Networks

Congestion Control

Adopted from material in “Computer Networking: A Top Down Approach” by Kurose and Ross and slides developed by William Conner

Congestion Control

Sections 3.6-3.7

TCP Congestion Control

- Underlying network might be overwhelmed by aggregate traffic load
  - Packet delays due to queues in routers
  - Packet loss due to buffer overflow at routers
- Not the same as flow control
- Rather than making situation worse with retransmissions, slow down transmission when the network is congested

TCP Congestion Control

- Congestion window (`cwnd`) limits how much unACKed data can be sent
- Maximum segment size (`MSS`)
- Additive increase: increase `cwnd` by 1 `MSS` every RTT until loss detected
- Multiplicative decrease: cut `cwnd` in half after loss detected

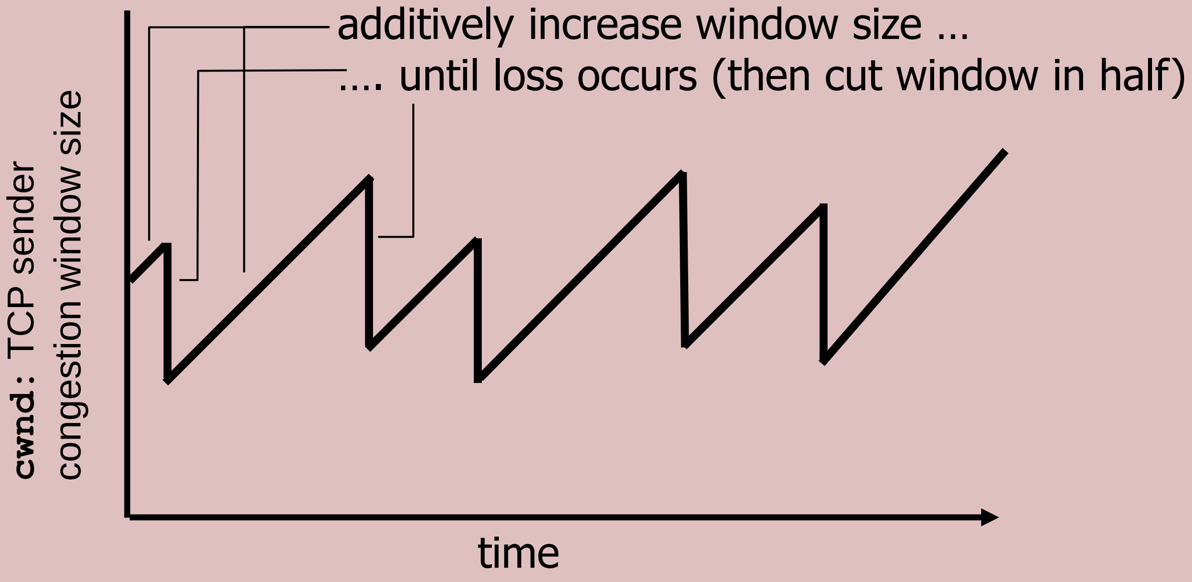
Maximum Segment Size (MSS)

- Maximum transmission unit (MTU) is the maximum frame for a given link layer (e.g., 1500 bytes for Ethernet and PPP)
- Maximum segment size is typically sender’s MTU minus 40 bytes for TCP/IP headers
- Path MTU (RFC 1191) is largest frame supported by all links in a given path

TCP Congestion Control

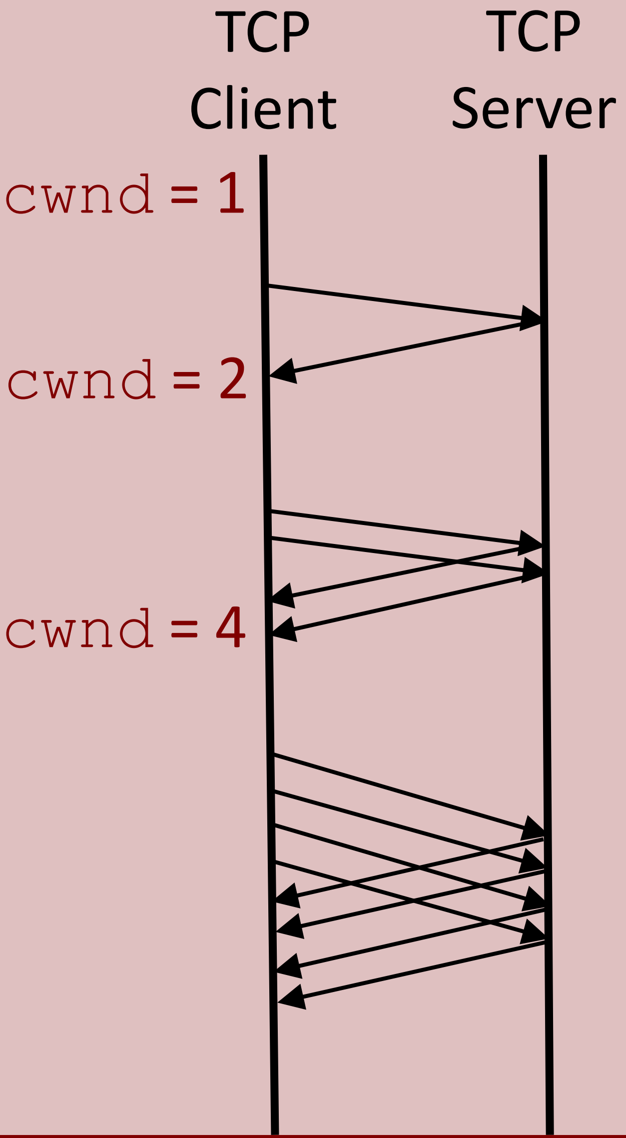
- AIMD: probe for usable bandwidth until loss occurs

AIMD saw tooth behavior: probing for bandwidth



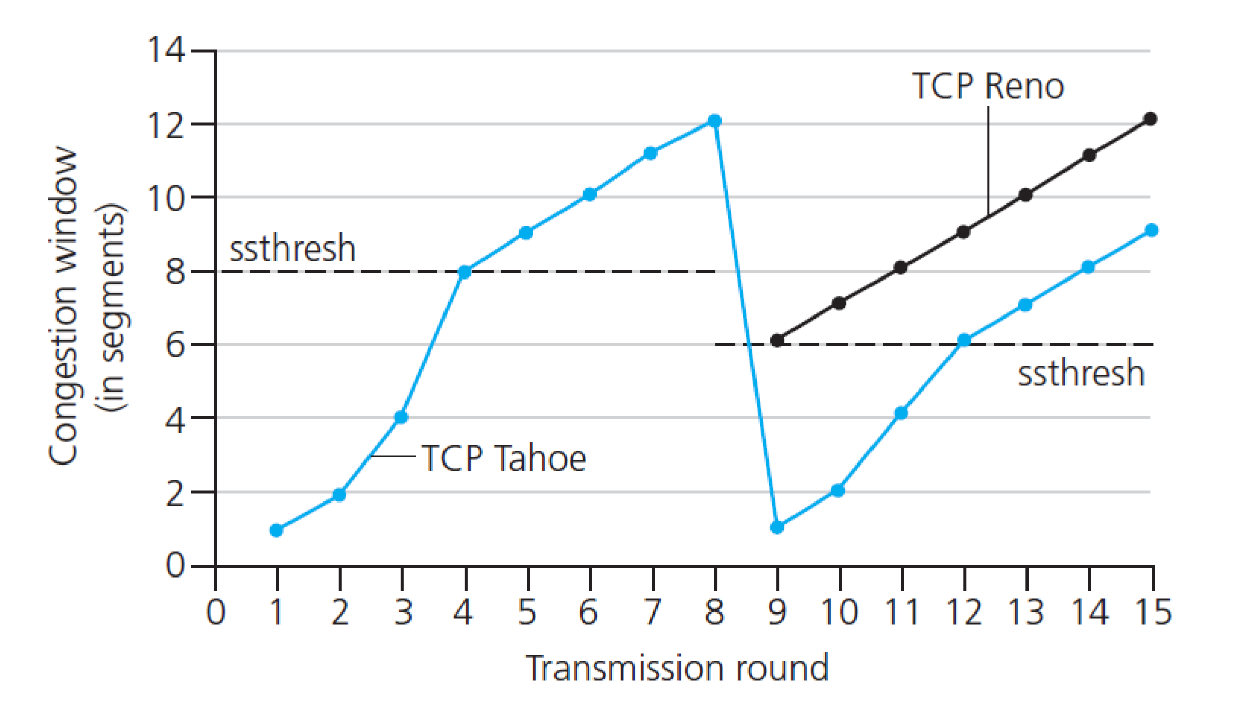
TCP Slow Start

- Increase transmission rate exponentially until first loss event
  - Initially, `cwnd` = 1 `MSS`
  - Double `cwnd` every RTT (i.e., increment for each ACK received)



TCP Congestion Avoidance

- Switch from exponential increase to linear increase when `cwnd` gets to ½ value before timeout
- Implemented with `ssthresh` variable set to ½ of `cwnd` just before loss event



TCP Congestion Control

- Loss indicated by timeout
  - `cwnd` set to 1 `MSS`
  - Window grows exponentially (slow start) to `ssthresh`, then linearly (congestion avoidance)

- Loss indicated by triple duplicate ACKs
  - TCP RENO cuts `cwnd` in half, then grows linearly from there (TCP fast recovery)
  - TCP TAHOE reacts as if timeout-based loss

TCP Flow Control

- Avoid overflowing receiver with too much data sent too fast
- Not the same as congestion control
- Receiver advertises free buffer space `rwnd` to sender in window size field of TCP header

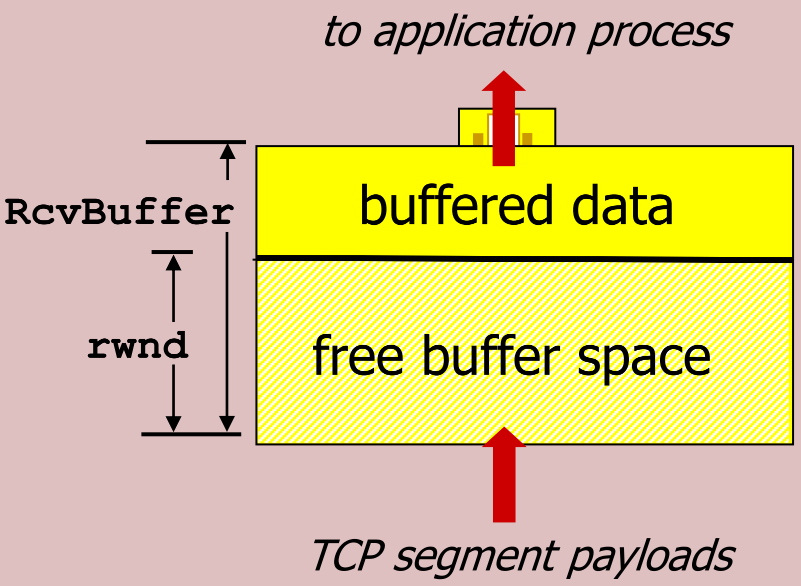
TCP Flow Control

- Receiver buffer

  `RcvBuffer`

  - Set via socket options (default is typically 4KB)
  - Some OSes auto adjust `RcvBuffer`

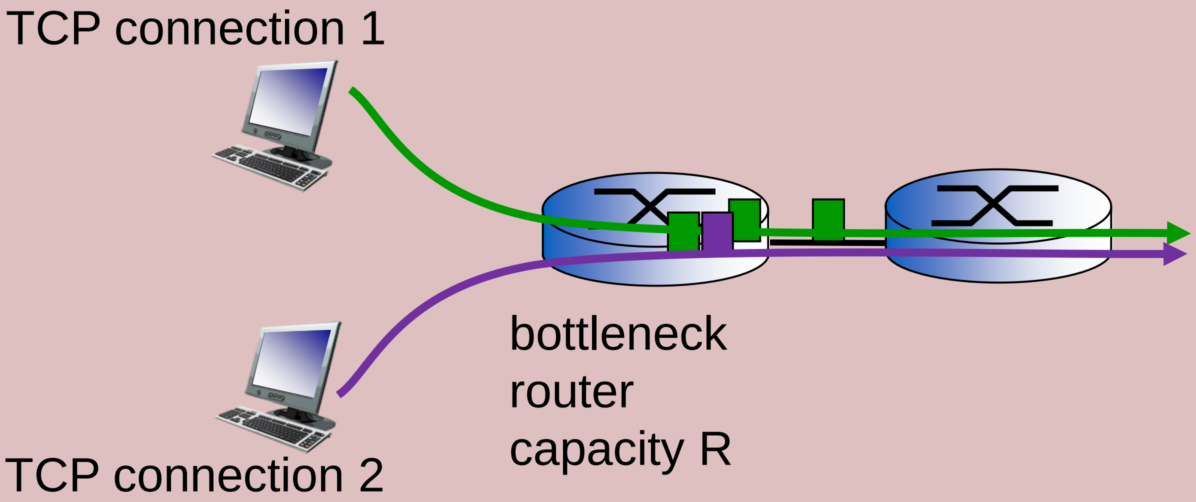
- Sender will send with W=min(`cwnd`, `rwnd`)



receiver-side buffering

TCP Fairness

- Goal: If *K* TCP connections share same bottleneck link of bandwidth *R*, then each connection *should* have an average rate of *R/K*



TCP (Un-)Fairness

- Multimedia applications often use UDP
  - Typically can tolerate *some* packet loss
  - Avoid throttling due to congestion control

- Multiple parallel TCP connections
  - Browser implementations do this
  - Example: Suppose 9 connections share link with capacity *R*
    - App 1 requests one connection (R/10 per app)
    - App 2 requests 10 connections (R/2 for app 2)
    - Single connection apps experience R/20

QUIC Review

- **Q**uick **U**DP **I**nternet **C**onnections
- Transport-like services implemented over UDP at the application layer
- Developed by Google initially as an experimental protocol
- Undergoing IETF standardization

QUIC

- Initially targeted improved transport performance between Chrome browser and Google services
- Over 1/3 of Google egress traffic (over 5% of internet traffic)
- Akamai CDN deployed in 2016

QUIC vs. TCP

- Many transport services overlap (congestion control, reliability, etc.)
- Multiple streams in single QUIC connection
- QUIC uses 64-bit connection IDs rather than 4-tuples that simplify migrating to different addresses and ports

Thank You!

Networks

Connectionless Transport - UDP

Adopted from material in “Computer Networking: A Top Down Approach” by Kurose and Ross and slides developed by William Conner

Connectionless Transport - UDP

Section 3.3

User Datagram Protocol

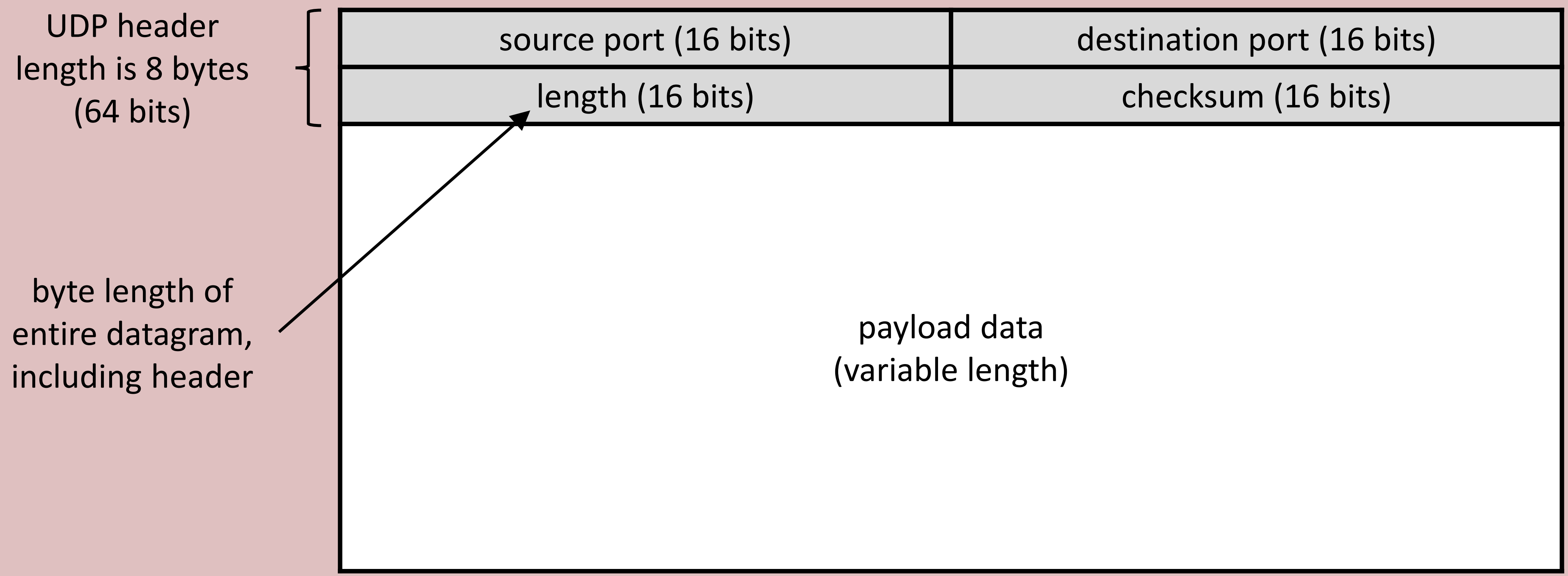
- Minimal best-effort transport protocol
  - Datagrams may be lost (unreliable)
  - Datagrams may arrive out of order

- Connectionless
  - No connection setup or teardown overhead
  - Each datagram handled independently

User Datagram Protocol

- No congestion or flow control
- Applications must implement desired services (e.g., reliability, flow control)
  - Streaming multimedia
  - DNS
  - QUIC (not really an application)

UDP Datagram Format



What is the largest possible UDP packet size?

UDP Checksum

- Also known as *Internet checksum* or *TCP checksum*
- Detect bit errors in datagrams
- Sender: set checksum header field to be checked by the receiver
- Receiver: verify checksum for received datagram

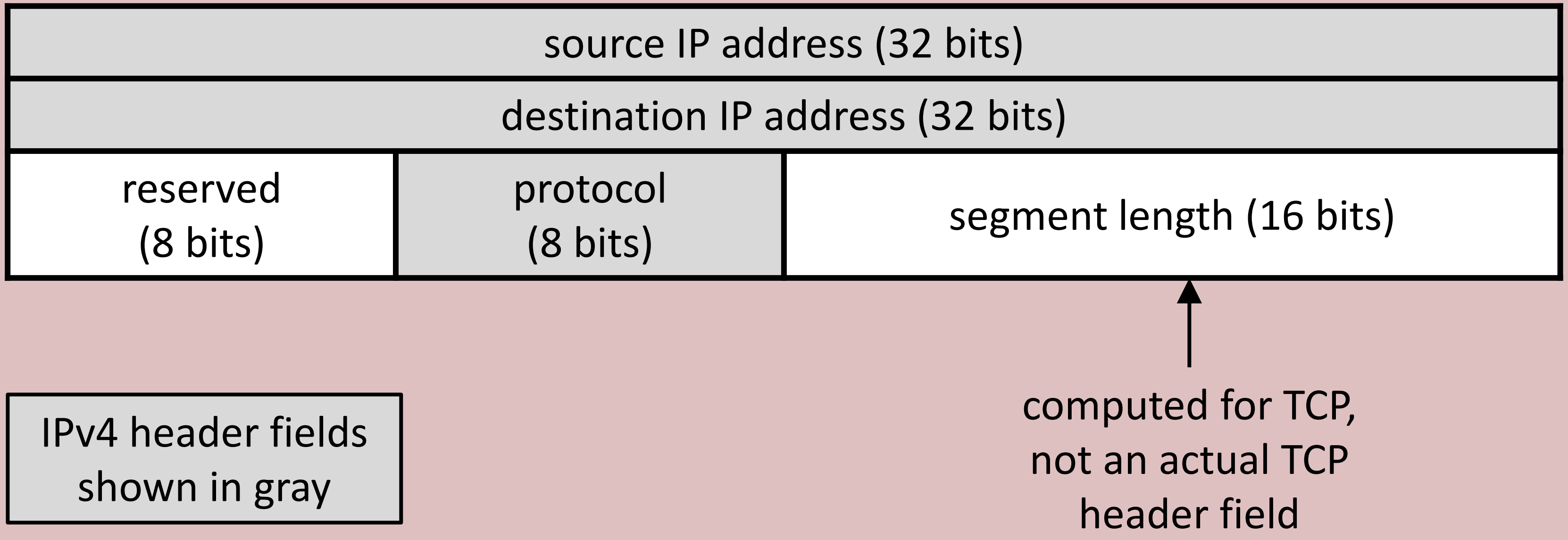
UDP Checksum

- One’s complement of the one’s complement sum of the following data treated as a sequence of 16-bit integers
  - Header (checksum field treated as all 0s for calculation)
  - Payload
  - Pad byte (if odd number of payload bytes)
  - Pseudo-header

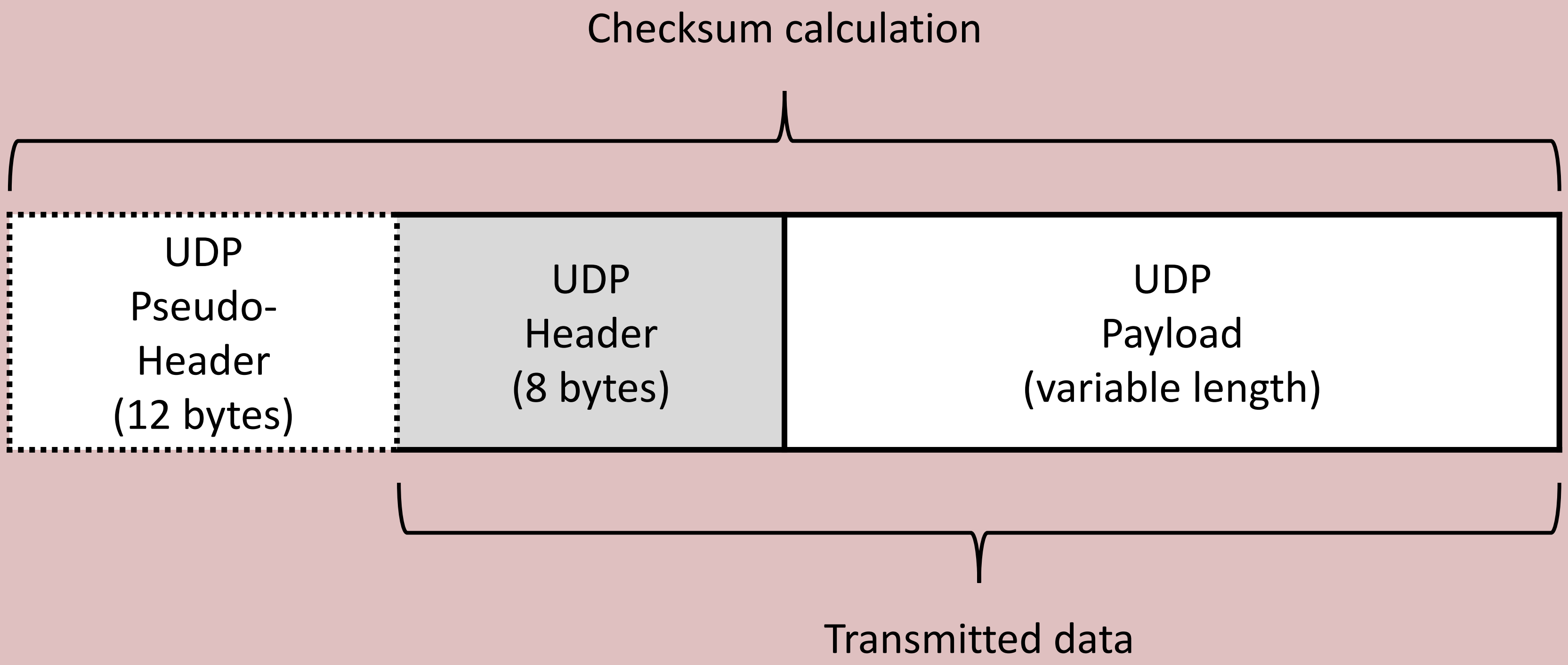
UDP Pseudo-Header

- Used for checksum computation *but not actually transmitted*
- Mostly IP header fields
  - 12 bytes for IPv4
  - 40 bytes for IPv6
- Protocol layering violation (why?)

UDP Pseudo-Header (IPv4)



UDP Pseudo-Header



UDP Checksum

Example: add two 16-bit integers

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

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

Thank You!

Networks

Connectionless Transport - UDP

Adopted from material in “Computer Networking: A Top Down Approach” by Kurose and Ross and slides developed by William Conner

Connectionless Transport - UDP

Section 3.3

User Datagram Protocol

- Minimal best-effort transport protocol
  - Datagrams may be lost (unreliable)
  - Datagrams may arrive out of order

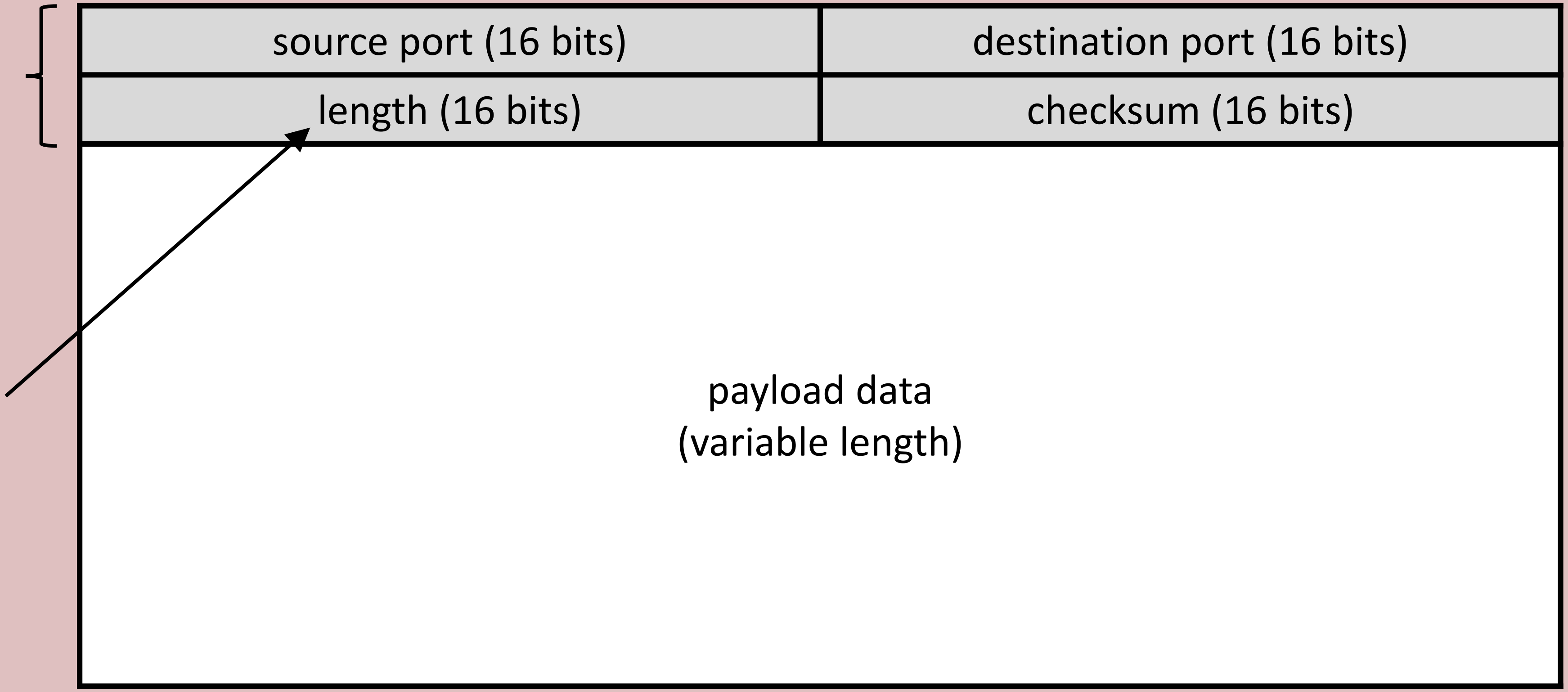
- Connectionless
  - No connection setup or teardown overhead
  - Each datagram handled independently

User Datagram Protocol

- No congestion or flow control
- Applications must implement desired services (e.g., reliability, flow control)
  - Streaming multimedia
  - DNS
  - QUIC (not really an application)

UDP Datagram Format

UDP header length is 8 bytes (64 bits)



byte length of entire datagram, including header

What is the largest possible UDP packet size?

UDP Checksum

- Also known as *Internet checksum* or *TCP checksum*
- Detect bit errors in datagrams
- Sender: set checksum header field to be checked by the receiver
- Receiver: verify checksum for received datagram

UDP Checksum

- One's complement of the one's complement sum of the following data treated as a sequence of 16-bit integers
  - Header (checksum field treated as all 0s for calculation)
  - Payload
  - Pad byte (if odd number of payload bytes)
  - Pseudo-header

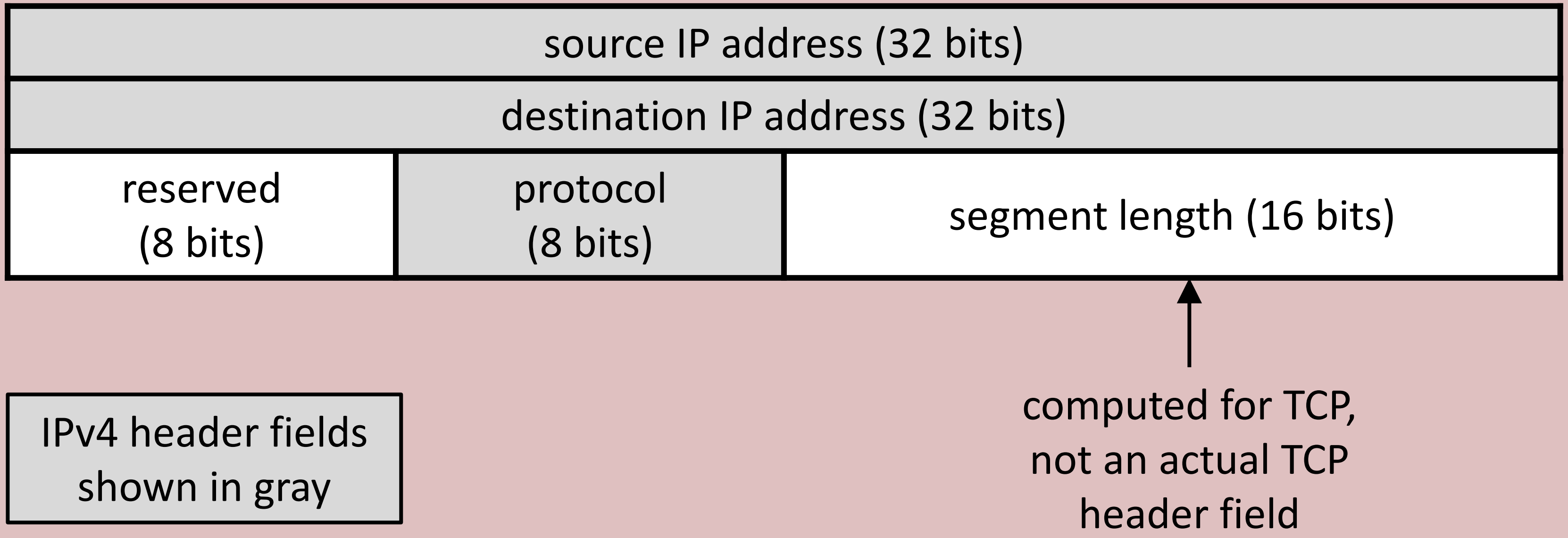
UDP Pseudo-Header

- Used for checksum computation *but not actually transmitted*

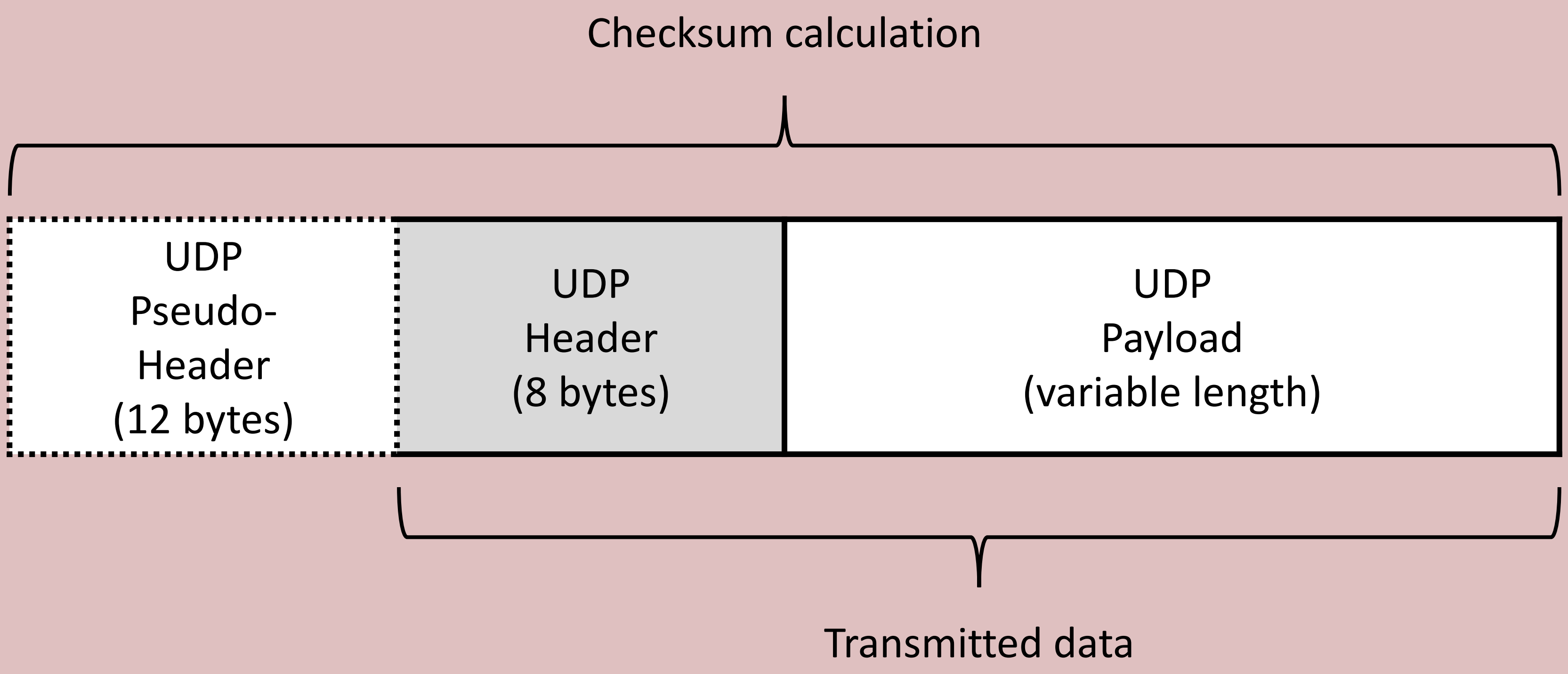
- Mostly IP header fields
  - 12 bytes for IPv4
  - 40 bytes for IPv6

- Protocol layering violation (why?)

UDP Pseudo-Header (IPv4)



UDP Pseudo-Header



UDP Checksum

Example: add two 16-bit integers

		1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
		1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	1	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

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

Thank You!

Networks

Transport-Layer Services

Adopted from material in “Computer Networking: A Top Down Approach” by Kurose and Ross and slides developed by William Conner

Transport-Layer Services

Section 3.1

Transport Layer

- Runs only on the end systems
- Provides logical process-to-process communication
  - Sender breaks application messages into UDP datagrams or TCP segments
  - Receiver reassembles for application
- Built on top of network layer services (host-to-host communication)

Inter-Office Mail Analogy

Company split between NY and LA offices

- Hosts = offices
- Processes = employees
- Application messages = company memos
- Transport layer = local mailroom that delivers individual envelopes to desks
- Network-layer = U.S. postal service

Transport Layer Protocols

- UDP: essentially datagram headers added to IP packets
- TCP: reliable, connection-oriented over IP
- QUIC: reliable, connection-oriented protocol over UDP (i.e., technically application layer)

Transport Layer Services

Service	UDP	TCP	QUIC
Reliable delivery		✓	✓
In-order delivery		✓	✓
Congestion control		✓	✓
Flow control		✓	✓
Connection-oriented		✓	✓
Delay guarantees			
Bandwidth guarantees			

Thank You!

Networks

Multiplexing and Demultiplexing

Adopted from material in “Computer Networking: A Top Down Approach” by Kurose and Ross and slides developed by William Conner

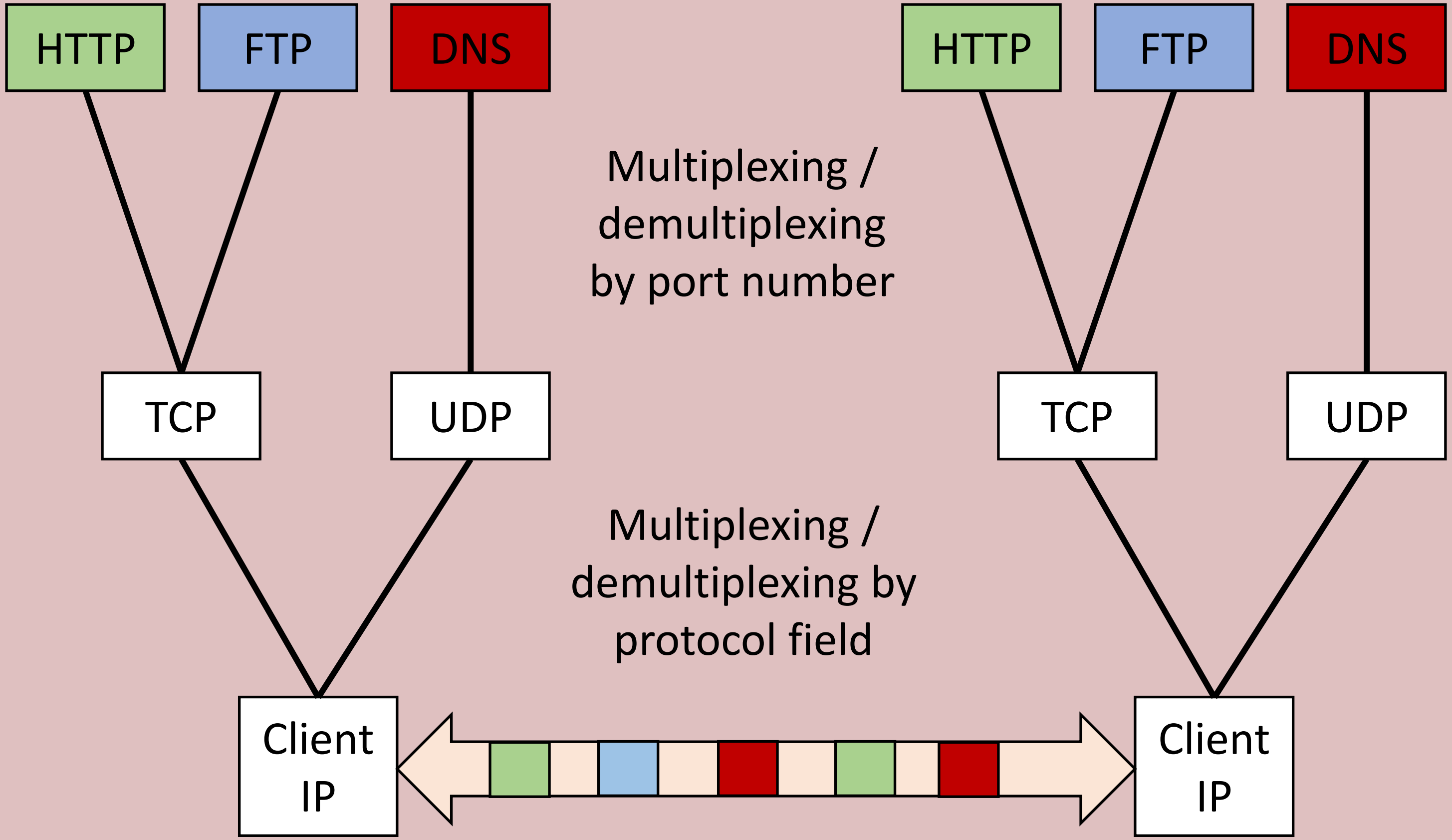
Multiplexing and Demultiplexing

Section 3.2

Protocol Multiplexing and Demultiplexing

- Multiplexing at sender
  - Handling messages from multiple sockets
  - Handling datagrams/segments from TCP or UDP
- Demultiplexing at receiver
  - Delivering packets to either UDP or TCP based on protocol number in IP header
  - Delivering datagrams/segments to correct socket based on port number in UDP/TCP header

Protocol Multiplexing and Demultiplexing



Thank You!

Networks

Transmission Control Protocol

Adopted from material in “Computer Networking: A Top Down Approach” by Kurose and Ross and slides developed by William Conner

Transmission Control Protocol

Section 3.5

Transmission Control Protocol (TCP)

- Originally published in RFC 793 in 1981
- Variant of Go-Back-N but also has SACK option for Selective Repeat
- Connection-oriented
- Reliable, in-order byte stream service
- Full duplex between two endpoints
- Congestion control
- Flow control

TCP Segment Header

<table>
<tr><td colspan="3">source port (16 bits)</td><td>destination port (16 bits)</td></tr>
<tr><td colspan="4">sequence number (32 bits)</td></tr>
<tr><td colspan="4">acknowledgement number (32 bits)</td></tr>
<tr><td>length
(4 bits)</td><td>unused
(4 bits)</td><td>flags
(8 bits)</td><td>window size (16 bits)</td></tr>
<tr><td colspan="3">checksum (16 bits)</td><td>urgent pointer (16 bits)</td></tr>
<tr><td colspan="4">options
(variable length)</td></tr>
</table>

TCP Segment Header

- Length: header length in 32-bit words (options are variable length), basic length is 20 bytes without any options
- Urgent pointer: rarely used method for marking special data
- Window size: number of bytes receiver is willing to accept (more later)
- Checksum: identical to UDP checksum, but over more fields
- Flags: mostly used for connection management

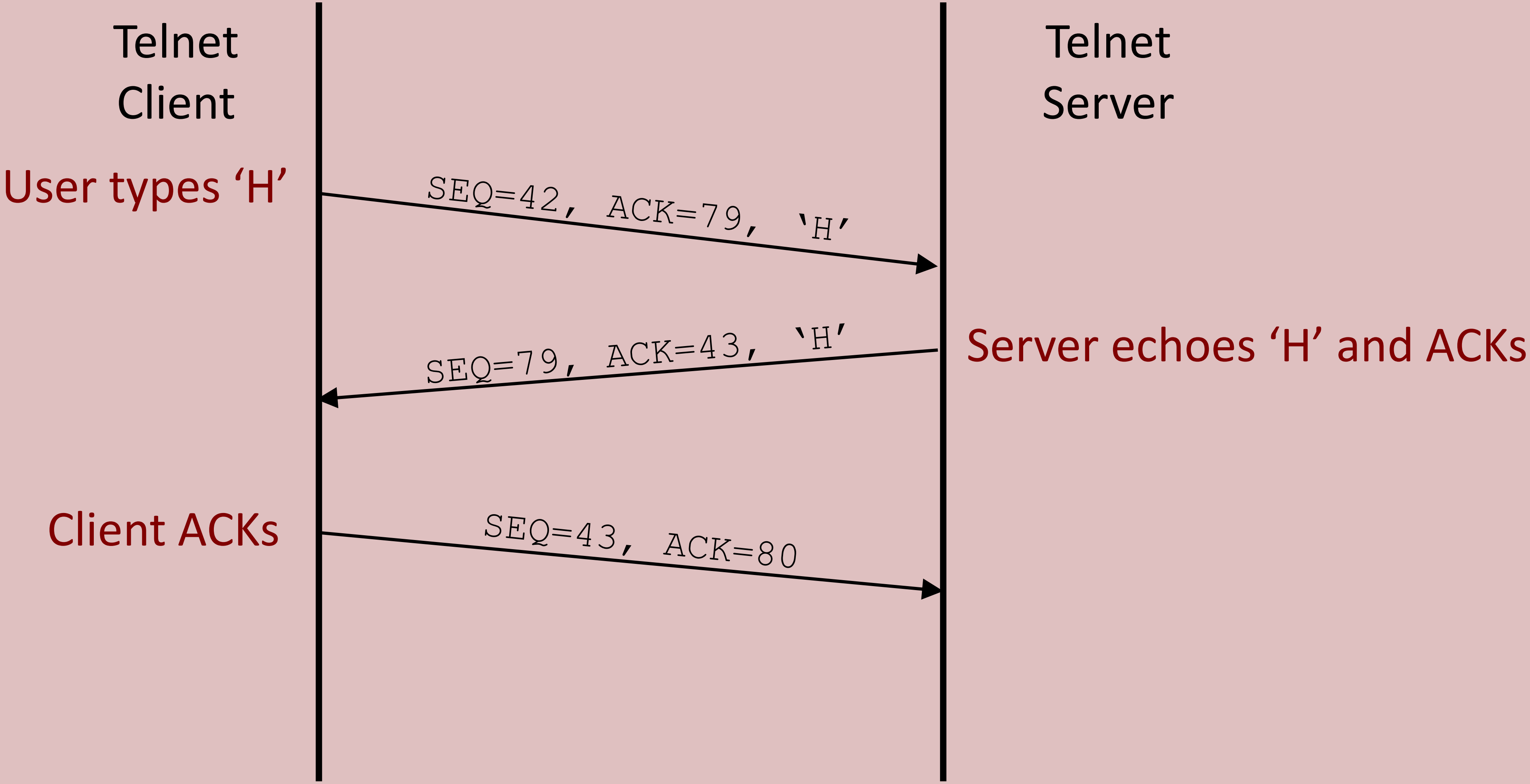
TCP Sequence Number

- 32-bit unsigned integer in TCP segment header
- Represents byte number in byte stream that first byte of segment represents
- Wraps back around to 0

TCP Acknowledgement Number

- Next sequence number expected to be received (cumulative ACK)
- Alternatively, sequence number of last successfully received byte of data plus 1
- Only valid if ACK flag is set

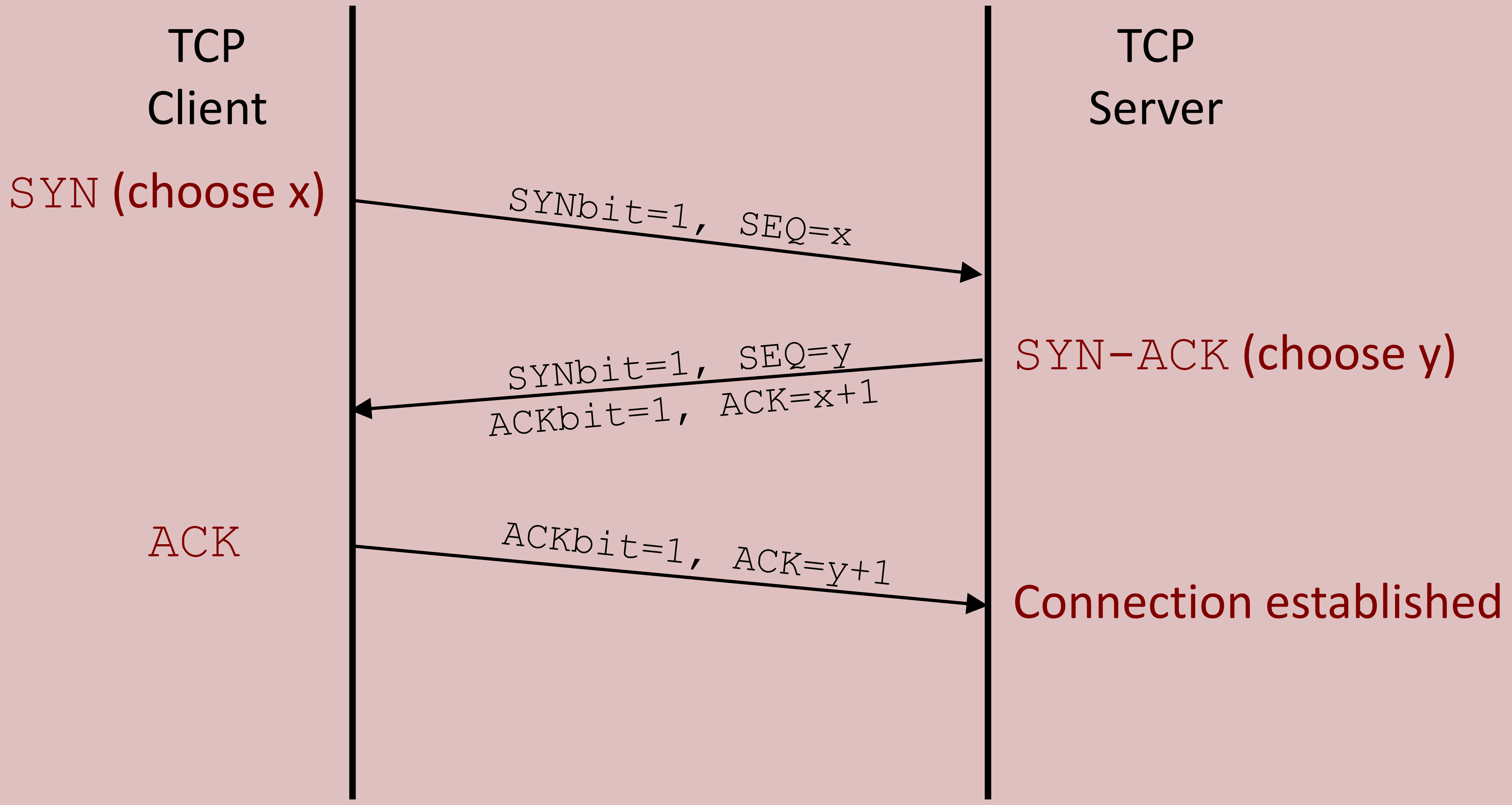
TCP SEQ and ACK Numbers



TCP Connections

- Perform 3-way handshake to establish the connection
  - Messages set SYN and/or ACK flags
  - Choose 32-bit initial sequence numbers
- Each side half-closes their end to terminate the connection (4 messages)
  - Messages set FIN or ACK flags

TCP Connection Establishment



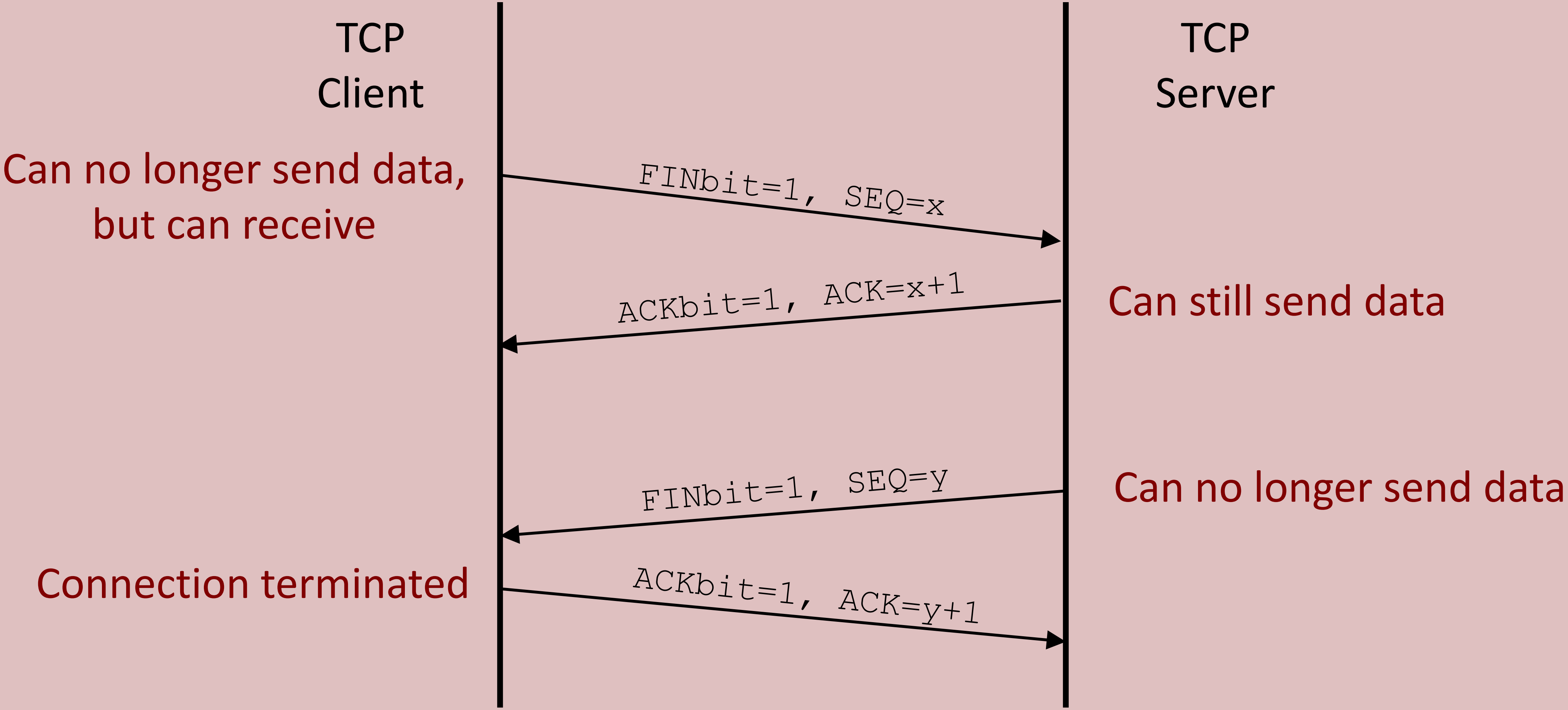
SYN Floods

- Denial-of-service attack on TCP servers
- Attacker repeatedly sends `SYN` segments to exhaust victim server
  - Typically uses spoofed IP address
  - Server allocates resources
  - Half-open connections
- Attacker never completes 3-way handshake

SYN Cookies

- Encode connection state in initial sequence number (ISN) chosen by server for TCP `SYN-ACK`
- Cryptographic hash of connection 4-tuple and secret value known to server
- Attackers from spoofed IP addresses cannot guess valid TCP `ACK`
  - Why not?

TCP Connection Termination



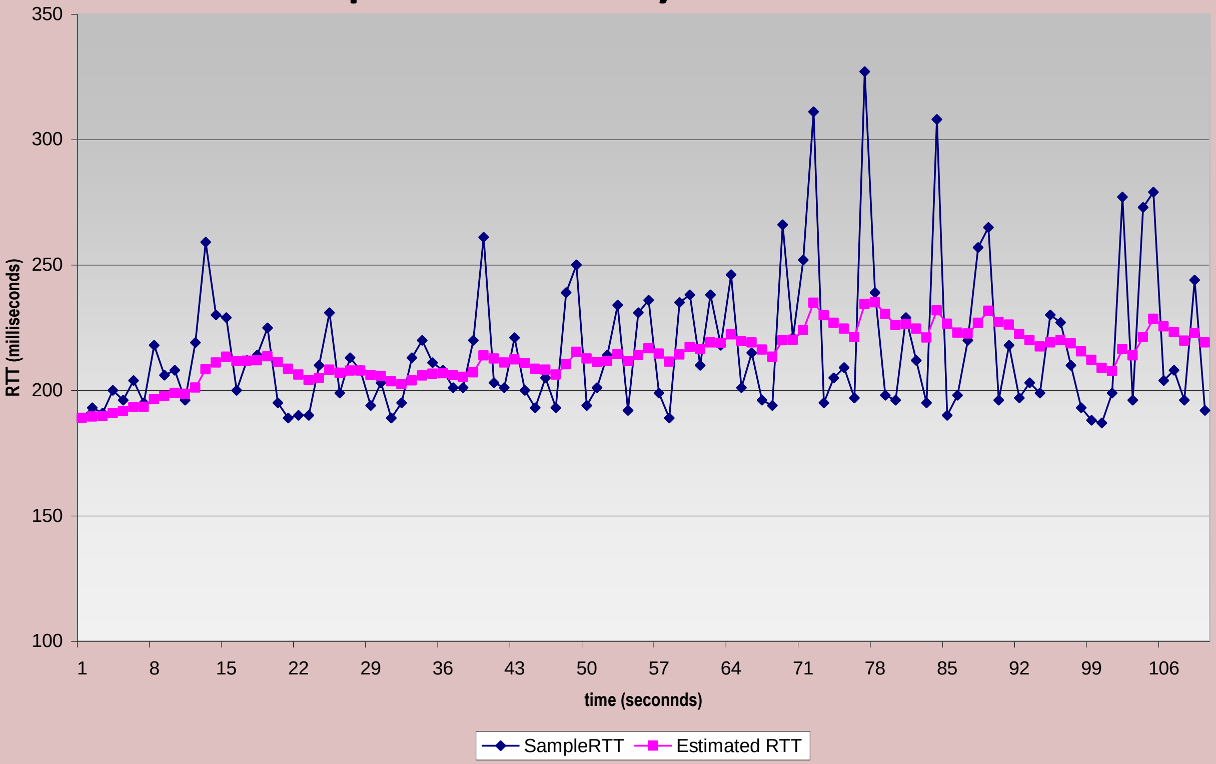
TCP Timeout

- If too short, unnecessary retransmissions
- If too long, slow reaction to lost segments
- Timeout interval should be longer than RTT, but RTT varies over time
- Solution: estimate RTT based on periodic RTT sample measurements
  - SampleRTT: measured time between segment transmission and ACK receipt
  - EstimatedRTT: average several recent measurements to avoid too much variation

TCP Timeout

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

- Exponentially weighted moving average
- Influence of past samples decreases exponentially fast
- Typical value: α=0.125



TCP Timeout

- Timeout interval: `EstimatedRTT` plus "safety margin"
  - Large variation in `EstimatedRTT` –> larger safety margin

- Estimate SampleRTT deviation from EstimatedRTT:

```
DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstmiatedRTT|
```

- Typical value: β=0.25

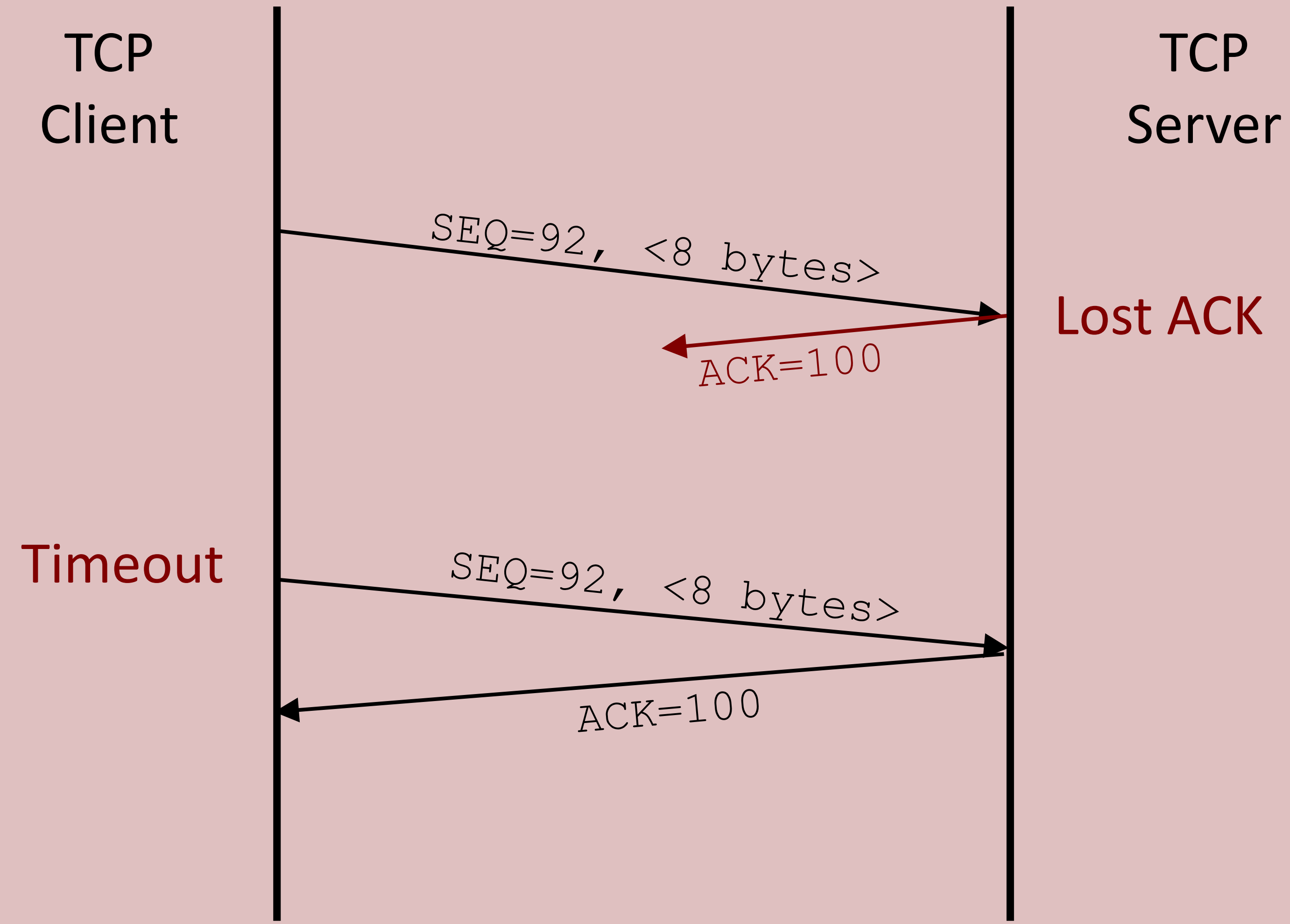
```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

Estimated RTT Safety Margin

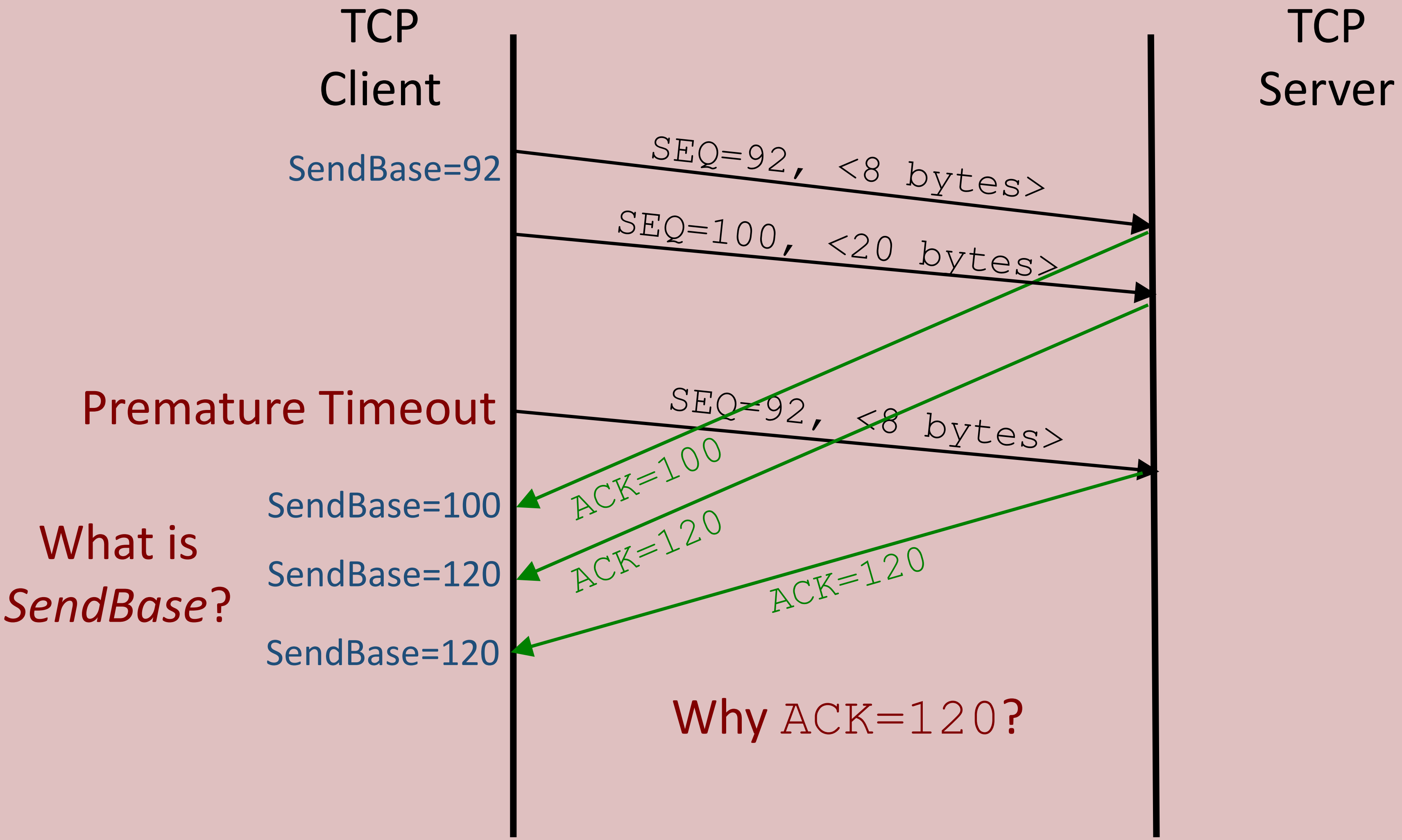
TCP Retransmission

- Single retransmission timer that gets reset for every received ACK or timeout
- Retransmission triggered by timeout
- *Fast retransmit* also triggered by three duplicate ACKs indicating out-of-order data received

TCP Retransmission – Lost ACK



TCP Retransmission – Premature Timeout



Why ACK=120?

TCP Retransmission – Cumulative ACKs

Why SEQ=120?

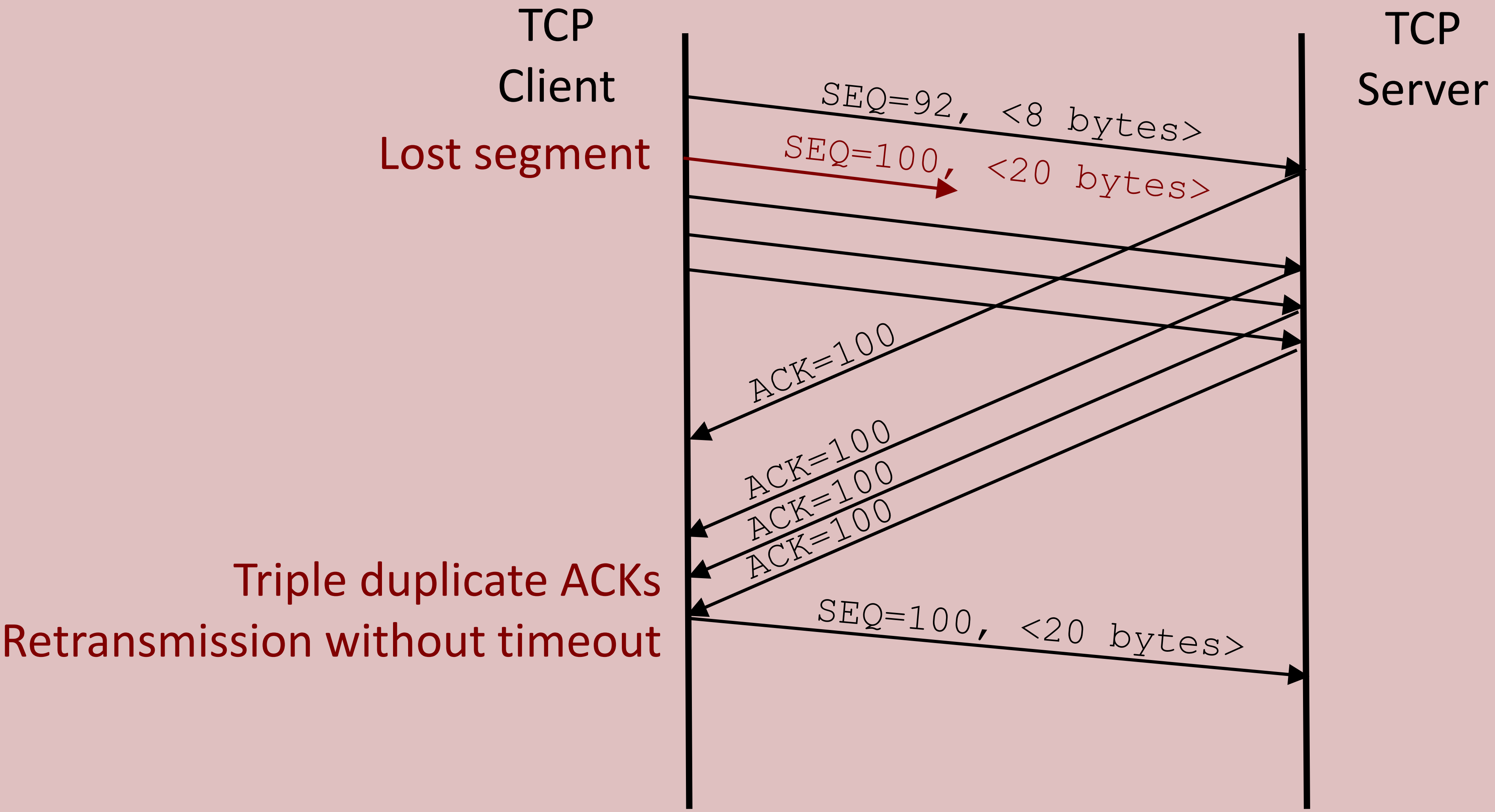
TCP ACK Generation

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 expected SEQ #. Gap detected.	Immediately send duplicate ACK, indicating SEQ # of next expected byte.
Arrival of segment that partially or completely fills gap.	Immediately send ACK, provided that segment starts at lower end of gap.

TCP Fast Retransmit

- Potentially long delays for timeout-based retransmissions (why?)
- Detect lost segments via duplicate ACKs
- If sender receives triple duplicate ACKs, then resend unACKed segment with smallest sequence number

TCP Fast Retransmit



Thank You!

Networks

Reliable Data Transfer

Adopted from material in “Computer Networking: A Top Down Approach” by Kurose and Ross and slides developed by William Conner

Reliable Data Transfer

Section 3.4

Unreliable Channels

- Packets might have bit errors
- Packets might be reordered
- Packets might be duplicated
- Packets might be dropped
- Can reliable data transfer be provided as a transport layer service?

Automatic Repeat Request (ARQ)

- Reliable data transfer protocols that retransmit until data is finally received

- ARQ mechanisms
  - Acknowledgements
  - Timeouts
  - Sequence numbers

- ARQ types
  - Stop-and-Wait
  - Go-Back-N
  - Selective Repeat

Acknowledgements (ACKs)

- Receiver sends signal back to sender that packet was successfully received
  - Handles lost or corrupted packets
  - Negative feedback version is a *NACK*

- Sender sends a packet, then
  - Retransmits packet if ACK not received
  - Retransmits packet if NACK received
  - Transmit next packet if ACK received

How can we determine this condition?

Timeouts

- Unlike NACK, absence of ACK is determined implicitly
- Sender sets a timer that triggers retransmission if an ACK is not received before the timer expires
- Minimum timer expiration period should be at least one RTT

What happens if the timer fires before the RTT?

Sequence Numbers

- Receiver needs to be able to detect duplicate packets
- Receiver needs to be able to handle out-of-order packets
- Sender assigns a unique sequence number to each packet for the receiver

Stop-and-Wait

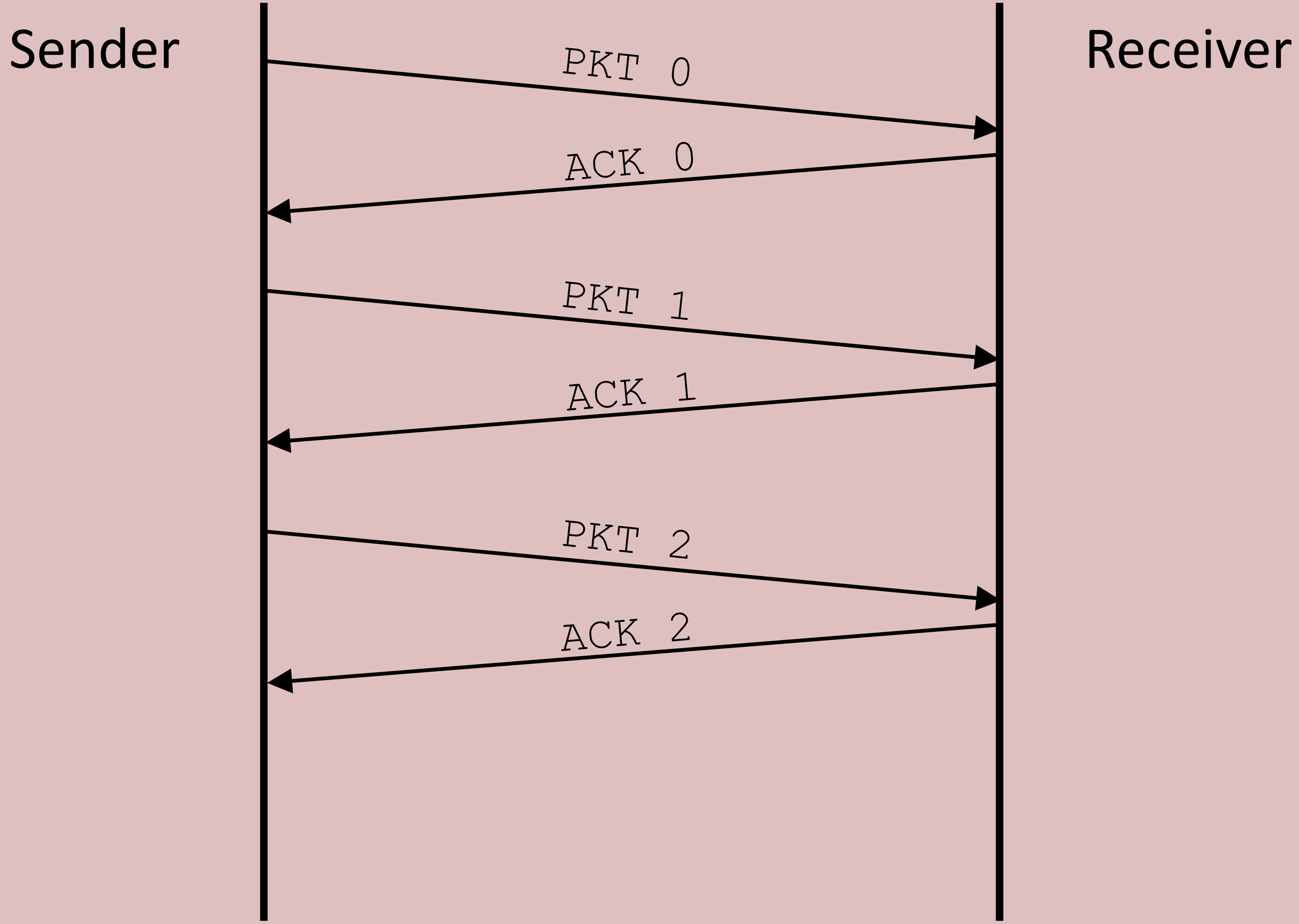
Sender:

- Send packet N
- Retransmit N (according to timer) until ACK received

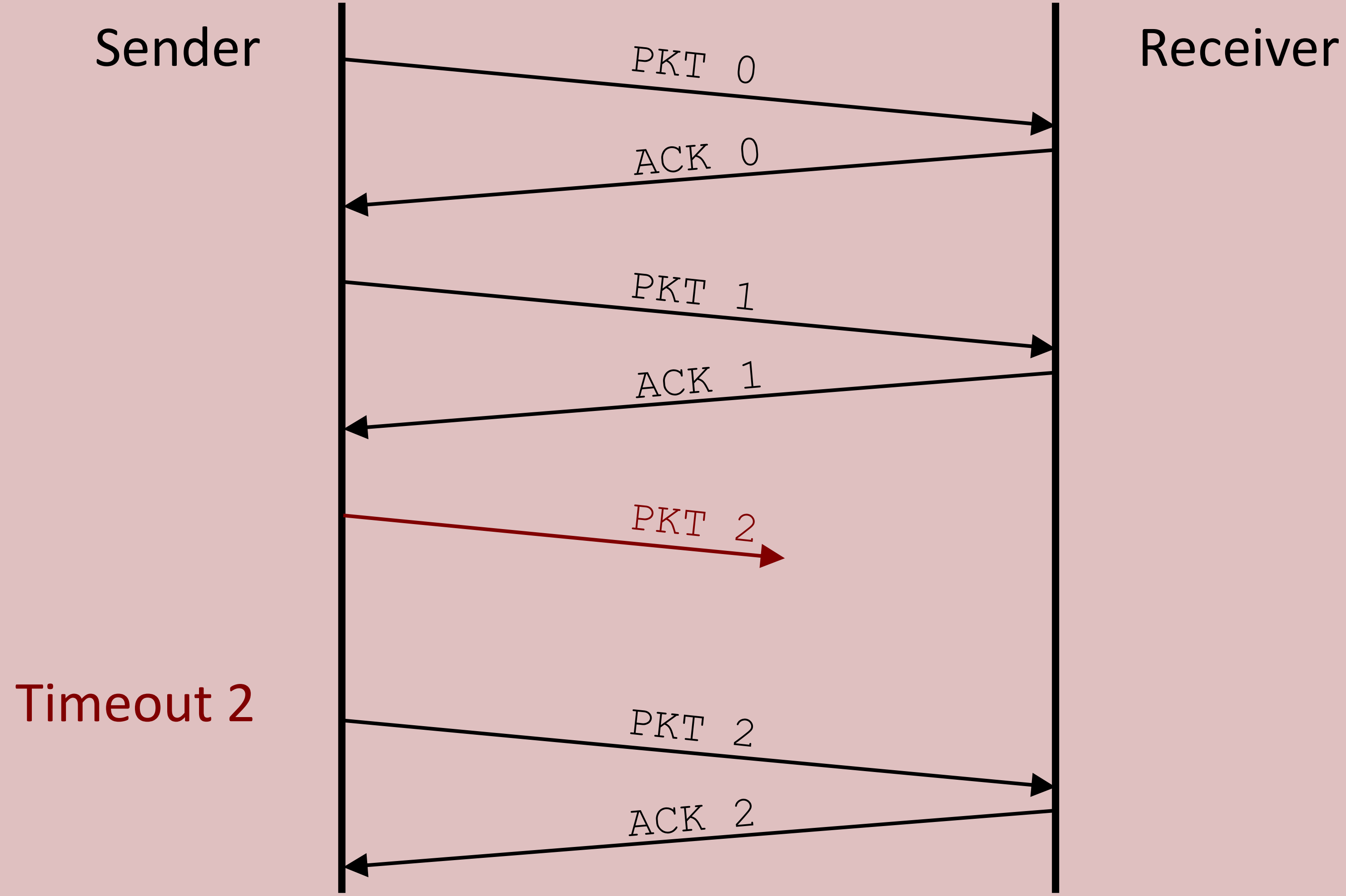
Receiver:

- Receive packet N
- Send ACK for packet N if checksum verifies

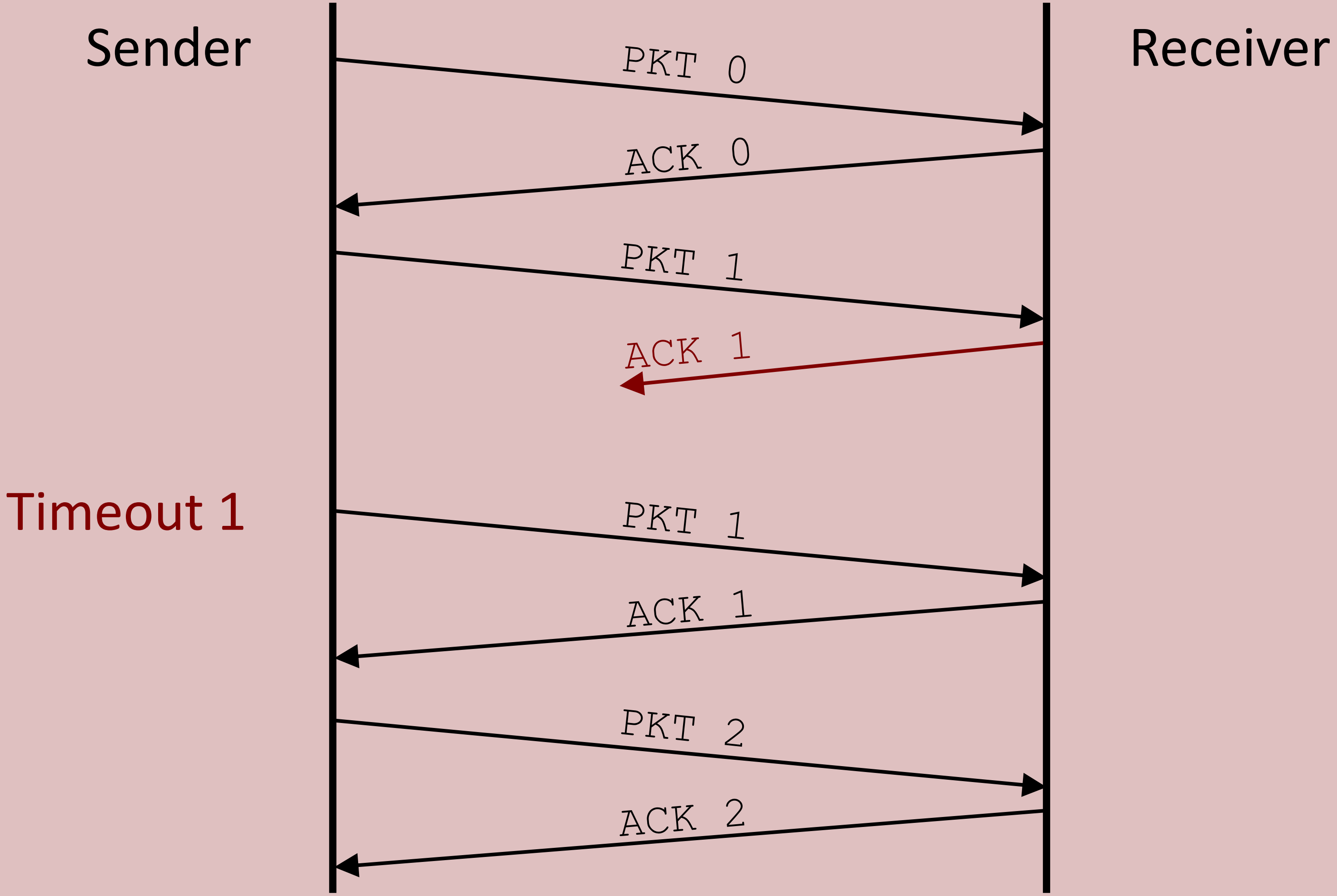
Stop-and-Wait – No Loss



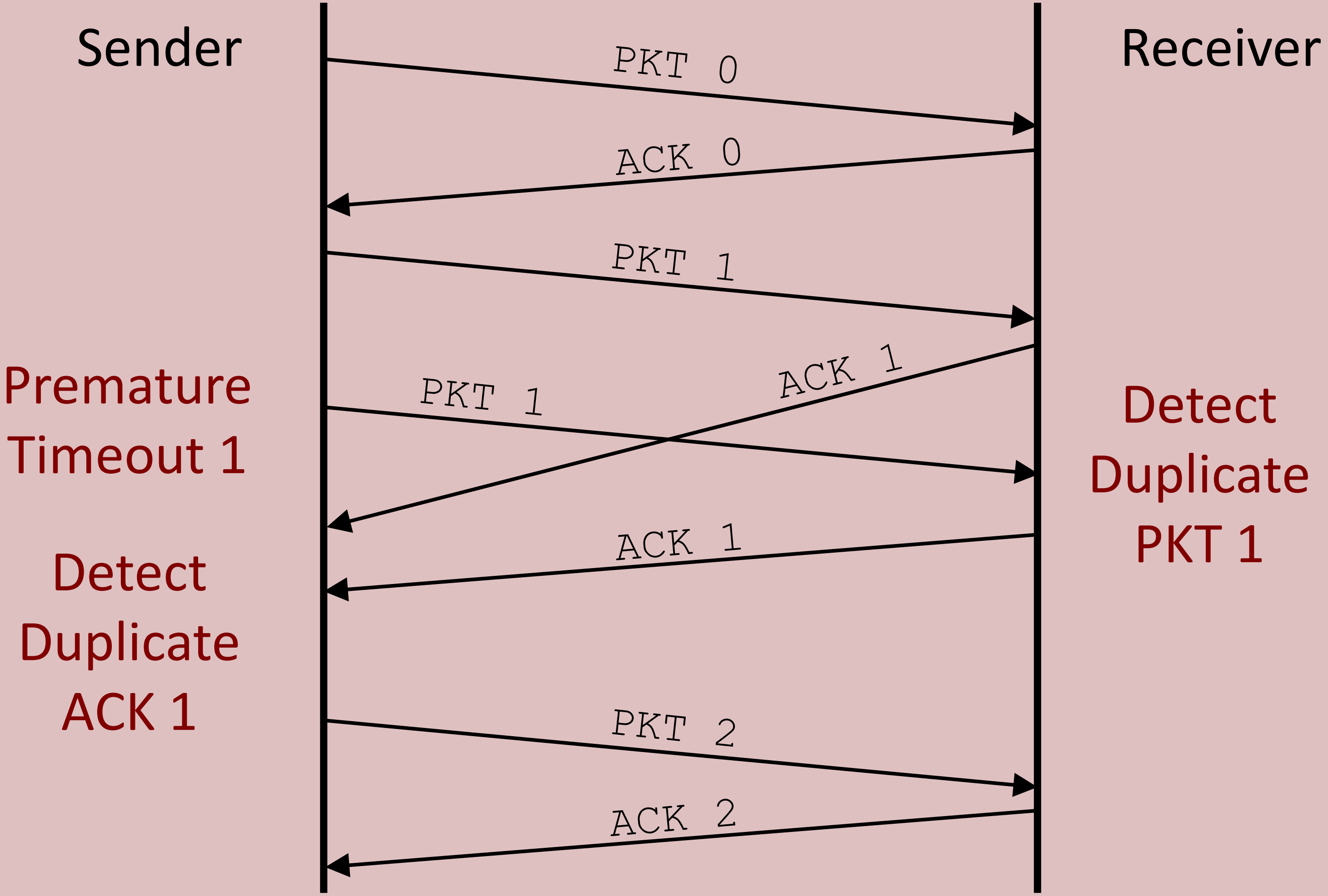
Stop-and-Wait – Packet Loss



Stop-and-Wait – ACK Loss



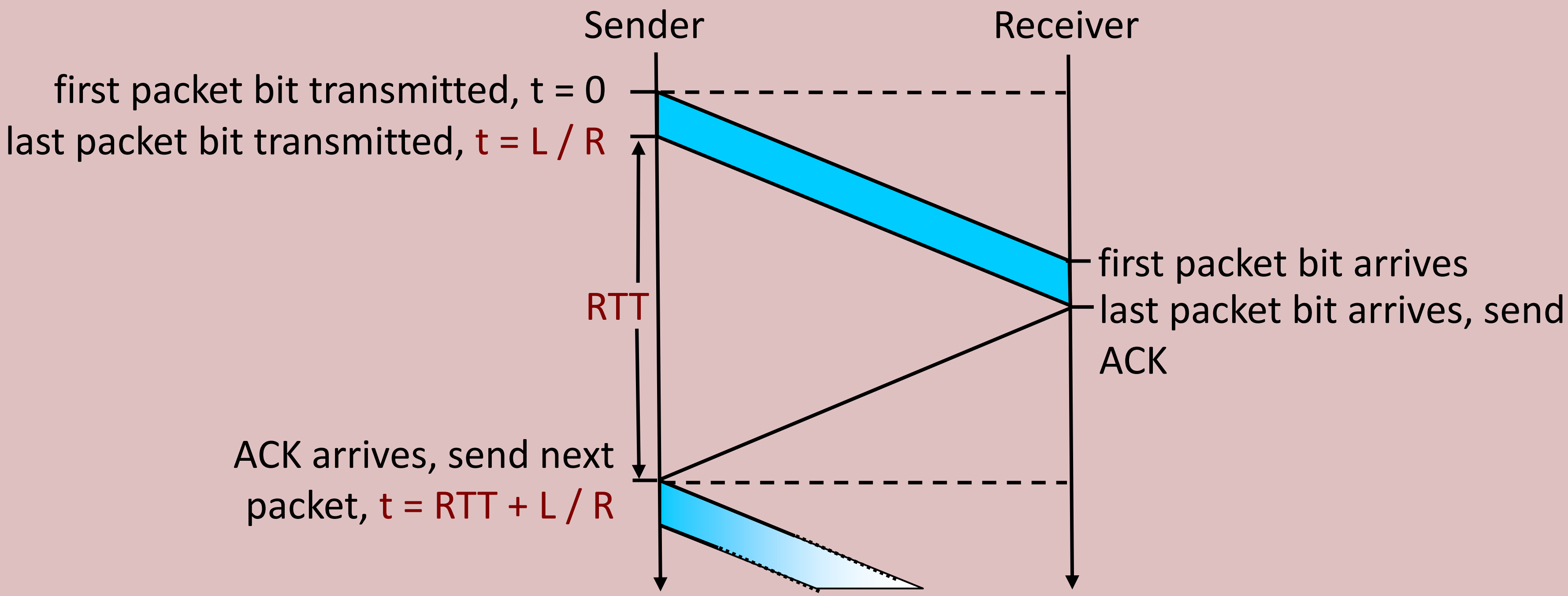
Stop-and-Wait – Premature Timeout



TFTP Lock Step ACKs

- Similar to two stop-and-wait protocol instances (one in each direction)
- Writer sends DATA and waits for ACKs
- Reader sends ACKs and waits for DATA (i.e., DATA is like an ACK of the ACK)
- DATA and ACKs can be retransmitted based on timers

Stop-and-Wait



$$U_{sender} = \frac{L / R}{RTT + L / R}$$

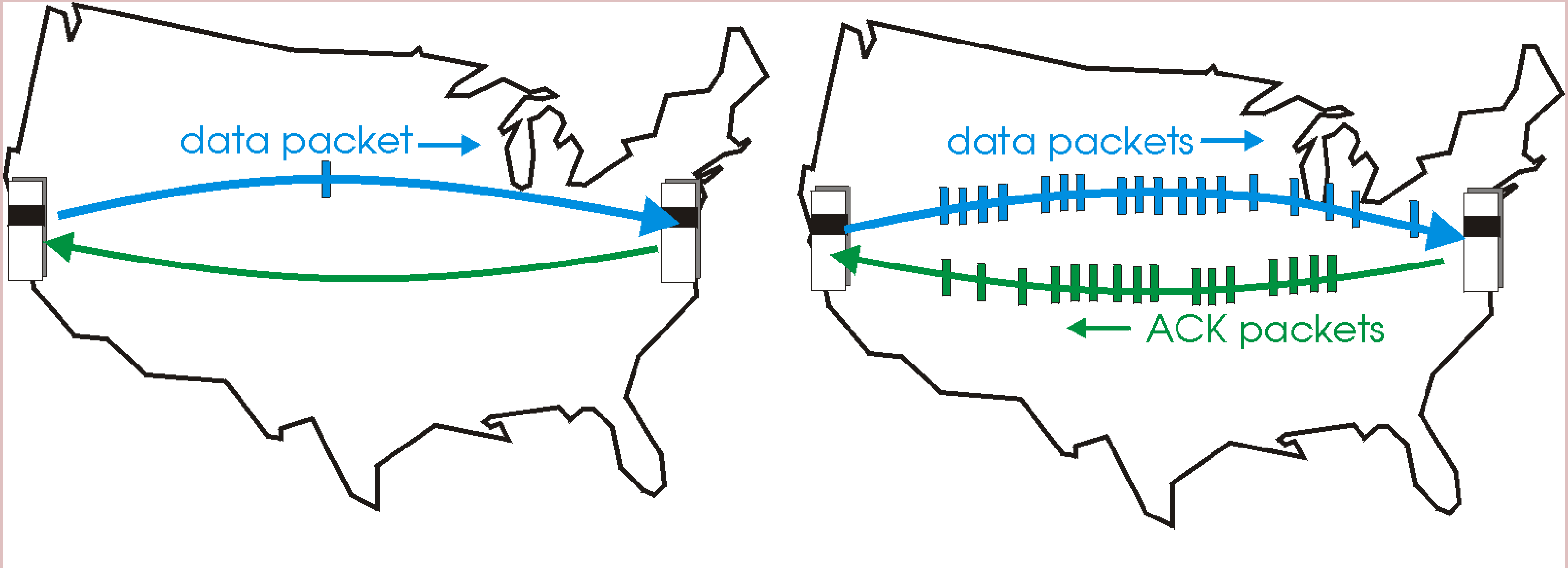
Very poor utilization for small packets, high transmission rates, and high RTTs!

Does this problem look familiar?

Pipelining

- Improves efficiency over stop-and-wait
- Sender is allowed to send from a sliding window
  - Collection of packets with a subset of unacknowledged packets
  - Fixed or variable window size
- Packet buffers are required at sender and also possibly at receiver

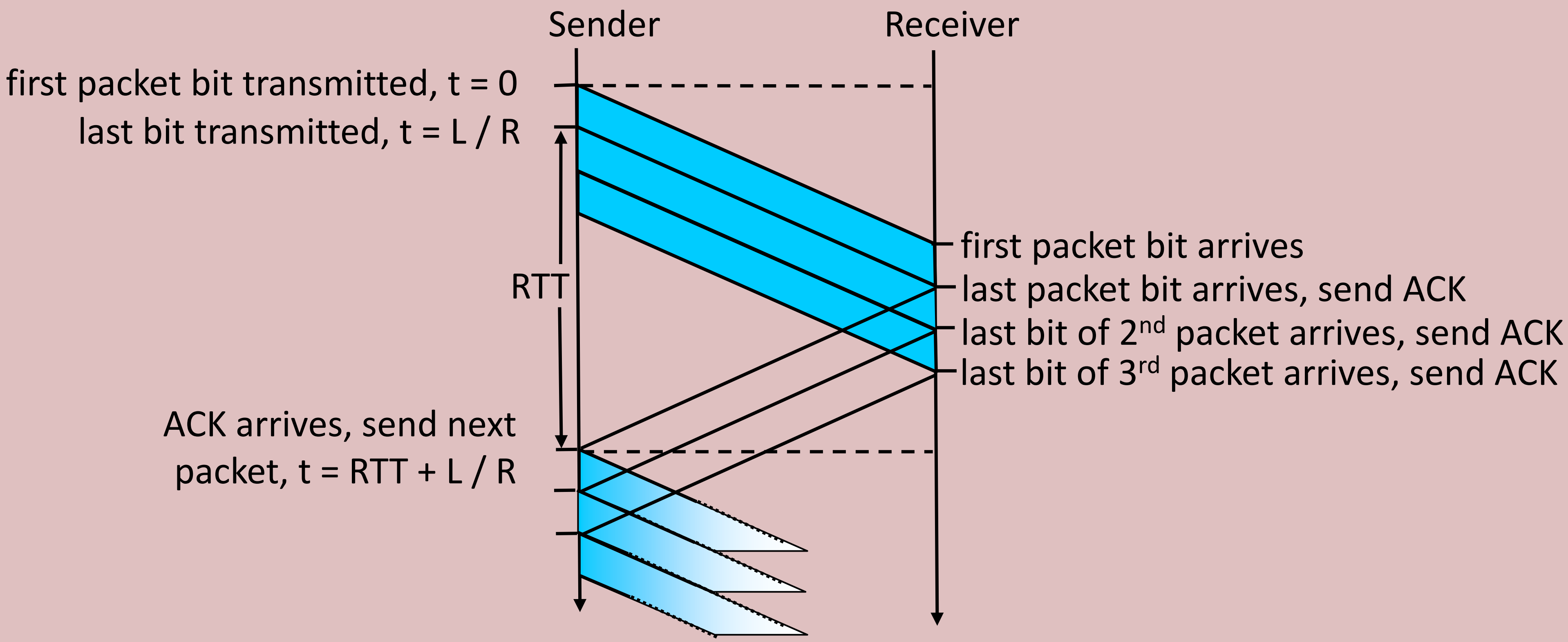
Pipelining



(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

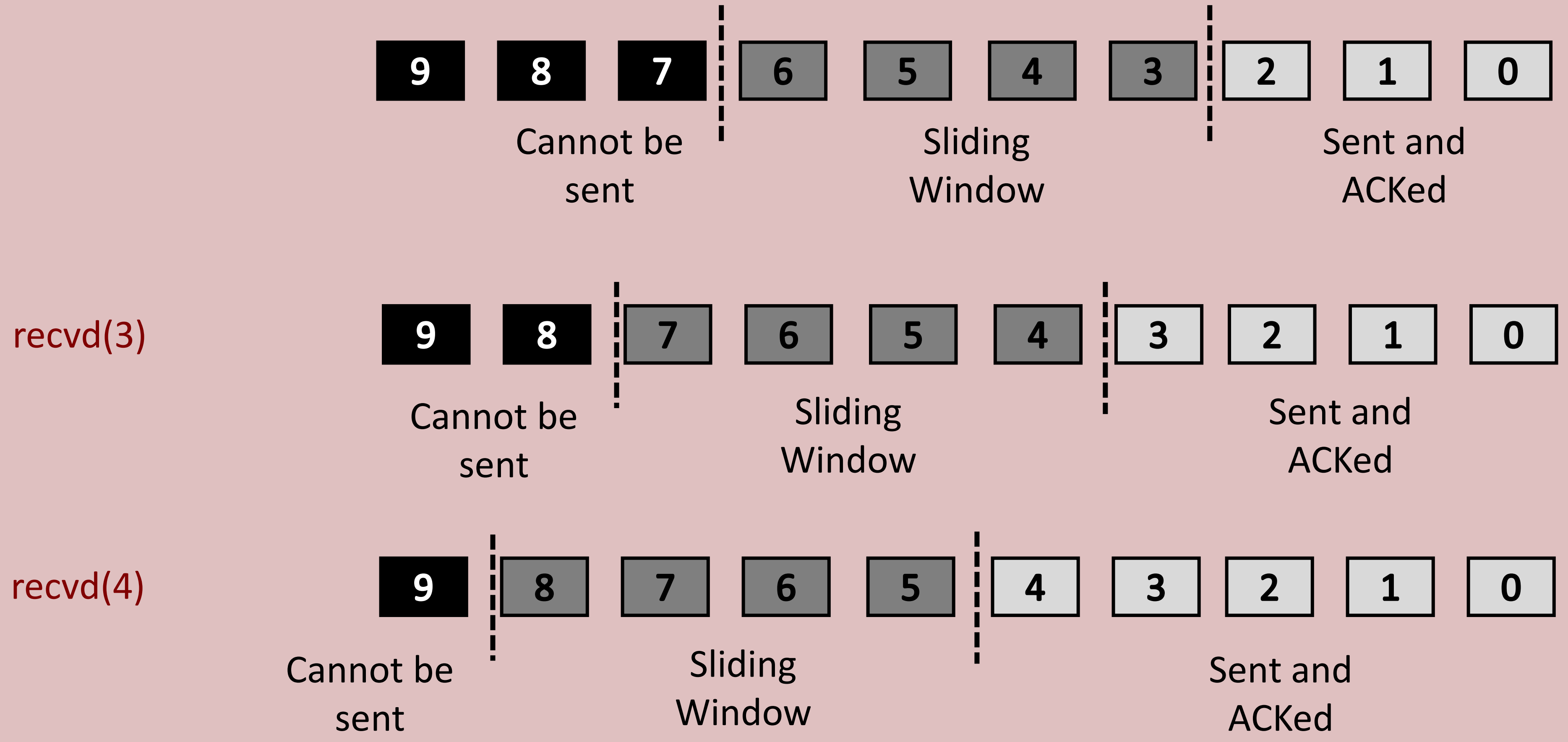
Pipelining



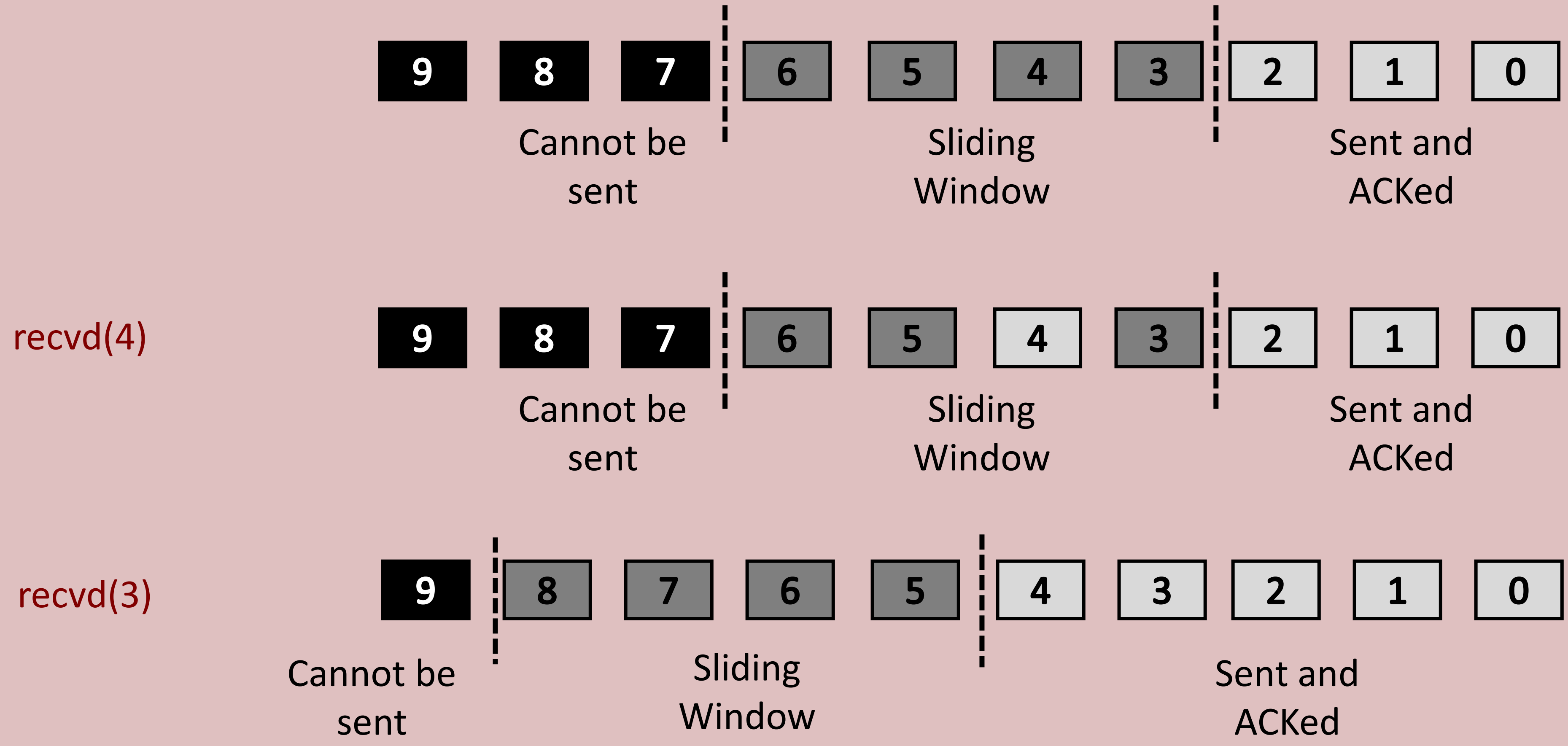
$$U_{sender} = \frac{3L/R}{RTT + L/R}$$

3-packet pipelining increases utilization by a factor of 3!

Sliding Windows



Sliding Windows



Go-Back-N

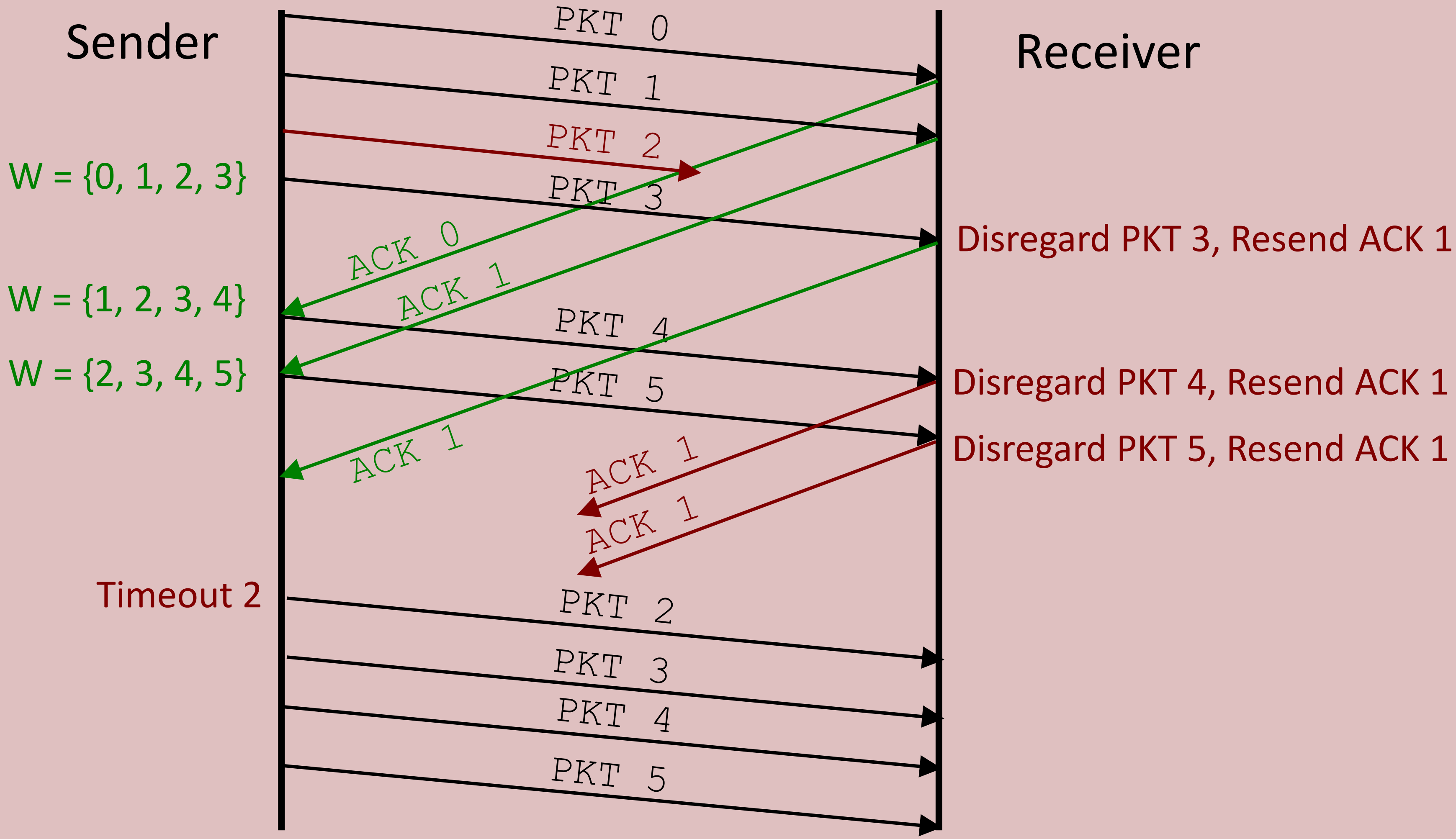
Sender:

- Allowed to have up to *N* unACKed packets in pipeline
- Maintains timer for oldest unACKed packet
- Retransmit all unACKed packets when timer expires

Receiver:

- Sends cumulative ACKs (i.e., ACK all packet numbers up to sequence number X)
- Do not ACK packet if there is a gap
- No need to buffer out-of-order packets

Go-Back-N


Sender
Receiver
PKT 0
PKT 1
PKT 2
W = {0, 1, 2, 3}
PKT 3
Disregard PKT 3, Resend ACK 1
ACK 0
ACK 1
W = {1, 2, 3, 4}
PKT 4
W = {2, 3, 4, 5}
Disregard PKT 4, Resend ACK 1
PKT 5
Disregard PKT 5, Resend ACK 1
ACK 1
ACK 1
ACK 1
Timeout 2
PKT 2
PKT 3
PKT 4
PKT 5

Selective Repeat

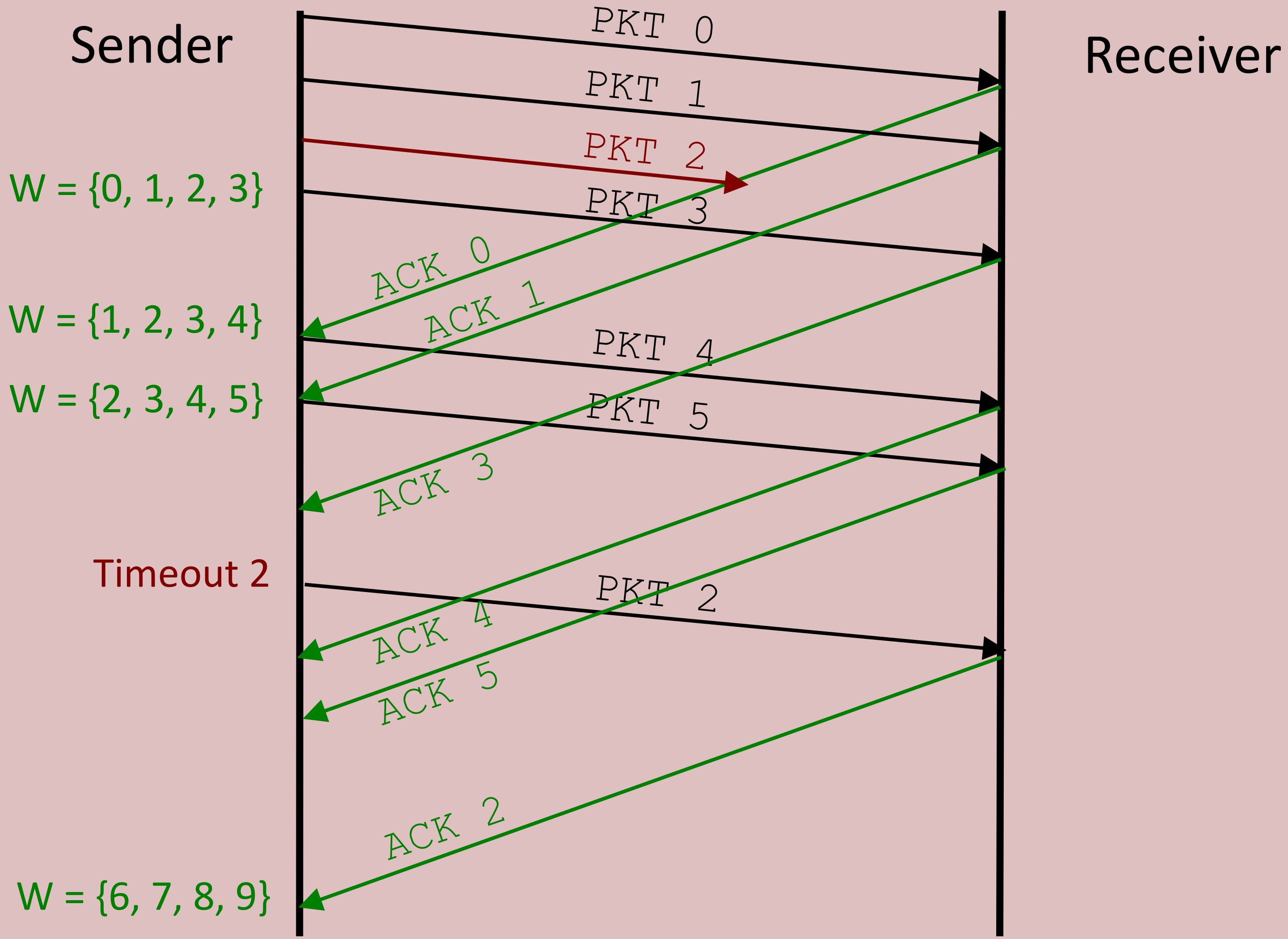
Sender:

- Allowed to have up to *N* unACKed packets in pipeline
- Maintains timer for each unACKed packet
- Retransmit unACKed packet when timer expires

Receiver:

- Sends individual ACK for each packet
- Must buffer out-of-order packets

Selective Repeat



Thank You!