CS 356: Introduction to Computer Networks

Lecture 19: Transmission Control Protocol (TCP)
Chap. 5.2

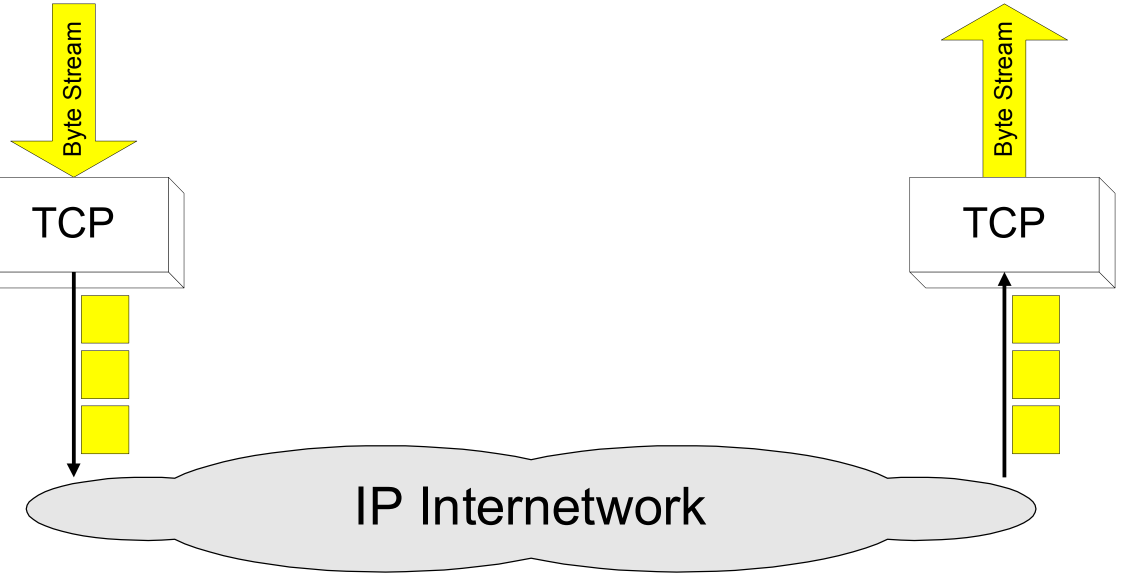
Xiaowei Yang
xwy@cs.duke.edu

Overview

- TCP
  - Connection management
  - Flow control
  - When to transmit a segment
  - Adaptive retransmission
  - TCP options
    - Modern extensions

Transmission Control Protocol

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

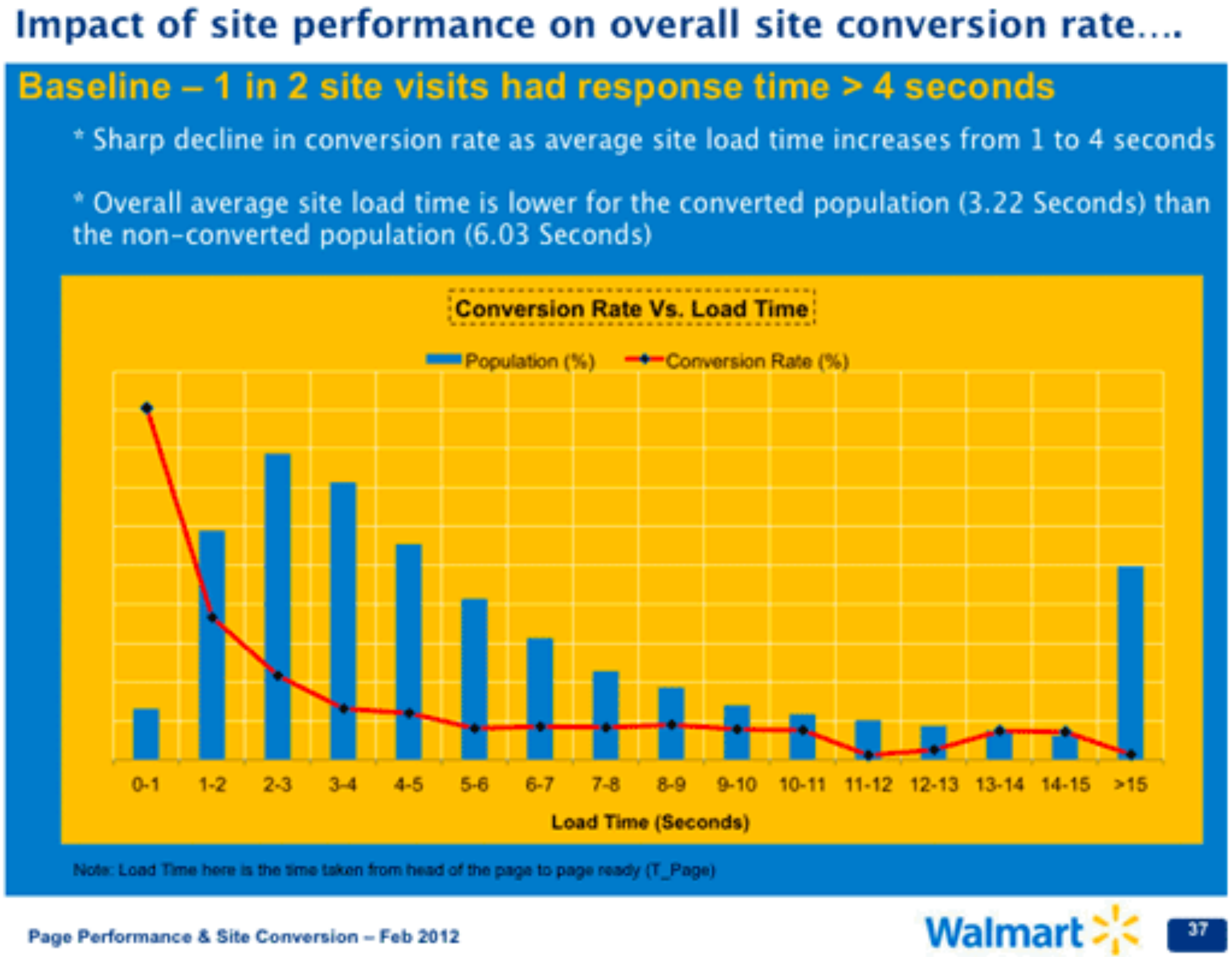


TCP performance is critical to business

	BOUNCE RATE	CONVERSION RATE	CART SIZE	PAGE VIEWS
200 ms	—	—	—	-1.2%
500 ms	-4.7%	-1.9%	—	-5.7%
1000 ms	-8.3%	-3.5%	-2.1%	-9.4%

— NO SIGNIFICANT CHANGE

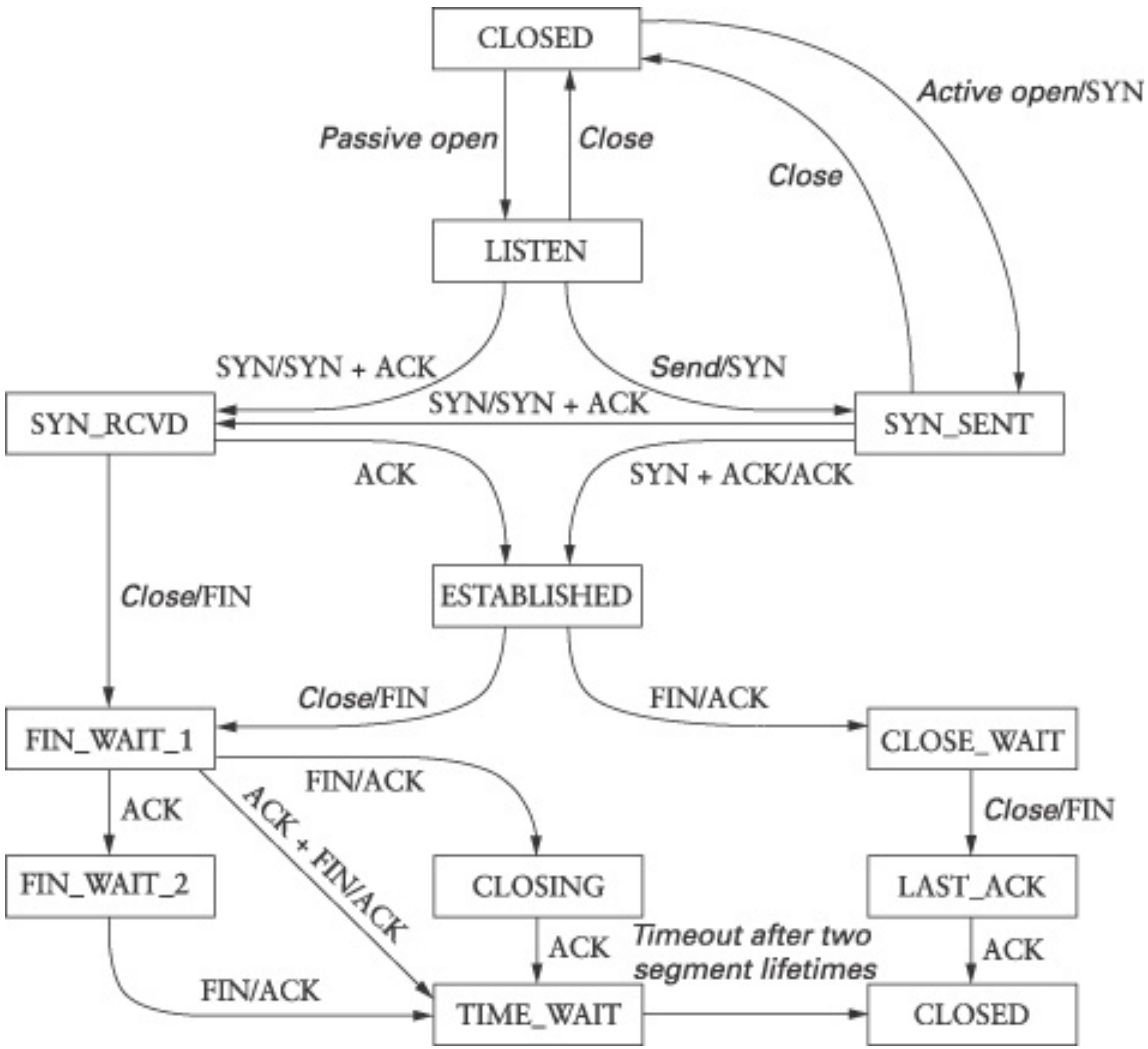
Source: http://www.webperformancetoday.com/2011/11/23/case-study-slow-page-load-mobile-business-metrics/



Source: http://www.webperformancetoday.com/2012/02/28/4-awesome-slides-showing-how-page-speed-correlates-to-business-metrics-at-walmart-com/

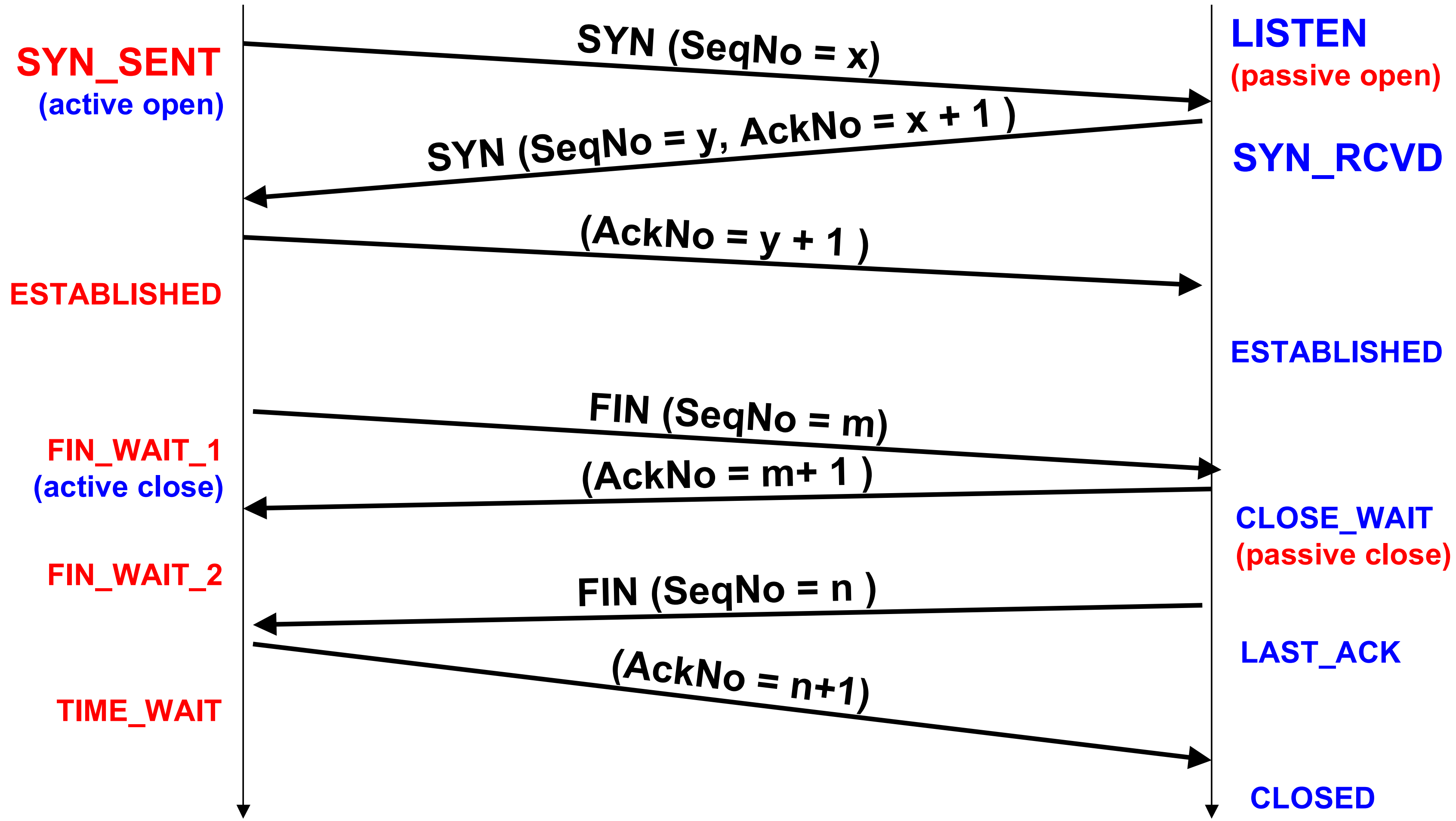
Connection establishment/tear down

- Active/passive open
- Active/passive close, simultaneous close



- Two events trigger transitions:
  - Packet arrival
  - Application operations
- Half-close end may still receive data
- Time_Wait
  - Must wait because ACK may be lost
  - The other end may retransmit FIN

TCP States in "Normal" Connection Lifetime

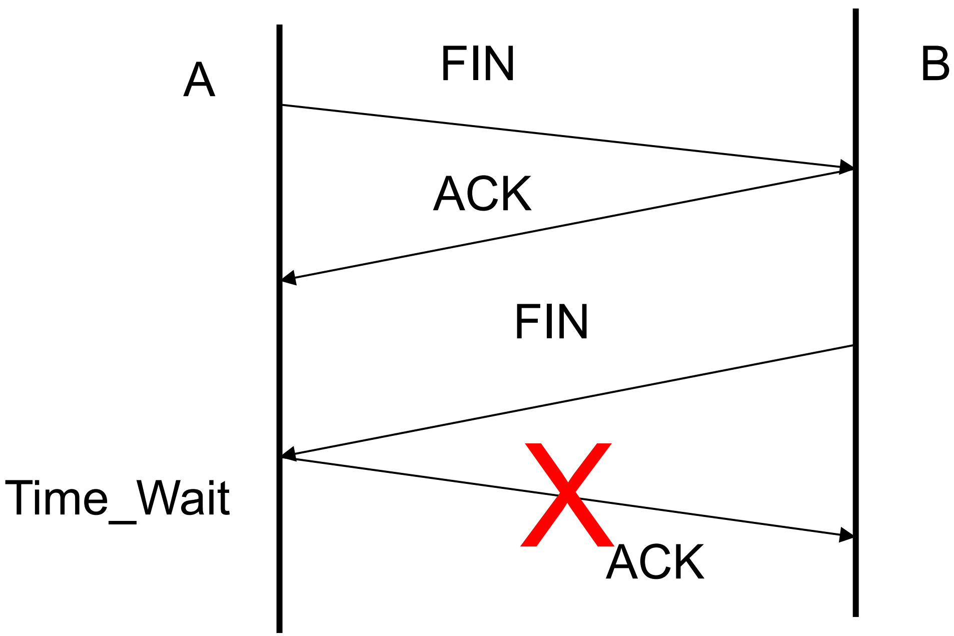


2MSL Wait State

2MSL= 2 * Maximum Segment Lifetime

2MSL Wait State = TIME_WAIT

- When TCP does an active close, and sends the final ACK, the connection **must stay in the TIME_WAIT state for twice the maximum segment lifetime**.
- The socket pair (srcIP, srcPort, dstIP, dstPort) cannot be reused



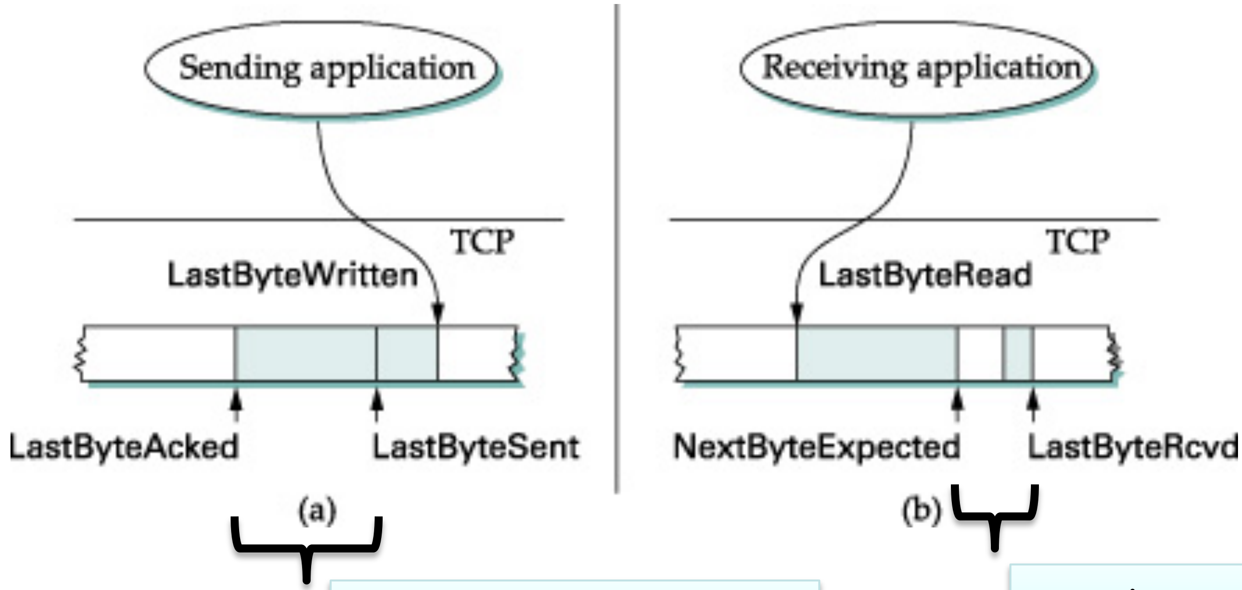
- Why?
- To prevent mixing packets from two different incarnations of the same connection

Resetting Connections

- Resetting connections is done by setting the RST flag
- **When is the RST flag set?**
  - Connection request arrives and no server process is waiting on the destination port
  - Abort a connection causes the receiver to throw away buffered data
  - Receiver does not acknowledge the RST segment
  - Abused in practice to block applications

Flow control

Sliding window revisited

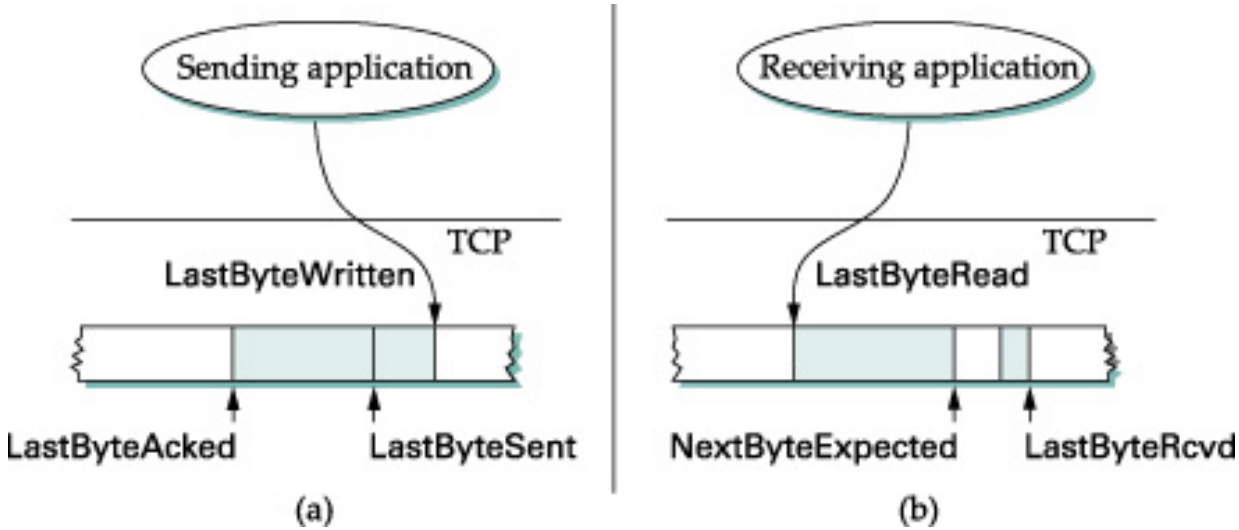


Sender Window Size

Receiver Window Size

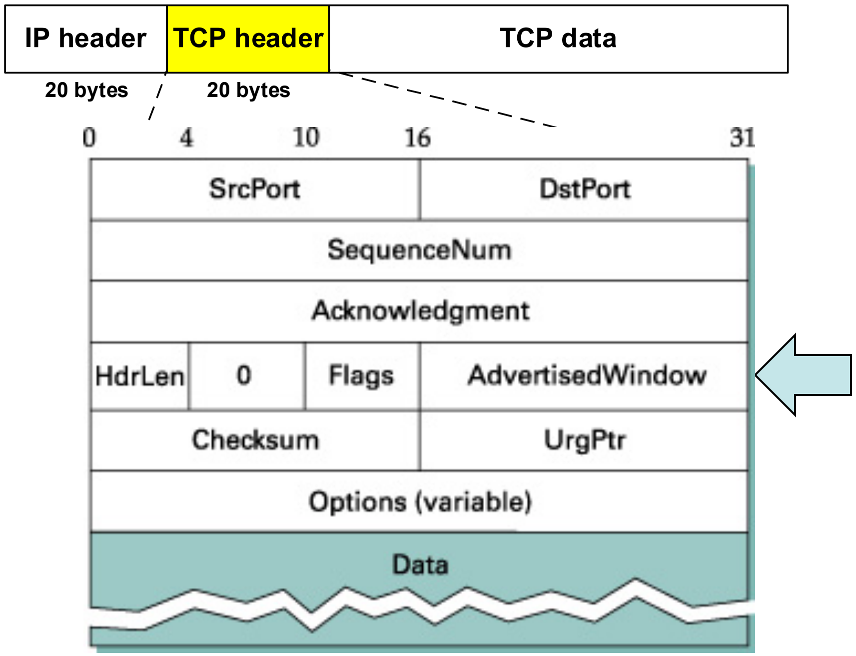
- Invariants
  - LastByteAcked ≤ LastByteSent
  - LastByteSent ≤ LastByteWritten
  - LastByteRead < NextByteExpected
  - NextByteExpected ≤ LastByteRcvd + 1
- Limited sending buffer and Receiving buffer

Buffer Sizes vs Window Sizes



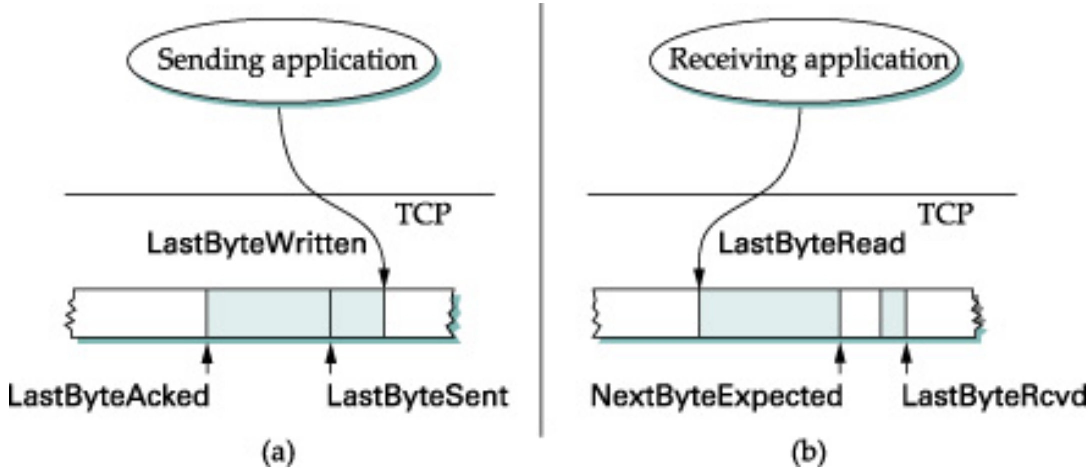
- Maximum SWS ≤ MaxSndBuf
- Maximum RWS ≤ MaxRcvBuf – ((NextByteExpected-1) – LastByteRead)

TCP Flow Control



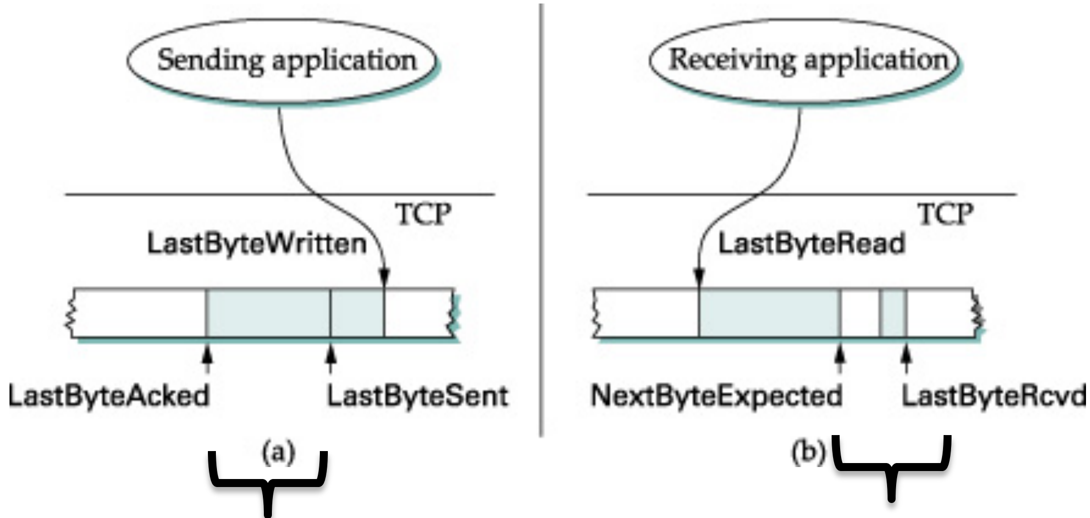
- Q: how does a receiver prevent a sender from overrunning its buffer?
- A: use AdvertisedWindow

Invariants for flow control



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

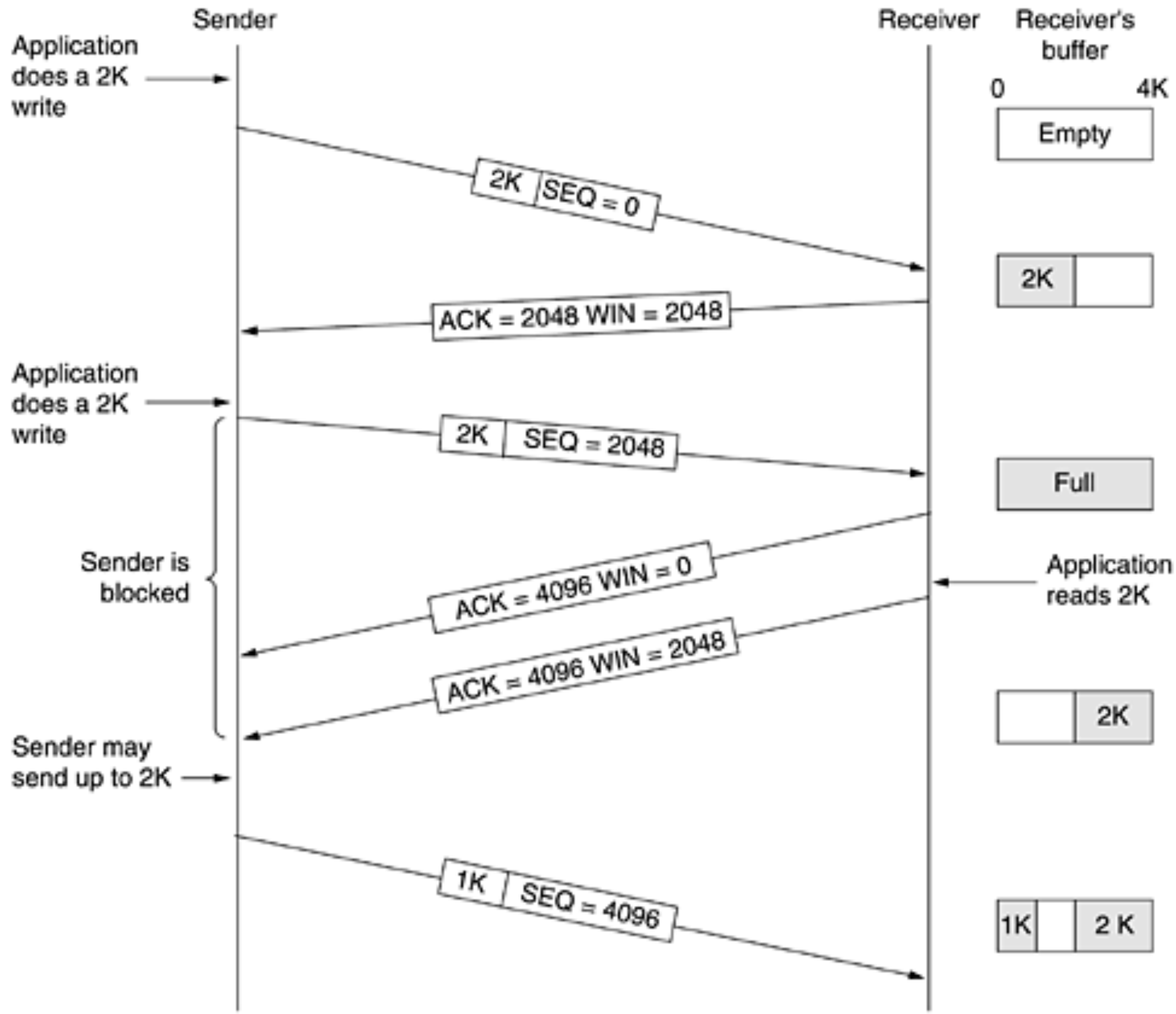
Invariants for flow control



- Sender side:

  EffectiveWindow = AdvertisedWindow – (LastByteSent – LastByteAcked)

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


Sender
Receiver
Receiver's buffer
0
4K
Application does a 2K write
Empty
2K SEQ = 0
2K
ACK = 2048 WIN = 2048
Application does a 2K write
2K SEQ = 2048
Full
Sender is blocked
ACK = 4096 WIN = 0
Application reads 2K
ACK = 4096 WIN = 2048
2K
Sender may send up to 2K
1K SEQ = 4096
1K
2 K

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 ✔
- Why?
  - Keeping the receive side simple → Smart sender/dumb receiver

When to send a segment?

- 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 data ✔
    - Why?
    - More efficient

Push flag

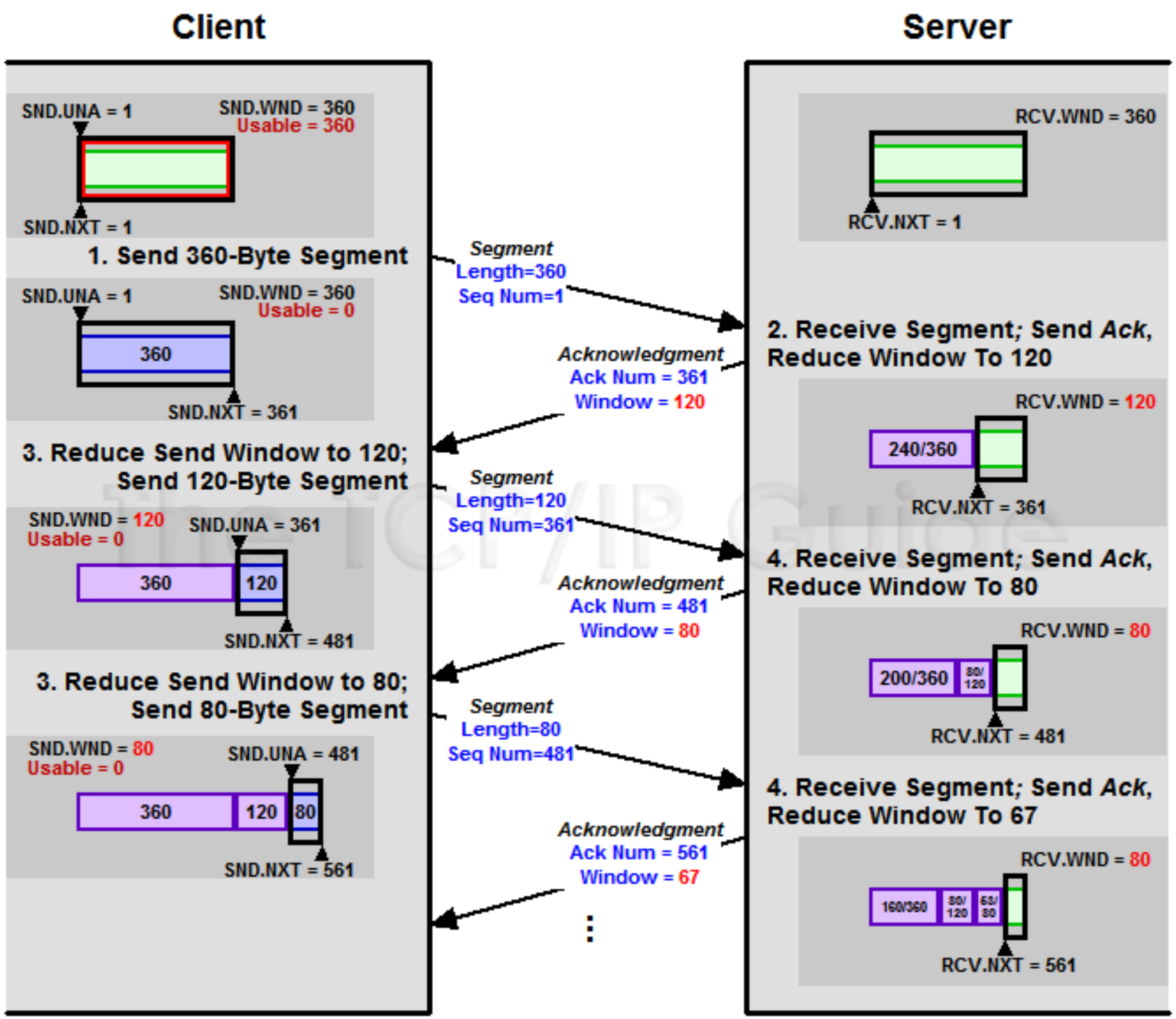
- What if App is interactive, e.g. ssh?
  - App sets the PUSH flag
  - Flush the sent buffer

Silly Window Syndrome

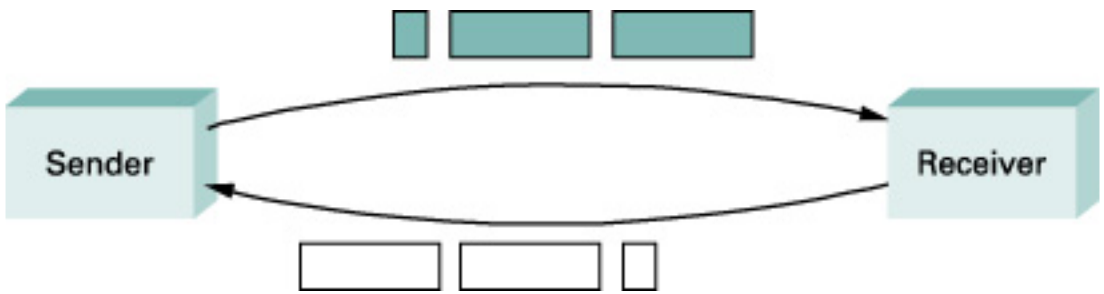
- Now considers flow control
  - Window opens, but does not have MSS bytes

- Design choice 1: 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

Sending smaller segments



Silly Window Syndrome



How to avoid Silly Window Syndrome

- Receiver side
  - Do not advertise small window sizes
  - Min (MSS, MaxRecvBuf/2)
- 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
  - Self-clocking

- 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

- TCP
  - Connection management
  - Flow control
  - When to transmit a segment
  - Adaptive retransmission
  - TCP options
    - 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
  - Duplicate ACKs (later)

How to set the timer

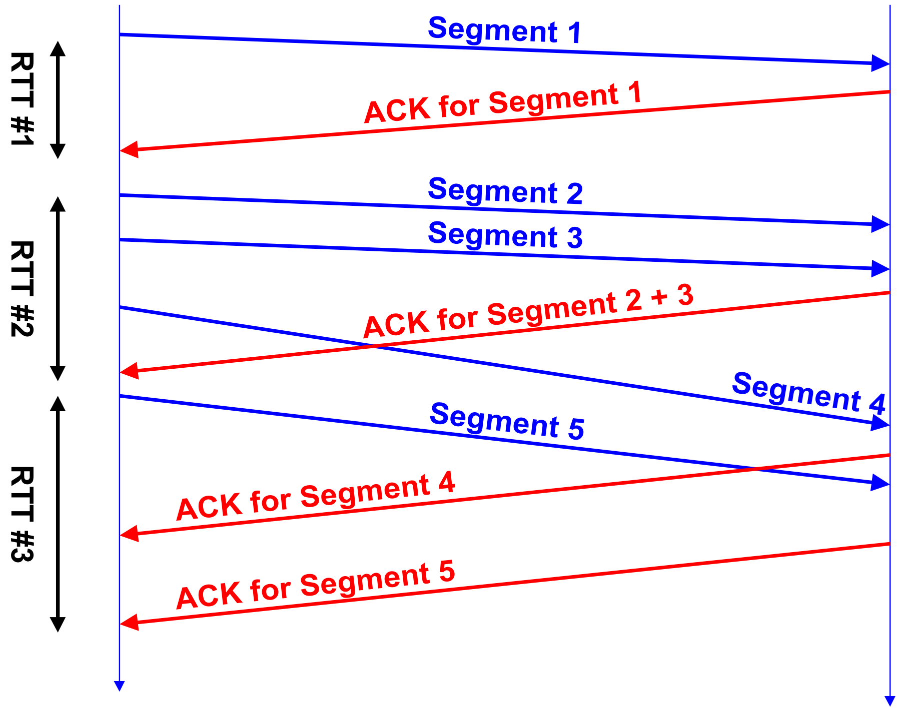
- Challenge: RTT unknown and variable
- Too small
  - Results in unnecessary retransmissions
- Too large
  - Long waiting time

Adaptive retransmission

- Estimate a RTO value based on round-trip time (RTT) measurements

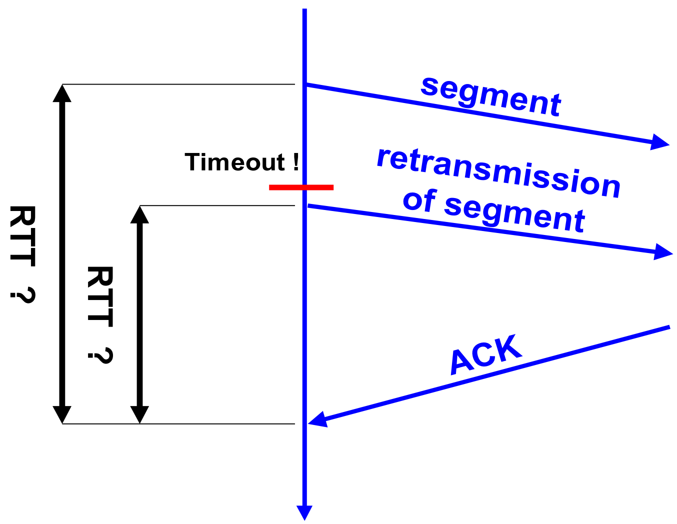
- Implementation: one timer per connection

- Q: Retransmitted segments?



Karn's Algorithm

- Ambiguity



- **Solution: Karn's Algorithm:**
  - Don't update RTT on any segments that have been retransmitted

Setting the RTO value

- Uses an exponential moving average (a low-pass filter) to estimate RTT (*srtt*) and variance of RTT (*rttvar*)
  - The influence of past samples decrease exponentially

- The RTT measurements are smoothed by the following estimators *srtt* and *rttvar*:

$$srtt_{n+1} = \alpha\ RTT + (1- \alpha)\ srtt_n$$
$$rttvar_{n+1} = \beta\,(\,|\,RTT - srtt_n\,|\,) + (1- \beta)\ rttvar_n$$
$$RTO_{n+1} = srtt_{n+1} + 4\ rttvar_{n+1}$$

  - The gains are set to $\alpha = 1/4$ and $\beta = 1/8$
  - Negative power of 2 makes it efficient for implementation

Setting the RTO value (cont'd)

- Initial value for RTO:
  - Sender should set the initial value of RTO to

$$RTO_0 = 3 \text{ seconds}$$

- RTO calculation after first RTT measurements arrived

$$srtt_1 = RTT$$
$$rttvar_1 = RTT / 2$$
$$RTO_1 = srtt_1 + 4\, rttvar_{n+1}$$

- When a timeout occurs , the RTO value is doubled

$$RTO_{n+1} = max\ (\ 2\ RTO_n\ ,\ 64)\ seconds$$

This is called *an exponential backoff*

Overview

- TCP
  - Connection management
  - Flow control
  - When to transmit a segment
  - Adaptive retransmission
  - TCP options
    - Modern extensions

TCP header fields

- **Options**: (type, length, value)
- TCP hdrlen field tells how long options are

End of Options	kind=0			
	1 byte			
NOP (no operation)	kind=1			
	1 byte			
Maximum Segment Size	kind=2	len=4	maximum segment size	
	1 byte	1 byte	2 bytes	
Window Scale Factor	kind=3	len=3	shift count	
	1 byte	1 byte	1 byte	
Timestamp	kind=8	len=10	timestamp value	timestamp echo reply
	1 byte	1 byte	4 bytes	4 bytes

TCP header fields

- **Options**:
  - **NOP** is used to pad TCP header to multiples of 4 bytes
  - **Maximum Segment Size**
  - **Window Scale Options**
    - Increases the TCP window from 16 to 32 bits, i.e., the window size is interpreted differently
    - This option can only be used in the SYN segment (first segment) during connection establishment time
  - **Timestamp Option**
    - Can be used for roundtrip measurements

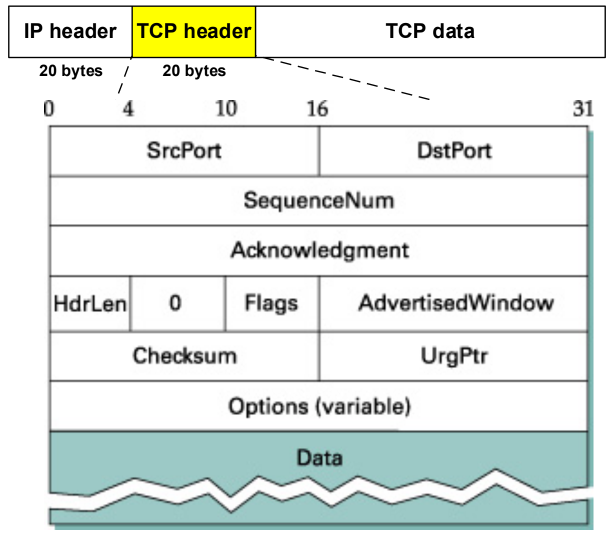
Modern TCP extensions

- 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

Improving RTT estimate

- TCP timestamp option
  - Old design
    - One sample per RTT
    - Using host timer
- More samples to estimate
  - Timestamp option
    - Current TS, echo TS

Increase TCP window size



- 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!

Protecting against Wraparound

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: Window scaling option

Kind = 3	Length = 3	Shift.cnt

Three bytes

- All windows are treated as 32-bit
- Negotiating shift.cnt in SYN packets
  - Ignore if SYN flag not set
- Sending TCP
  - Real available buffer >> self.shift.cnt → AdvertisedWindow
- Receiving TCP: stores other.shift.cnt
  - AdvertisedWindow << other.shift.cnt → Maximum Sending Window

Protect Against Wrapped Sequence Number

- 32-bit sequence number space
- Why sequence numbers may wrap around?
  - High speed link
  - On an OC-45 (2.5Gbps), it takes 14 seconds < 2MSL
- Solution: compare timestamps
  - Receiver keeps recent timestamp
  - Discard old timestamps

Selective Acknowledgement

- More when we discuss congestion control
- If there are holes, ack the contiguous received blocks to improve performance

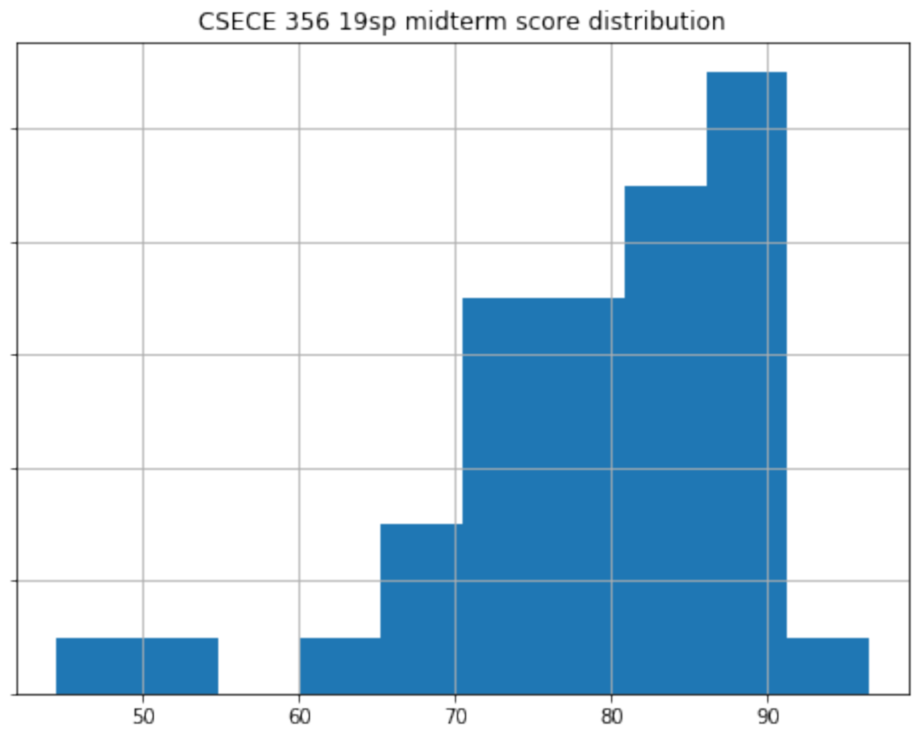
Summary

- Nitty-gritty details about TCP
  - Connection management
  - Flow control
  - When to transmit a segment
  - Adaptive retransmission
  - TCP options
  - Modern extensions
  - Next: Congestion Control
    - How does TCP keeps the pipe full?

Midterm statistics

Overall Scores

- MAXIMUM
  - 96.5
- MEAN
  - 79.79
- MEDIAN
  - 82.0
- STD DEV
  - 10.69



Quiz Score

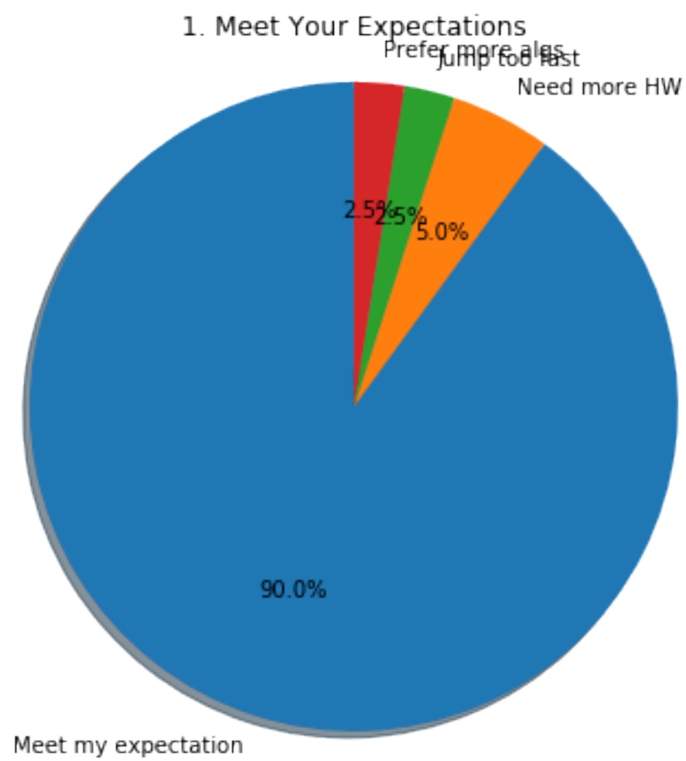
- Use mean to evaluate
- Take Quiz 1 82.021739
- Not Take Quiz 1 72.157895

- Take Quiz 2 82.025000
- Not Take Quiz 2 73.500000

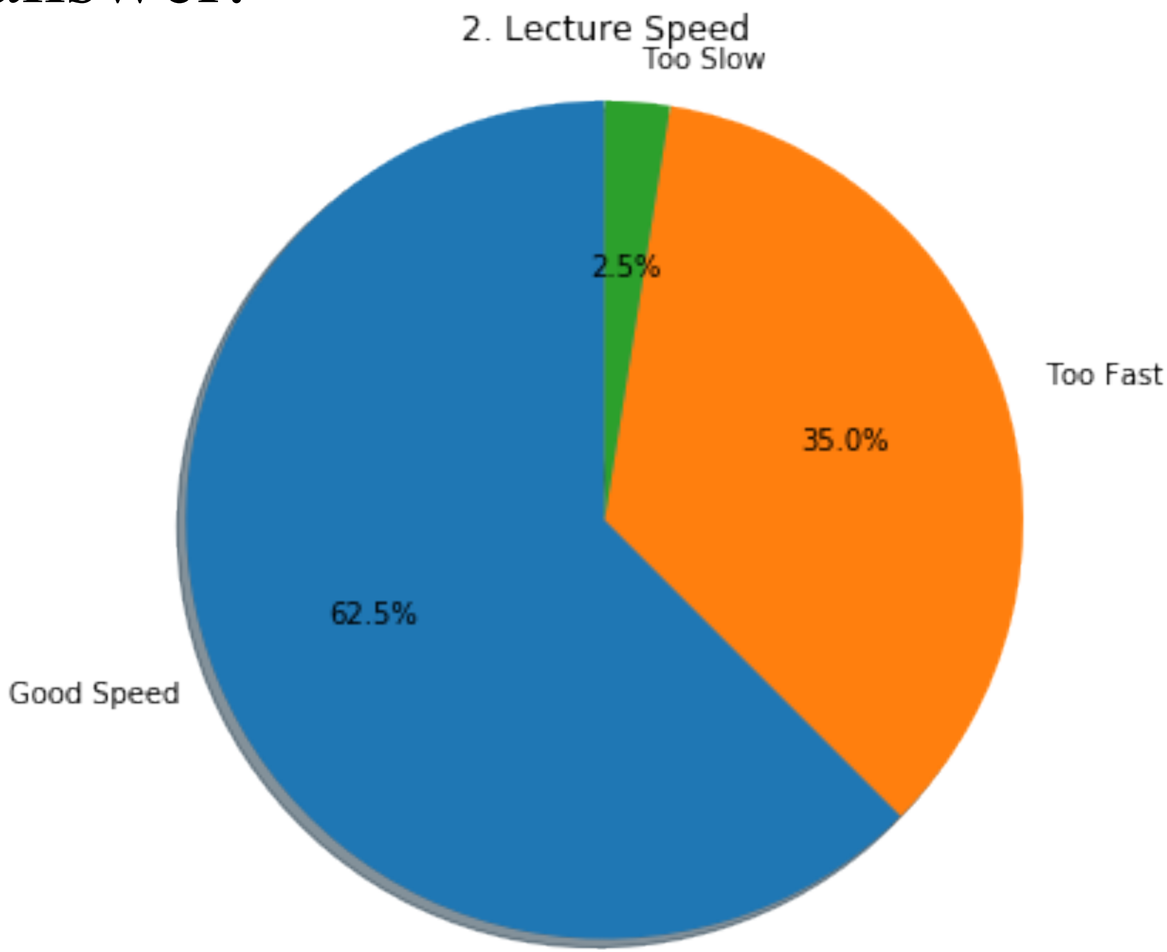
- Either 1 or 2 81.800000
- Neither 1 nor 2 66.958333

Survey results

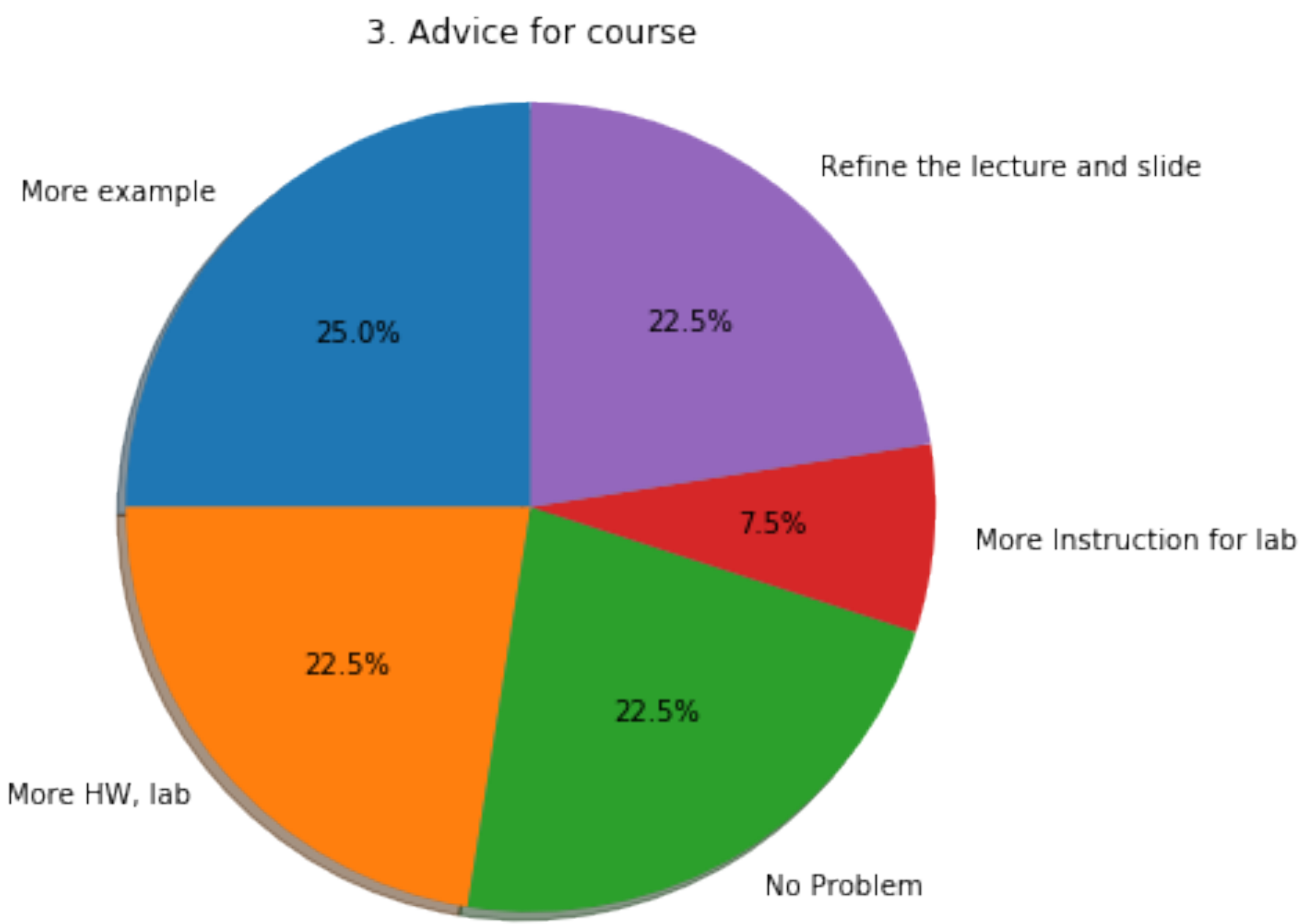
- 1. Has the course met your expectations of an undergraduate networking class so far? If not, please explain the areas where the course has not met your expectations.



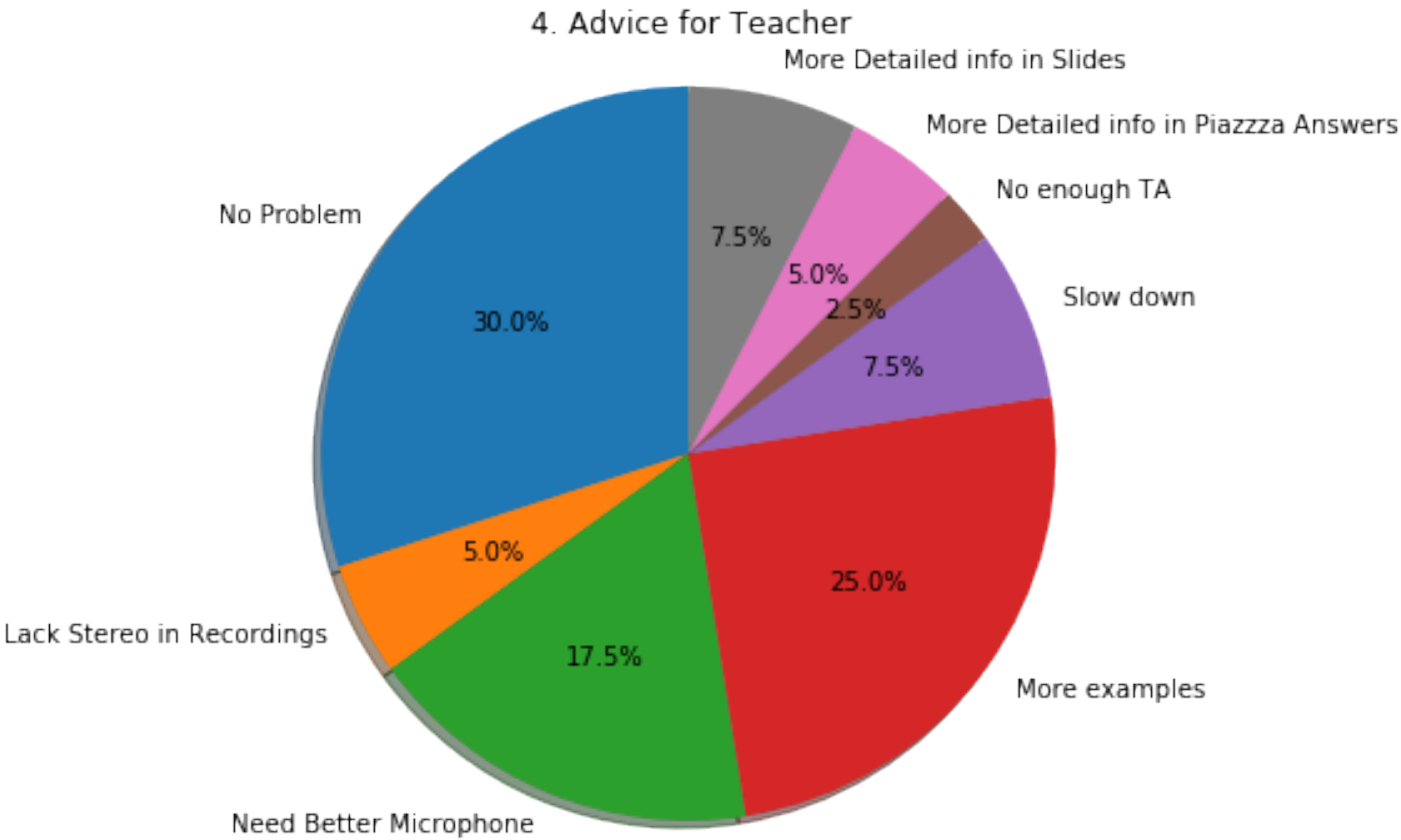
- 2. Are the lectures given a) at an appropriate speed; b) too fast; c) too slow? Please circle one answer.



- 3. Do you have any suggestion on how the course materials might be improved?



- 4. Do you have any suggestion on how the instructor might improve her teaching quality?



- 5. Do you have any suggestion on how the TA might improve his teaching quality?