vs. SCTP

Attribute	*TCP*	*UDP*	*SCTP*
Reliability	Reliable	Unreliable	Reliable
Connection Service	Connection-oriented	Connectionless	Connection-oriented
Transmission	Byte-oriented	Message-oriented	Message-oriented
Flow Control	Yes	No	Yes
Congestion Control	Yes	No	Yes
Fault Tolerance	No	No	Yes
Data Delivery	Strictly Ordered	Unordered	Both

Summary

- We covered:
  - Transport layer services of UDP, TCP, and SCTP
  - Differences between different transport-layer protocols in terms of reliability, transmission, flow control, congestion control, fault tolerance, and data delivery
- Extra reading: www.ietf.org/rfc
  - A discussion of UDP can be found in RFC 768
  - A discussion of TCP can be found in the following RFCs: 675, 700, 721, 761, 793, 879, 1078, 1106, 1110, 1114, 1145, 1263, 1323, 1337, 1379, 1644, 1693, 1901, 1905, 2001, 2018, 2488, 2580
  - A discussion of SCTP can be found in the following RFCs: 2960, 3257, 3284, 3285, 3286, 3309, 3436, 3554, 3708, 3375