NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Week 12: Transport Layer

EE3017/IM2003 Computer Communications

School of Electrical and Electronic Engineering

Prof. Cheng Tee Hiang

Room: S1-B1a-29

Email: ethcheng@ntu.edu.sg

Phone: 6790-4534

Topic Outline

Topic Outline

Wireless Local Area Network (WLAN)

- Transport Service and Protocols
- User Diagram Protocol (UDP)
- Transmission Control Protocol (TCP)
- TCP Sequence Number and Acknowledgement
- Round-trip Time and Timeout
- Retransmission and Fast Retransmit
- Flow Control
- Connection Managements

Recommended reading:
Section 3.3, Pages 236 to 240 and Section 3.5, Pages 268 to 297 of the recommended textbook
(Page numbers are based on 5th or 2010 Edition.)

Learning Objectives

Learning Objectives

By the end of this topic, you should be able to:

- Explain Transport Layer services and protocols.
- Explain the differences between Transport layer and Network Layer services.
- Explain User Datagram Protocol (UDP).
- Explain Transmission Control Protocol (TCP).
- Explain the various fields in a TCP segment.
- Explain the TCP sequence and acknowledgment sequence and how they are used for error control.
- Compute the TCP round-trip time and timeout interval.
- Explain TCP flow control.
- Explain TCP connection managements.

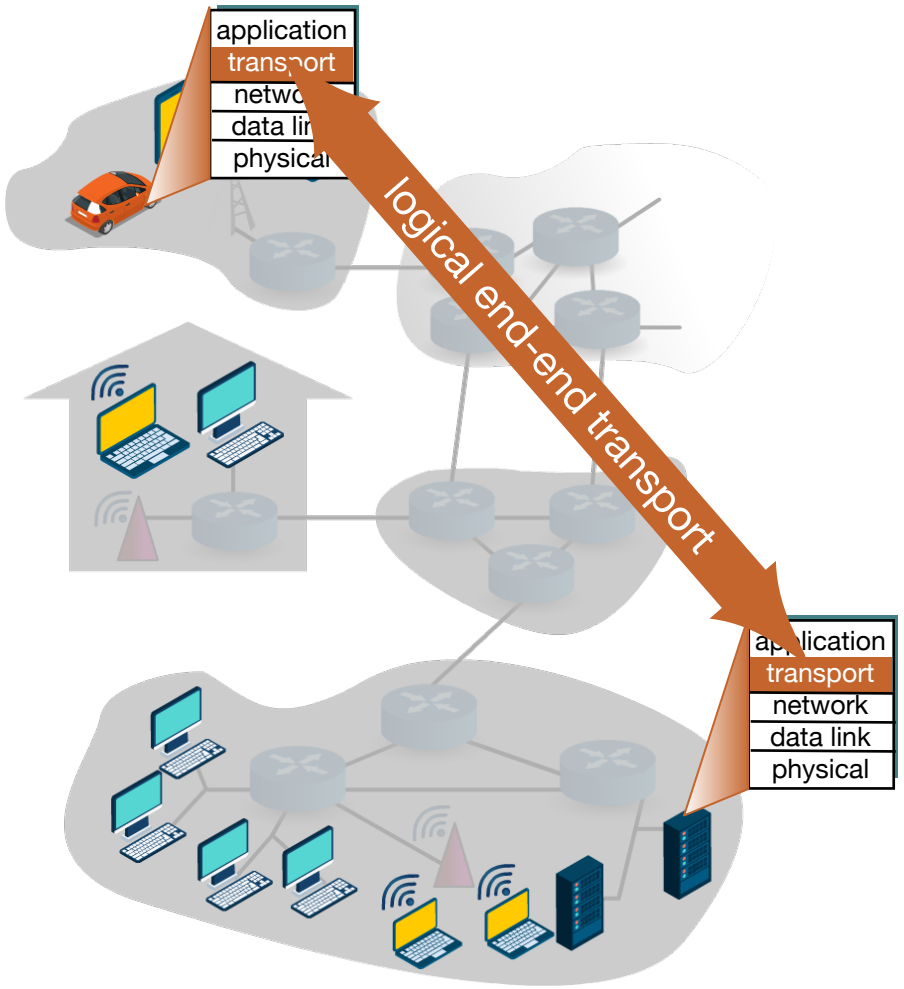
Transport Service and Protocols

Transport Service and Protocols

- Provide **logical communication** between application processes running on different hosts.

- Transport protocols run in end systems.
  - Sender side: breaks application messages into **segments**, passes to network layer.
  - Receiver side: reassembles segments into messages, passes to application layer.

- More than one transport protocol available to applications
  - Internet: TCP and UDP



Transport vs. Network Layer

Transport Logical communication between processes.

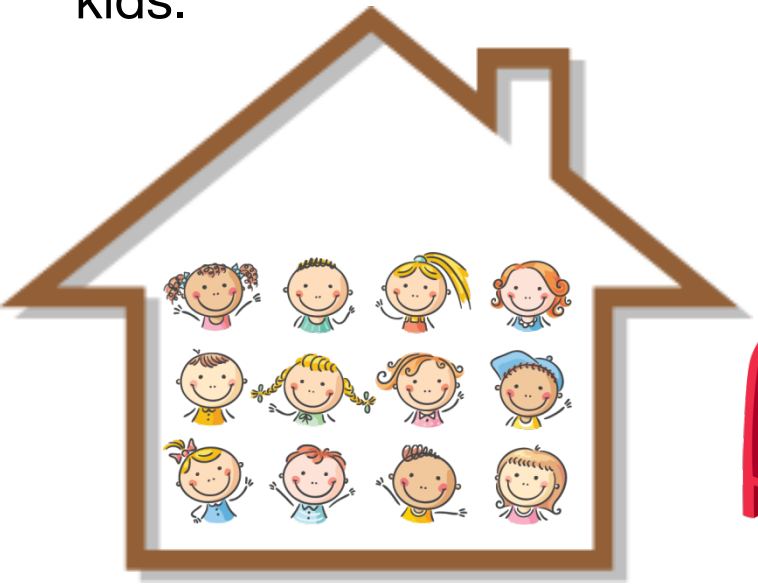
- Relies on, enhances, network layer services.

Network Logical communication between hosts.

Household analogy: 12 kids sending letters to 12 kids.

Transport vs. Network Layer

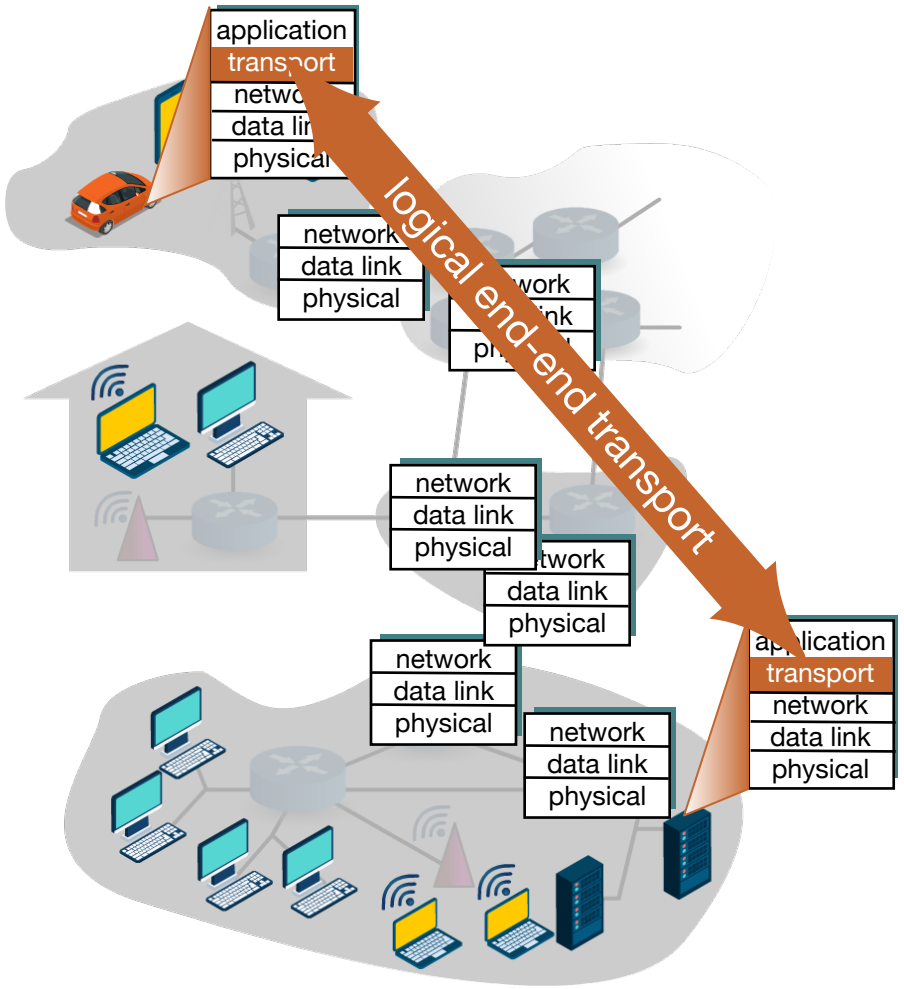
Household analogy: 12 kids sending letters to 12 kids.



Hosts = Houses
Processes = Kids
Application message = Letters in envelops
Transport layer protocol = House keepers
Network layer protocol = Postal service

Internet Transport-Layer Protocols

- Reliable, in-order delivery: TCP
  - Congestion control
  - Flow control
  - Connection setup
- Unreliable, unordered delivery: UDP
  - No-frills extension of “best-effort” IP
- Services not available:
  - Delay guarantees
  - Bandwidth guarantees



Notes: Transport Layer

- A transport-layer protocol provides for logical communication between application process running on different hosts. By logical communications, we mean that from an application's perspective, it is as if the hosts running the processes were directly connected.

- Transport-layer protocols are implemented in the end systems but not in network routers. On the sending side, the transport layer converts the message it receives from a sending application process into segments. This is done by breaking the message into smaller chunks adding a transport layer to each chunk. The transport layer then passes the segment to the network layer, which encapsulates it into a network-layer packet, which is also known as a datagram.

- The datagram will be routed from one router to another router and eventually it will reach the receiving end system. There, the network layer extracts the transport-layer segment from the datagram and passes the segment to the transport layer.

Notes: Transport Layer

- Two main transport-layer protocols are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

- UDP provides an unreliable, connectionless service to the invoking application. TCP provides a reliable, connection-oriented service to the invoking application. When designing a network application, the application developer must specify one of these two transport protocols.

- Due to the connection-oriented nature, a TCP connection needs to be set up between two processes in two end systems over the network before data could be exchanged. The connection also needs to be torn down after the data transfer is completed.

- TCP uses a network congestion-avoidance algorithm to minimise congestion within the Internet. It also uses flow control to prevent the sender overflowing the receiver's buffet. *Congestion control will not be covered in this course.*

- TCP and UDP are unable to provide bandwidth and delay guarantees.

User Datagram Protocol (UDP)

User Datagram Protocol (UDP) [RFC768]

- "No frills," "bare bones" Internet transport protocol

- "Best effort" service, UDP segments may be:
  - Lost
  - Delivered out of order to applications

- **Connectionless**:
  - No handshaking between UDP sender, receiver
  - Each UDP segment handled independently of others

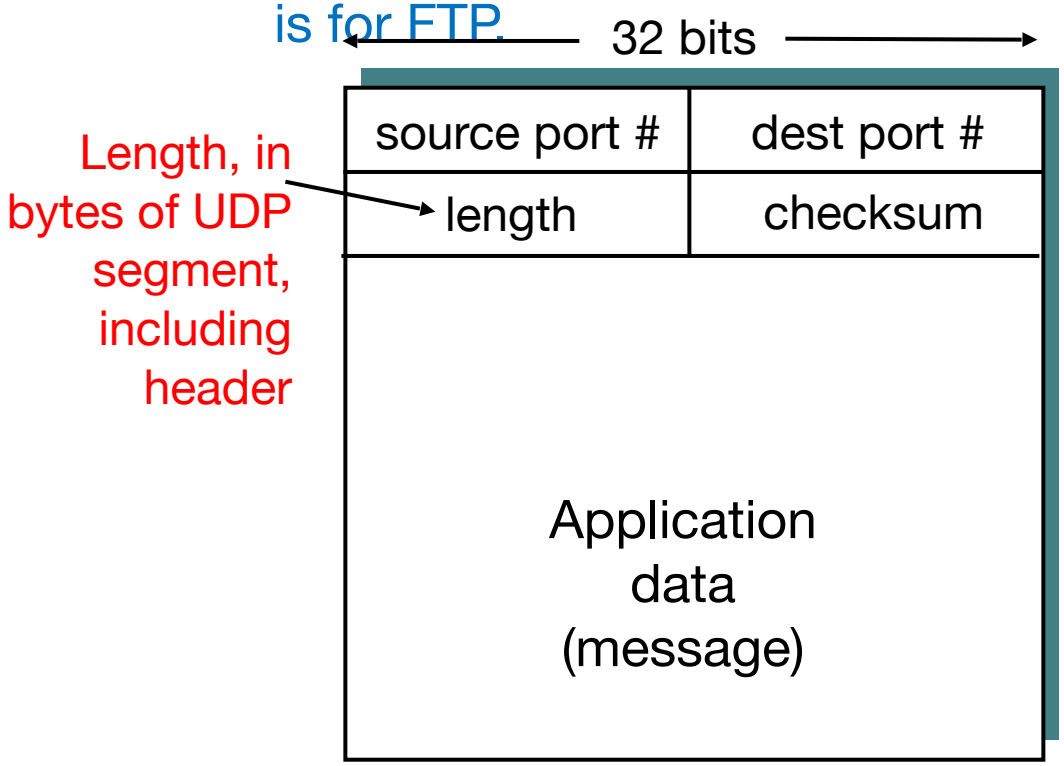
Why is there a UDP?

- No connection establishment (which can add delay)
- Simple: no connection state at sender, receiver
- Small segment header
- No congestion control: UDP can blast away as fast as desired

User Datagram Protocol (UDP)

- Often used for streaming multimedia applications
  - Loss tolerant
  - Rate sensitive
- Other UDP uses:
  - Domain Name System (DNS)
  - Simple Network Management Protocol (SNMP)
- Reliable transfer over UDP: add reliability at application layer
  - Application-specific error recovery!

Port numbers ranging from 0 to 1023 are called well-known port number; e.g. 80 is for HTTP and 21 is for FTP.



UDP segment format

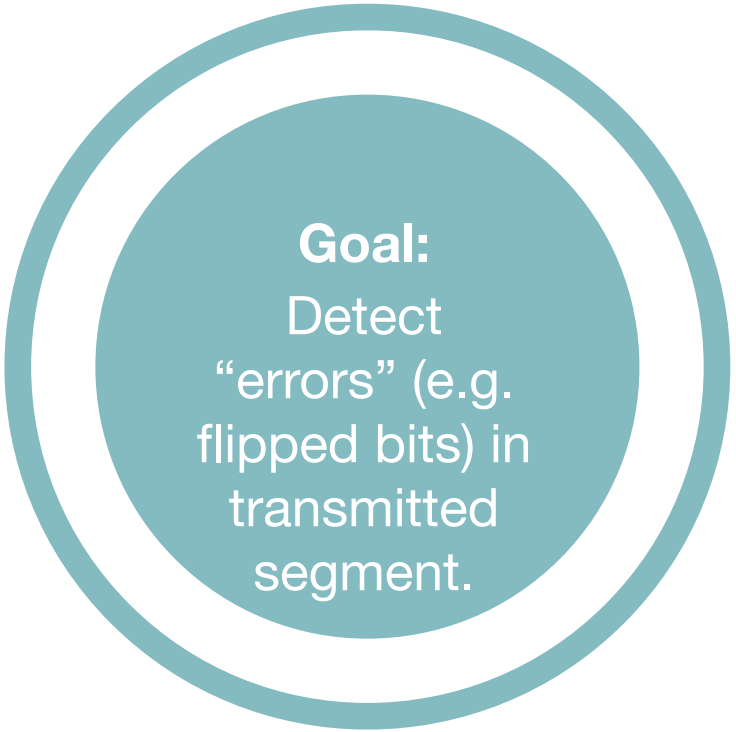
User Datagram Protocol (UDP): Additional Information

Port Number	Protocol	Application
20	TCP	FTP data
21	TCP	FTP control
22	TCP	SSH
23	TCP	Telnet
25	TCP	SMTP
53	UDP, TCP	DNS
67, 68	UDP	DHCP
69	UDP	TFTP
80	TCP	HTTP (WWW)
110	TCP	POP3
161	UDP	SNMP
443	TCP	HTTPS (SSL)
16, 384 - 32, 767	UDP	RTP-based voice (VoIP) and video

UDP Checksum

Sender

- Treat segment contents as sequence of 16-bit integers
- Checksum: addition (1's complement sum) of segment contents
- Sender puts checksum value into UDP checksum field



Receiver

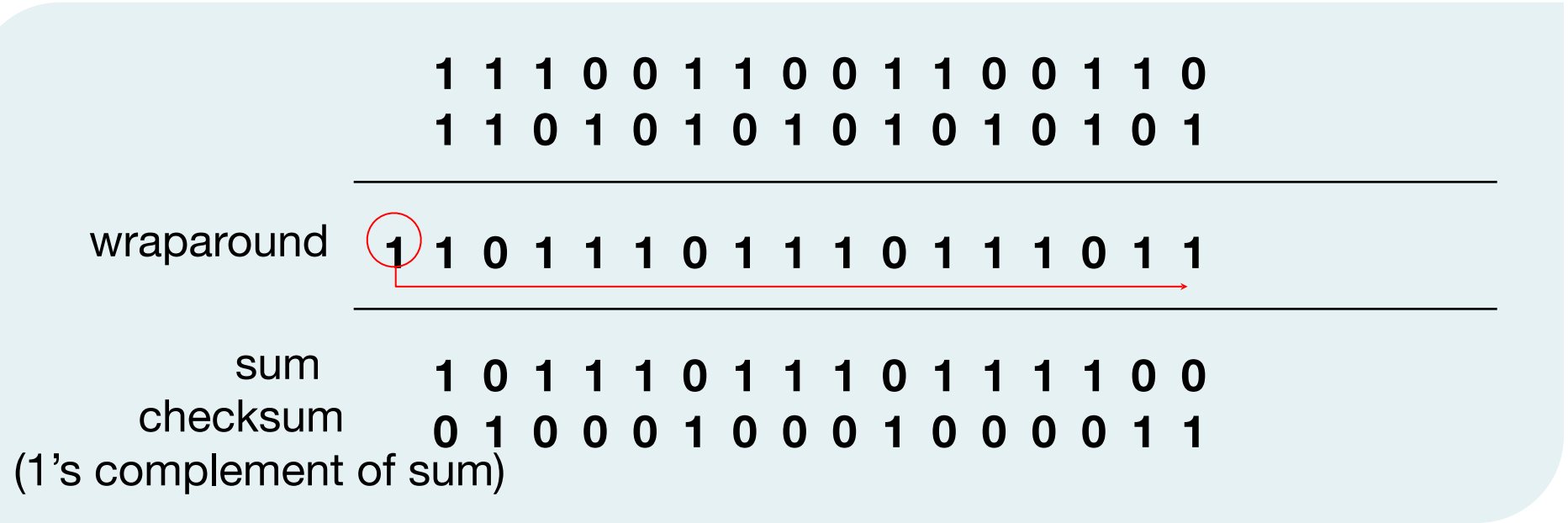
- Compute checksum of received segment
- Check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. But maybe errors nonetheless? More later …

Internet Checksum Example

Note: When adding numbers, a carryout from the most significant bit needs to be added to the result

Example:

Add two 16-bit integers.



Notes: UDP

- UDP is a “no-frills” and “bare-bones” transport layer protocol. It provides the “best-effort” service and will not be able to guarantee that the segments will be delivered error-free and in-sequence. It is connectionless in that no connection needs to be set up and torn down. When an application message is received, the UDP layer will insert a simple header that includes the source port and destination port, length of the segment (including header), and the checksum.

- Due to the low overhead and simplicity, it is suitable to be used for multimedia streaming applications, which could tolerate some data loss. It is also suitable and has been used for Domain Name System (DNS) service and Simple Network Management Protocol (SNMP) because of its simplicity and low overhead.

- The checksum field in the UDP segment header provides error control over the entire UDP segment.

  How checksum can be computed will be covered in the last module of this course.

Transmission Control Protocol (TCP)

Transmission Control Protocol (TCP): Overview [RCFs: 793, 1122, 1323, 2018, 2581]

Point-to-point:
- One sender, one receiver

Pipelined:
- TCP congestion and flow control set window size

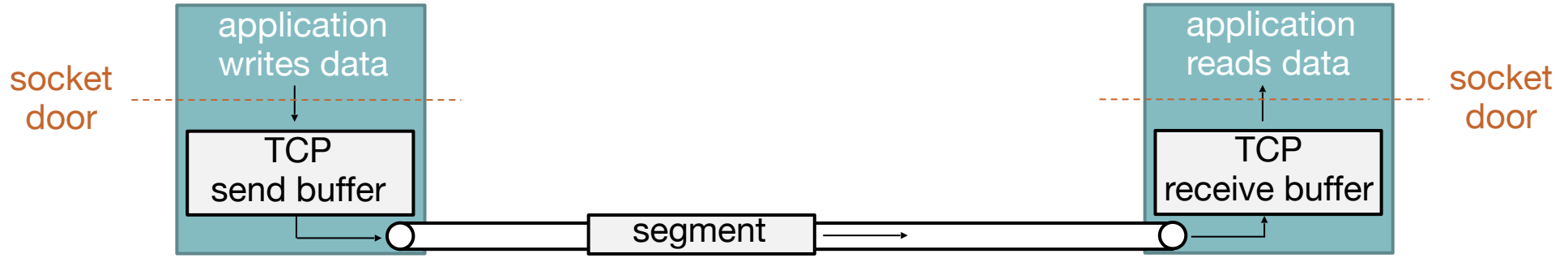
Full duplex data:
- Bi-directional data flow in same connection
- MSS: maximum segment size

Reliable, in-order byte steam:
- No “message boundaries”

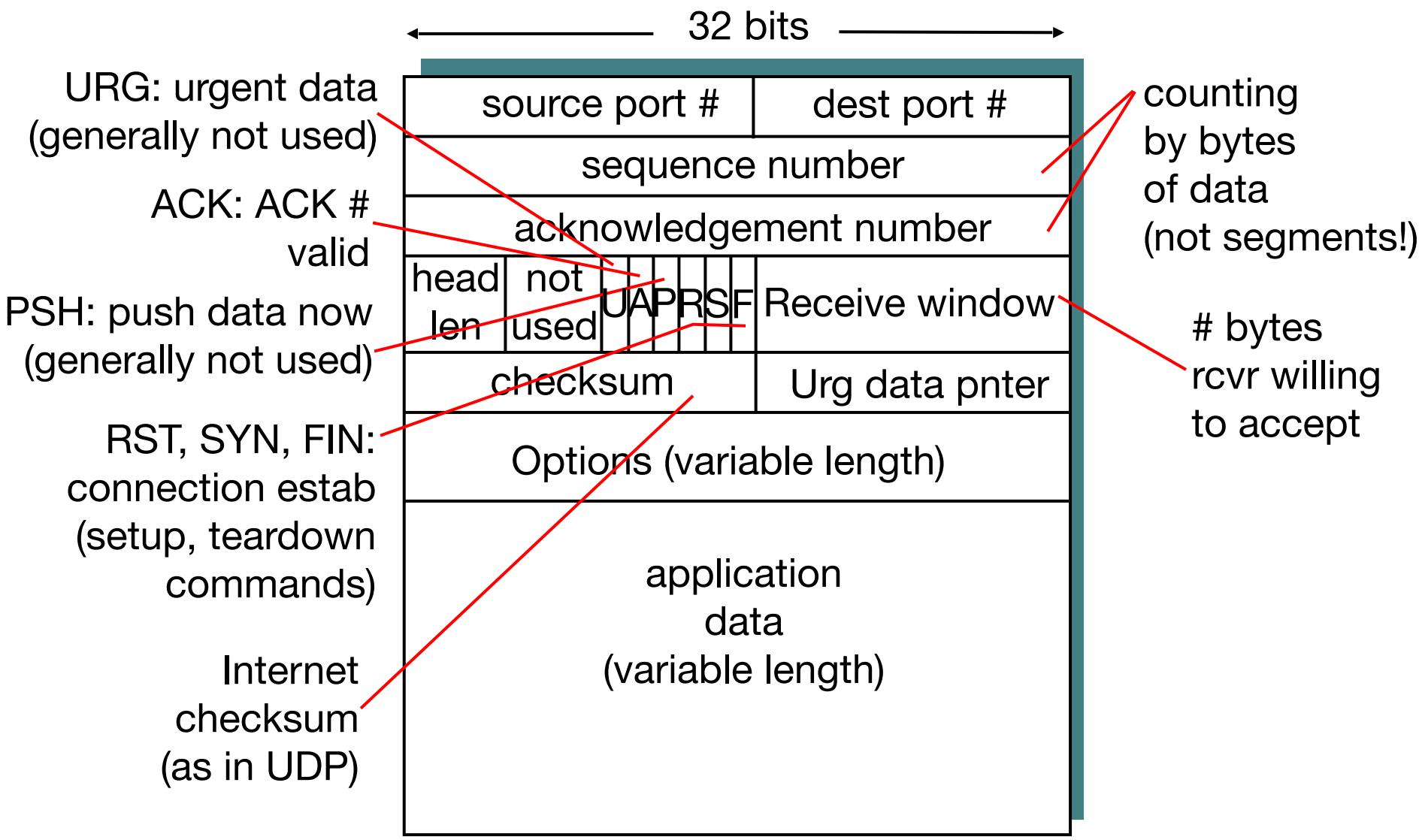
Send and receive buffers

Connection-oriented:
- Handshaking (exchange of control messages) in it’s sender, receiver state before data exchange

Flow controlled:
- Sender will not overwhelm receiver



TCP Segment Structure



Notes: TCP Segment Structure

- As with UDP, TCP header includes source and destination port numbers, as well as a checksum field, which provides error control over the entire TCP segment.
- The 32-bit sequence number field and the 32-bit acknowledgement number field are used by TCP sender and receiver in implementing a reliable data transfer service.
- The 16-bit receive window field is used for flow control.
- The 4-bit header length field specifies the length of the TCP header in 32-bit words. The TCP header can be of variable length due to the TCP options field.
- The optional and variable-length options field is used for some optional functions, which will not be covered in this course.
- The flag field has 6 bits. ACK is used to indicate that the value carried in the acknowledgment field is valid. RST, SYN, and FIN are used for connection setup and teardown.
  We will not cover PSH, URG and urgent data pointer field in this course.

TCP Sequence Number and Acknowledgement

TCP Sequence Number and Acknowledgement (ACK)

Sequence numbers (#'s):

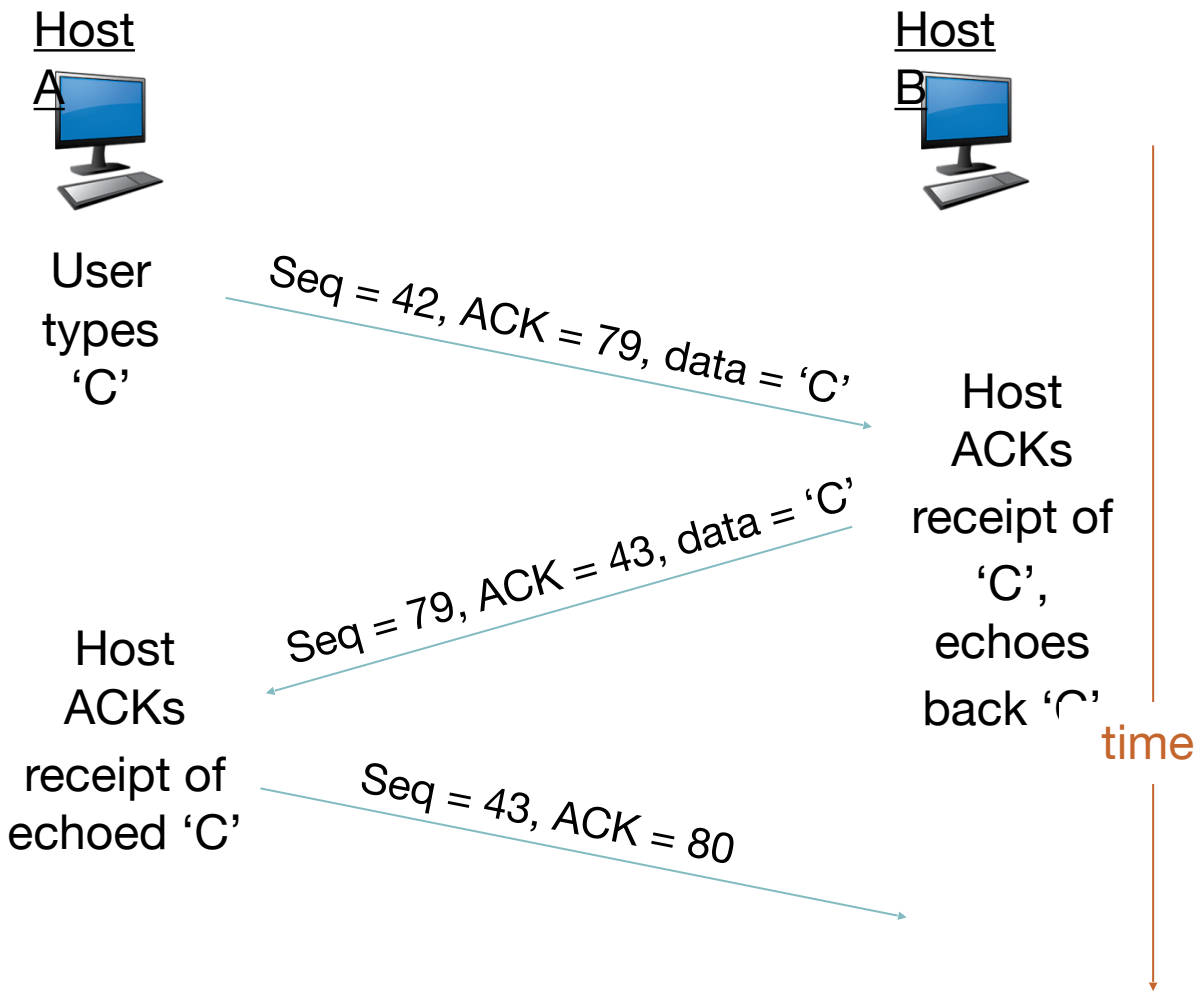
- Byte stream "number" of first byte in segment's data

ACKs:

- Sequence # of next byte expected from other side
- Cumulative ACK

How receiver handles out-of-order segments?

TCP specification doesn't say, it is up to the implementer.



Simple Telnet Scenario

Example

A web server needs to sends a webpage of 5,000 bytes to the client over a TCP connection. The maximum segment size (MSS) is 1,460 bytes and the total header and trailer size is 66 bytes. Assuming that the initial sequence number is 2,380. How many TCP segments need to be transmitted and what are the sequence numbers for each of these segments?

Max. no. of data bytes in each segment
= 1,460 – 66 = 1,394

5,000 / 1,394 = 3.6; hence, number of segments is 4.

Sequence numbers are 2,380 (1st segment);
2,380 + 1,394 = 3,774 (2nd segment);
3,774 + 1,394 = 5,168 (3rd segment);
5,168 + 1,394 = 6,562 (4th segment).

Round-trip Time and Timeout

TCP Round-trip Time and Timeout

How to set TCP timeout value?

- longer than RTT
  - but RTT varies
- too short: premature timeout
  - unnecessary retransmissions
- too long: slow reaction to segment loss

How to estimate RTT?

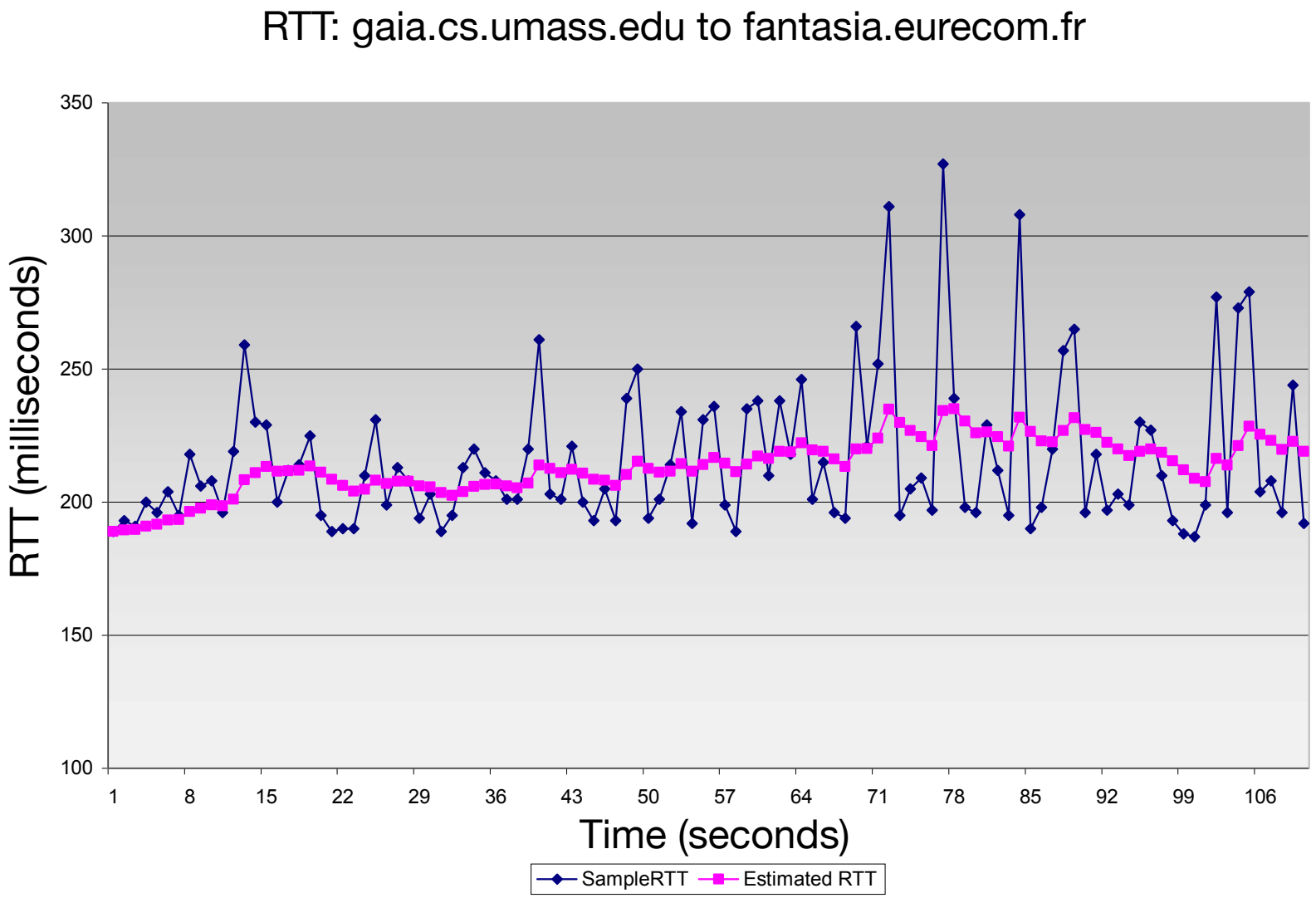
- `SampleRTT`: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- `SampleRTT` will vary, want estimated RTT "smoother"
  - average several recent measurements, not just current `SampleRTT`

TCP Round-trip Time and Timeout

$$\texttt{EstimatedRTT} = (1-\alpha)*\texttt{EstimatedRTT} + \alpha*\texttt{SampleRTT}$$

- Exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: $\alpha = 0.125$

Example RTT Estimation



TCP Round-trip Time and Timeout

Setting the timeout:

- `EstimatedRTT` plus "safety margin"
  - large variation in `EstimatedRTT` → larger safety margin
- first estimate of how much `SampleRTT` deviates from `EstimatedRTT`:

DevRTT = (1-β)*DevRTT +
β*|SampleRTT-EstimatedRTT|

(typically, β = 0.25)

Then set timeout interval:

TimeoutInterval = EstimatedRTT + 4*DevRTT

Example

Assuming that $\alpha = 0.25$ and $\beta = 0.4$. Further assume that at time t, estimatedRTT is 10.0 seconds and DevRTT is 2.0 seconds. Compute the estimatedRTT and DevRTT if the next sampleRTT is 20.0.

new estimatedRTT = (1 - α) × old estimatedRTT + α × sampleRTT

= 0.75 × 10.0 + 0.25 × 20.0 = 12.5 sec

new DevRTT = (1 - β) × old DevRTT + β × |SampleRTT - new EstimatedRTT| = 0.6 × 2.0 + 0.4 × |20.0 – 12.5| = 4.2 seconds

TCP Sender Events

Data received from application:

- Create segment with seq #
- Seq # is byte-stream number of first data byte in segment
- Start timer if not already running (think of timer as for oldest unacked segment)
- Expiration interval: TimeOutInterval

Timeout:

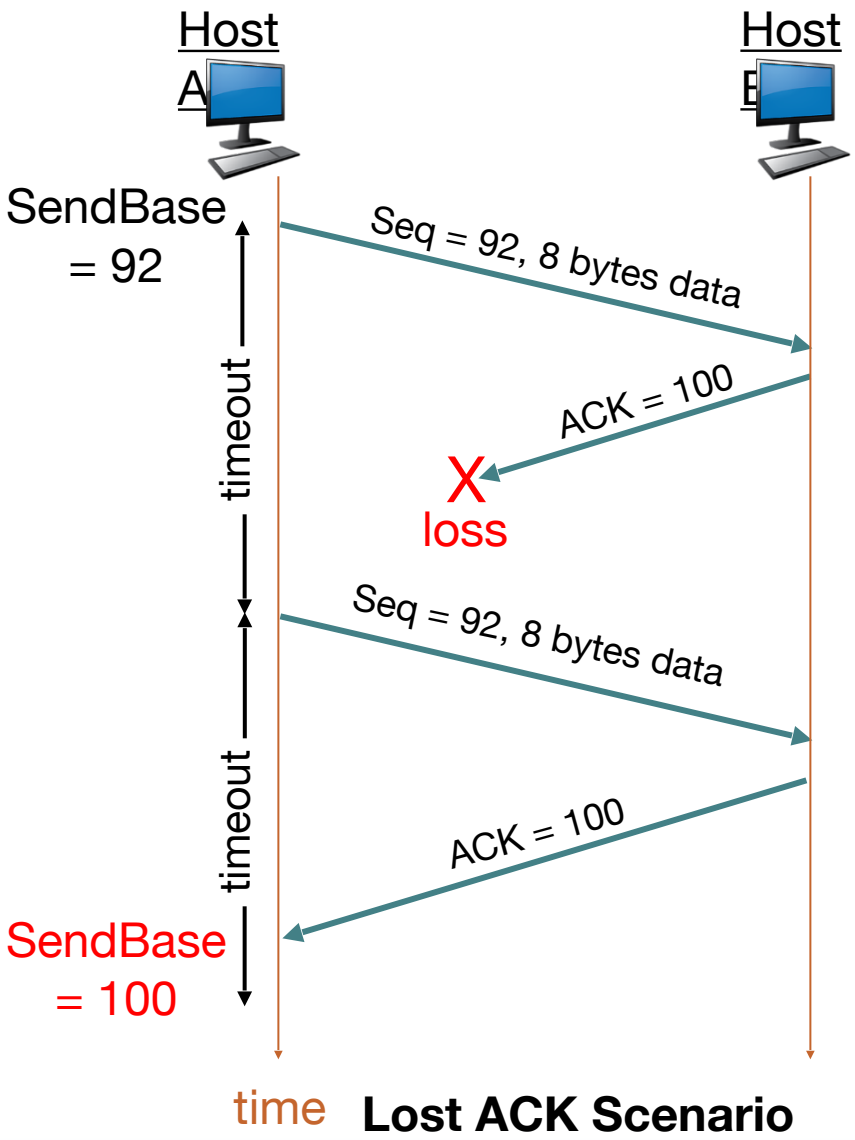
- Retransmit segment that caused timeout
- Restart timer

ACK received:

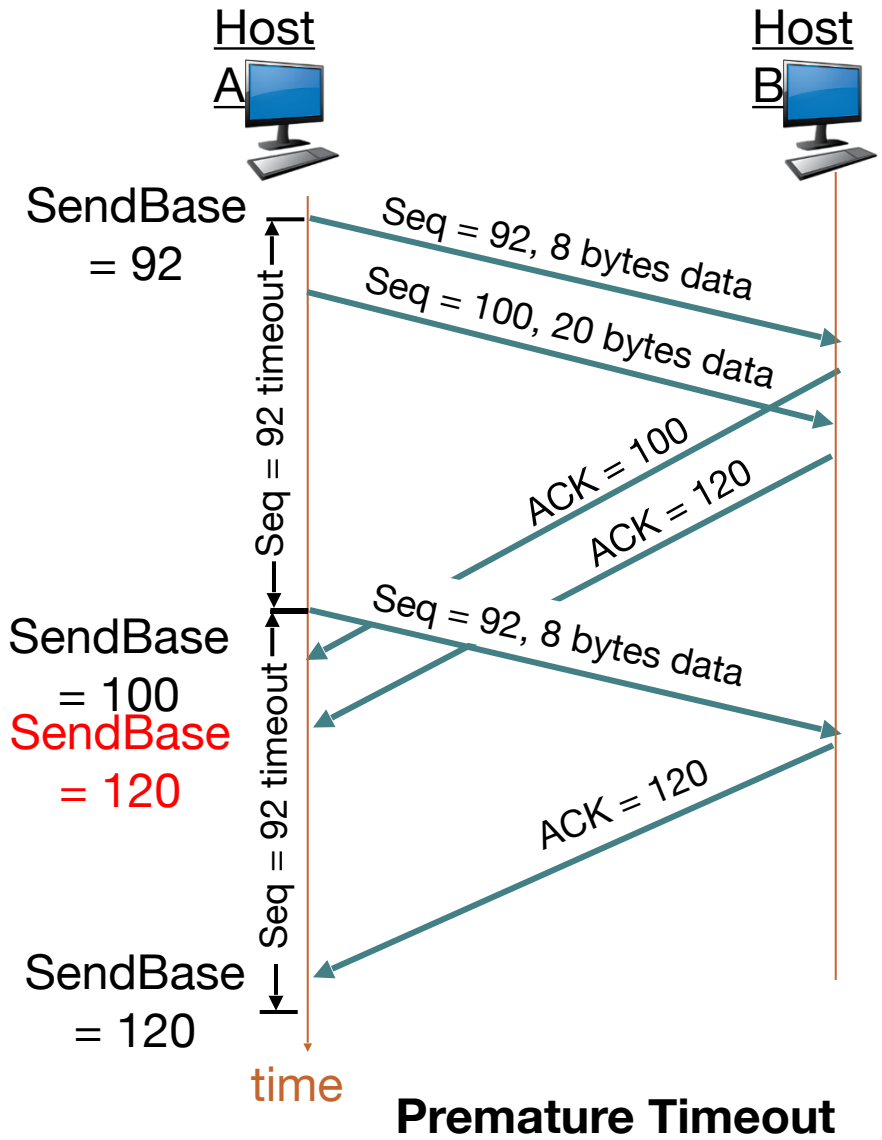
- If acknowledges previously unacked segments
  - update what is known to be acked
  - start timer if there are outstanding segments

Retransmission and Fast Retransmit

TCP: Retransmission Scenarios

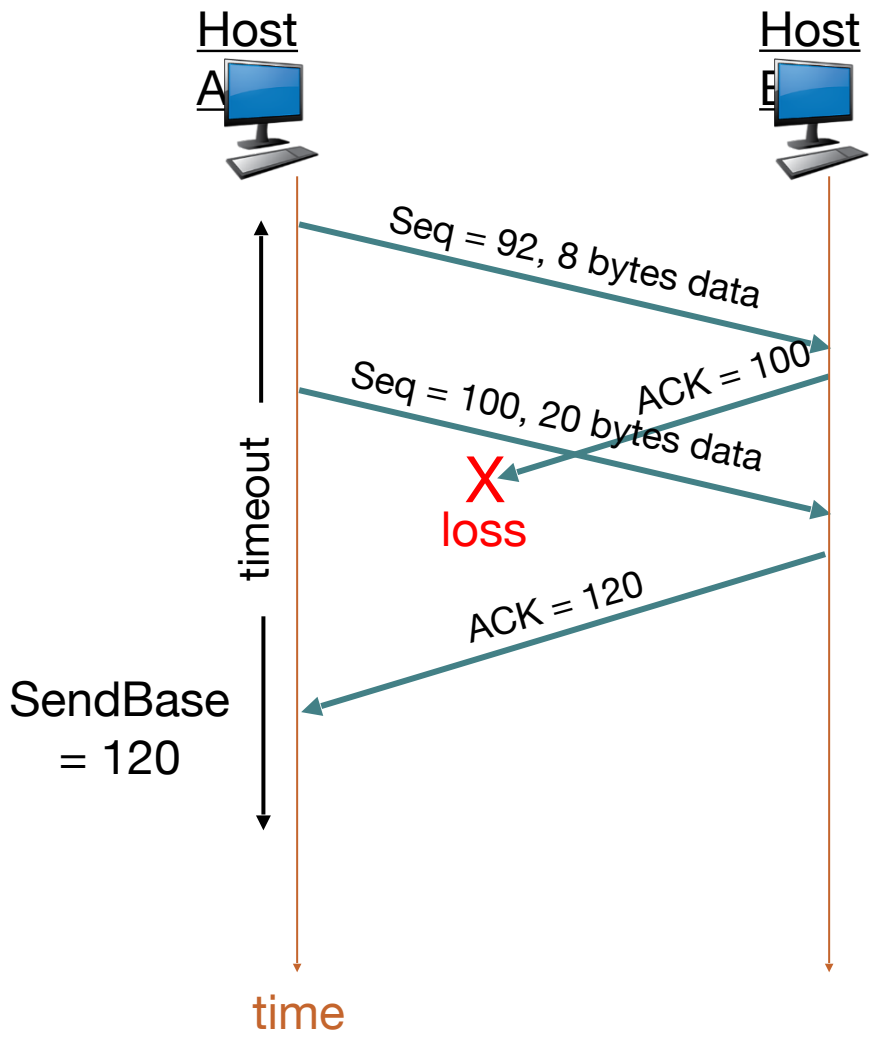


Lost ACK Scenario



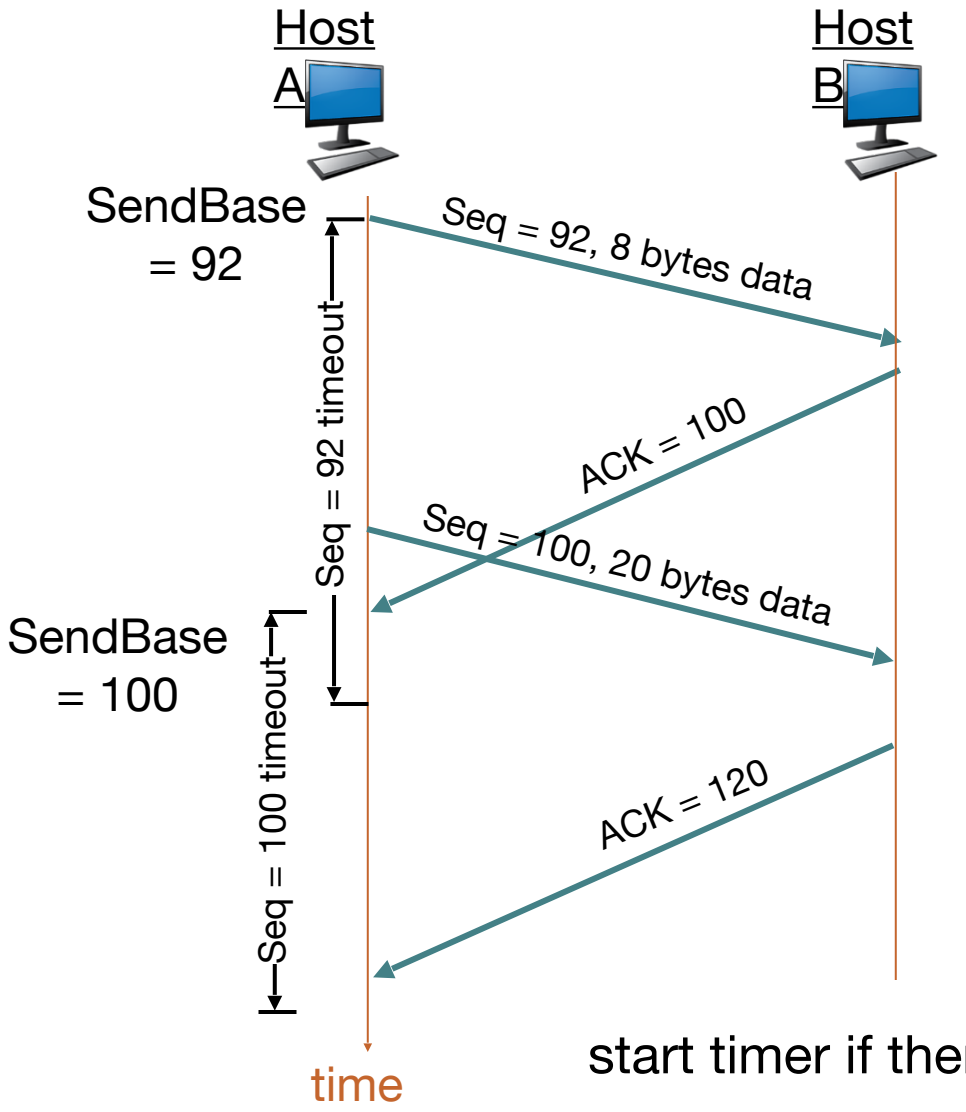
Premature Timeout

TCP: Retransmission Scenarios (more)



Cumulative ACK Scenario

TCP: Retransmission Scenarios (more)



start timer if there are outstanding segments

Fast Retransmit

Time-out period often relatively long:

- Long delay before resending lost packet.

Detect lost segments via duplicate ACKs.

- Sender often sends many segments back-to-back
- If segment is lost, there will likely be many duplicate ACKs.

If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:

- Fast retransmit: resend segment before timer expires

Fast Retransmit Algorithm

```
event: ACK received, with ACK field value of y
            if (y > SendBase) {
                SendBase = y
                if (there are currently not-yet-acknowledged
segments)
                    start timer
                }
            else {
                increment count of dup ACKs received for y
                if (count of dup ACKs received for y = 3) {
                    resend segment with sequence number y
                }
```

a duplicate ACK for already ACKed segment

fast retransmit

Notes: Fast Retransmit

- One of the problems with timeout-triggered retransmission is that the timeout period can be relatively long. When a segment is lost, this long timeout period forces the sender to delay resending the lost packet, thereby increasing the end-to-end delay.

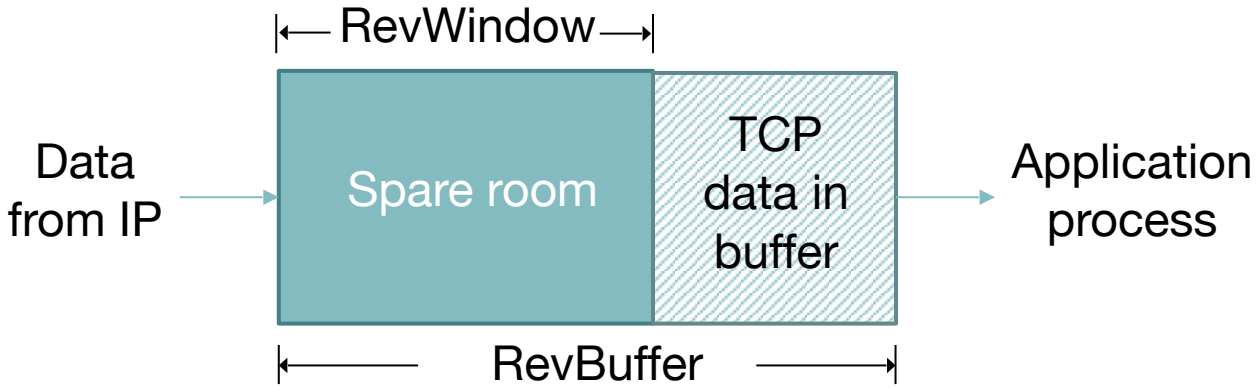
- The sender can often detect packet loss well before the timeout event occurs by noting so-called duplicate ACKs. A duplicate ACK is an ACK that acknowledges a segment for which the sender has already received an earlier acknowledgement.

- As the sender often sends a large number of segments back to back, if one segment is lost, there will likely be many back-to-back duplicate ACKs. If the TCP sender receives 3 duplicate ACKs for the same data, it takes this as an indication that the segment following the segment that has been ACKed three times has been lost.

Flow Control

Flow Control

- Receive side of TCP connection has a receive buffer:



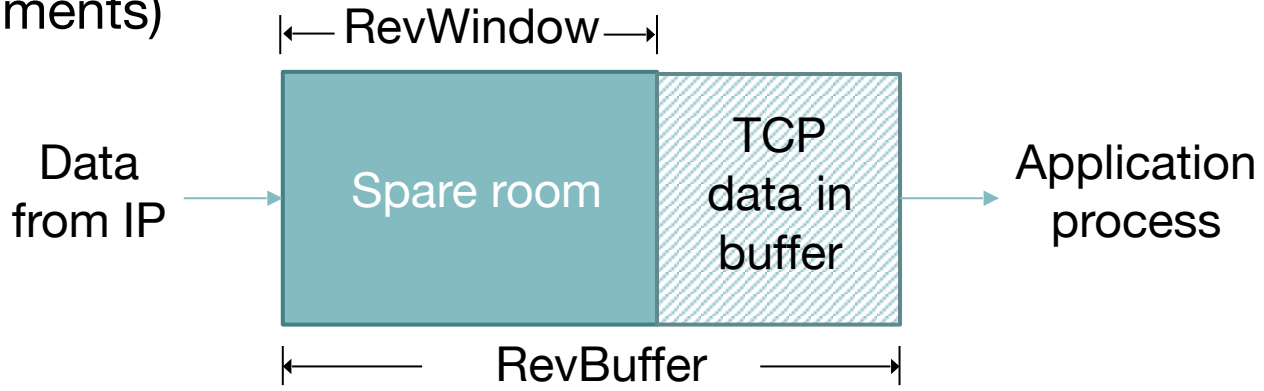
- Application process may be slow at reading from buffer.

- Speed-matching service: matching the send rate to the receiving application's drain rate.

> **Flow control**
> Sender won't overflow receiver's buffer by transmitting too much, too fast

TCP Flow Control: How it works

(Suppose TCP receiver discards out-of-order segments)

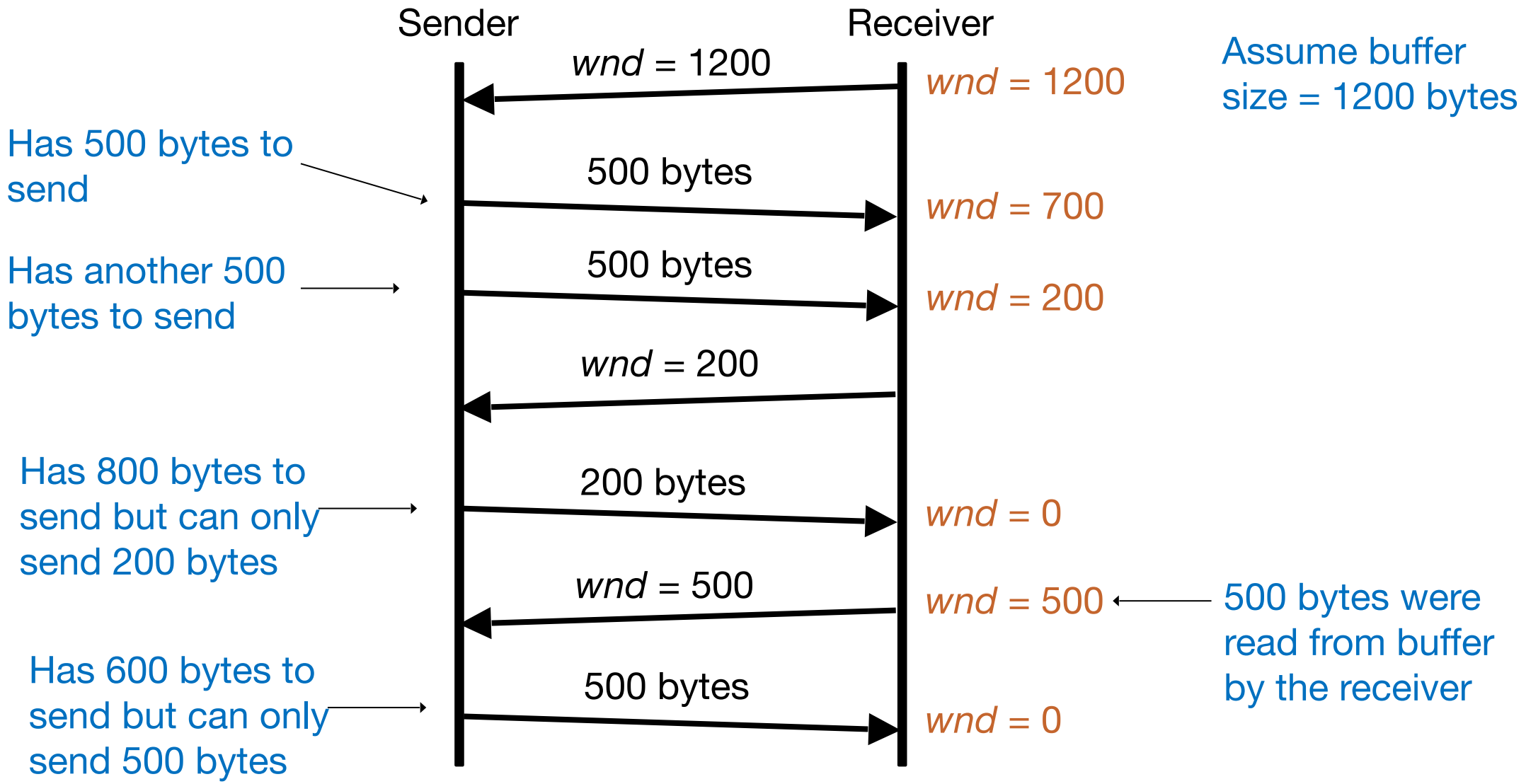


- Spare room in buffer
  = `RcvWindow`
  = `RcvBuffer-[LastByteRcvd - LastByteRead]`

- Rcvr advertises spare room by including value of `RcvWindow` in segments

- Sender limits unACKed data to `RcvWindow`
  - guarantees receive buffer doesn't overflow

TCP Flow Control: Example



Notes: TCP Flow Control

- TCP flow control is used to prevent the sender overflowing the receiver's buffer. This is achieved by the sender maintaining a variable called receive window.

- Suppose that Host A is sending a large file to Host B over a TCP connection. Host B allocates a receiver buffer with size denotes as RevBuffer. Define LastByteRead as the number of the last byte in the data stream read from the buffer by the application process in B, and LastByteRcvd as the number of last byte in the data stream that has arrived from the network that has been placed in the receive buffer at B. As TCP is not permitted to overflow the allocated buffer, we must have:

  $$LastByteRcvd - LastByteRead <= RcvBuffer$$

  The receive window, denoted RcvWindow is set to the amount of spare room in the buffer:

  $$RcvWindow = RcvBuffer - (LastByteRcvd - LastByteRead)$$

Connection Managements

TCP Connection Management

Recall:

TCP sender, receiver establish “connection” before exchanging data segments.

Initialise TCP variables:

- seq. #s
- buffers, flow control information (e.g. `RcvWindow`)

Client: connection initiator

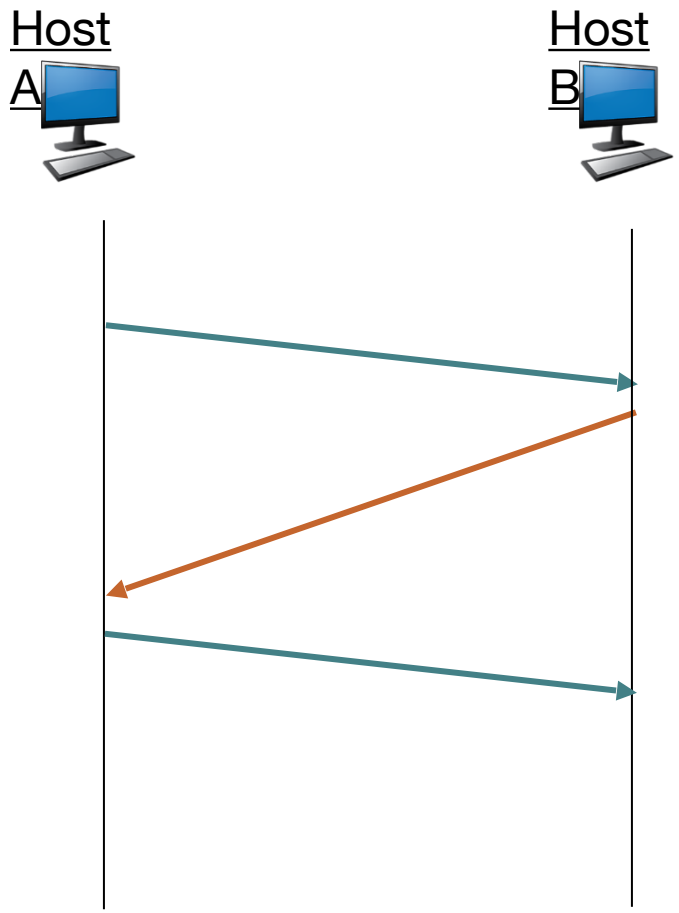
```
Socket clientSocket = new
Socket("hostname","port number");
```

Server: contacted by client

```
Socket connectionSocket =
welcomeSocket.accept();
```

TCP Connection Management

Three-way Handshake



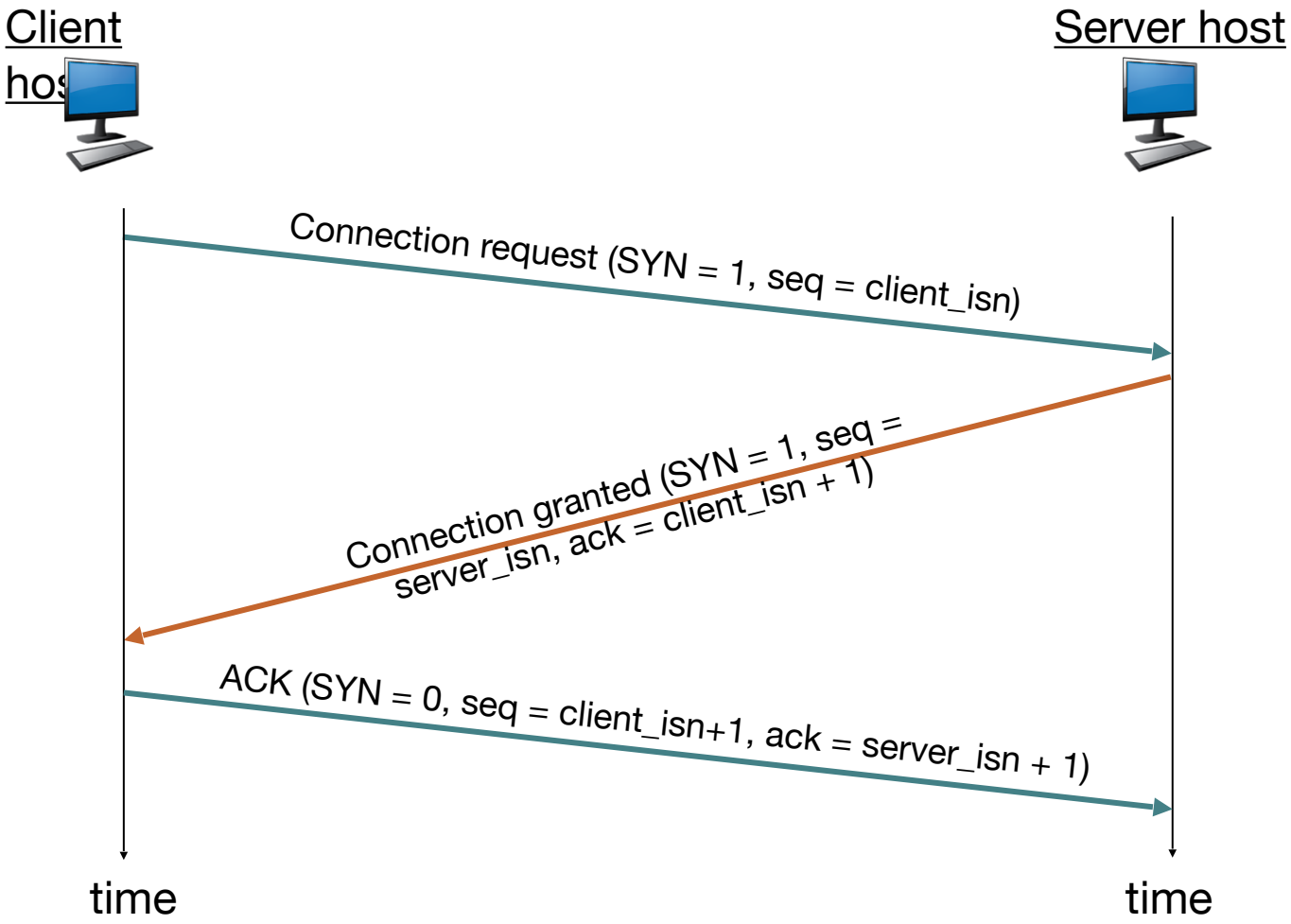
1. Client host sends TCP SYN segment to server
   - specifies initial seq #
   - no data

2. Server host receives SYN, replies with SYNACK segment
   - server allocates buffers
   - specifies server initial seq. #

3. Client receives SYNACK, replies with ACK segment, which may contain data

TCP Connection Management

Opening a connection:



isn: initial sequence number

Notes: Opening a Connection

Suppose a process in one host wants to initiate a connection with another process in another host, the client application process first informs the client TCP that it wants to establish a TCP connection with the TCP in the server in the following manner:

- The client-side TCP sends a TCP connection request message with no application-layer data, SYN bit set to 1, and a randomly chosen initial sequence number (client_isn) and puts this number in the sequence number field of the initial TCP SYN segment. This segment is encapsulated within an IP datagram and sent to the server.

- Once the IP datagram arrives at the server, the server extracts the TCP SYN segment from the datagram, allocates the TCP buffers to the connection, and then sends a connection-granted segment, which also does not contain any application data, to the client TCP. In the segment header, SYN is set to 1, the acknowledgement field is set to client_isn+1, and a randomly chosen server_isn into the sequence number field.

Notes: Opening a Connection

- Upon receiving the connection-granted segment, the client also allocates buffers and variables to the connection. The client host then sends the server yet another segment; this segment acknowledges the server's connection-granted segment. In the segment header, the SYN bit is set to zero, since the connection is established. This third stage of the three-way handshake may carry client-to-server data in the segment payload.

- Once the three steps have been completed, the client and server host can send segments containing data to each other.

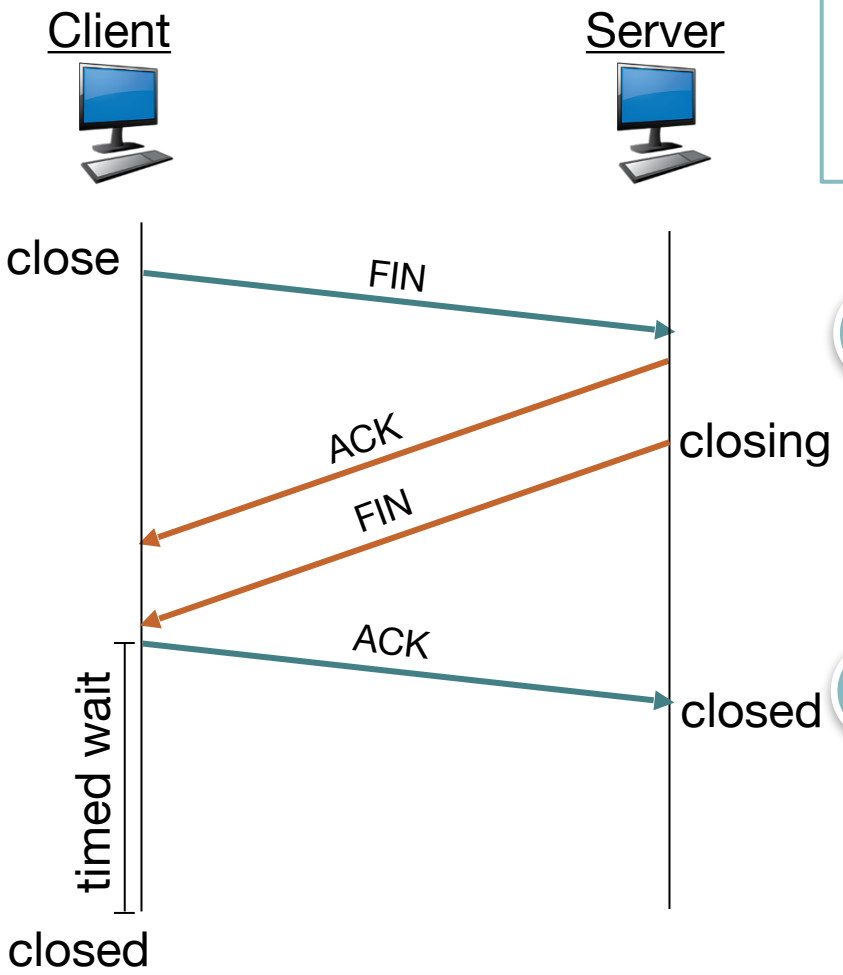
TCP Connection Management (cont'd)

Closing a connection:

client closes socket:

`clientSocket.close();`

1. Client end system sends TCP FIN control segment to server.



2. Server receives FIN, replies with ACK. Closes connection, sends FIN.

3. Client receives FIN, replies with ACK.
   - Enters "timed wait" - will respond with ACK to received FINs.

4. Server receives ACK. Connection closed.

Note:

Timed wait allows the lose of ACK in transit, which will cause FIN to be re-transmitted by the server.

Notes: Closing a Connection

- Suppose the client application decides to close the connection. (Note that the server could also choose to close the connection.) This causes the client TCP to send a TCP segment with FIN bit set to 1 and waits for a TCP segment from the server with an acknowledgement.

- When it receives the acknowledgement, the client TCP waits for another segment from the server with the FIN bit set to 1; after receiving this segment, the client TCP acknowledges the server's segment and enters another wait state. This allows the TCP client to resend the final acknowledgement in case the ACK is lost.

- After the wait, the connection formally closes and all resources on the client side are released. Note that the wait is necessary in case the acknowledgment sent by the client is lost and not received by the server, which will result in the server to re-transmit the segment with FIN set to 1.

Summary

Summary

Key points discussed in this topic:

- The transport-layer protocol provides for logical communication between application processes running on different hosts and are implemented in the end systems. On the sending side, the transport layer coverts the message it receives from a sending application process into segments. The transport layer then passes the segment to the network layer, which encapsulates it into a network-layer packet, which is also known as a datagram.

- UDP provides an unreliable, connectionless service to the invoking application. TCP provides a reliable, connection-oriented service to the invoking application.

- The TCP segment structure includes the source and destination port numbers, checksum field, sequence number field, acknowledgement number field, receive window field, header length field, optional and variable-length options field, flag fields and urgent data pointer.

Summary

Key points discussed in this topic (cont'd):

- Sequence number is the byte stream "number" of first byte in segment's data. Acknowledgement is the sequence number of the next byte expected from other side.

- Computing TCP Round Trip Time:

  $$\text{EstimatedRTT} = (1-\alpha)*\text{EstimatedRTT} + \alpha*\text{SampleRTT}$$

- Computing TCP Timeout:

  $$\text{DevRTT} = (1-\beta)*\text{DevRTT} + \beta*|\text{SampleRTT}-\text{EstimatedRTT}|$$

- Computing TimeoutInterval:

  $$\text{TimeoutInterval} = \text{EstimatedRTT} + 4*\text{DevRTT}$$

Summary

Key points discussed in this topic (cont'd):

- TCP flow control is used to prevent the sender overflowing the receiver's buffer. This is achieved by the sender maintaining a variable called receive window.

- TCP uses a three-way handshake to establish a connection between the client and server.