ECE 356/COMPSI 356
Computer Network Architecture

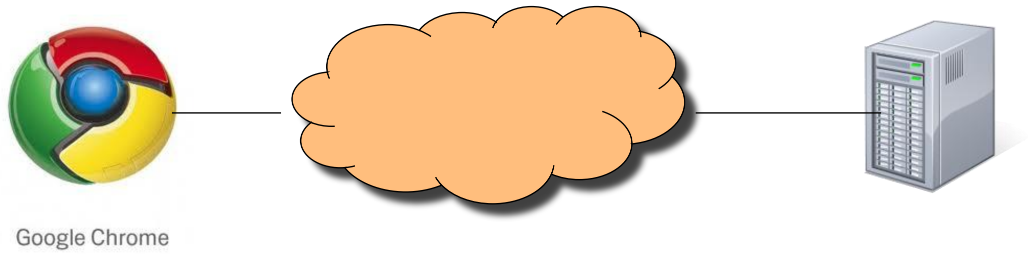
How TCP Achieves Reliable Operation

Monday October 28th, 2019

Recap

- Last lecture:
  - ➢Transport control
  - ➢UDP
- Readings for this lecture: **PD 5.2.2 – 5.2.6**

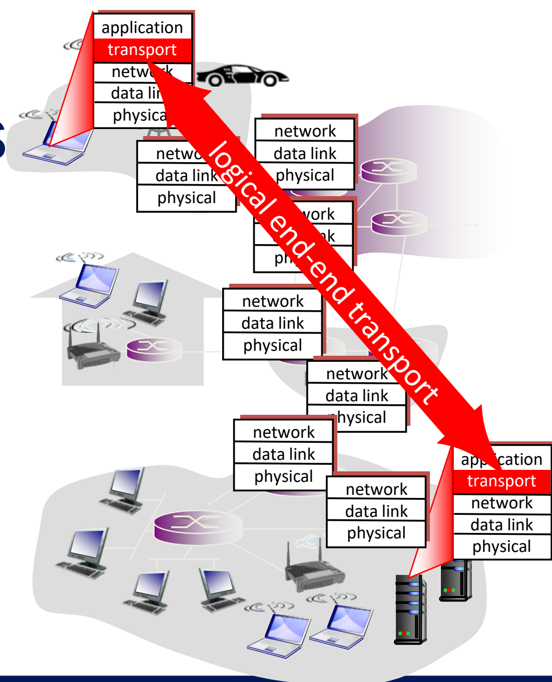
Transport Layer Design Goals



- A process-to-process communication channel
  - As if the hosts running processes were directly connected
  - Upper-layer: application
  - Lower-layer: network (IP)

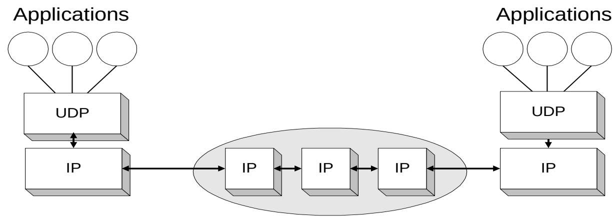
Transport Layer: End-to-End Protocols

- Implemented in end systems
- *Not implemented in network routers*
  - Routers act only on network-layer fields of the datagram



User Datagram Protocol (UDP)

- Minimal transport service: non-guaranteed datagram delivery
- *"A no-frills, bare-bones transport protocol"*
- "*Almost a null protocol*"
- Only provides:
  - Multiplexing by port number
  - Checksumming of data
- Has important advantages over TCP
- No connection setup: ***connectionless***



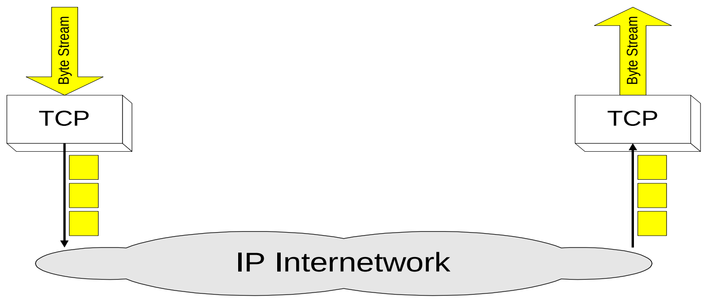
Lecture Outline

- **Transport Control Protocol (TCP)**
  - TCP segment format (PD 5.2.2)
  - Adaptive retransmission intervals (PD 5.2.6)
  - Reliable data transfer (PD 5.2.4)
  - TCP flow control (PD 5.2.4)
  - TCP connection establishment and termination (PD 5.2.3)
- Next lecture: TCP congestion control

Overview

TCP = Transmission Control Protocol

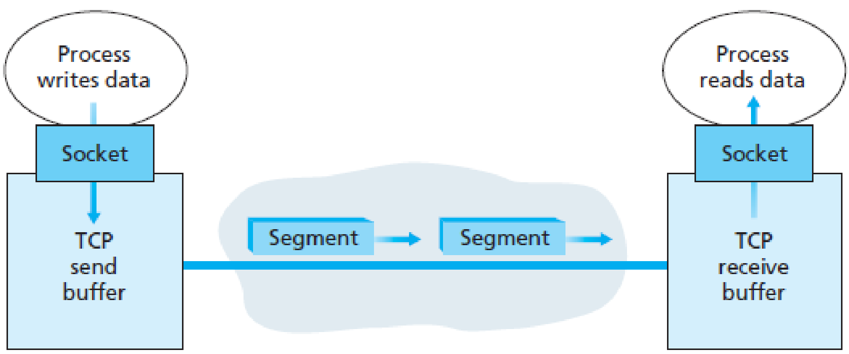
- Connection-oriented protocol
- Provides a **reliable unicast** end-to-end byte stream over an unreliable internetwork



7

Duke UNIVERSITY

TCP Manages a Byte Stream



- One-way shown for simplicity; bi-directional in general
- Transmitting *segments*: carrying a segment of the byte stream

8

Duke UNIVERSITY

Unique Design Challenges

- We've learned how to reliably transmit over a direct link
  - ➢ Coding/encoding, framing, sliding window
- What's new?
  1. Process-to-process communication → connection setup
  2. Heterogeneity
     - ➢ Bandwidth varies: how fast should the sender send?
     - ➢ RTT varies: when should a sender time out?
  3. Out of order
  4. Resource sharing
     - ➢ Many senders share a link in the middle of the network

TCP: Connection-Oriented

- Host processes must first "handshake" with each other
  - ➢ Exchange messages
  - ➢ Establish the parameters of data transfer
- Note: state is established in the ***end hosts***, <u>not the intermediate routers</u>
  - ➢ Intermediate routers are oblivious to TCP connections
  - ➢ Note the difference with circuit switching
- Full-duplex
- Point-to-point only: no broadcast, no multicast

TCP: Key Points to Remember

- Connection-oriented unicast operation
- Reliable, in-order byte stream service
- Flow control: not to overrun a receiver
- Congestion control: not to congest the network

11

Duke UNIVERSITY

Lecture Outline

- Transport Control Protocol (TCP)
  - ➢ **TCP segment format** (PD 5.2.2)
  - ➢ Adaptive retransmission intervals (PD 5.2.6)
  - ➢ Reliable data transfer (PD 5.2.4)
  - ➢ TCP flow control (PD 5.2.4)
  - ➢ TCP connection establishment and termination (PD 5.2.3)

12

Duke UNIVERSITY

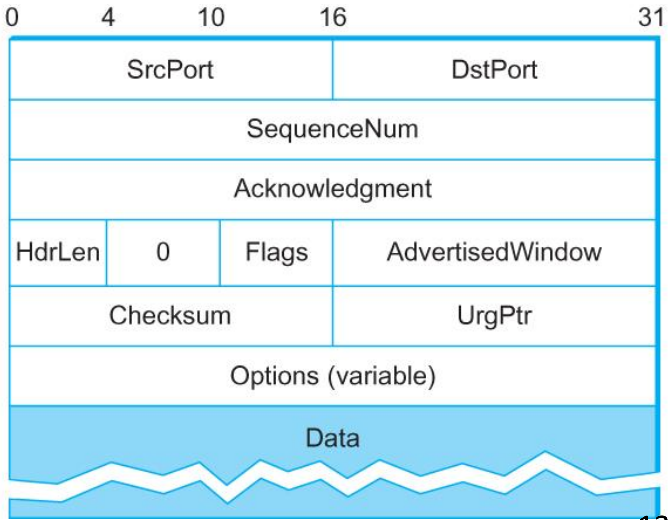
TCP Segment Format

TCP segments have a 20 byte header with >= 0 bytes of data

- Note similarities and differences with UDP header

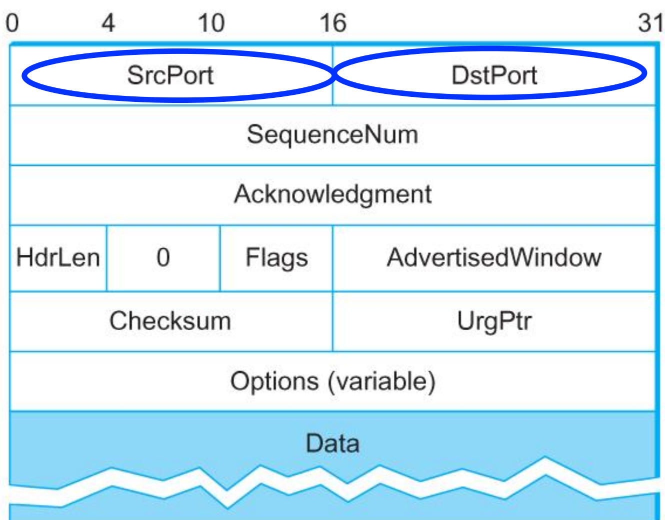
Source Port (2 bytes)	Destination Port (2 bytes)
Length (2 bytes)	Checksum (2 bytes)

UDP Header



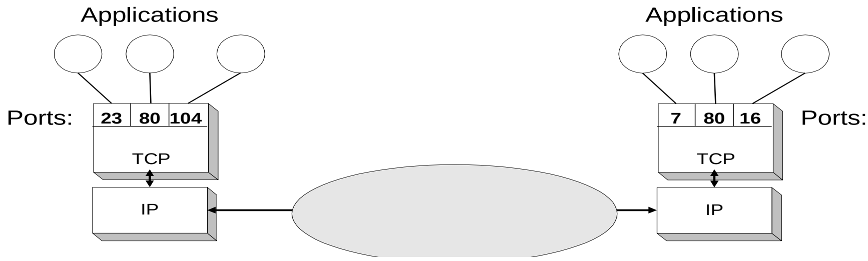
TCP Ports (1/2)

- Same as UDP
- A port number identifies the endpoint of a connection



TCP Ports (2/2)

- A pair <IP address, port number> identifies one endpoint of a connection
- Two pairs <client IP address, client port number> and <server IP address, server port number> identify a TCP connection

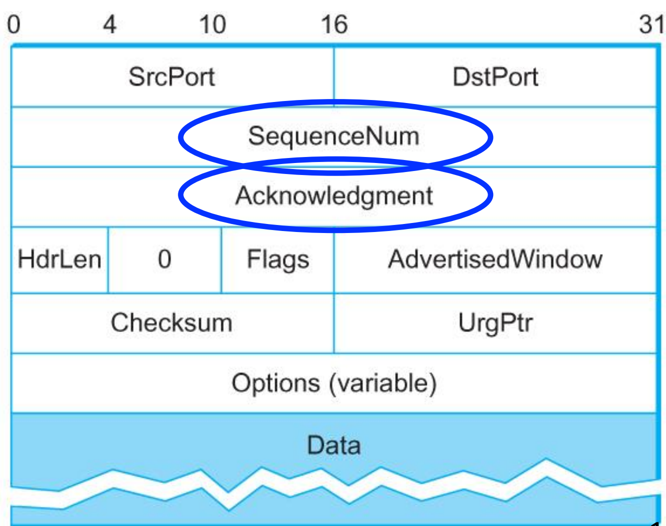


15

Duke UNIVERSITY

TCP: Reliable Communications

- Via sequence numbers and acknowledgements

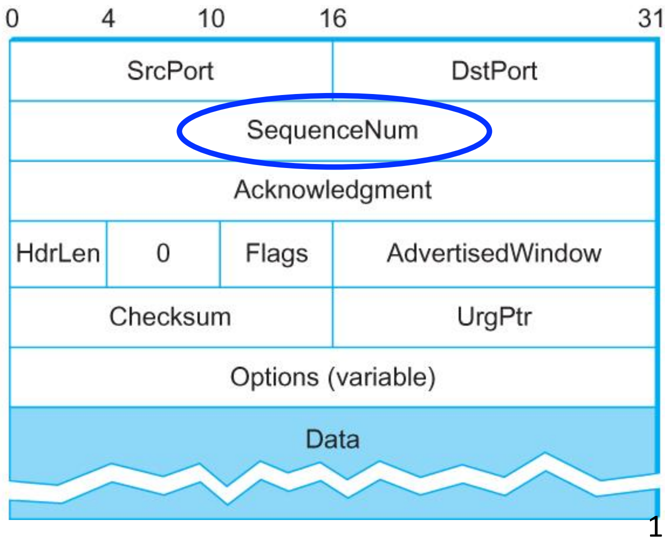


16

Duke UNIVERSITY

TCP Sequence Number

- Identifies the first byte in the segment
- Initial Sequence Number of a connection is set during connection establishment

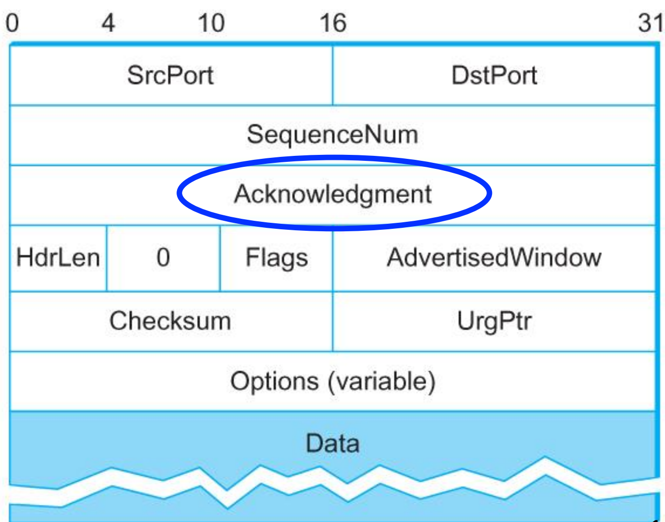


17

Duke UNIVERSITY

TCP Acknowledgement

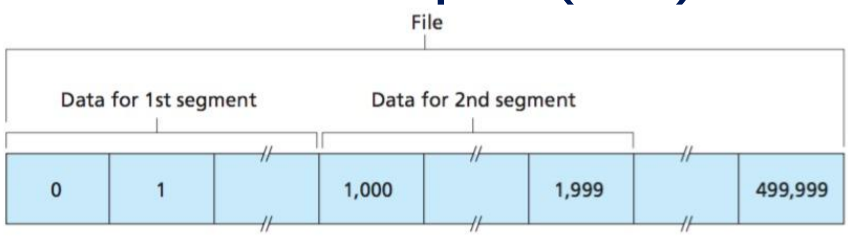
- Acknowledgements are piggybacked
- The AckNo contains the next SeqNo that a host is expecting
- ACK is cumulative



18

Duke UNIVERSITY

Sequence Numbers and ACKs: An Example (1/2)



- Example: host A is sending 500,000 bytes to host B, when maximum segment size is 1,000 bytes
  - First sequence number: 0, second one: 1,000, third one: 2,000…
  - Acknowledgement number is the next number expected from host B

19

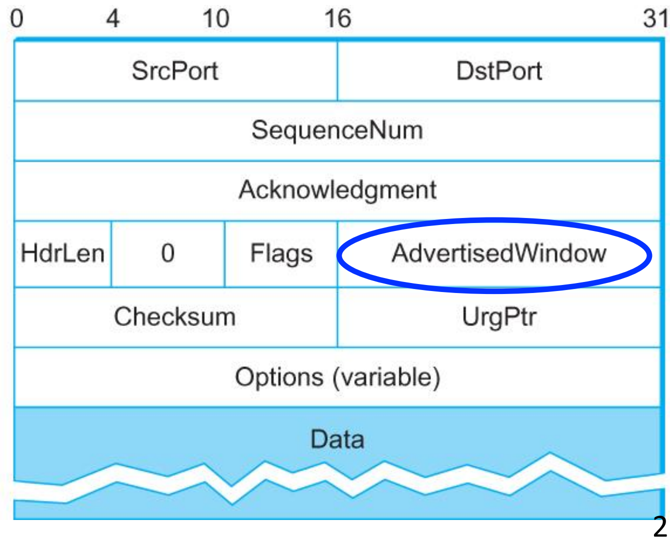
Sequence Numbers and ACKs: An Example (2/2)

- Acknowledgement number is the next number expected from host B
  - E.g., host A received bytes 0 – 535 from host B, and is about to send a segment: host A puts 536 in its acknowledgement field
- Acknowledgements are cumulative:
  - A received 0 – 535 and 900 – 1000: it puts 536 in the acknowledgement

20

TCP Advertised Window (1/2)

- Used to implement flow control
- Each side of the connection advertises the window size
- Window size is the *maximum number of bytes that a receiver can accept*

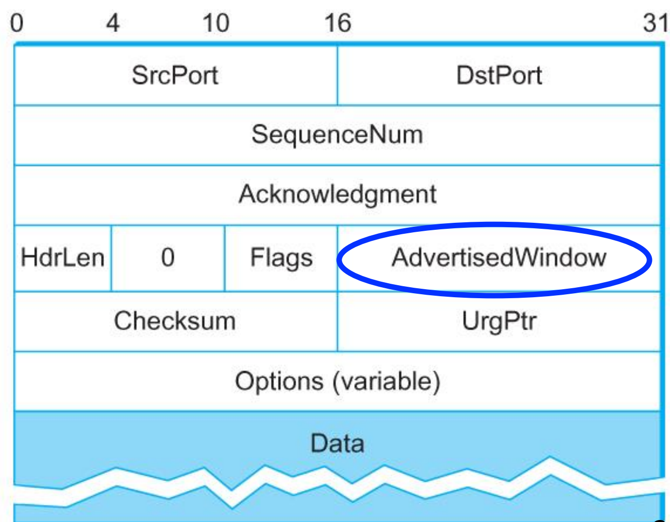


21

Duke UNIVERSITY

TCP Advertised Window (2/2)

- Included in every segment: **dynamic**
- Maximum window size is 2^{16}-1= 65,535 bytes
  - ➢ Problematic for high-speed links



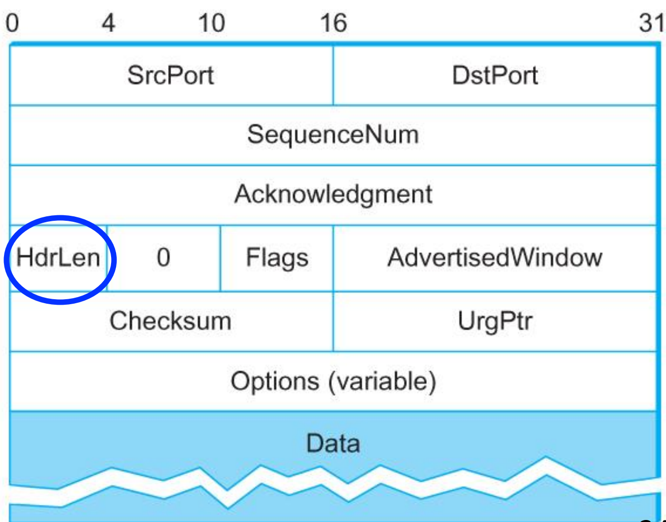
22

Duke UNIVERSITY

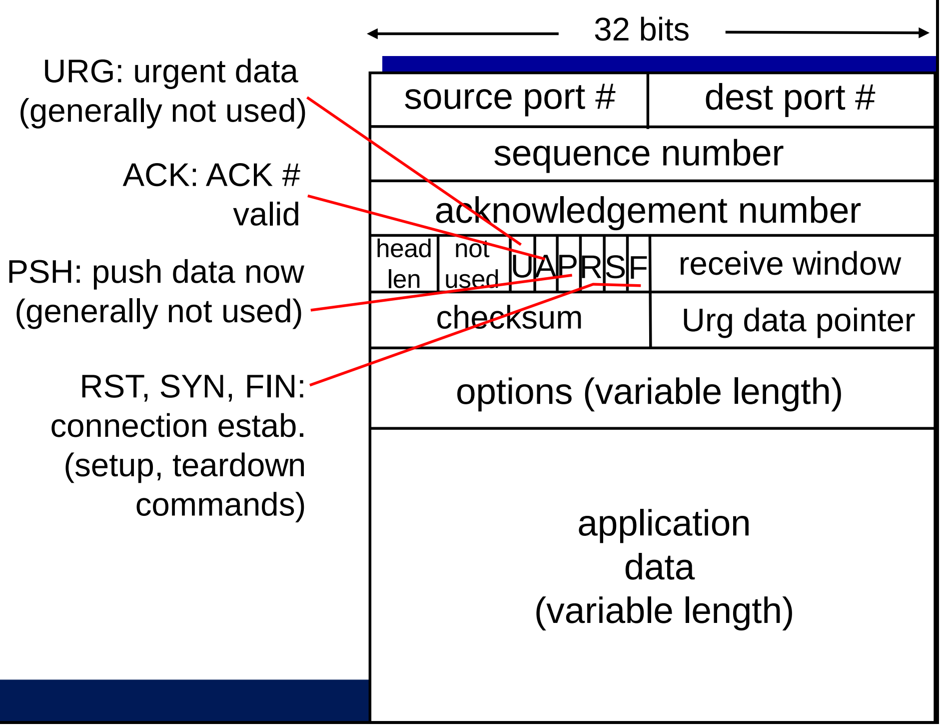
A Simplified TCP Process

TCP Header Length

- Variable-length header
  - Minimum: 20 bytes
- Header length: a 4-bit field
- Length of header in 32-bit words



TCP Flag Bits



Duke UNIVERSITY

TCP Flag Bits: URG, ACK, PSH

- URG: Urgent pointer is valid (not encouraged to use)
  - If the bit is set, the following bytes contain an urgent message in the range:

    SeqNo <= urgent message < SeqNo+urgent pointer
- ACK: Acknowledgement Number is valid
  - Segment contains a valid ACK
- PSH: PUSH Flag (generally not used)
  - Notification from sender to the receiver that the receiver should pass all data that it has to the application.
  - Normally set by a sender when the sender's buffer is empty

26

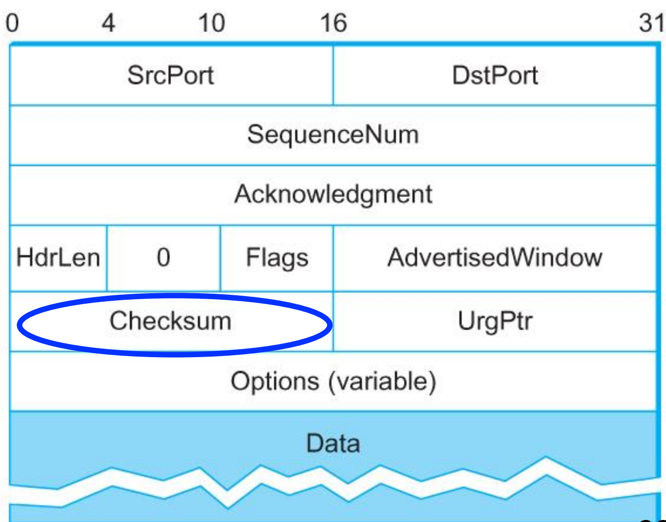
Duke UNIVERSITY

TCP Flag Bits: Connection Establishment

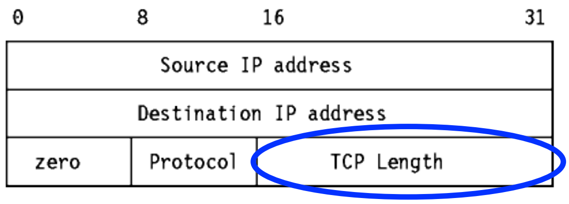
- RST: Reset the connection
  - Receiver of a RST terminates the connection and indicates higher layer application about the reset
- SYN: Synchronize sequence numbers
  - Sent in the first packet when initiating a connection
- FIN: Sender is finished with sending
  - Used for closing a connection

TCP Checksum

- Similar mechanism as UDP
- Cover TCP header, data, and a **pseudo-header**



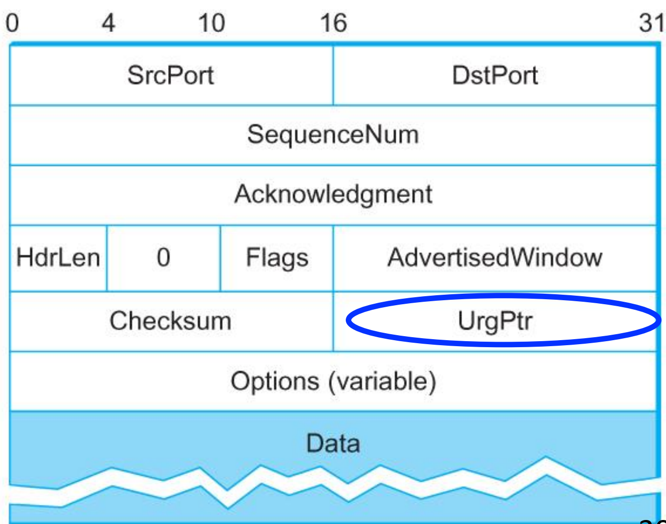
TCP Pseudo-Header



- Interesting part: TCP length
  - The length of the TCP segment, including both header and data
  - *Not a specific header field: it is computed*
- If TCP length is odd, one pad byte of zero will be added to the end for a 16-bit checksum computation

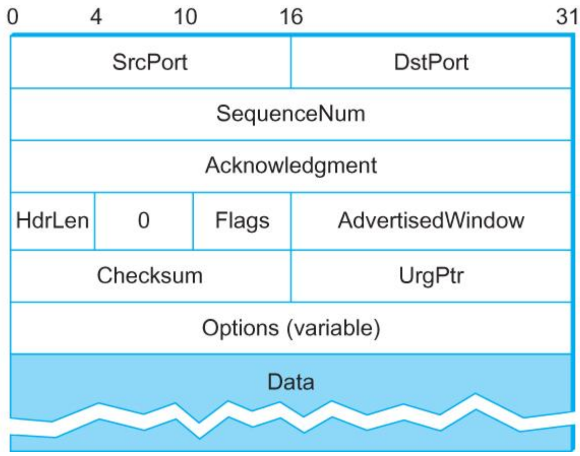
TCP URG Pointer

- Urgent Pointer:
  - Only valid if URG flag is set



TCP Segment Format: Key Points to Remember

- Like UDP, provides for demultiplexing and checksumming
- Also has dedicated fields for
  - ➢ Reliable communications
  - ➢ Flow control
  - ➢ Connection establishment



31

Duke UNIVERSITY

Lecture Outline

- Transport Control Protocol (TCP)
  - ➢ TCP segment format (PD 5.2.2)
  - ➢ **Adaptive timeout intervals** (PD 5.2.6)
  - ➢ **Reliable data transfer** (PD 5.2.4)
  - ➢ TCP flow control (PD 5.2.4)
  - ➢ TCP connection establishment and termination (PD 5.2.3)

32

Duke UNIVERSITY

Reliable Data Transfer in TCP

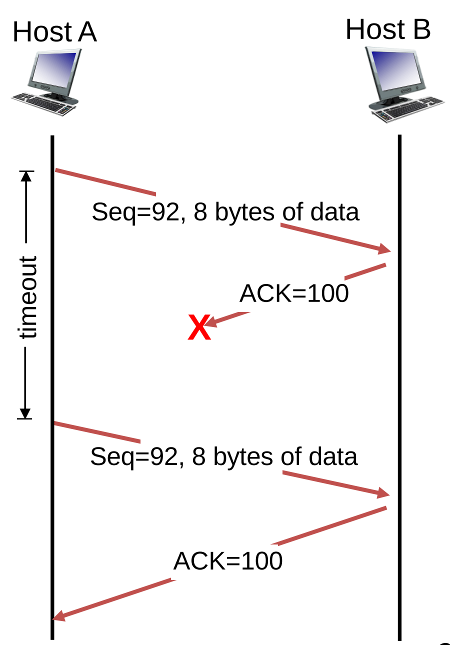
- Ensures that the data stream read out of the TCP receive buffer is:
  - Without gaps
  - Without duplication
  - In sequence
- *Exactly the same* byte stream that was sent

33

Duke UNIVERSITY

Same Core Principles as What We Studied for Link Layer Reliability

- Differences include:
  - Adaptive timeout values
  - One timeout variable per connection
  - Retransmissions triggered by timeouts and duplicate acknowledgements

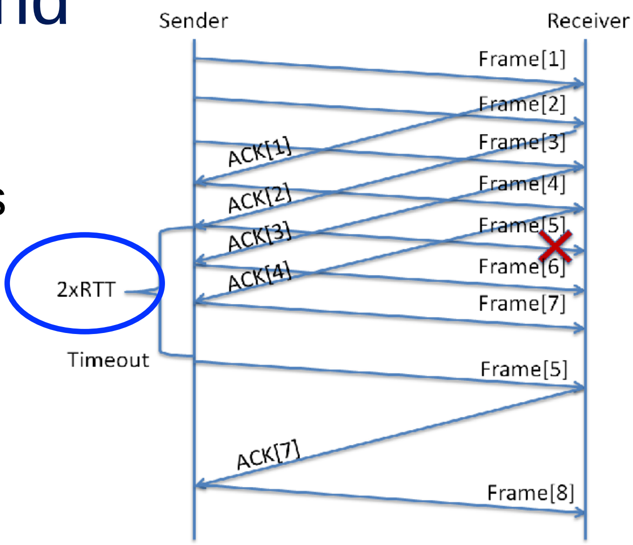


34

Duke UNIVERSITY

RTT Estimation and Timeout

- Reliable communications requires timeouts and retransmissions
- How should we set timeout values?
    - Assumed to be given before



How to Set TCP Timeout Value?

- *Definitely* needs to be longer than RTT
    - But RTT varies in practice
- If timeout is too short:
    - Premature timeout
    - Unnecessary retransmissions
- If timeout is too long:
    - Slow reaction to segment loss
    - Large average delays when the number of retransmissions is large

RTT Estimation and Timeout

- Set timeout to RTT + “safety margin”
- Two parts:
  - ➢Estimating RTT
  - ➢Calculating the additional margin

37

Duke UNIVERSITY

Estimating RTT

- *SampleRTT*: measured time from segment transmission until ACK receipt
  - ➢Ignore retransmissions
- SampleRTT varies. We want estimated RTT to be “smoother”
- Solution: average several recent measurements, not just current SampleRTT

38

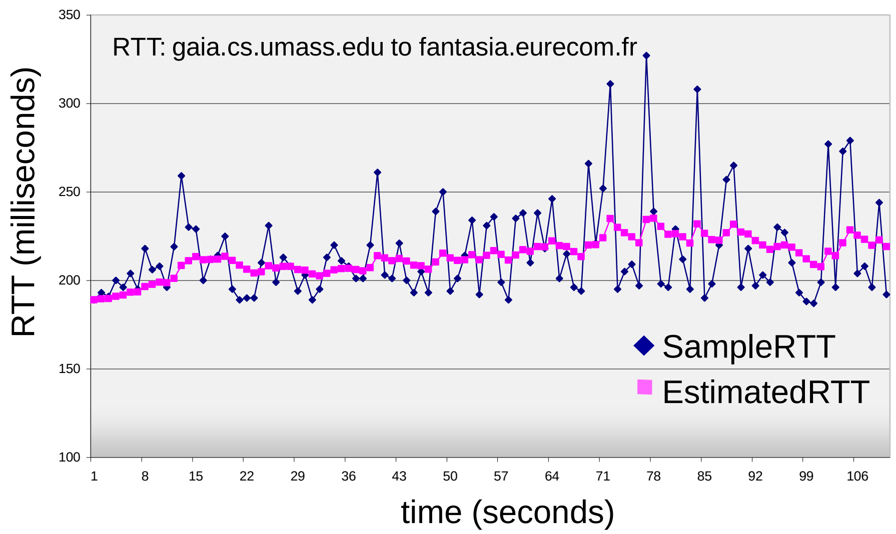
Duke UNIVERSITY

RTT Estimates: Exponential Weighted Moving Average

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

- Influence of past sample decreases exponentially fast
- Typical value: $\alpha = 0.125$
  - 1/8: efficient implementation due to the use of the power of 2

Sample And Estimated RTT: An Example



“Safety Margin”: Based on the Variability in RTT

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

- Also exponentially weighed
- Typical value: β = 0.25
  - ➢ Also a power of 2: 1/4

Setting and Managing Timeout Interval

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

estimated RTT

“safety margin”

- Initial value: 1s
- When a timeout occurs, TimeoutInterval is doubled
  - ➢ Goes back to the formula-based calculation after a segment is received

Adaptive Timeout Intervals: Key Points to Remember

- Timeout interval is not fixed
  - It is set to a sender-estimated **estimated RTT** + **a safety margin**
  - Both are dynamic metrics, recalculated with each recorded RTT
- Estimated RTT is calculated based on exponential averaging of sample RTT values
- Safety margin is calculated based on the variability in the RTT

43

Duke UNIVERSITY

Lecture Outline

- Transport Control Protocol (TCP)
  - TCP segment format (PD 5.2.2)
  - Adaptive timeout intervals (PD 5.2.6)
  - **Reliable data transfer** (PD 5.2.4)
  - TCP flow control (PD 5.2.4)
  - TCP connection establishment and termination (PD 5.2.3)

44

Duke UNIVERSITY

Reliable Data Transfer in TCP

- Familiar reliable communication mechanisms
  - Cumulative acknowledgements
- New: previously assumed that an individual timer is associated with each transmitted packet
  - TCP uses a single retransmission timer
- New: retransmissions triggered by:
  - Timeout events
  - Duplicate ACKs

3 Types of TCP Sender Events

- Data received from the application
- Timeout
- ACK received

Duke UNIVERSITY

TCP Sender Events: Data Received from an Application

- Create segment with seq #
  - Seq # is byte-stream number of first data byte in segment
- Pass the segment to IP
- Start timer if not already running
  - Think of timer as for oldest unacked segment
  - Expiration interval: TimeOutInterval

47

Duke UNIVERSITY

TCP Sender Events: Timeout and ACK Received

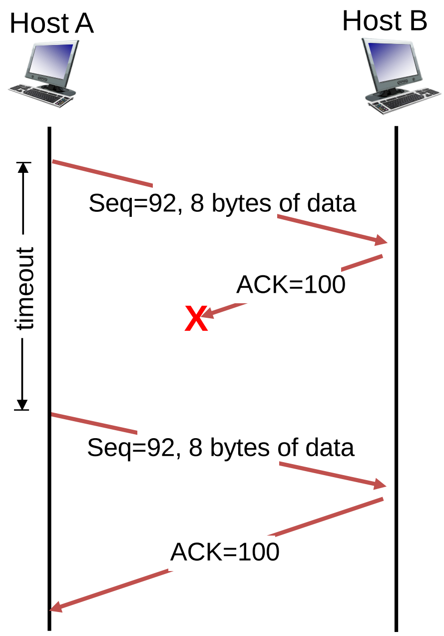
- **Timeout:**
  - Retransmit segment that caused timeout
  - Restart timer
- **ACK Received:**
  - If ack acknowledges previously unacked segments:
    - Update what is known to be ACKed
    - Start timer if there are still unacked segments

48

Duke UNIVERSITY

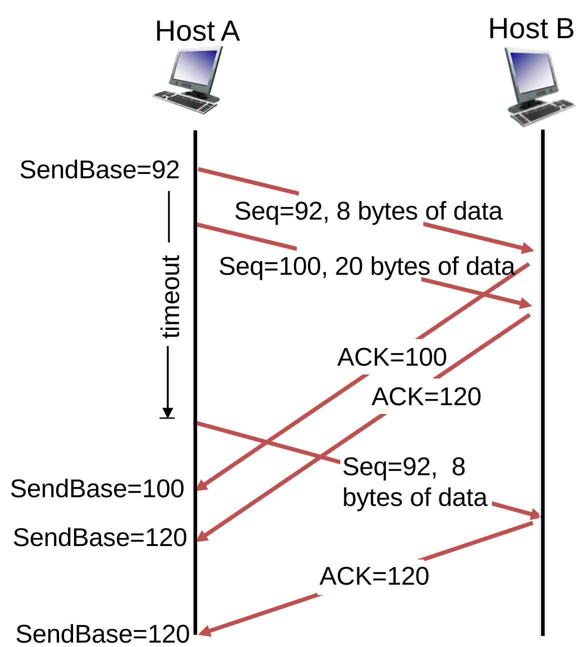
TCP Retransmission Scenarios (1/3)

- Retransmission due to a lost ACK
- Segment times out, and is retransmitted
- New (duplicate) data ignored
- ACK retransmitted



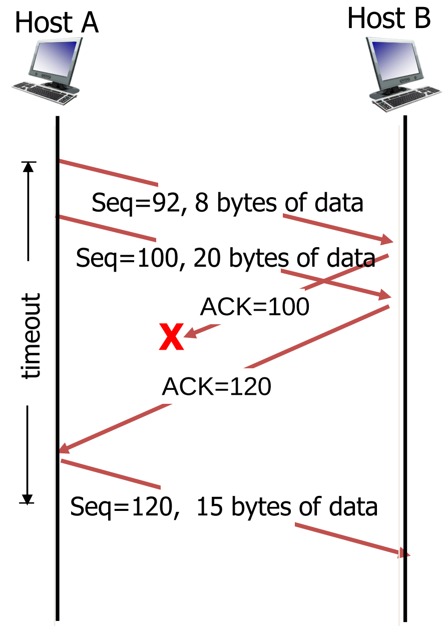
TCP Retransmission Scenarios (2/3)

- Cumulative ACKs
- Both ACKs do not make it back in time
- Segment 92 is retransmitted
- Both segments are acknowledged
    - ➢ If ACK arrives before the new timeout value, segment 100 is not retransmitted



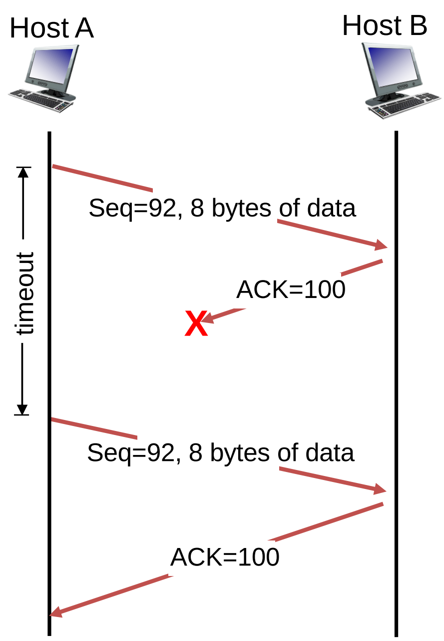
TCP Retransmission Scenarios (3/3)

- Cumulative ACKs
- ACK 100 is lost
- ACK 120 arrives before timeout
- Host A knows that **everything** was received
  - ➢ No data is retransmitted



Doubling Timeout Interval After a Timeout

- Set to twice the previous value
  - ➢ Rather than deriving from RTT measures
- Double for each successive timeout
  - ➢ Similar to exponential backoff
- A form of congestion control
  - ➢ Assume that losses are due to network being overloaded



TCP ACK Generation

Event at receiver	*TCP receiver action*
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed.	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK.
Arrival of in-order segment with expected seq #. One other segment has ACK pending.	Immediately send single cumulative ACK, ACKing both in-order segments.
Arrival of out-of-order segment higher-than-expect seq. #s. Gap detected.	Immediately send ***duplicate ACK,*** indicating seq. # of next expected byte.
Arrival of segment that partially or completely fills gap.	Immediate send ACK, provided that segment starts at lower end of gap.

TCP Fast Retransmit (1/2)

- 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

TCP Fast Retransmit (2/2)

- If sender receives 3 ACKs for same data ("*triple duplicate ACKs*") resend unacked segment with smallest seq #
  - Likely that unacked segment is lost
  - Do not wait for timeout

TCP Reliable Data Transfer: Key Points to Remember

- TCP uses **cumulative acknowledgements** and retransmissions to ensure reliable in-order data transfer
  - Similar to what we have seen for link-layer reliability before
- Uses one retransmission timer per connection
- Doubles timeout interval after a timeout
  - Form of congestion control
- Uses *duplicate acknowledgements* to retransmit segments before timeouts

Lecture Outline

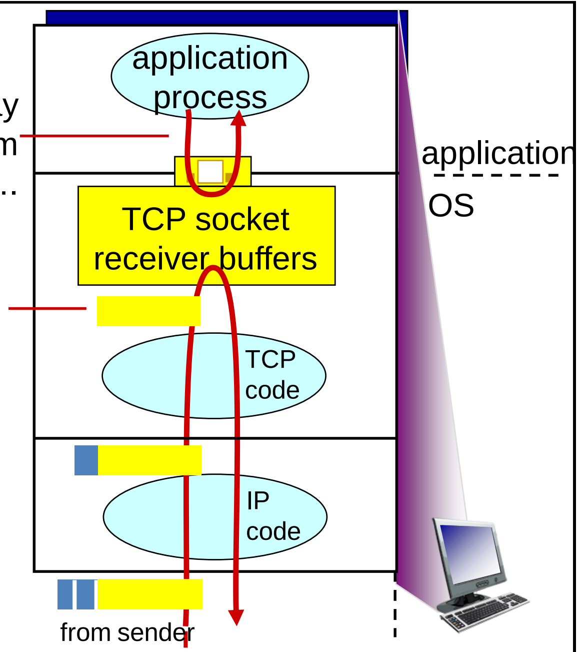
- Transport Control Protocol (TCP)
  - TCP segment format (PD 5.2.2)
  - Adaptive retransmission intervals (PD 5.2.6)
  - Reliable data transfer (PD 5.2.4)
  - **TCP flow control** (PD 5.2.4)
  - TCP connection establishment and termination (PD 5.2.3)

Flow Control: Definition (1/2)

- TCP hosts set up *receive buffers* for the data, on both sides of a connection

Application may remove data from TCP socket buffers ….

… slower than TCP receiver is delivering (sender is sending)



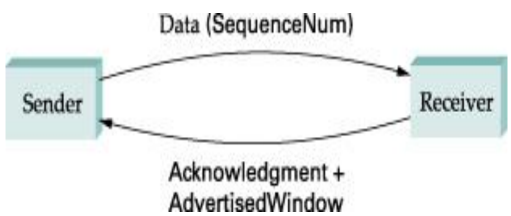
Receiver protocol stack

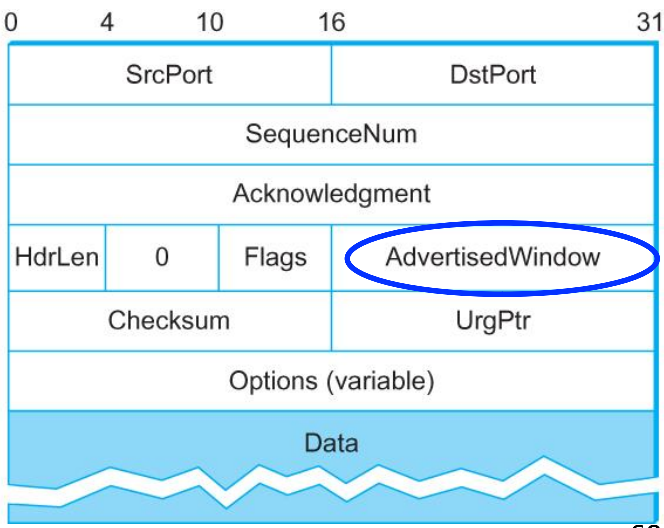
Flow Control: Definition (1/2)

- **Flow control**: receiver controls sender so sender won't overflow receiver's buffer by transmitting too much, too fast
- **Congestion control**: throttling the sender due to congestion on the network

Flow Control: Use Receive Window Size

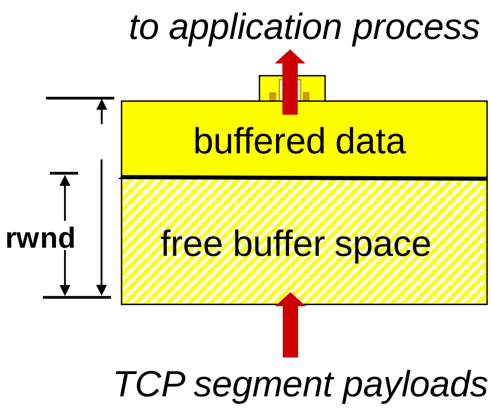
- Same concept as we've seen for link layer flow control
- Explicitly signaled in segment headers





TCP Flow Control (1/2)

- Receiver "advertises" free buffer space by including *rwnd* value in TCP header of receiver-to-sender segments
  - RcvBuffer size set via socket options (typical default is 4096 bytes)



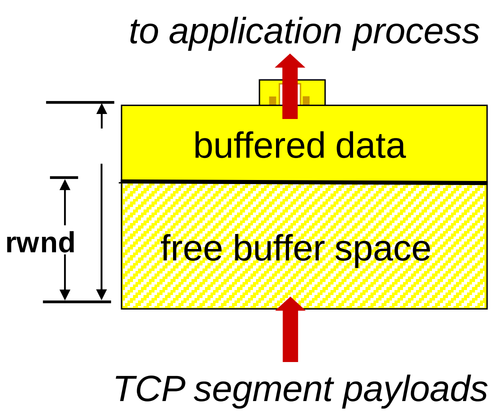
Receiver-side buffering

61

Duke UNIVERSITY

TCP Flow Control (2/2)

- Sender limits amount of unacked ("in-flight") data to receiver's rwnd value
- Guarantees receive buffer will not overflow



Receiver-side buffering

62

Duke UNIVERSITY

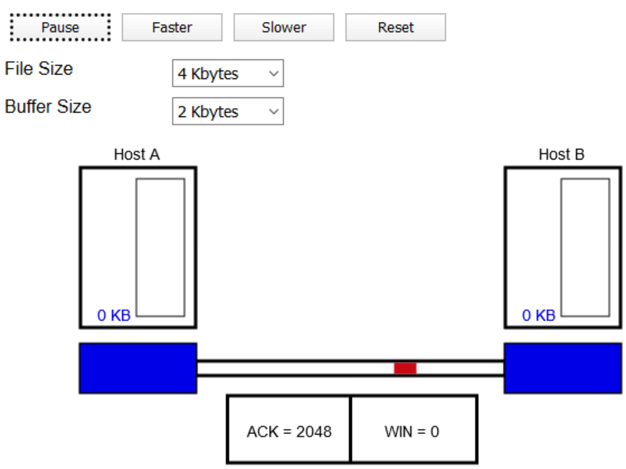
Window Probes

- What if a receiver advertises a window size of zero?
  - ➢ Problem: Receiver can't send more ACKs as sender stops sending more data
- Design choices
  - ➢ Receivers send duplicate ACKs when window opens
  - ➢ Sender sends periodic 1 byte probes

Duke UNIVERSITY

Flow Control: An Interactive Demonstration

- A visualization of the process is available at:
  - ➢ https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/flow-control/index.html



64

Duke UNIVERSITY

No Flow Control in UDP: What Happens?

- Segments may be lost at the receiver due to buffer overflow
- Typical implementation: UDP appends segments in a finite-sized buffer at the receiving process
- If the process does not read the segments fast enough, the buffer will overflow and the segments will be dropped

65

Duke UNIVERSITY

TCP Flow Control: Key Points to Remember

- **Receiver controls sender** by <u>explicitly stating</u> how much space is available in the receive buffer
  - Information included in segment headers
  - Highly dynamic
- Approach guarantees that receive buffer will not overflow

66

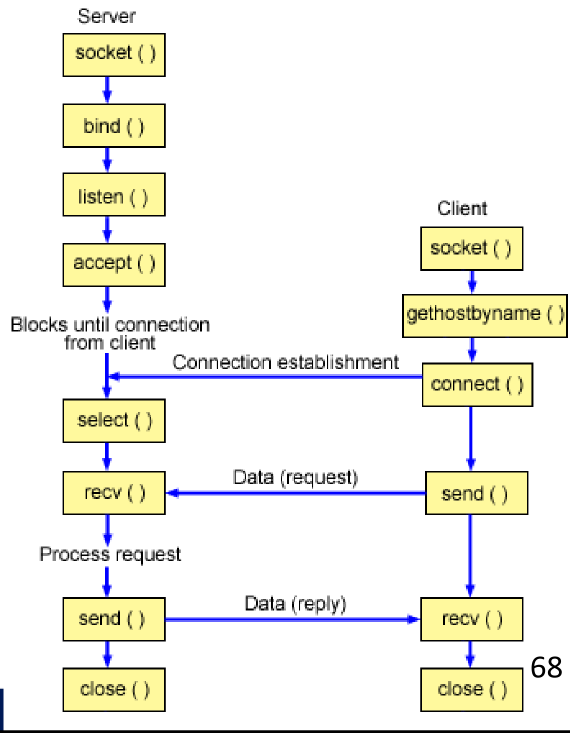
Duke UNIVERSITY

Lecture Outline

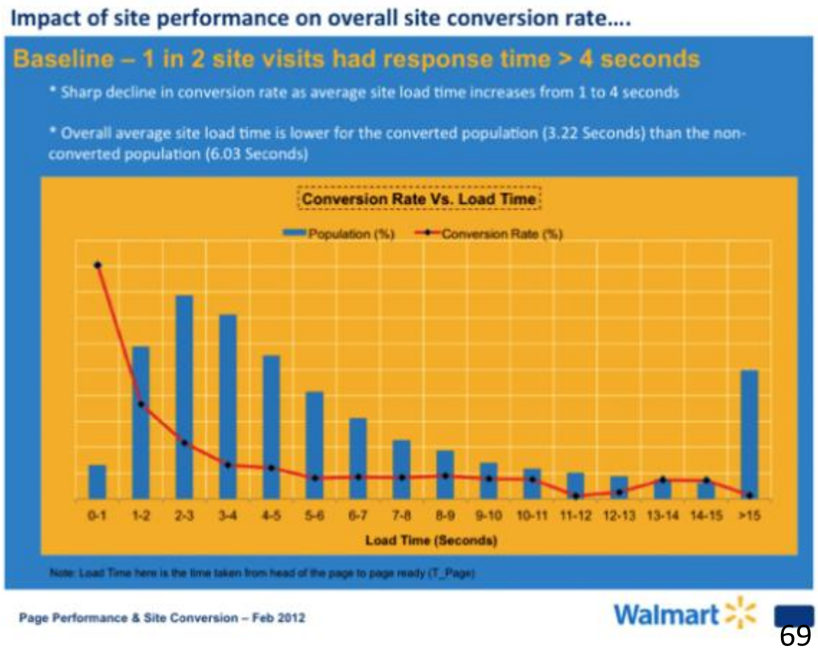
- Transport Control Protocol (TCP)
  - ➢TCP segment format (PD 5.2.2)
  - ➢Adaptive retransmission intervals (PD 5.2.6)
  - ➢Reliable data transfer (PD 5.2.4)
  - ➢TCP flow control (PD 5.2.4)
  - ➢**TCP connection establishment and termination** (PD 5.2.3)

Recap: Setting Up TCP Sockets

- TCP is *connection-oriented*
- Processes must first "handshake" with each other
  - ➢ Agree to establish connection
  - ➢ Agree on connection parameters



TCP Performance is Critical to Business



69

Duke UNIVERSITY

Recap: Segment Header Flags

URG: urgent data (generally not used)

ACK: ACK # valid

PSH: push data now (generally not used)

RST, SYN, FIN: connection estab. (setup, teardown commands)

32 bits

source port #								dest port #
sequence number								
acknowledgement number								
head len	not used	U	A	P	R	S	F	receive window
checksum								Urg data pointer
options (variable length)								
application data (variable length)								

Duke UNIVERSITY

Recap: TCP Flag Bits: Connection Management

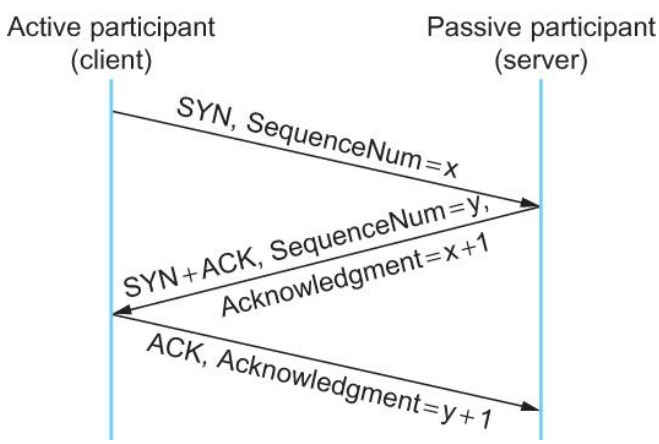
- RST: Reset the connection
  - ➢ Receiver of a RST terminates the connection and indicates higher layer application about the reset
- SYN: Synchronize sequence numbers
  - ➢ Sent in the first packet when initiating a connection
- FIN: Sender is finished with sending
  - ➢ Used for closing a connection

71

Duke UNIVERSITY

TCP Connection Establishment

- TCP uses a **three-way handshake** to open a connection



72

Duke UNIVERSITY

Three-way Handshake: Step 1

- Client sends a special segment: no data, SYN bit set to 1
  - "**SYN segment**"
- Picks a randomly chosen initial sequence number *client_isn* and sends it over

Three-way Handshake: Step 2

- SYN segment arrives to the server
- Server allocates TCP buffers and variables
- Sends a "connection granted" segment
  - No application-layer data
  - SYN bit set to 1
  - Acknowledgement is set to *client_isn+1*
  - Chooses its own initial sequence number *server_isn*, puts it in the sequence number field
  - **"SYNACK" segment**

Three-way Handshake: Step 3

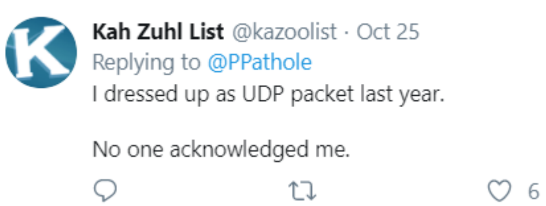
- Client receives SYNACK segment
- Client allocates buffers and variables
- Client sends to server another segment:
  - Acknowledgement: server_isn+1
  - SYN bit is set to 0
  - May carry payload
- SYN bit set to zero in all subsequent packets

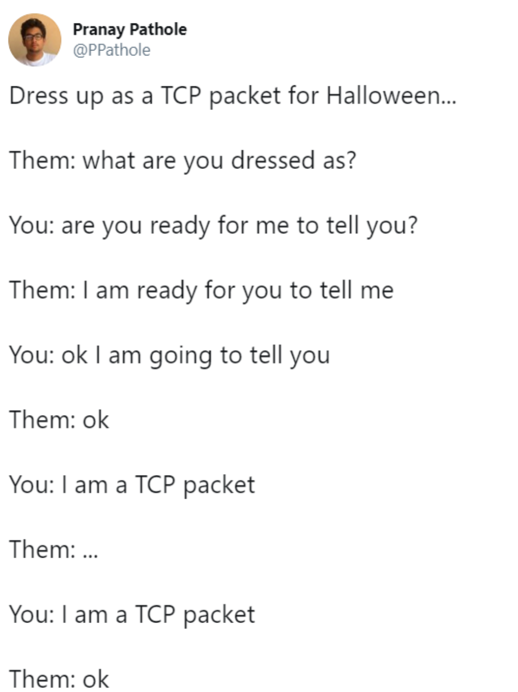
75

Duke UNIVERSITY

Dress Up as a TCP Packet

- A 3-way handshake example
- More protocol jokes: https://twitter.com/PPathole/status/1187371220238508035

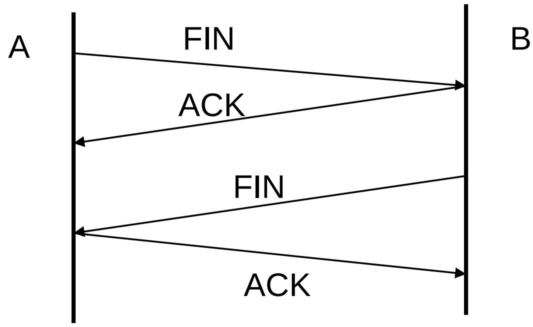




Duke UNIVERSITY

TCP Connection Termination

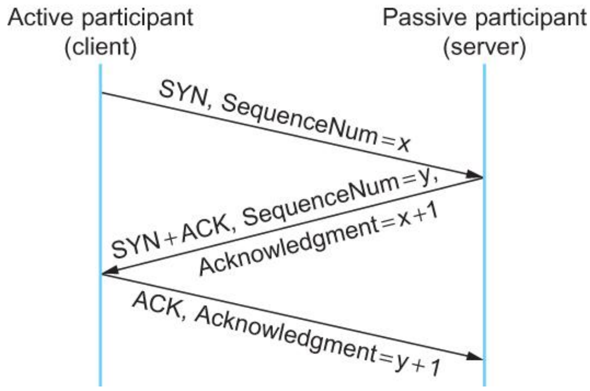
- Need to de-allocate buffers and variables
- Each end of the data flow must be shut down independently
- If one end is done it sends a FIN segment. The other end sends ACK
- Four messages to completely shut down a connection



Duke UNIVERSITY

TCP Connection Management: Key Points to Remember

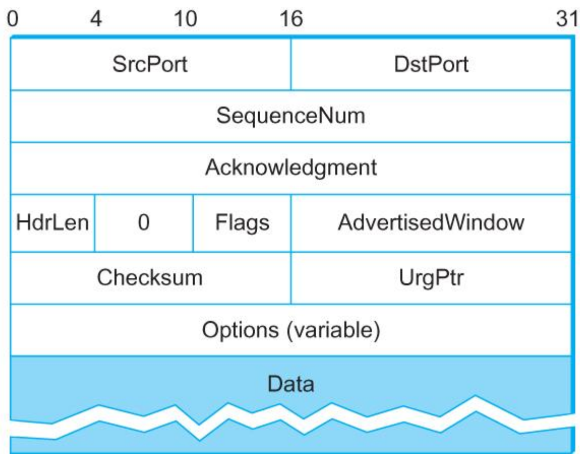
- Need to allocate resources on both ends of the communication
- Connection established via a three-way handshake
- Connections need to be torn down, to deallocate the resources



Duke UNIVERSITY

Lecture Summary (1/2)

- TCP segment header has dedicated fields for:
  - ➢ Demultiplexing by port numbers
  - ➢ Checksumming
  - ➢ Reliable communications
  - ➢ Flow control
  - ➢ Connection establishment



Lecture Summary (2/2)

- Core mechanisms discussed:
  - ➢ Reliable data transfer, including adaptive timeout values
  - ➢ TCP flow control
  - ➢ TCP connection establishment and termination

Next Lecture

- TCP congestion control