ECE/COMPSCI 356 Computer Network Architecture

Lecture 19: TCP Reliable Transmission

Neil Gong

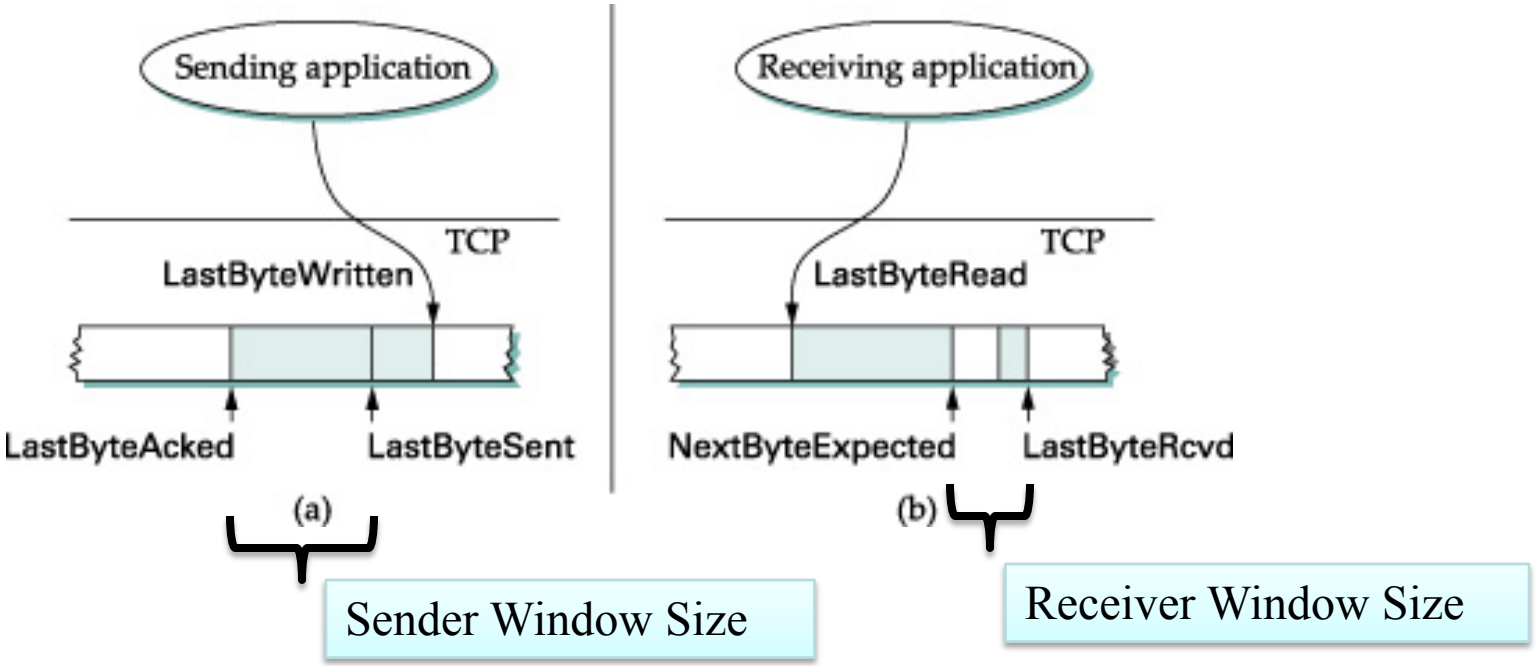
neil.gong@duke.edu

Slides credit: Xiaowei Yang, PD

Roadmap

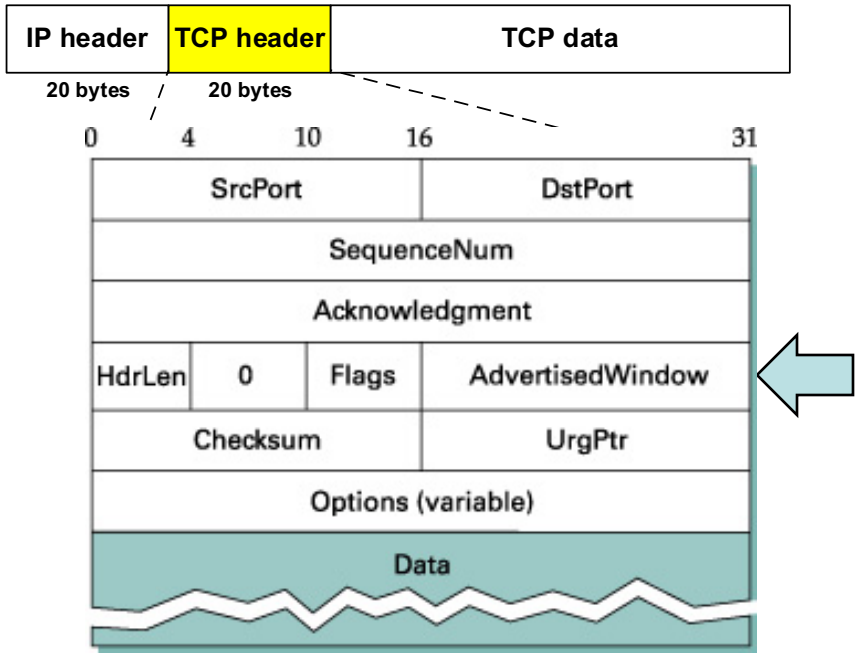
- Reliable transmission via sliding window
  - Flow control
  - When to transmit a segment
  - Adaptive retransmission
  - Modern extensions

Sliding window revisited



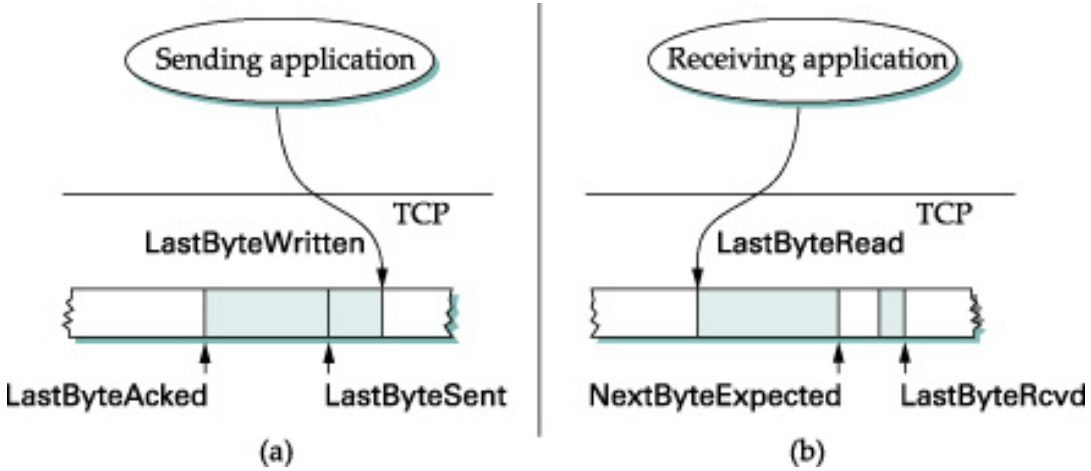
- Invariants
  - LastByteAcked ≤ LastByteSent
  - LastByteSent ≤ LastByteWritten
  - LastByteRead < NextByteExpected
  - NextByteExpected ≤ LastByteRcvd + 1
- Limited sending buffer and Receiving buffer

TCP Flow Control



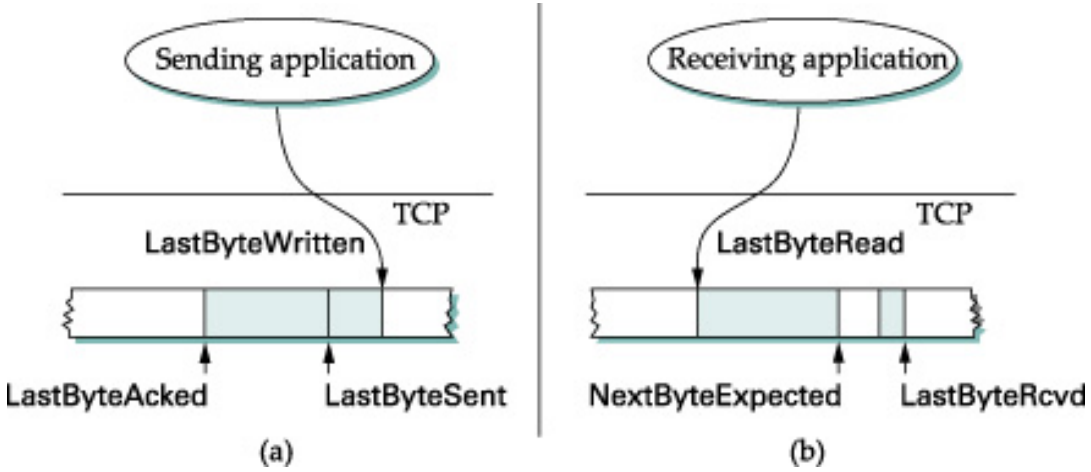
- Q: how does a receiver prevent a sender from overrunning its buffer?
- A: use AdvertisedWindow

Invariants for flow control



- Receiver side:
  - LastByteRcvd – LastByteRead ≤ MaxRcvBuf
  - AdvertisedWindow = MaxRcvBuf – ((NextByteExpected - 1) – LastByteRead)

Invariants for flow control



- Sender side:

  EffectiveWindow = AdvertisedWindow – (LastByteSent – LastByteAcked)

  – LastByteWritten – LastByteAcked ≤ MaxSndBuf
    - Sender process would be blocked if send buffer is full

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
    - Communication gets stuck

- Solution
  - Receiver sends duplicate ACKs when window opens
  - Sender sends periodic 1 byte probes ✔
- Why?
  - Keeping the receive side simple → Smart sender/dumb receiver

When to send a segment (assume no flow control)?

- App writes bytes to a TCP socket
- TCP decides when to send a segment

- Design choices when window opens:
  - Send whenever data available
  - Send when collected Maximum Segment Size (MSS) data ✔
    - More efficient

Push flag

- What if App is interactive, e.g. ssh?
  - App sets the PUSH flag
  - Flush the send buffer

Silly Window Syndrome

- Now considers flow control
  - Window opens, but does not have MSS bytes

- Potential solution: send all it has
  - E.g., sender sends 1 byte, receiver acks 1, acks opens the window by 1 byte, sender sends another 1 byte, and so on
  - Silly Window Syndrome

How to avoid Silly Window Syndrome

- Receiver side
  - Do not advertise small window sizes
- Sender side
  - Wait until it has a large segment to send
  - Q: How long should a sender wait?

Sender-Side Silly Window Syndrome avoidance

- Nagle's Algorithm
- Interactive applications may turn off Nagle's algorithm using the TCP_NODELAY socket option

```
When app has data to send
    if data and window >= MSS
        send a full segment
    else
        if there is unACKed data
            buffer new data until ACK
        else
            send all the new data now
```

TCP window management summary

- Receiver uses AdvertisedWindow for flow control
- Sender sends probes when AdvertisedWindow reaches zero
- Silly Window Syndrome avoidance
  - Receiver: do not advertise small windows
  - Sender: Nagle's algorithm

Overview

- Reliable transmission via sliding window
  - Flow control
  - When to transmit a segment
  - **Adaptive retransmission**
  - Modern extensions

TCP Retransmission

- A TCP sender retransmits a segment when it assumes that the segment has been lost

- How does a TCP sender detect a segment loss?
  - Timeout
  - Selective ACKs

How to set the timer

- Challenge: RTT unknown and variable
- Too small
  - Results in unnecessary retransmissions
- Too large
  - Long waiting time

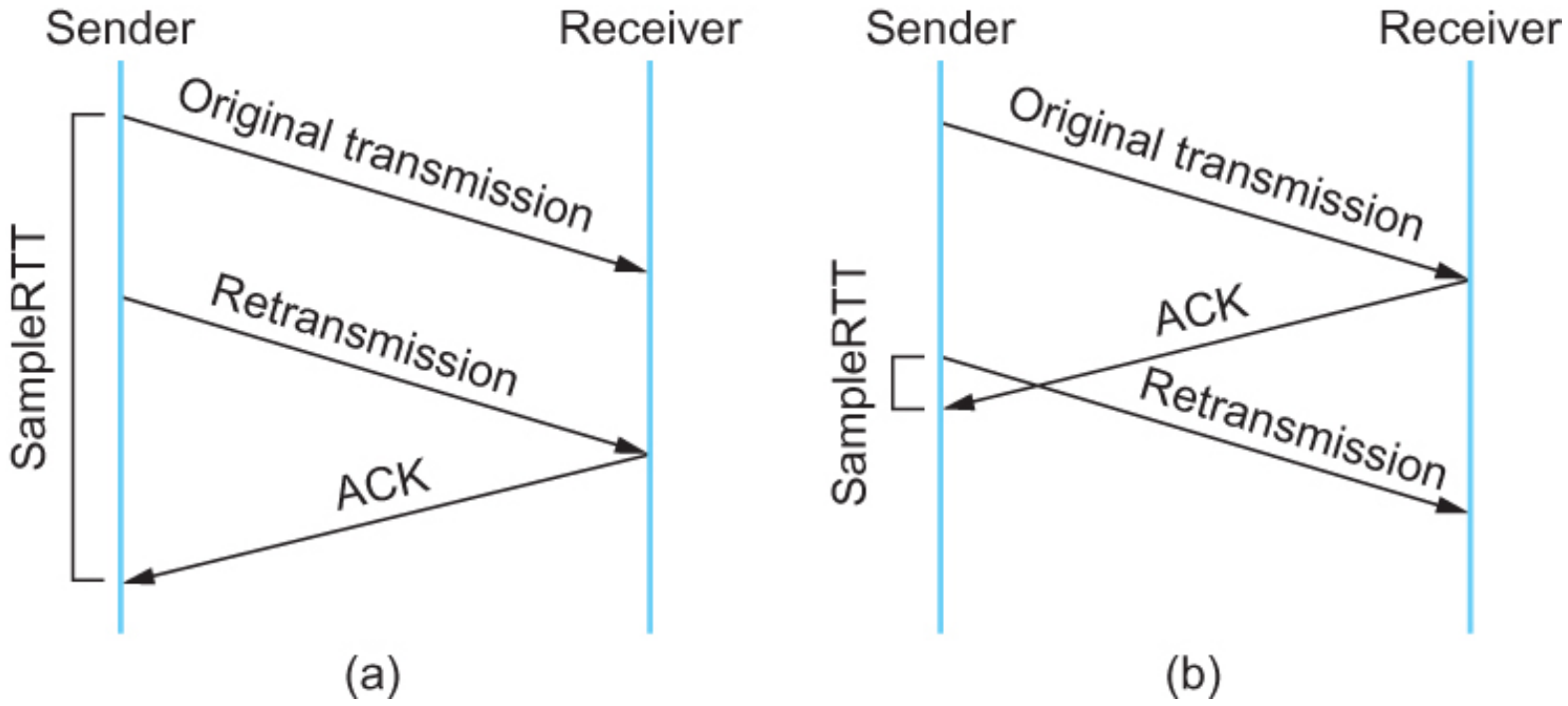
Estimating RTT

- Original Algorithm
  - Measure **`SampleRTT`** for each segment/ ACK pair
  - Compute weighted average of RTT
    - **`EstRTT`** = α x **`EstRTT`** + (1 - α)x **`SampleRTT`**
    - α between 0.8 and 0.9
  - Set timeout based on **`EstRTT`**
    - **`TimeOut`** = **`2`** x **`EstRTT`**

Estimating RTT

- Problem
  - ACK does not really acknowledge a transmission
    - It actually acknowledges the receipt of data
  - When a segment is retransmitted and then an ACK arrives at the sender
    - It is impossible to decide if this ACK should be associated with the first or the second transmission for calculating RTTs

Illustration of the problem



Associating the ACK with (a) original transmission versus (b) retransmission

Karn/Partridge Algorithm to solve the problem

- Do not sample RTT when retransmitting
- Double timeout after each retransmission

Karn/Partridge Algorithm

- Karn-Partridge algorithm was an improvement over the original approach, but it does not eliminate congestion

- We need to understand how timeout is related to congestion
  - If you timeout too soon, you may unnecessarily retransmit a segment which adds load to the network

Karn/Partridge Algorithm

- Main problem
  - variance of Sample RTTs is not considered.
- If the variance among Sample RTTs is small
  - Then the Estimated RTT can be better trusted
  - No need to multiply this by 2 to compute the timeout

Karn/Partridge Algorithm

- A large variance in the samples suggest that timeout value should not be tightly coupled to the Estimated RTT
- Jacobson/Karels proposed a new scheme for TCP retransmission

Jacobson/Karels Algorithm

- Difference = SampleRTT − EstimatedRTT
- EstimatedRTT = EstimatedRTT + (δ× Difference)
- Deviation = Deviation + δ (|Difference| − Deviation)
- TimeOut = μ × EstimatedRTT + ϕ × Deviation
  - δ is between 0 and 1
  - Based on experience, μ is typically set to 1 and ϕ is set to 4. Thus, when the variance is small, TimeOut is close to EstimatedRTT; a large variance causes the deviation term to dominate the calculation.

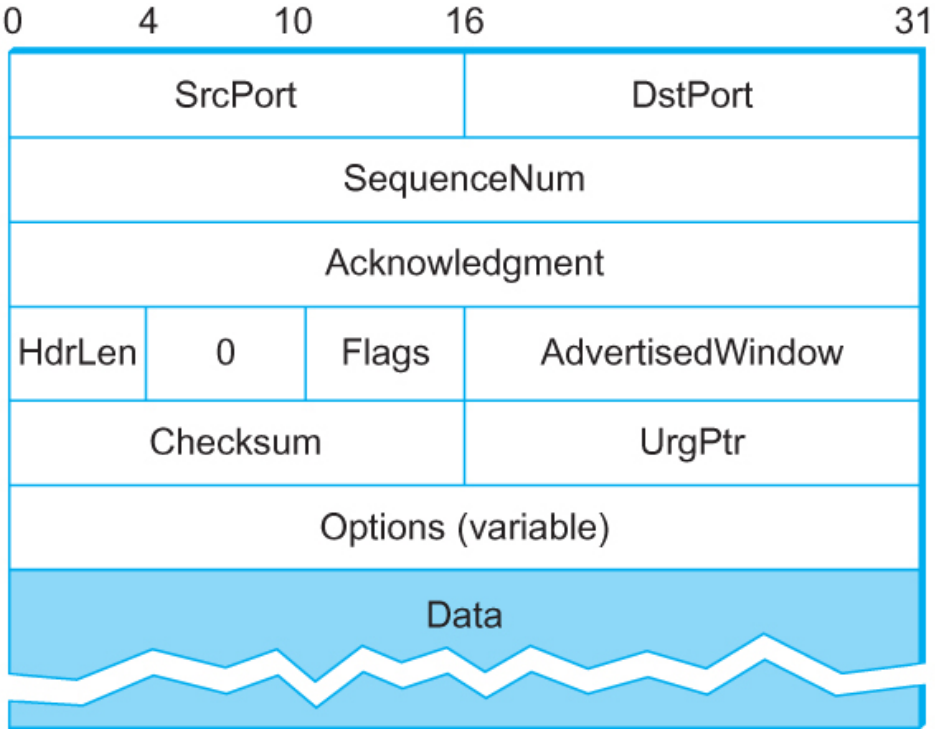
Overview

- Reliable transmission via sliding window
  - Flow control
  - When to transmit a segment
  - Adaptive retransmission
  - **Modern extensions**

Modern TCP extensions

- Timestamp
- Window scaling factor
- Protection Against Wrapped Sequence Numbers (PAWS)
- Selective Acknowledgement (SACK)
- References
  - http://www.ietf.org/rfc/rfc1323.txt
  - http://www.ietf.org/rfc/rfc2018.txt

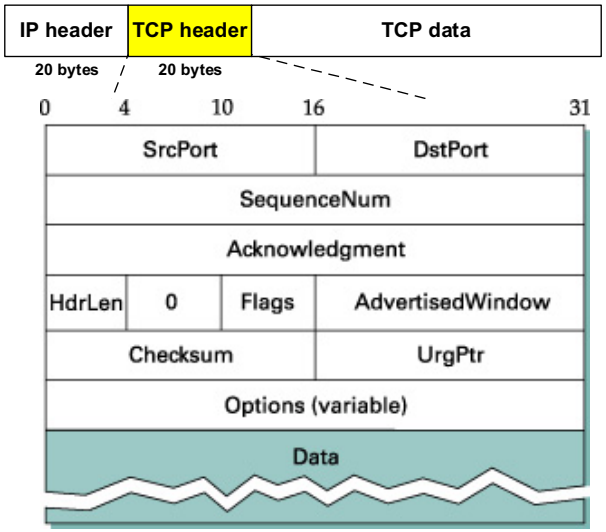
Options define the extensions



RTT estimate via timestamp

- Sender includes a timestamp in a segment
- Receiver echoes the timestamp in an ACK
- RTT for a segment = current_time – timestamp in ACK

TCP window size is small



- 16-bit window size
- Maximum send window <= 65535B
- Suppose a RTT is 100ms
- Max TCP throughput = 65KB/100ms = 5Mbps
- Not good enough for modern high speed links!

Solution: Window scaling option

- Scale windows by a factor
  - Each unit is 2 bytes instead of one byte
  - Each unit is 4 bytes instead of one byte

Sequence number wraps around

- 32-bit sequence number space
- TCP requires no wrap around in 120-second period of time (life time of a packet on the Internet)
- Sequence numbers may wrap around in 120-second period of time for high speed link

Bandwidth	Time until Wraparound
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
Fast Ethernet (100 Mbps)	6 minutes
OC-3 (155 Mbps)	4 minutes
OC-12 (622 Mbps)	55 seconds
OC-48 (2.5 Gbps)	14 seconds

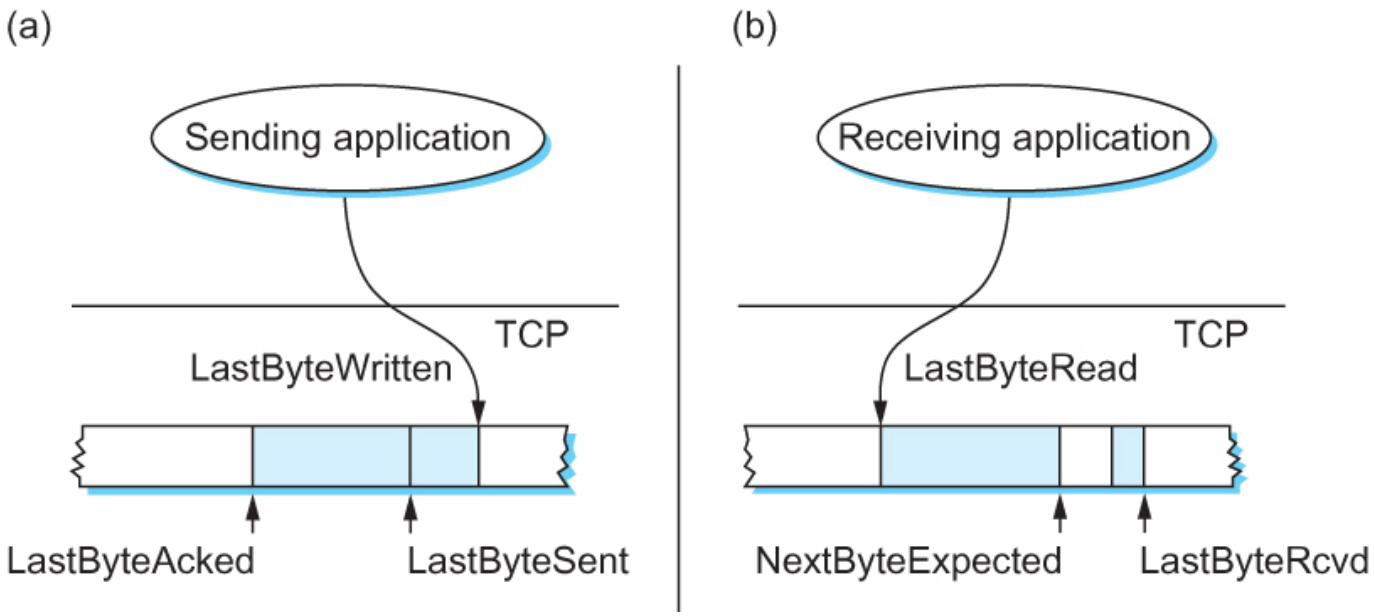
Time until 32-bit sequence number space wraps around.

Solution: Compare timestamp

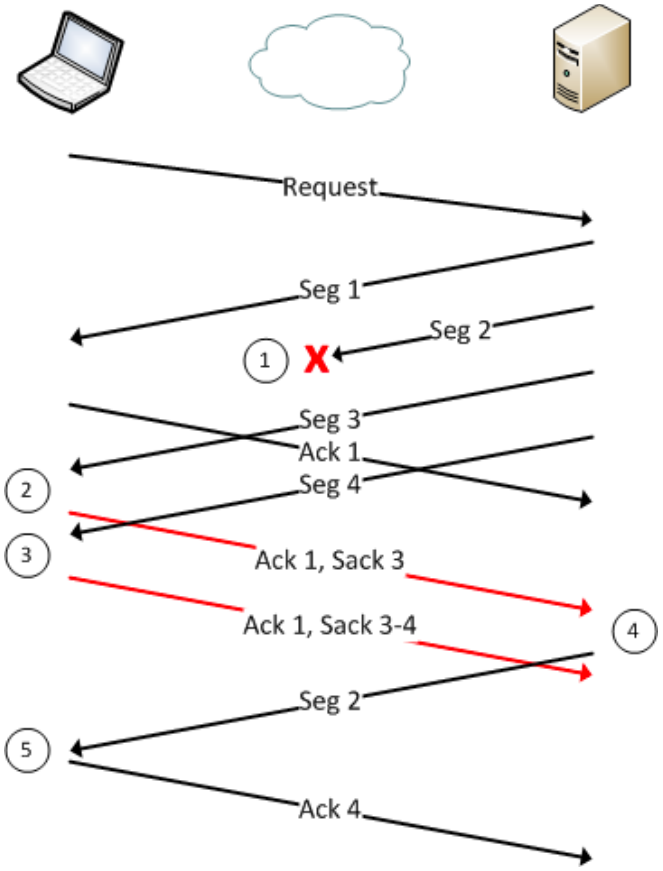
- Receiver keeps the segment with the latest timestamp
- Discard duplicate segments with old timestamps

Selective Acknowledgement

- Ack the received blocks even if not contiguous



Duplicate and Selective Acknowledgement



Summary

- Reliable transmission via sliding window
  - Flow control
  - When to transmit a segment
  - Adaptive retransmission
  - Modern extensions