COMP 3331/9331: Computer Networks and Applications

Week 5

Transport Layer (Continued)

Reading Guide: Chapter 3, Sections: 3.5

Announcements

- Tutorial 1 in Week 5
  - Problem solving prep for exam
- Assignment 1
  - Have you started?
  - Do not delay
  - Be careful about plagiarism
  - Read specification thoroughly
  - Post questions on forum
- Mid-semester Exam in Week 6
  - Monday, 29th August during regular lecture hours
  - Details at end of slide set

Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

  Pipelined protocols

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

Self Study

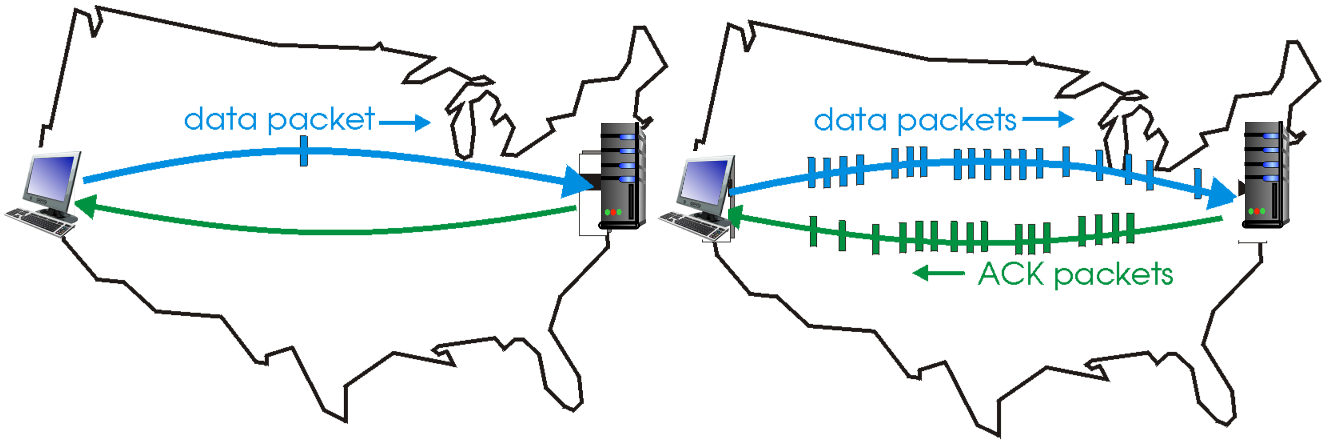
Practice Problem: RDT

http://www-net.cs.umass.edu/kurose_ross/interactive/rdt22.php

Pipelined protocols

pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged pkts

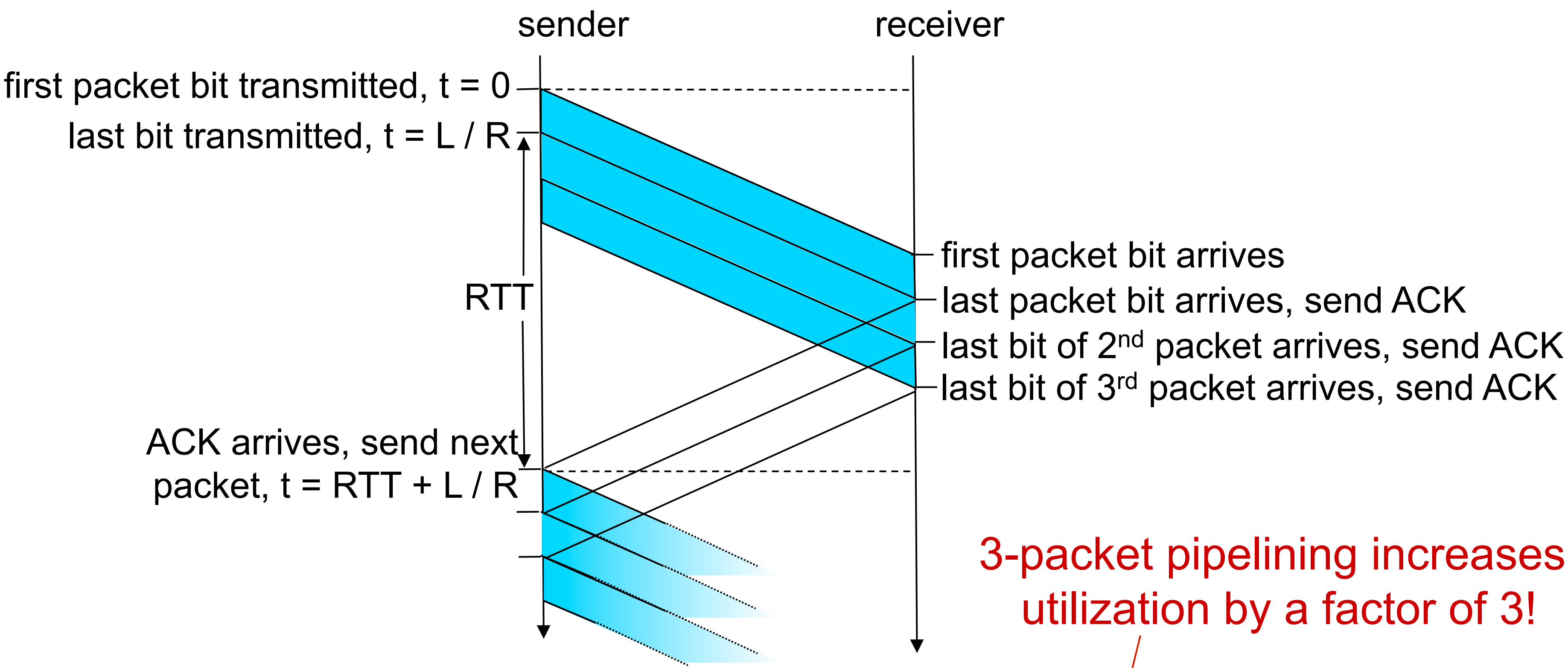
- range of sequence numbers must be increased
- buffering at sender and/or receiver



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

- two generic forms of pipelined (sliding window) protocols: *go-Back-N, selective repeat*

Pipelining: increased utilization



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

$$U_{sender} = \frac{3L / R}{RTT + L / R} = \frac{3 \times 125}{100+125} = 1.67$$

Pipelined protocols: overview

Go-back-N:

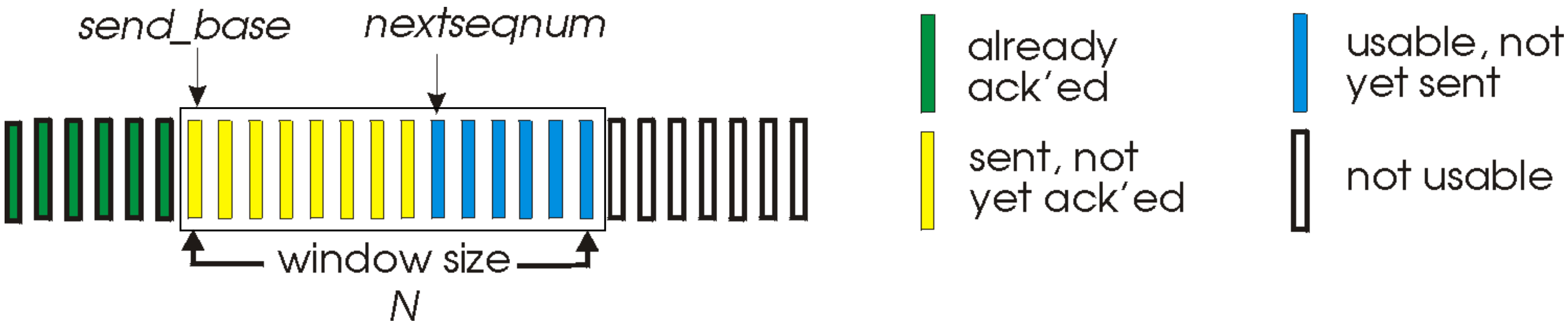
- sender can have up to N unacked packets in pipeline
- receiver only sends *cumulative ack*
  - doesn't ack packet if there's a gap
- sender has timer for oldest unacked packet
  - when timer expires, retransmit *all* unacked packets

Selective Repeat:

- sender can have up to N unack'ed packets in pipeline
- rcvr sends *individual ack* for each packet
- sender maintains timer for each unacked packet
  - when timer expires, retransmit only that unacked packet

Go-Back-N: sender

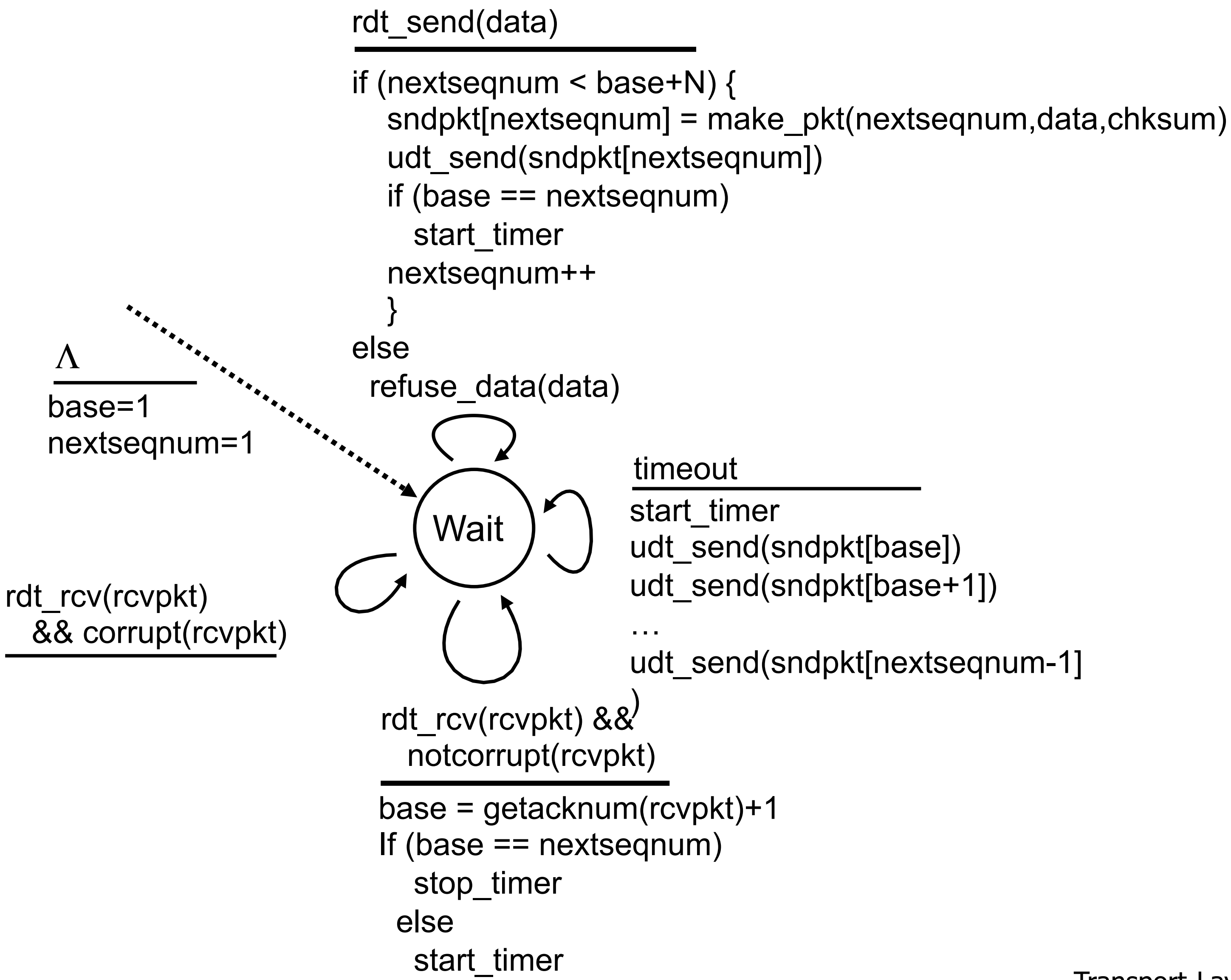
- k-bit seq # in pkt header
- "window" of up to N, consecutive unack'ed pkts allowed



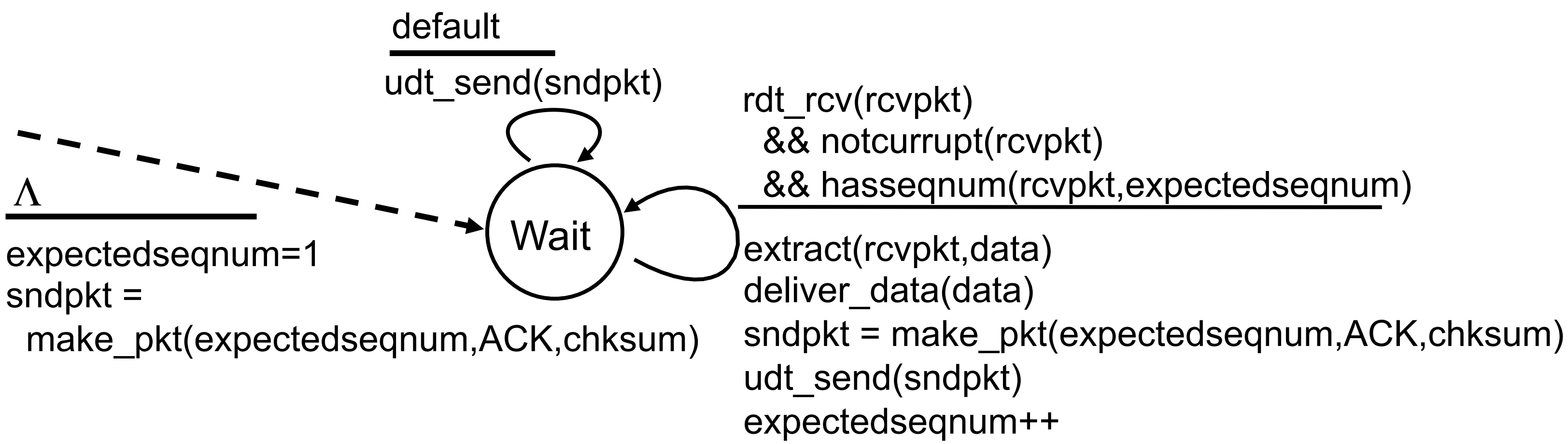
- ACK(n): ACKs all pkts up to, including seq # n - *"cumulative ACK"*
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- *timeout(n):* retransmit packet n and all higher seq # pkts in window

Applet: http://media.pearsoncmg.com/aw/aw_kurose_network_2/applets/go-back-n/go-back-n.html

GBN: sender extended FSM



GBN: receiver extended FSM

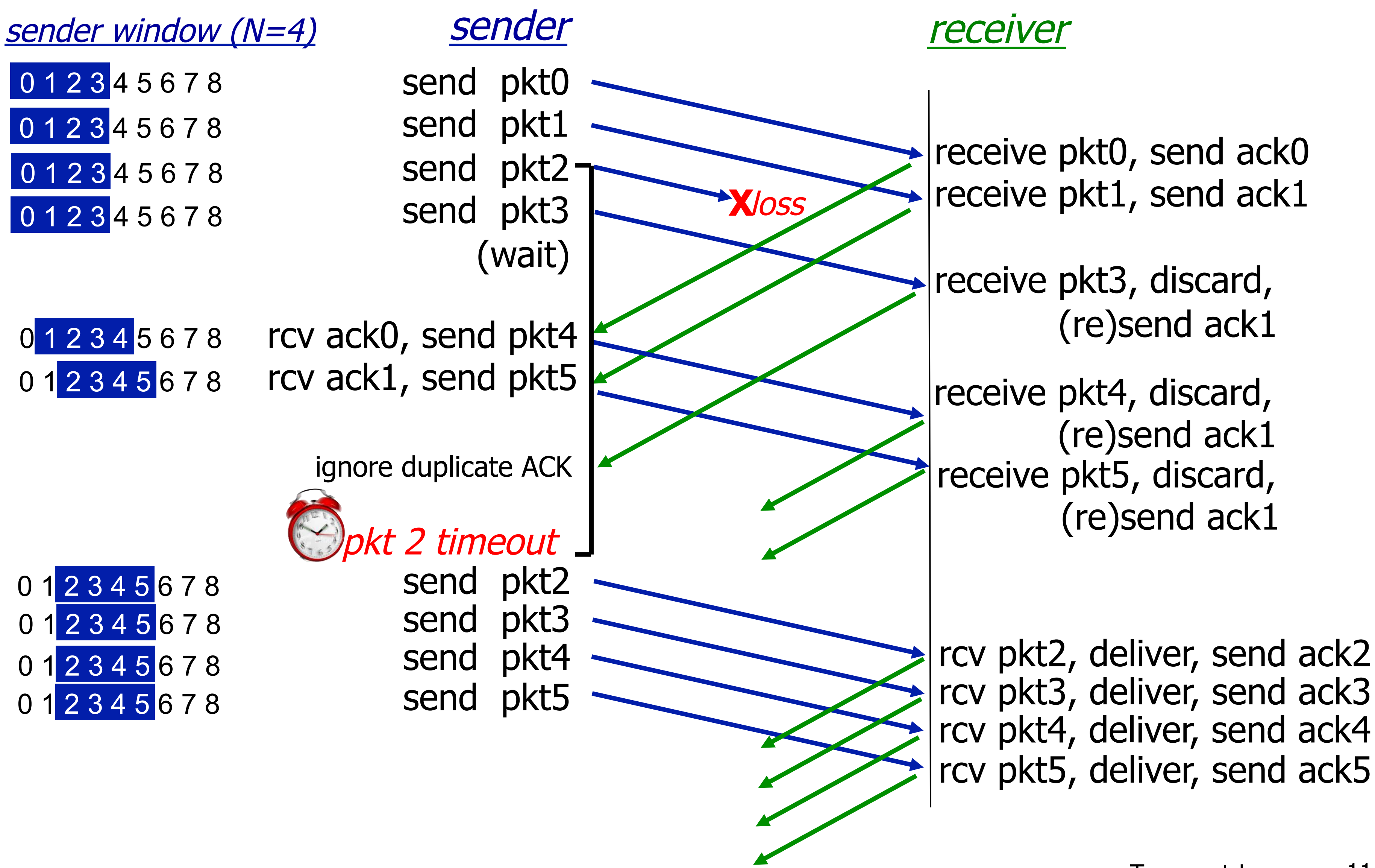


ACK-only: always send ACK for correctly-received pkt with highest *in-order* seq #

- may generate duplicate ACKs
- need only remember **`expectedseqnum`**

- out-of-order pkt:
  - discard (don't buffer): *no receiver buffering!*
  - re-ACK pkt with highest in-order seq #

GBN in action

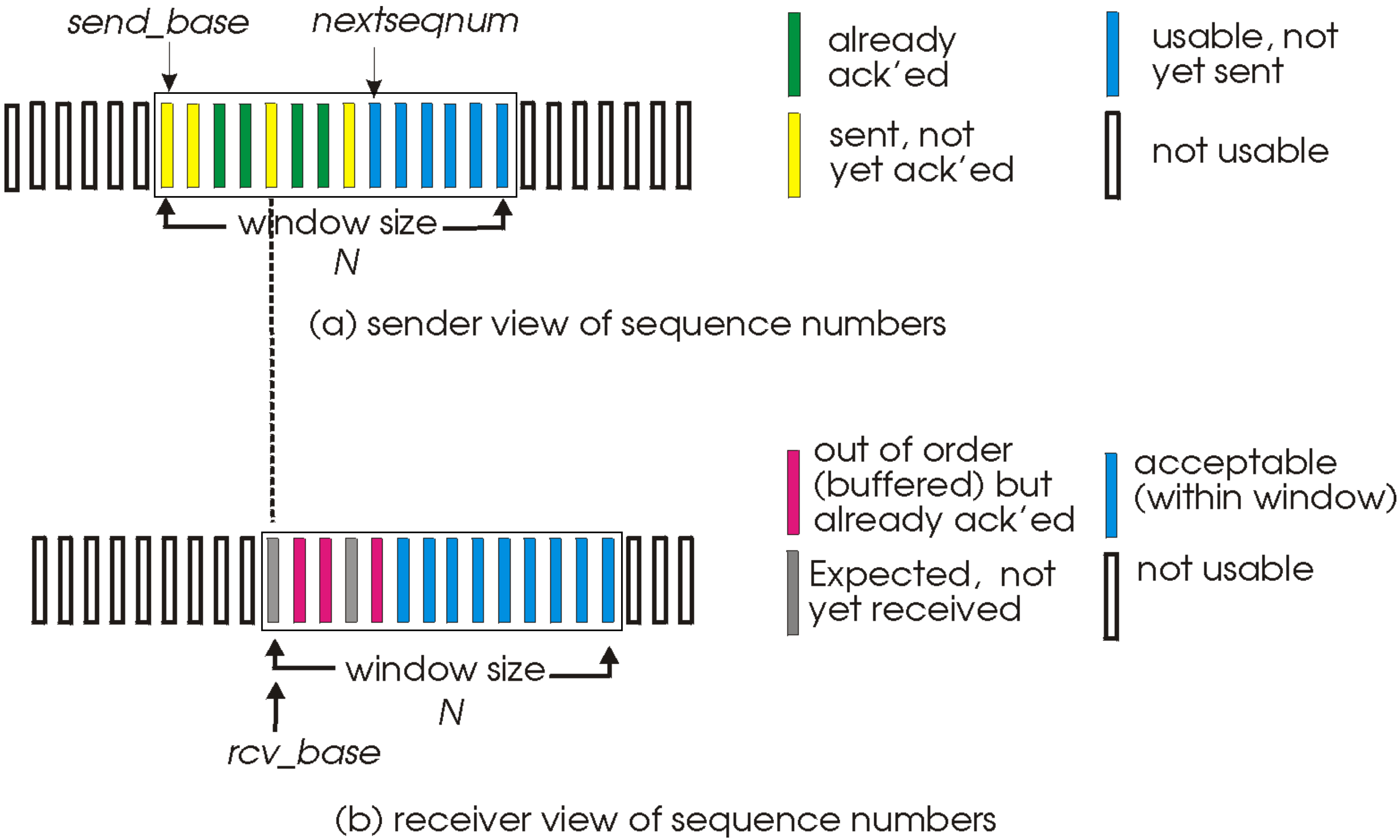


Selective repeat

- receiver *individually* acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt
- sender window
  - *N* consecutive seq #'s
  - limits seq #s of sent, unACKed pkts

Applet: http://media.pearsoncmg.com/aw/aw_kurose_network_3/applets/SelectRepeat/SR.html

Selective repeat: sender, receiver windows



Selective repeat

sender

data from above:

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n) in [sendbase,sendbase+N]:

- mark pkt n as received
- if n smallest unACKed pkt, advance window base to next unACKed seq #

receiver

pkt n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

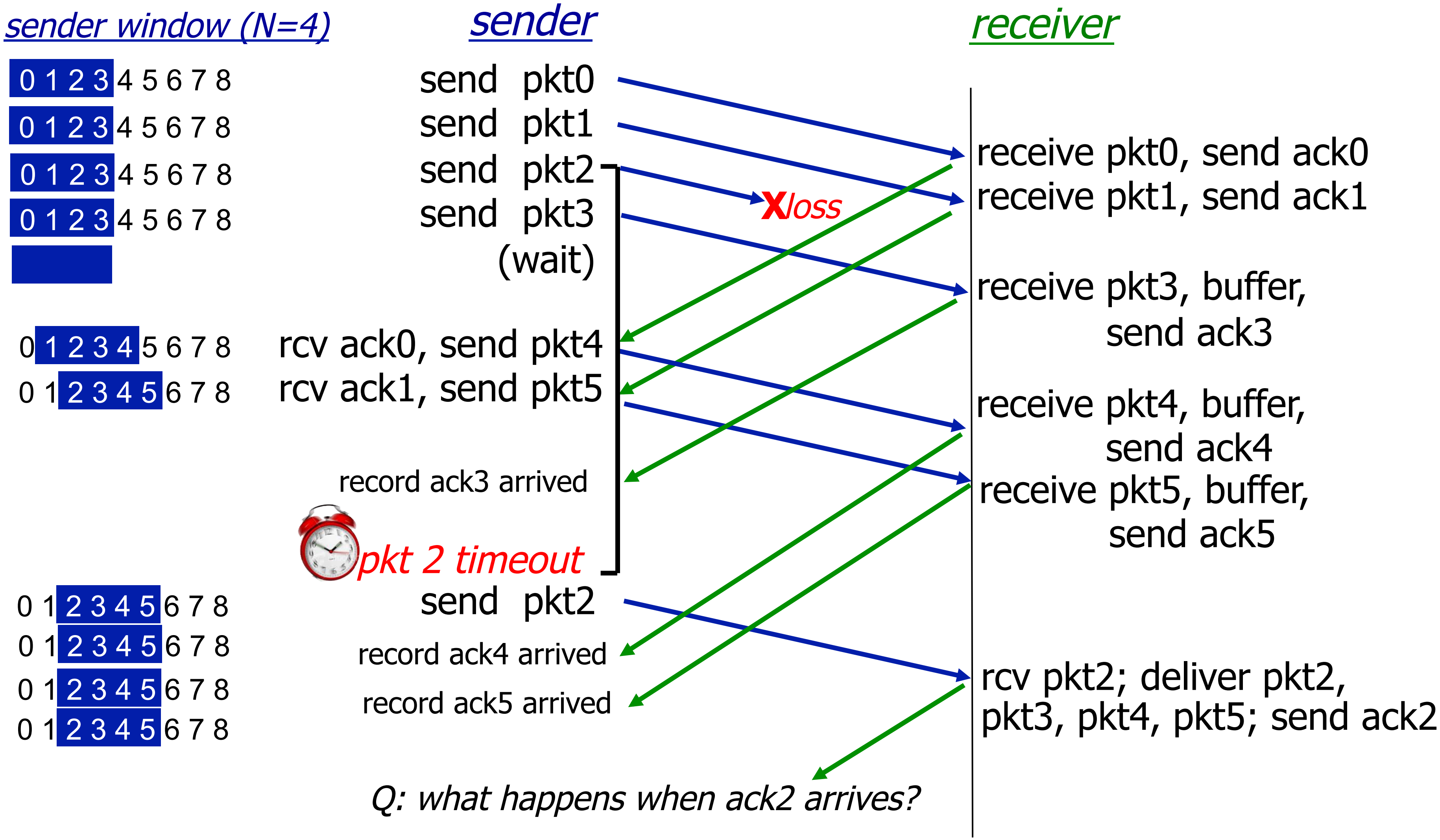
pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

Selective repeat in action



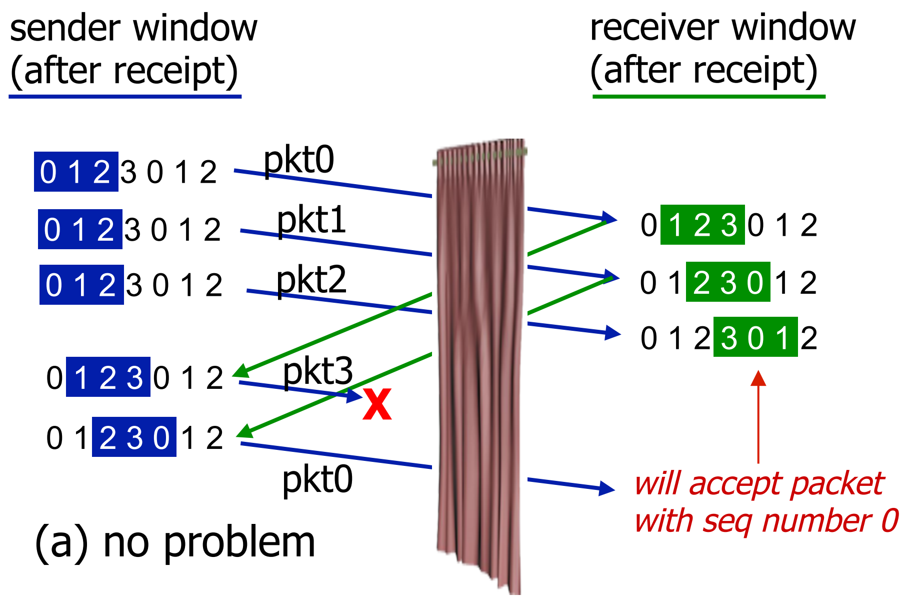
Selective repeat: dilemma

example:

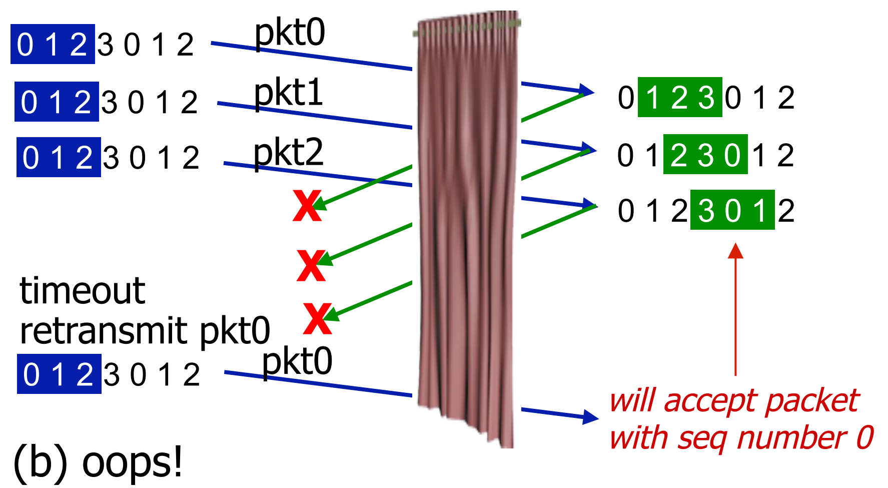
- seq #'s: 0, 1, 2, 3
- window size=3
- receiver sees no difference in two scenarios!
- duplicate data accepted as new in (b)

Q: what relationship between seq # size and window size to avoid problem in (b)?

A: window size must be less than or equal to half the size of the sequence number space



receiver can't see sender side.
receiver behavior identical in both cases!
something's (very) wrong!



Observations

- With sliding windows, it is possible to fully utilize a link (or path), provided the window size is large enough. Throughput is ~ (n/RTT)
  - Stop & Wait is like n = 1.
- Sender has to buffer all unacknowledged packets, because they may require retransmission
- Receiver may be able to accept out-of-order packets, but only up to its buffer limits
- Implementation complexity depends on protocol details (GBN vs. SR)

Recap: components of a solution

- Checksums (for error detection)
- Timers (for loss detection)
- Acknowledgments
  - cumulative
  - selective
- Sequence numbers (duplicates, windows)
- Sliding Windows (for efficiency)

- Reliability protocols use the above to decide when and what to retransmit or acknowledge

Quiz: GBN vs. SR

- Which of the following is *not* true?

A. GBN uses cumulative ACKs, SR uses individual ACKs
B. Both GBN and SR use timeouts to address packet loss
C. GBN maintains a separate timer for each outstanding packet
D. SR maintains a separate timer for each outstanding packet
E. Neither GBN nor SR use NACKs

Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

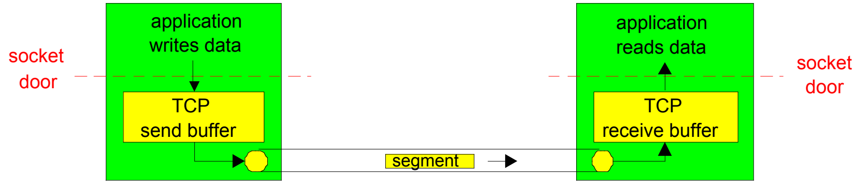
3.7 TCP congestion control

Practical Reliability Questions

- How do the sender and receiver keep track of outstanding pipelined segments?
- How many segments should be pipelined?
- How do we choose sequence numbers?
- What does connection establishment and teardown look like?
- How should we choose timeout values?

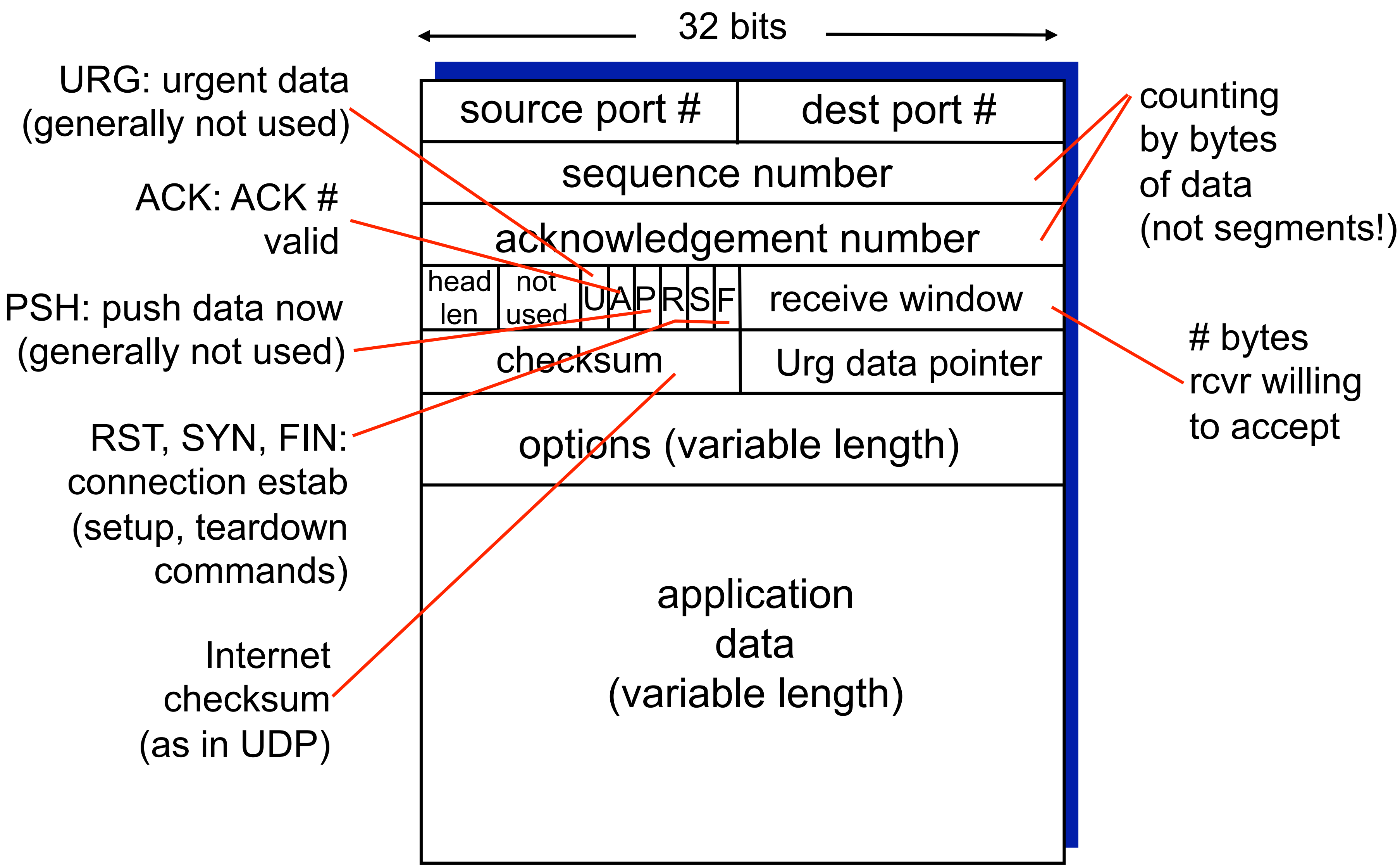
TCP: Overview RFCs: 793,1122,1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- send and receive buffers



- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

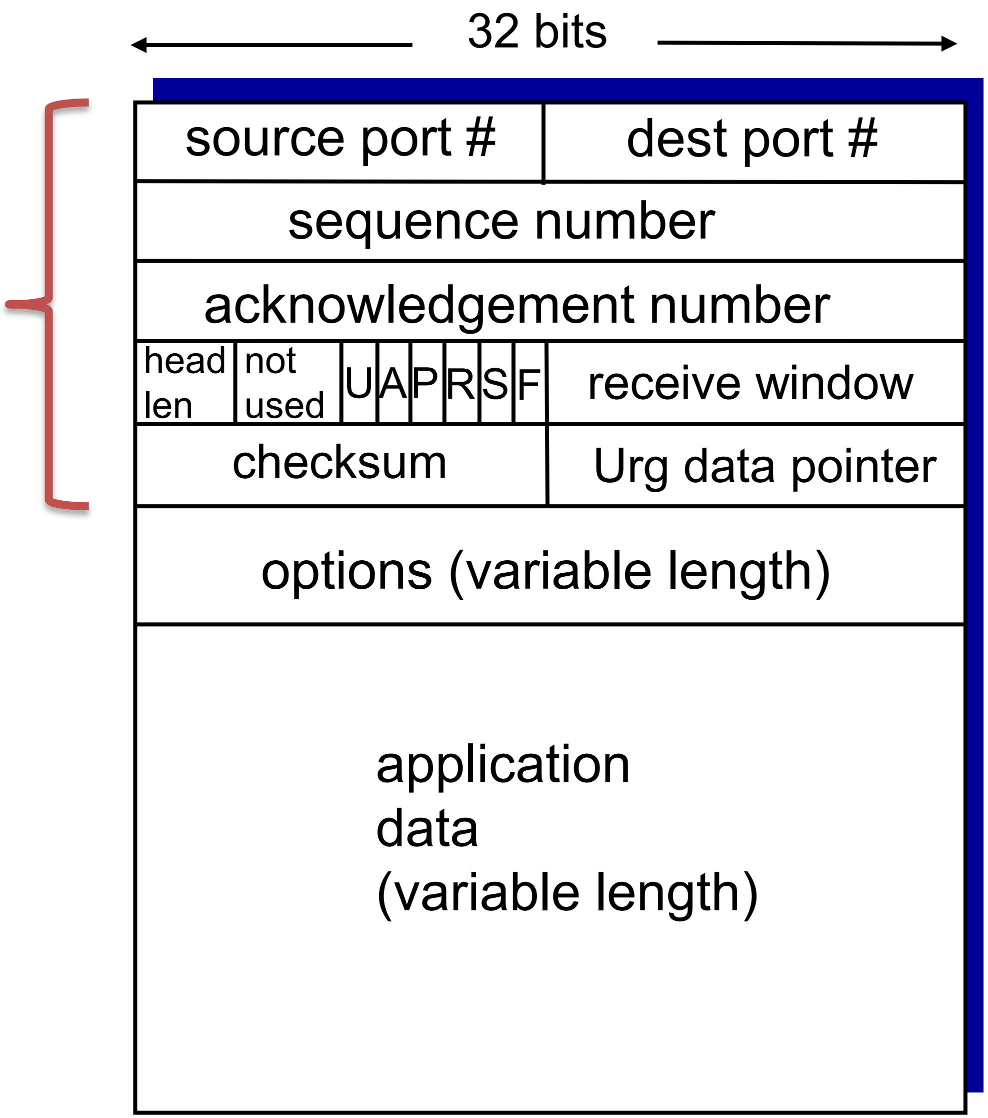
TCP segment structure



TCP segment structure

20 Bytes

(UDP was 8)



Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

Recall: Components of a solution for reliable transport

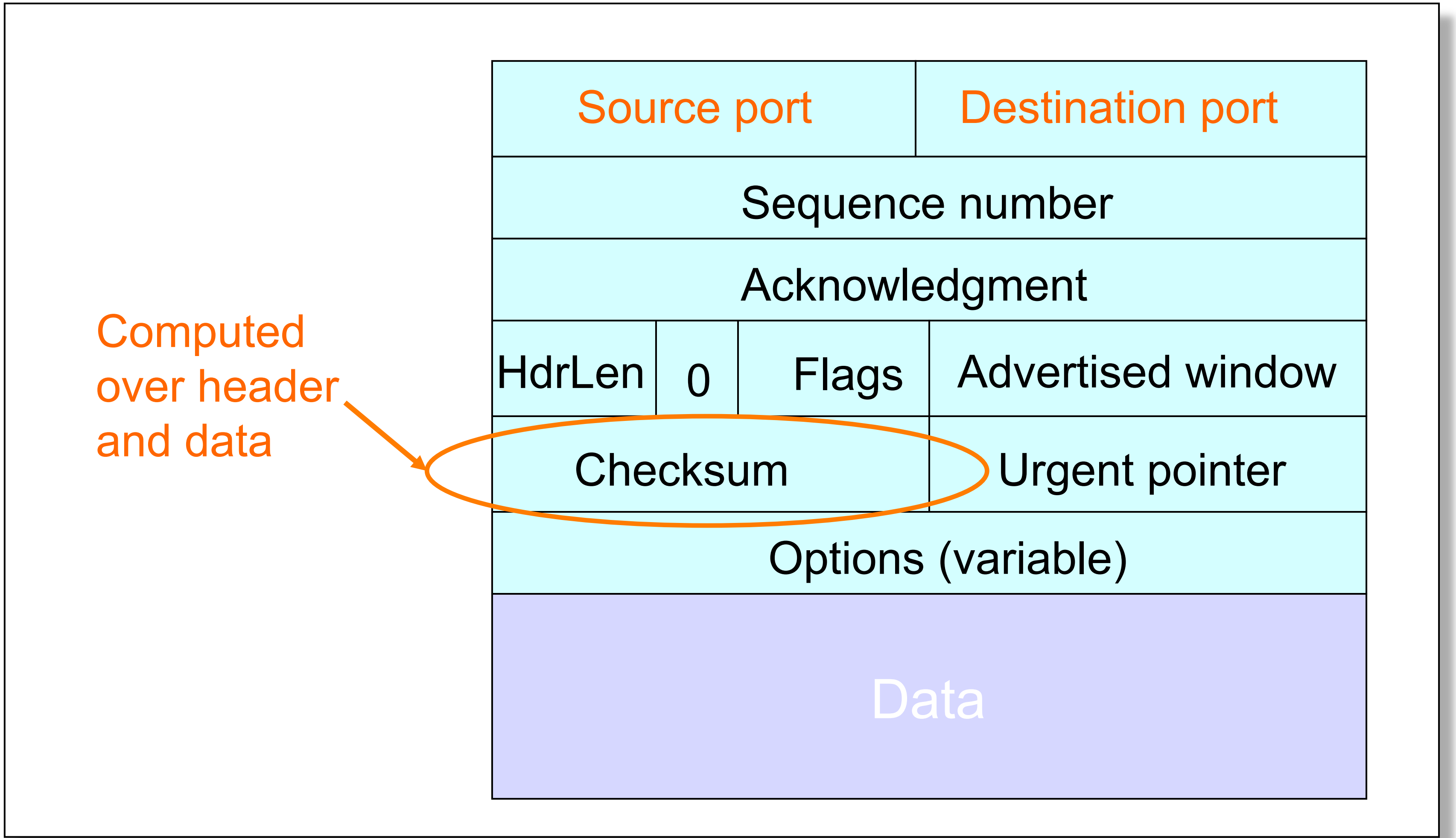
- Checksums (for error detection)
- Timers (for loss detection)
- Acknowledgments
  - cumulative
  - selective
- Sequence numbers (duplicates, windows)
- Sliding Windows (for efficiency)
  - Go-Back-N (GBN)
  - Selective Replay (SR)

What does TCP do?

Many of our previous ideas, but some key differences

- Checksum

TCP Header


Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
Flags
Advertised window
Checksum
Urgent pointer
Options (variable)
Data
Computed over header and data

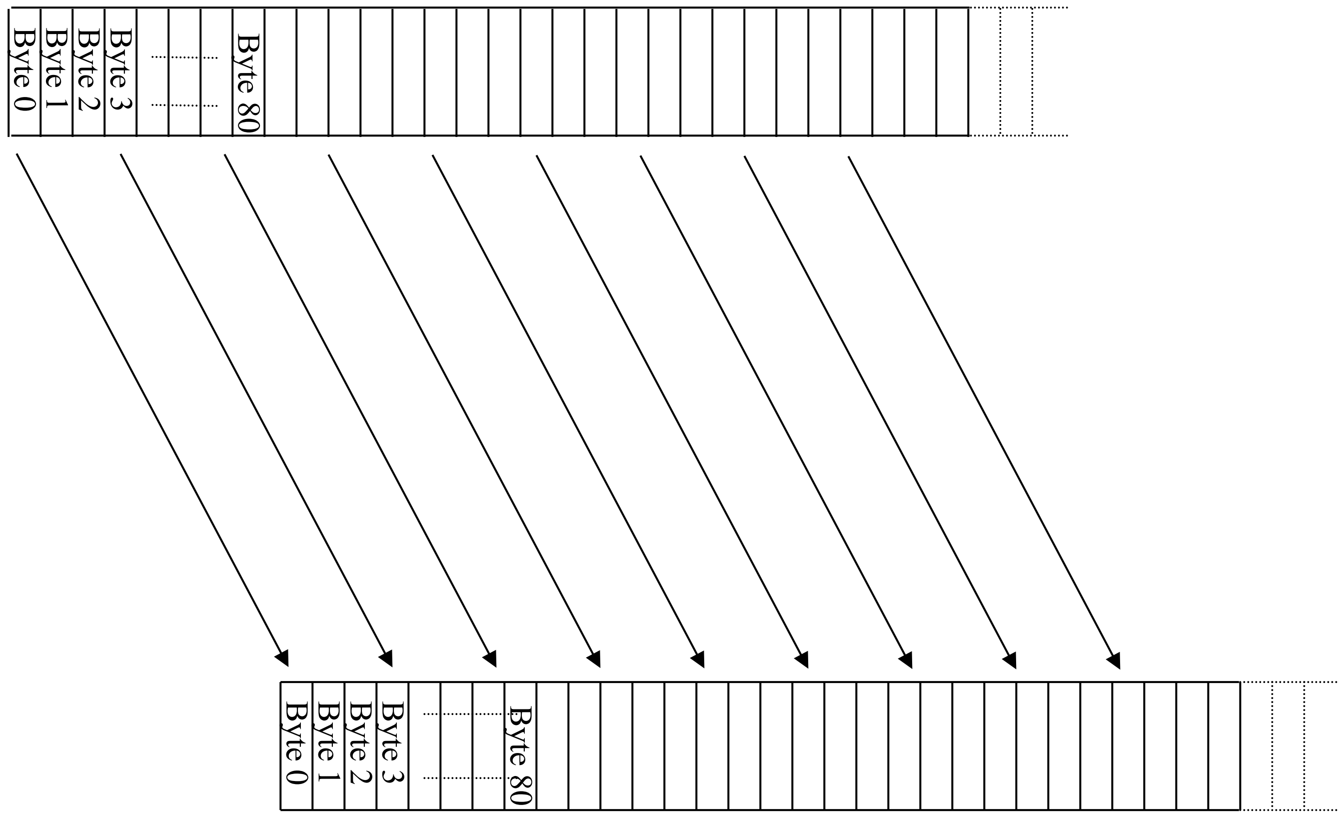
What does TCP do?

Many of our previous ideas, but some key differences

- Checksum
- **Sequence numbers are byte offsets**

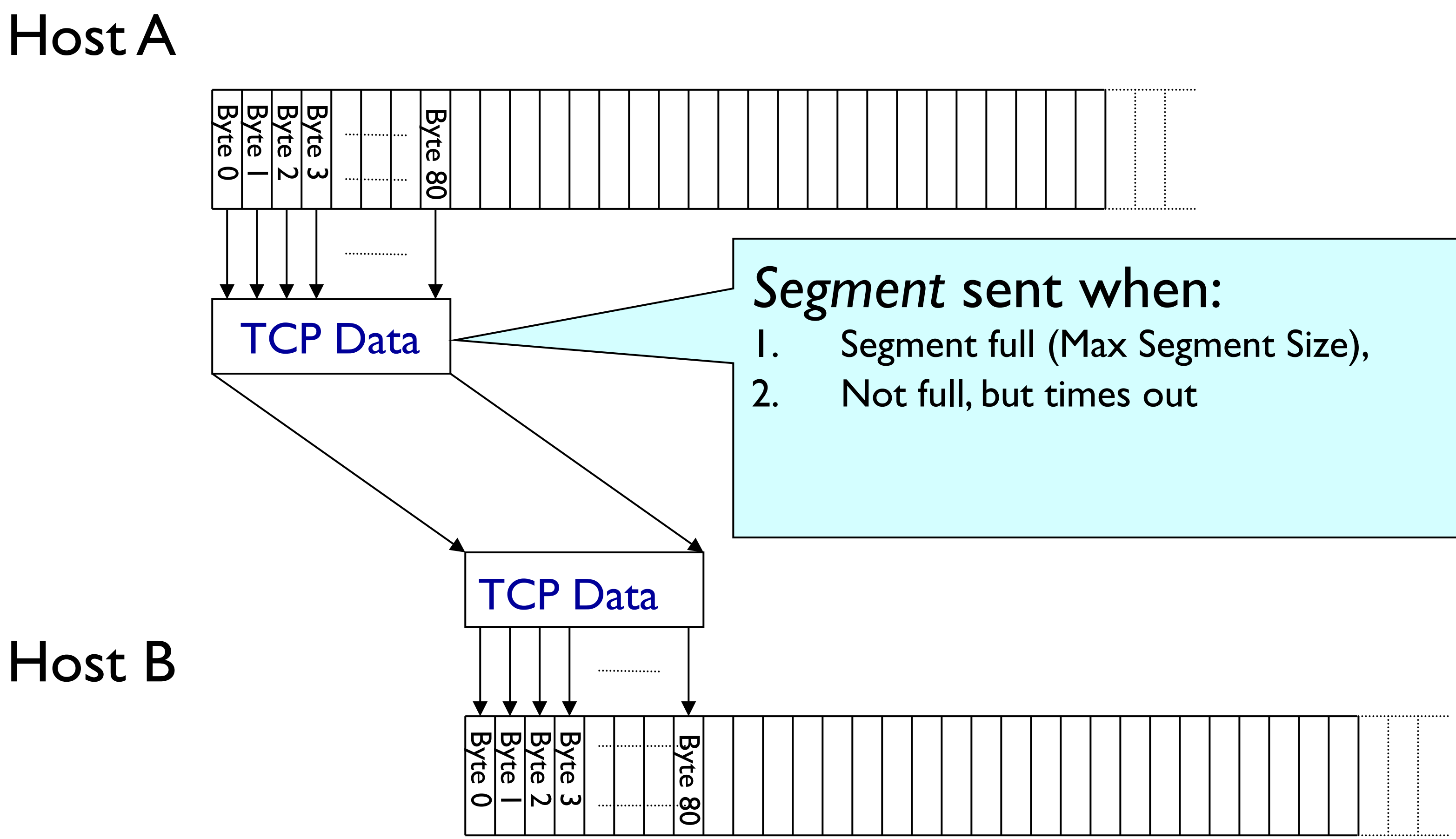
TCP "Stream of Bytes" Service ..

Application @ Host A



Application @ Host B

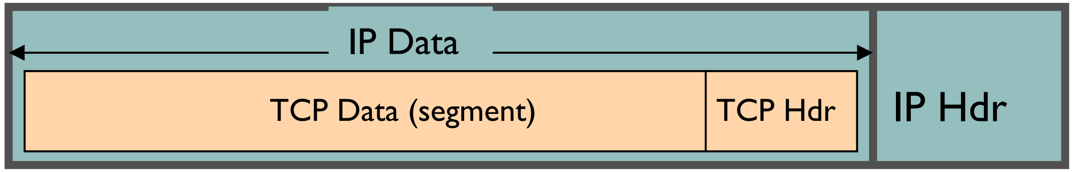
.. Provided Using TCP "Segments"



Segment sent when:

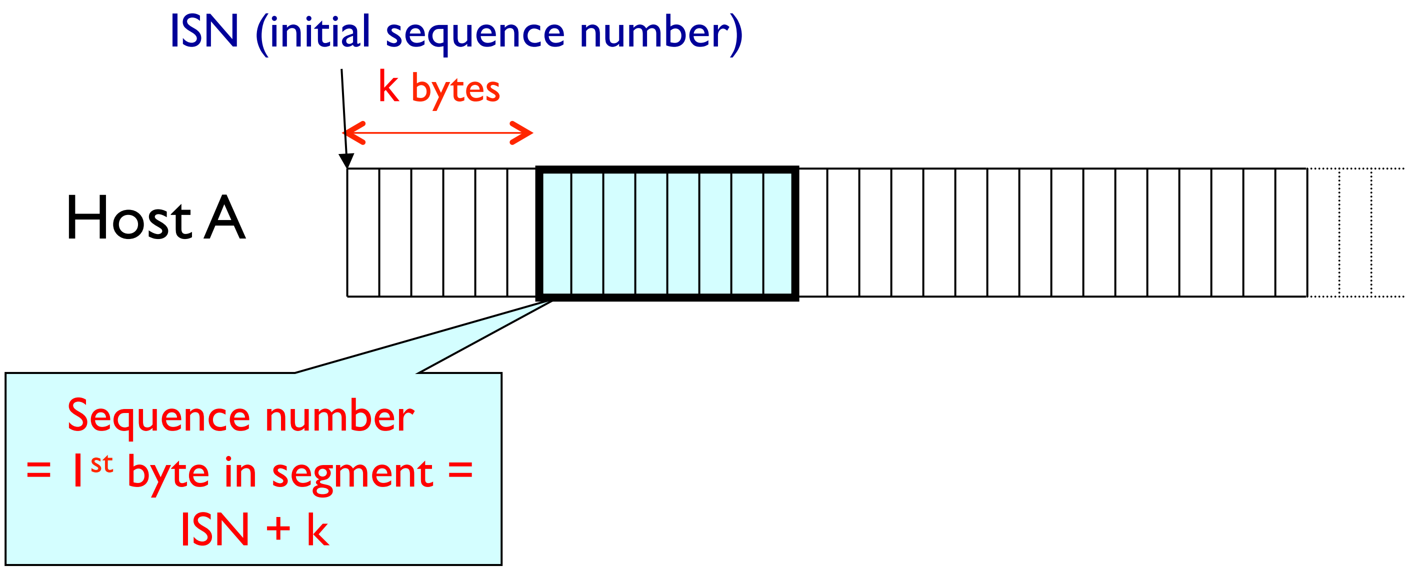
1. Segment full (Max Segment Size),
2. Not full, but times out

TCP Segment

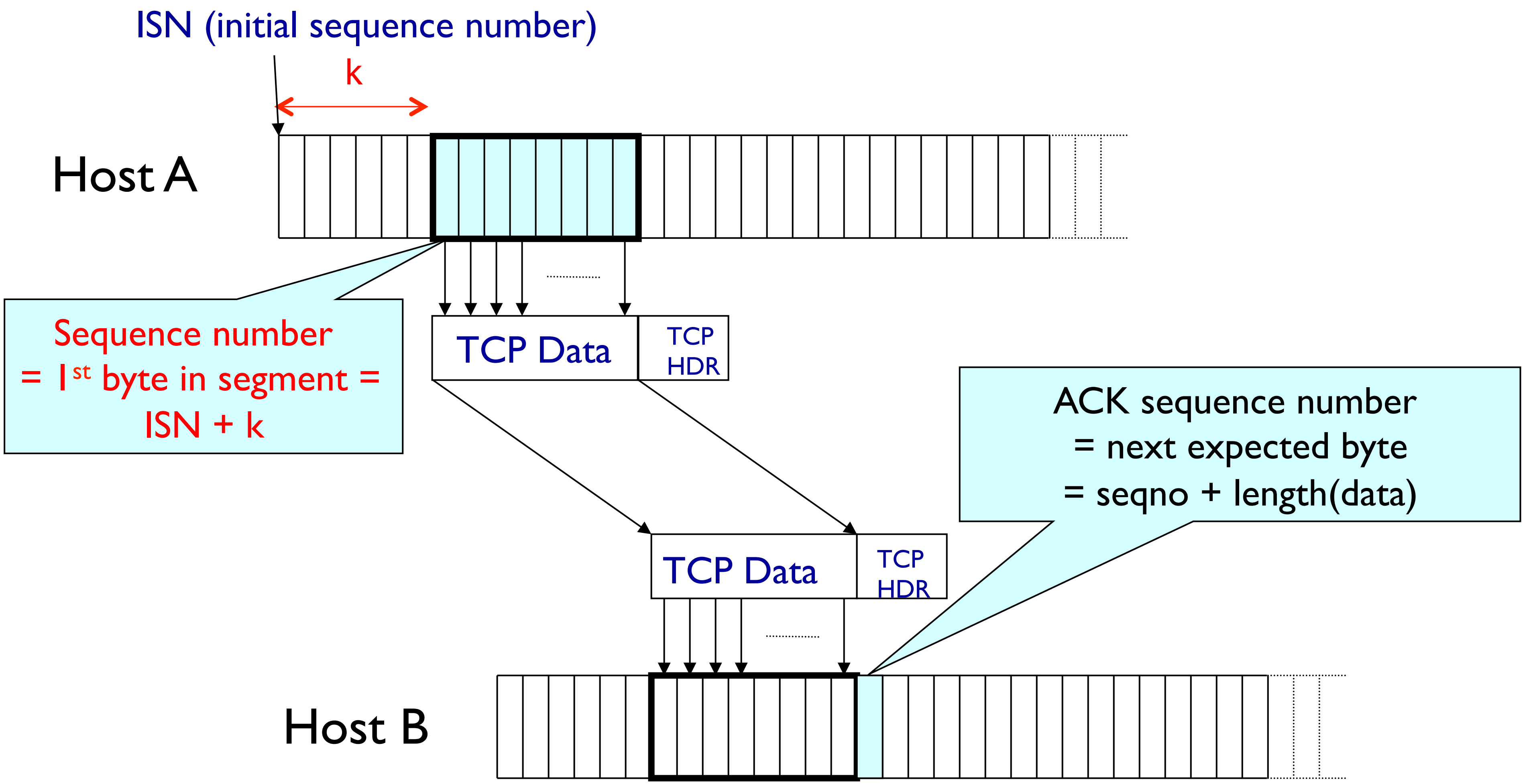


- IP packet
  - No bigger than Maximum Transmission Unit (MTU)
  - E.g., up to 1500 bytes with Ethernet
- TCP packet
  - IP packet with a TCP header and data inside
  - TCP header ≥ 20 bytes long
- TCP **segment**
  - No more than Maximum Segment Size (MSS) bytes
  - E.g., up to 1460 consecutive bytes from the stream
  - MSS = MTU – (IP header) – (TCP header)

Sequence Numbers



Sequence Numbers



What does TCP do?

Most of our previous tricks, but a few differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)

ACKing and Sequence Numbers

- Sender sends packet
  - Data starts with sequence number X
  - Packet contains B bytes [X, X+1, X+2, ….X+B-1]
- Upon receipt of packet, receiver sends an ACK
  - If all data prior to X already received:
    - ACK acknowledges X+B (because that is next expected byte)
  - If highest in-order byte received is Y s.t. (Y+1) < X
    - ACK acknowledges Y+1
    - Even if this has been ACKed before

Normal Pattern

- Sender: seqno=X, length=B
- Receiver: ACK=X+B
- Sender: seqno=X+B, length=B
- Receiver: ACK=X+2B
- Sender: seqno=X+2B, length=B

- Seqno of next packet is same as last ACK field

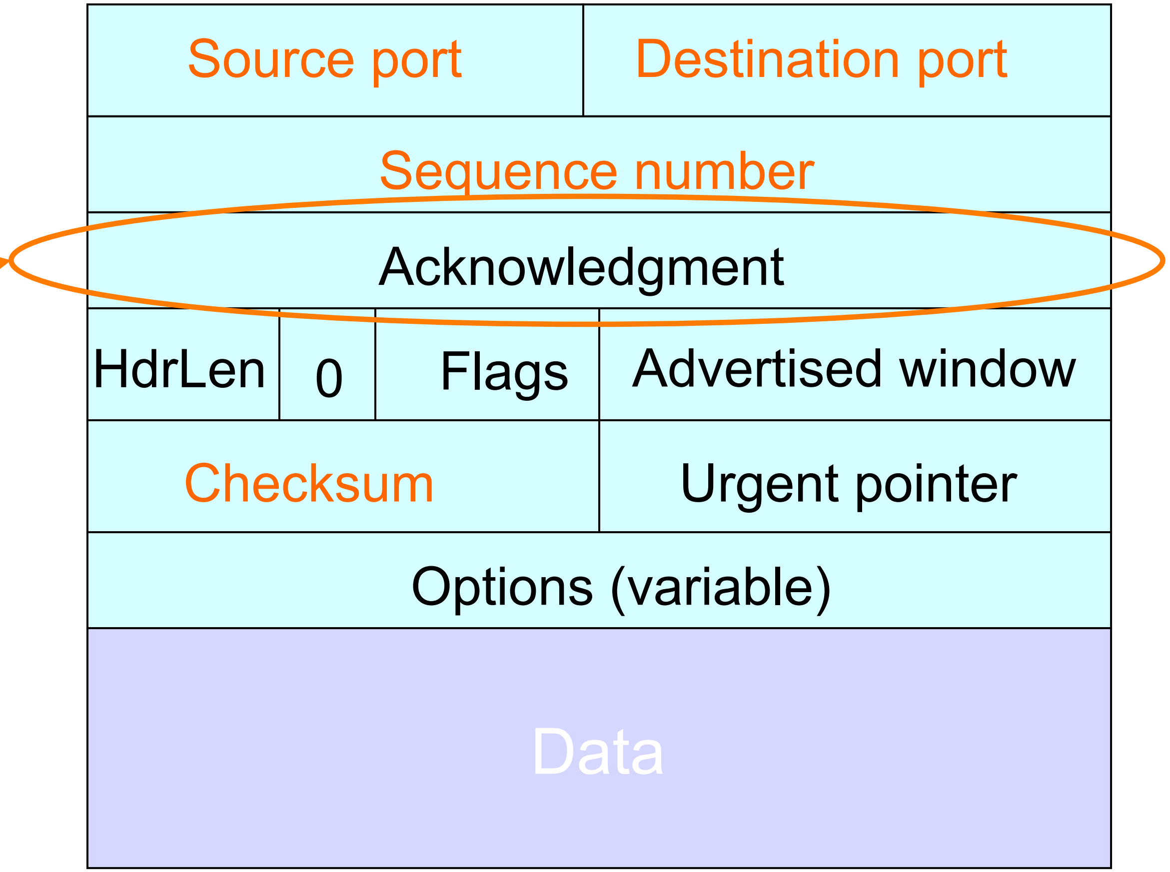
Packet Loss

- Sender: seqno=X, length=B
- Receiver: ACK=X+B
- Sender: ~~seqno=X+B, length=B~~ LOST

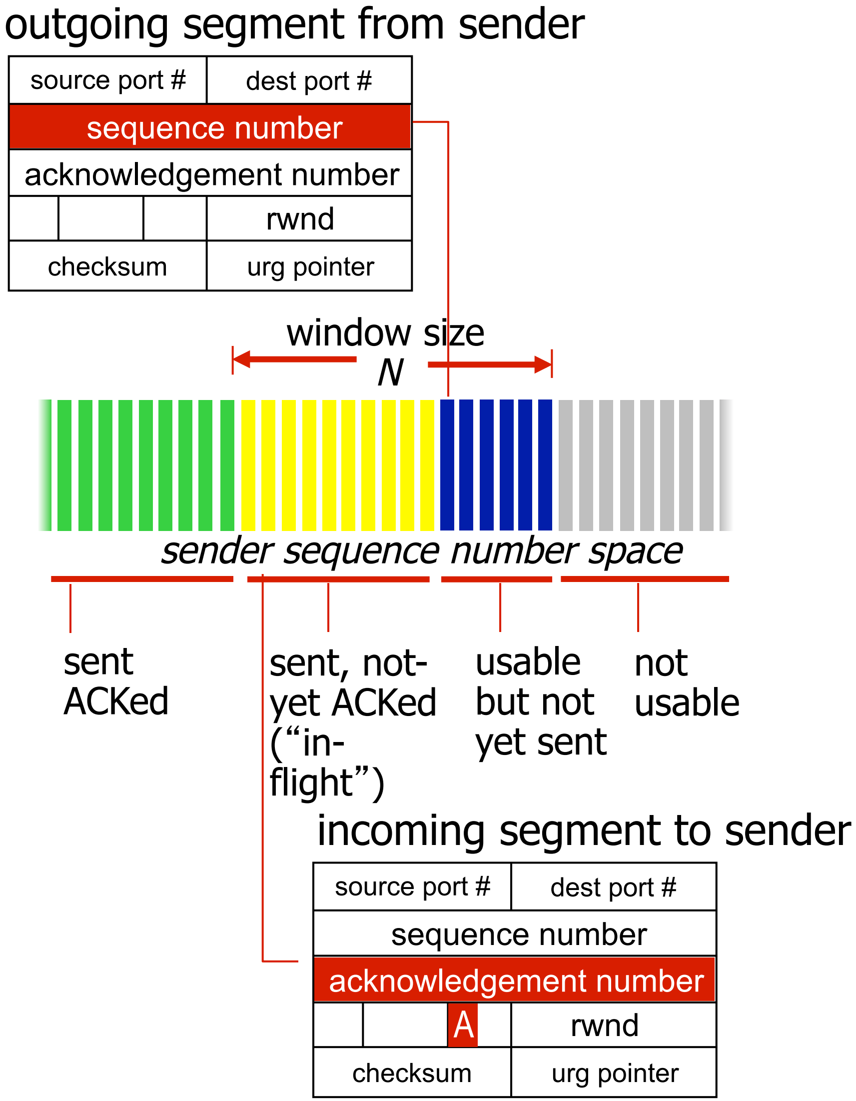
- Sender: seqno=X+2B, length=B
- Receiver: ACK = X+B

TCP Header

Acknowledgment gives seqno just beyond highest seqno received **in order**
("What Byte is Next")



TCP seq. numbers, ACKs



sequence numbers:

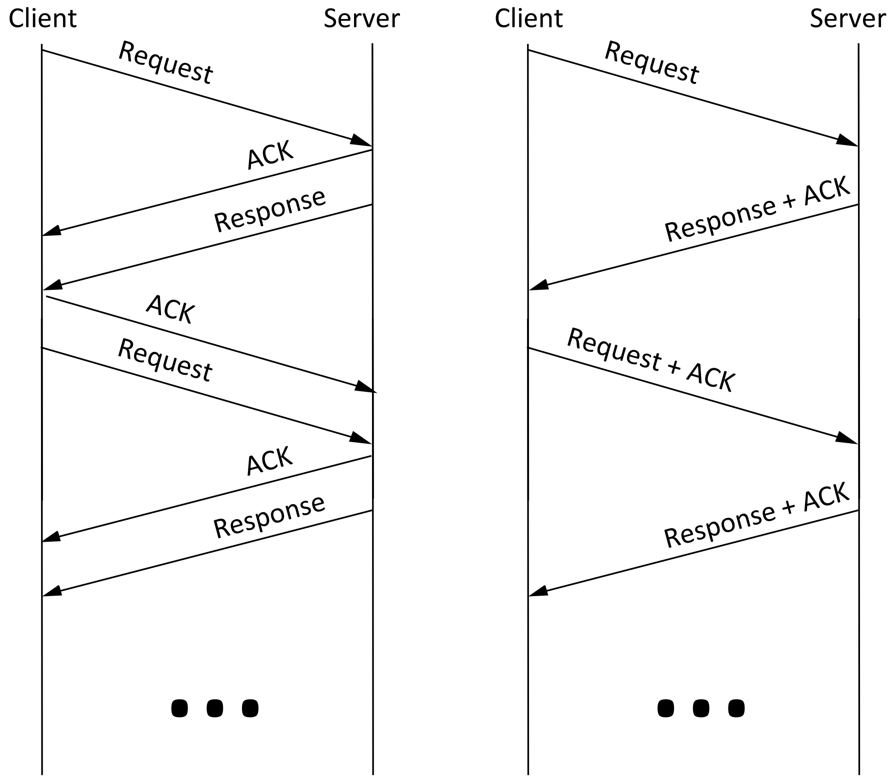
- byte stream "number" of first byte in segment's data

acknowledgements:

- seq # of next byte expected from other side
- cumulative ACK

Piggybacking

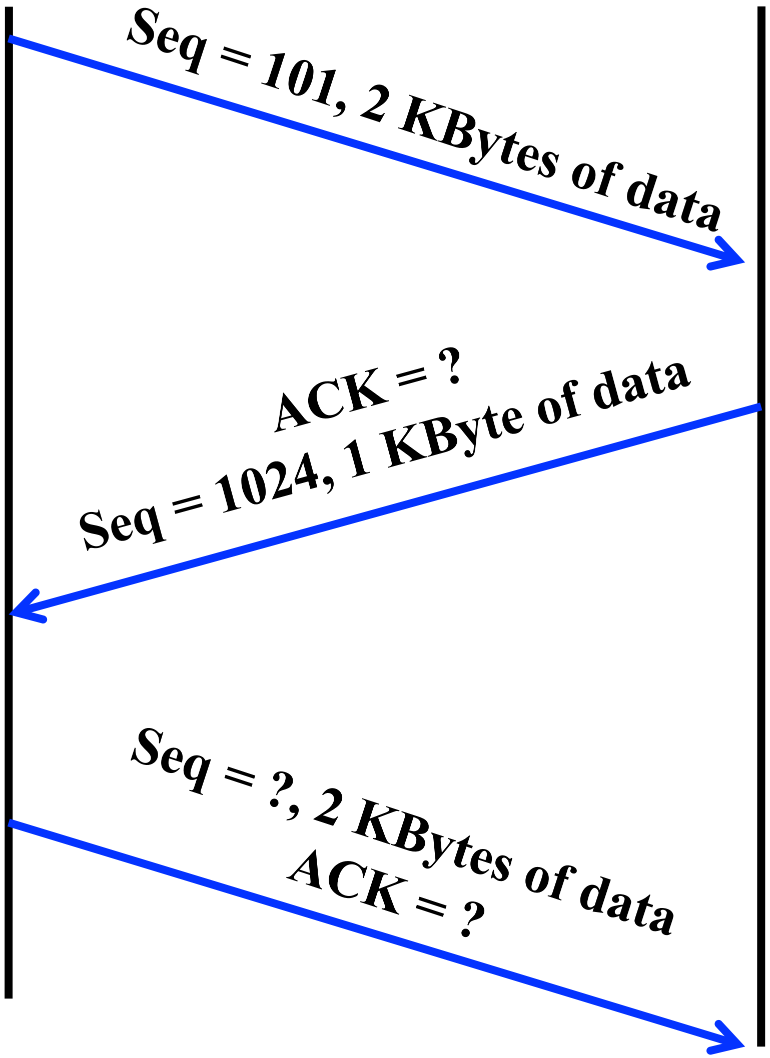
- So far, we've assumed distinct "sender" and "receiver" roles
- In reality, usually both sides of a connection send some data



Without Piggybacking

With Piggybacking

Quiz


Seq = 101, 2 KBytes of data
ACK = ?
Seq = 1024, 1 KByte of data
Seq = ?, 2 KBytes of data
ACK = ?

What does TCP do?

Most of our previous tricks, but a few differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)
- Receivers can buffer out-of-sequence packets (like SR)

Loss with cumulative ACKs

- Sender sends packets with 100B and seqnos.:
  - 100, 200, 300, 400, 500, 600, 700, 800, 900, …
- Assume the fifth packet (seqno 500) is lost, but no others
- Stream of ACKs will be:
  - 200, 300, 400, 500, 500, 500, 500,…

What does TCP do?

Most of our previous tricks, but a few differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)
- Receivers do not drop out-of-sequence packets (like SR)
- Sender maintains a single retransmission timer (like GBN) and retransmits on timeout

TCP round trip time, timeout

Q: 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 and connection has lower throughput

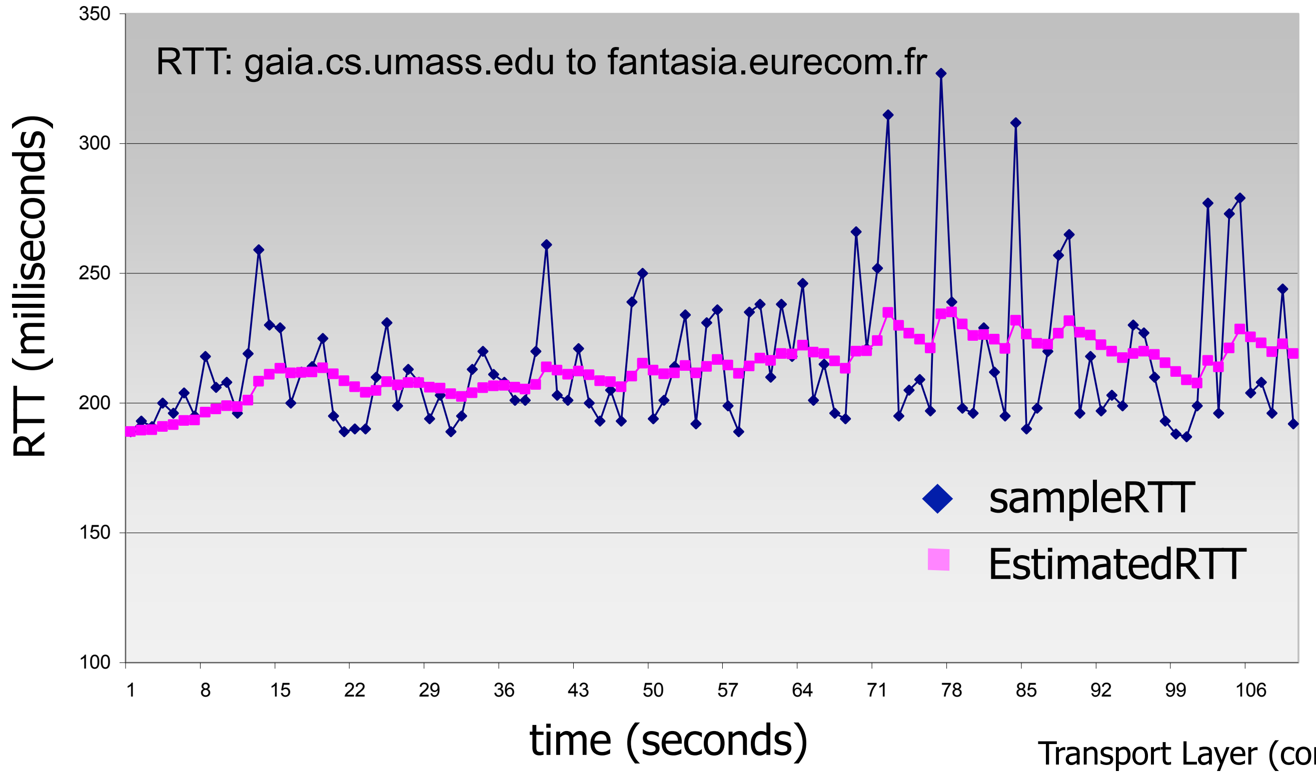
Q: how to estimate RTT?

- **`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, timeout

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

- exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: α = 0.125



TCP round trip time, 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-EstimatedRTT|
              (typically, β = 0.25)
```

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

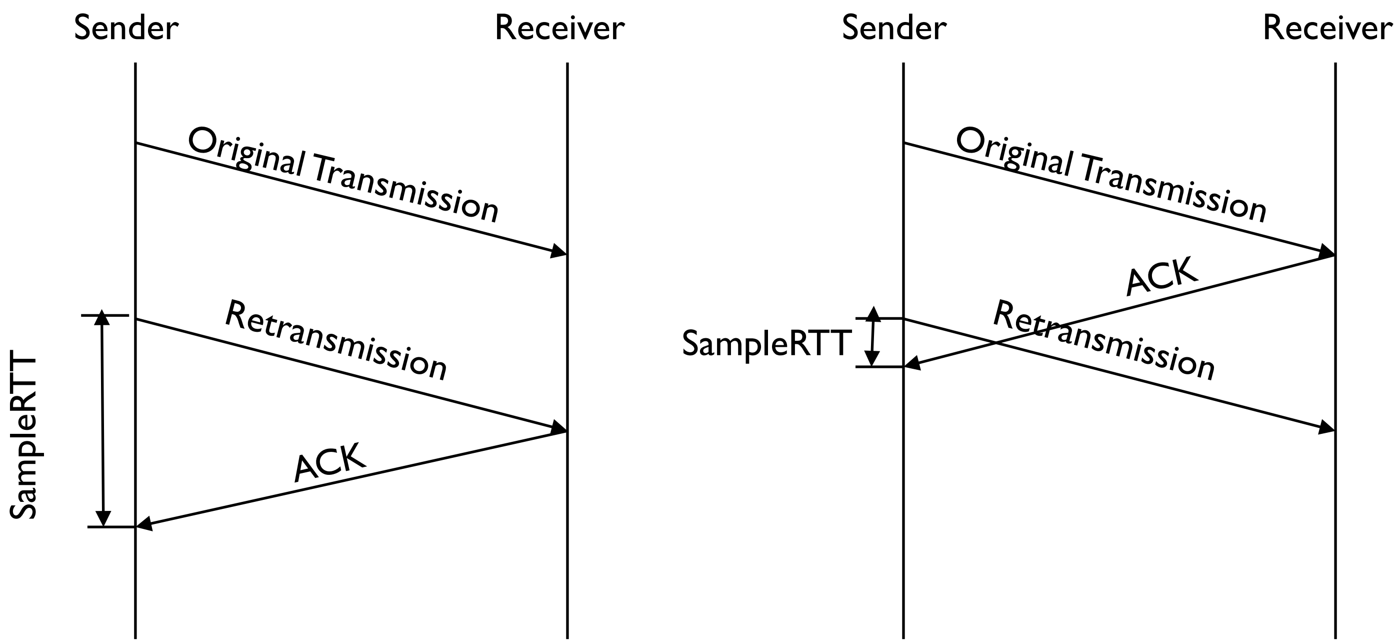
estimated RTT

"safety margin"

Practice Problem:
http://wps.pearsoned.com/ecs_kurose_compnetw_6/216/55463/14198700.cw/index.html

Why exclude retransmissions in RTT computation?

- How do we differentiate between the real ACK, and ACK of the retransmitted packet?



TCP sender events:

PUTTING IT TOGETHER

data rcvd from app:

- 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:

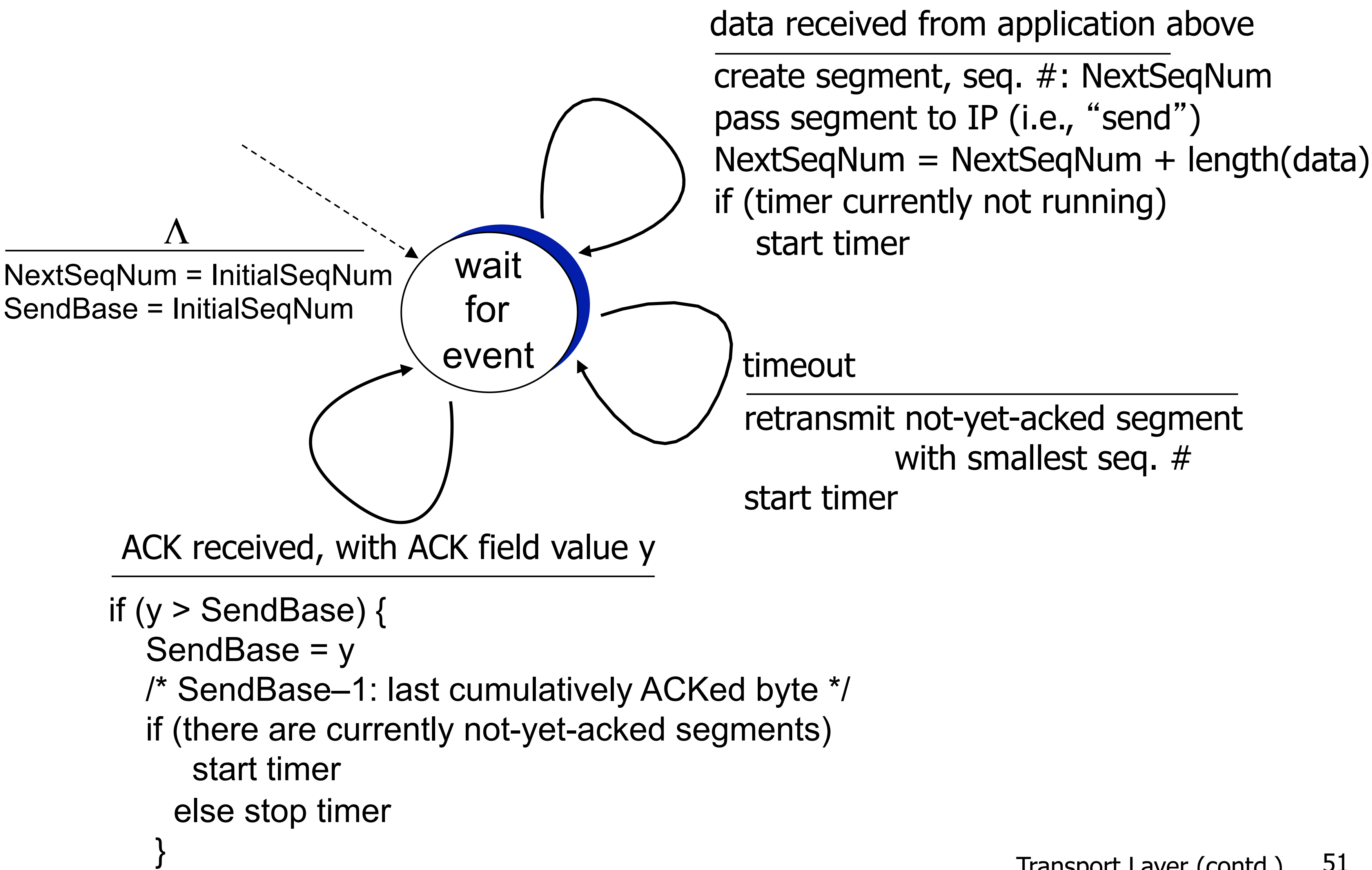
- retransmit segment that caused timeout
- restart timer

ack rcvd:

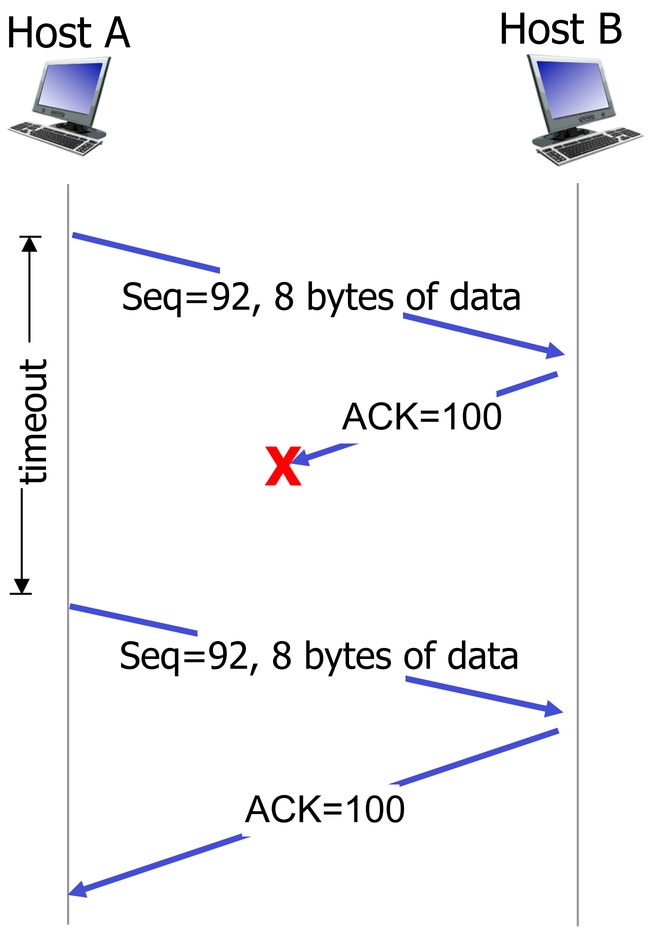
- if ack acknowledges previously unacked segments
  - update what is known to be ACKed
  - start timer if there are still unacked segments

TCP sender (simplified)

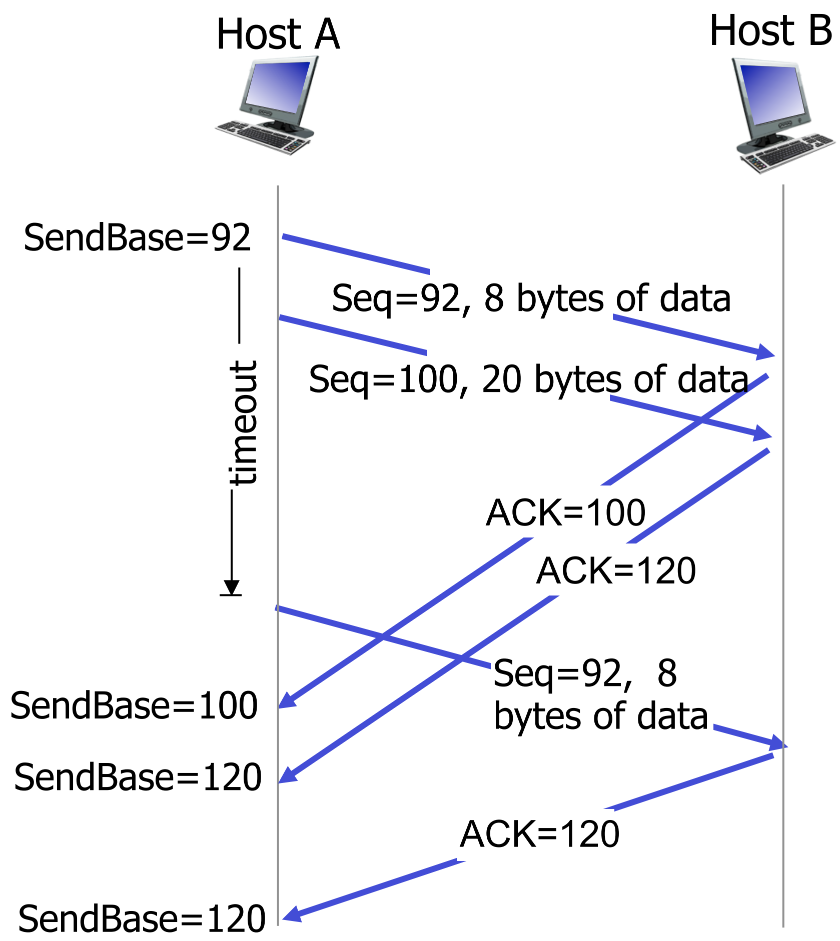
PUTTING IT TOGETHER



TCP: retransmission scenarios

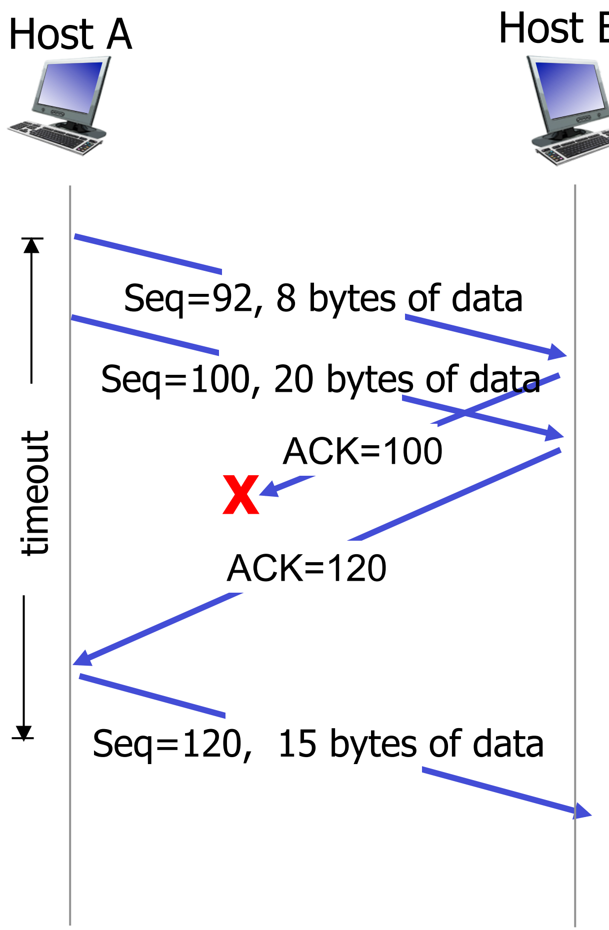


lost ACK scenario



premature timeout

TCP: retransmission scenarios



cumulative ACK

TCP ACK generation [RFC 1122, RFC 2581]

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. # . 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

What does TCP do?

Most of our previous tricks, but a few differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)
- Receivers may not drop out-of-sequence packets (like SR)
- Sender maintains a single retransmission timer (like GBN) and retransmits on timeout
- Introduces fast retransmit: optimisation that uses duplicate ACKs to trigger early retransmission

TCP fast retransmit

- time-out period often relatively long:
  - long delay before resending lost packet
- "Duplicate ACKs" are a sign of an <u>isolated</u> loss
  - The lack of ACK progress means that packet hasn't been delivered
  - Stream of ACKs means some packets are being delivered
  - Could trigger resend on receiving "k" duplicate ACKs (TCP uses k = 3)

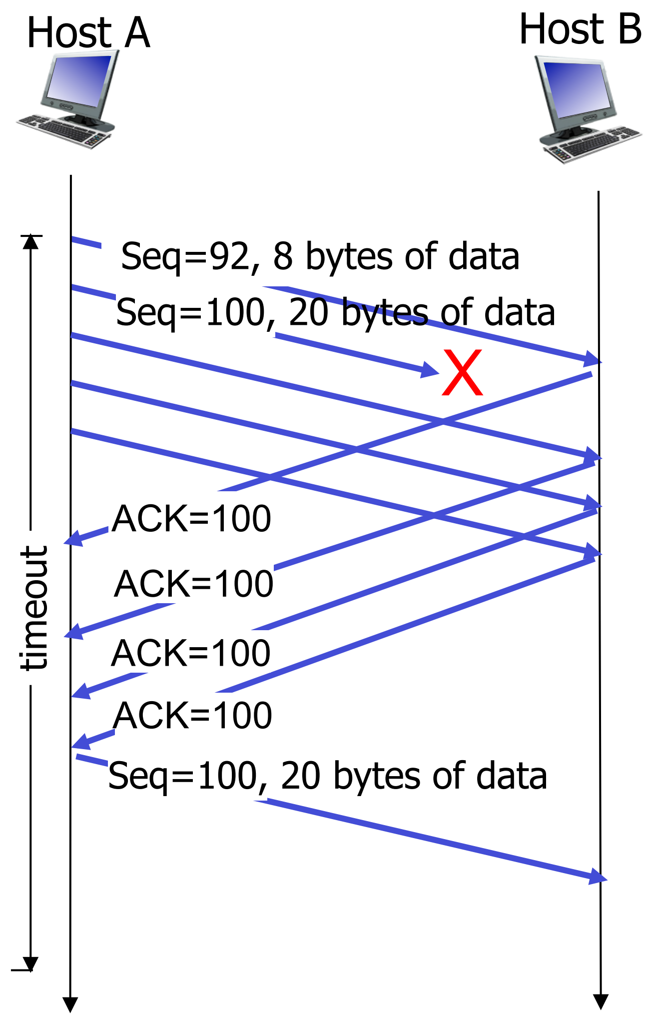
TCP fast retransmit

if sender receives 3 duplicate ACKs for same data

("triple duplicate ACKs"), resend unacked segment with smallest seq #

- likely that unacked segment is lost, so don't wait for timeout

TCP fast retransmit



fast retransmit after sender receipt of triple duplicate ACK

What does TCP do?

Most of our previous ideas, but some key differences

- Checksum
- Sequence numbers are byte offsets
- Receiver sends cumulative acknowledgements (like GBN)
- Receivers do not drop out-of-sequence packets (like SR)
- Sender maintains a single retransmission timer (like GBN) and retransmits on timeout
- Introduces fast retransmit: optimization that uses duplicate ACKs to trigger early retransmission

Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

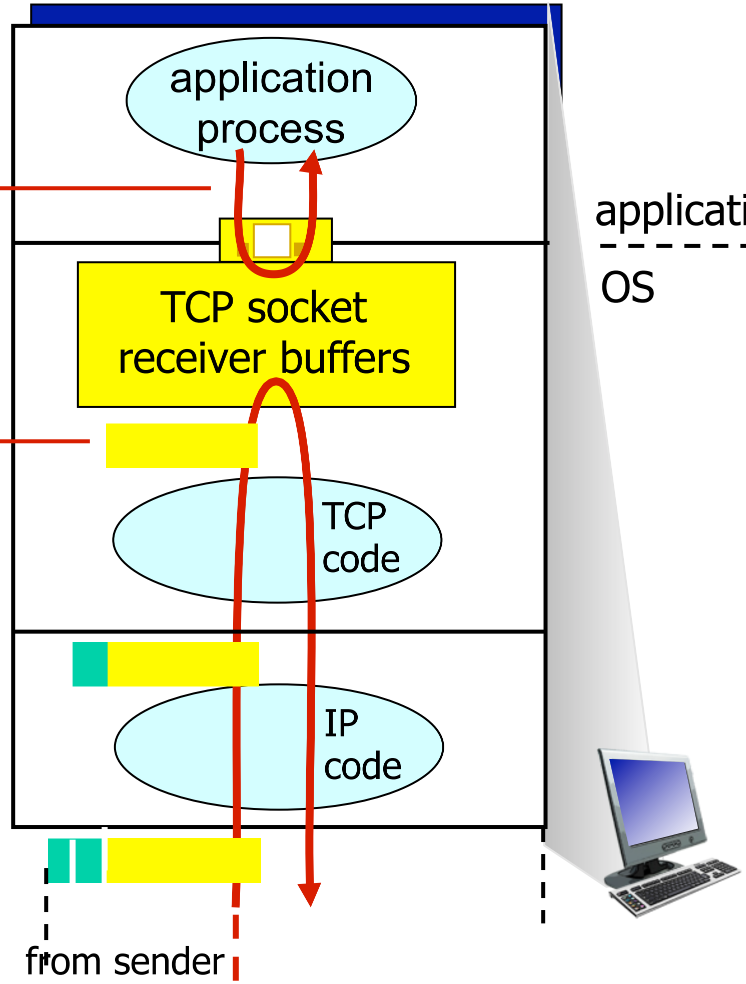
3.6 principles of congestion control

3.7 TCP congestion control

TCP flow control

application may remove data from TCP socket buffers ….

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



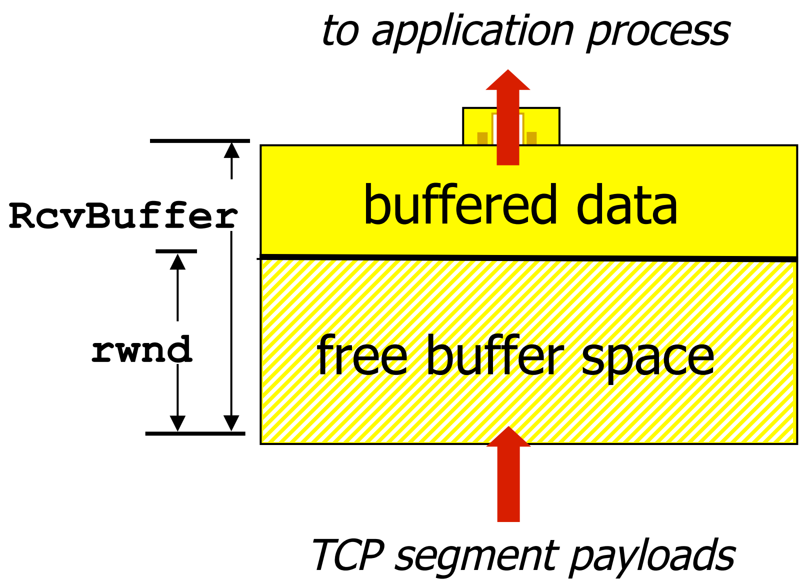
receiver protocol stack

flow control

receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

TCP flow control

- 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)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unacked ("in-flight") data to receiver's **`rwnd`** value
- guarantees receive buffer will not overflow



receiver-side buffering

http://media.pearsoncmg.com/aw/aw_kurose_network_4/applets/flow/FlowControl.htm

Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

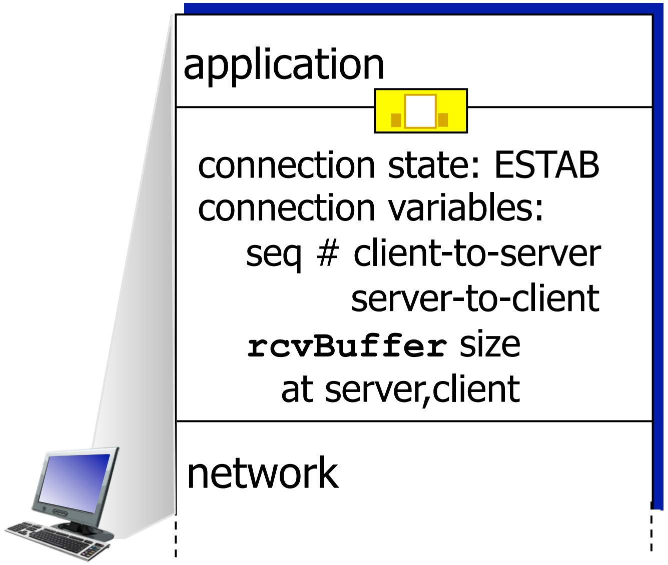
3.6 principles of congestion control

3.7 TCP congestion control

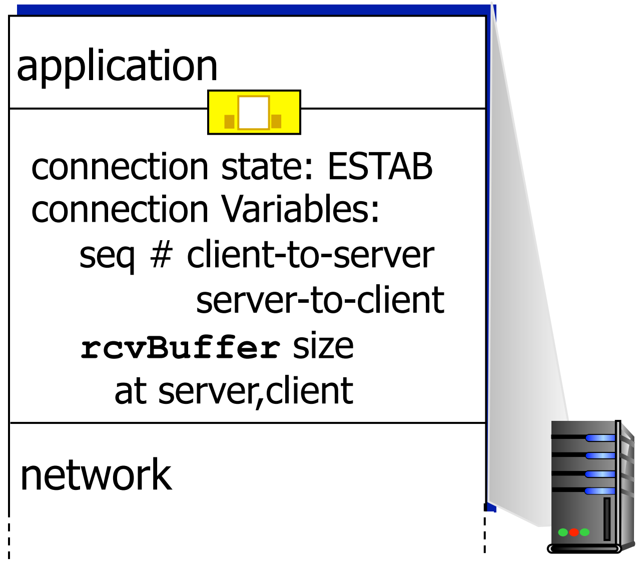
Connection Management

before exchanging data, sender/receiver "handshake":

- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters




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




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

Initial Sequence Number (ISN)

- Sequence number for the very first byte

- Why not just use ISN = 0?

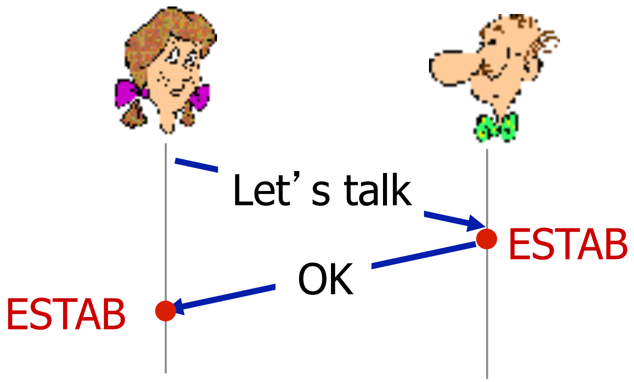
- Practical issue
  - IP addresses and port #s uniquely identify a connection
  - Eventually, though, these port #s do get used again
  - … small chance an old packet is still in flight
  - Easy to hijack a TCP connection (security threat)

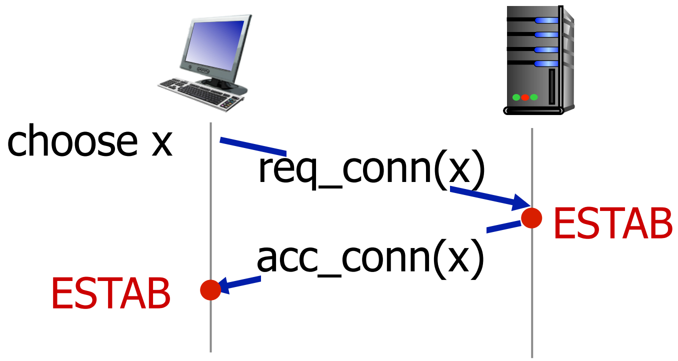
- TCP therefore requires changing ISN

- Hosts exchange ISNs when they establish a connection

Agreeing to establish a connection

2-way handshake:



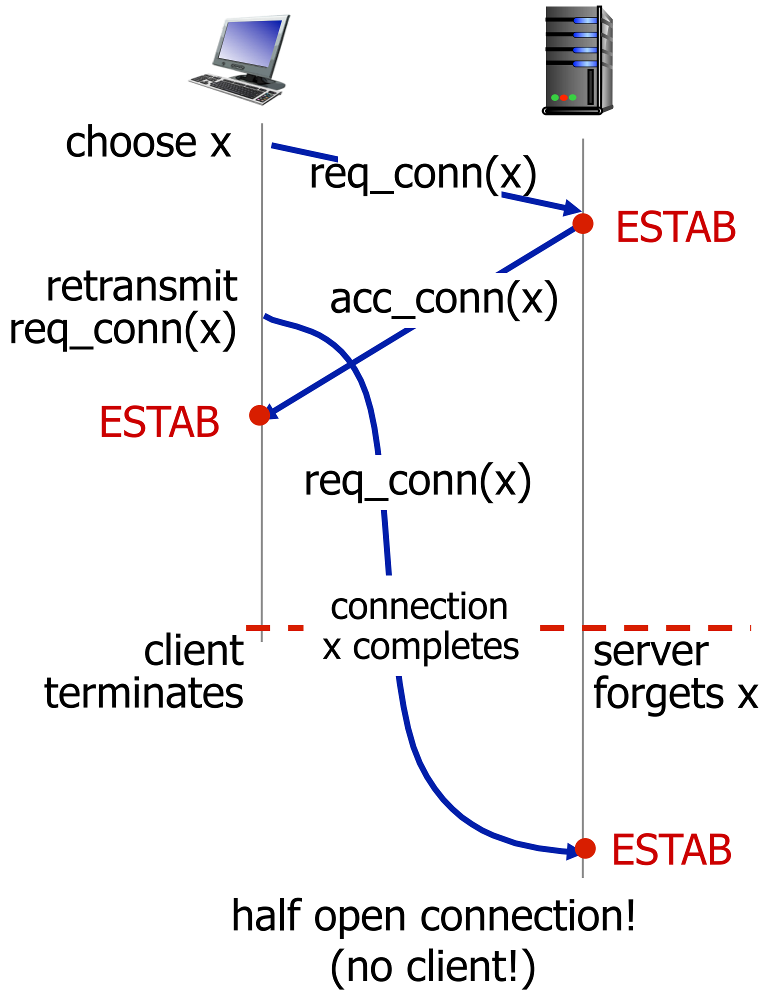


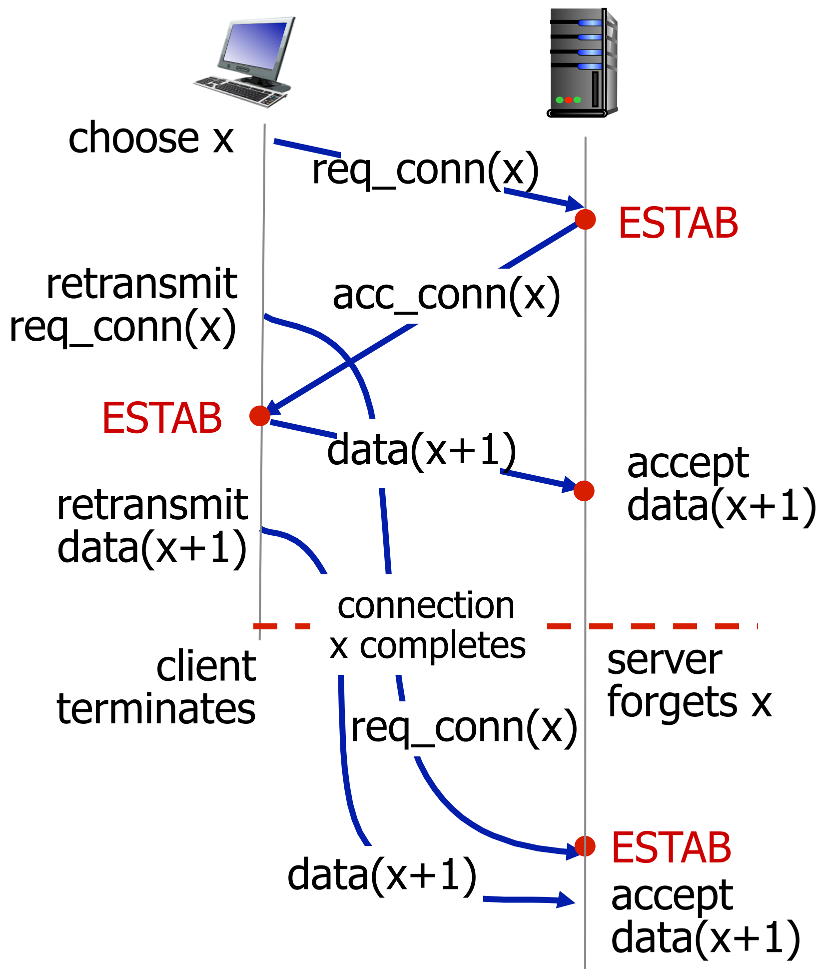
Q: will 2-way handshake always work in network?

- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

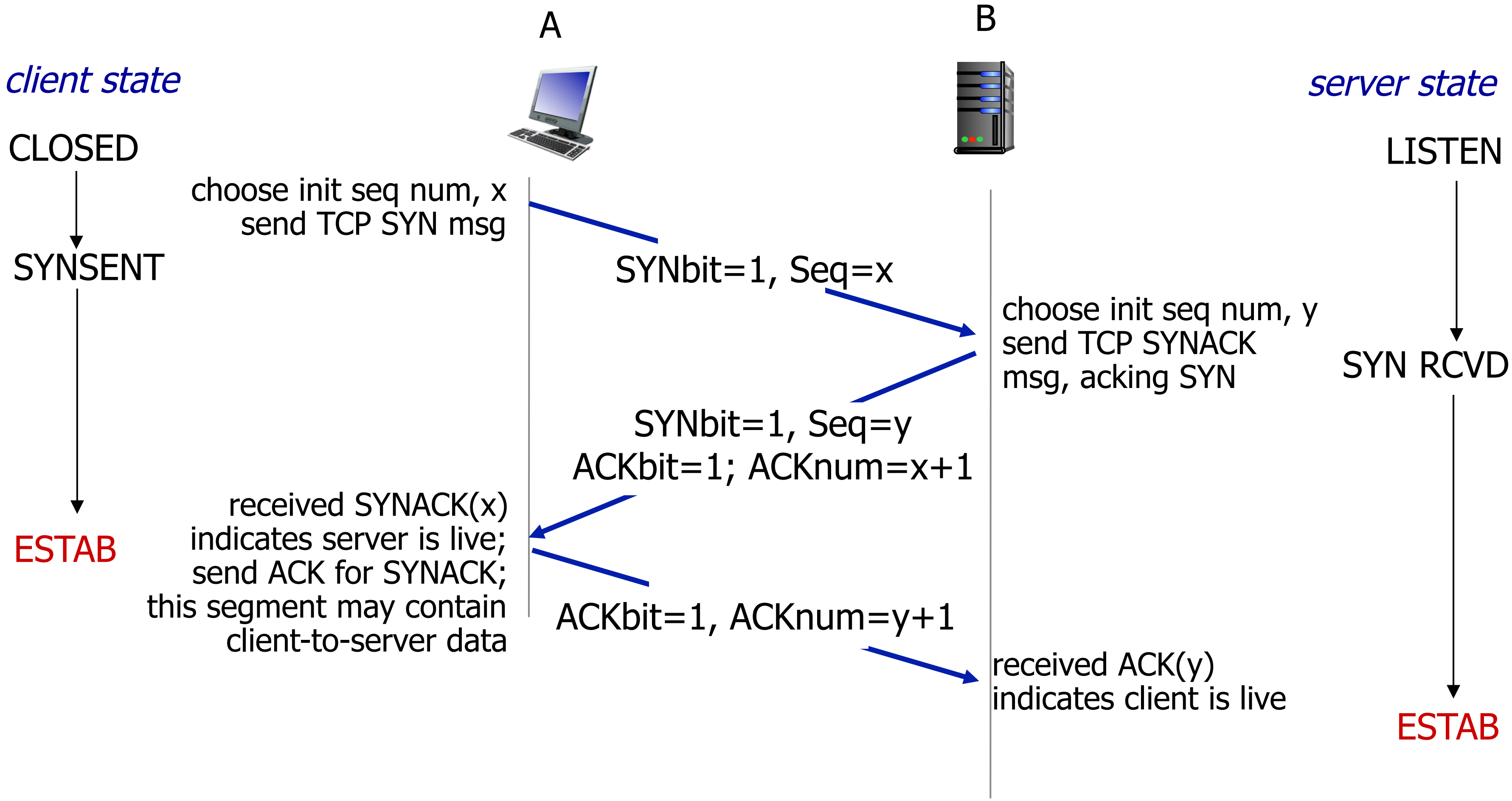
Agreeing to establish a connection

2-way handshake failure scenarios:

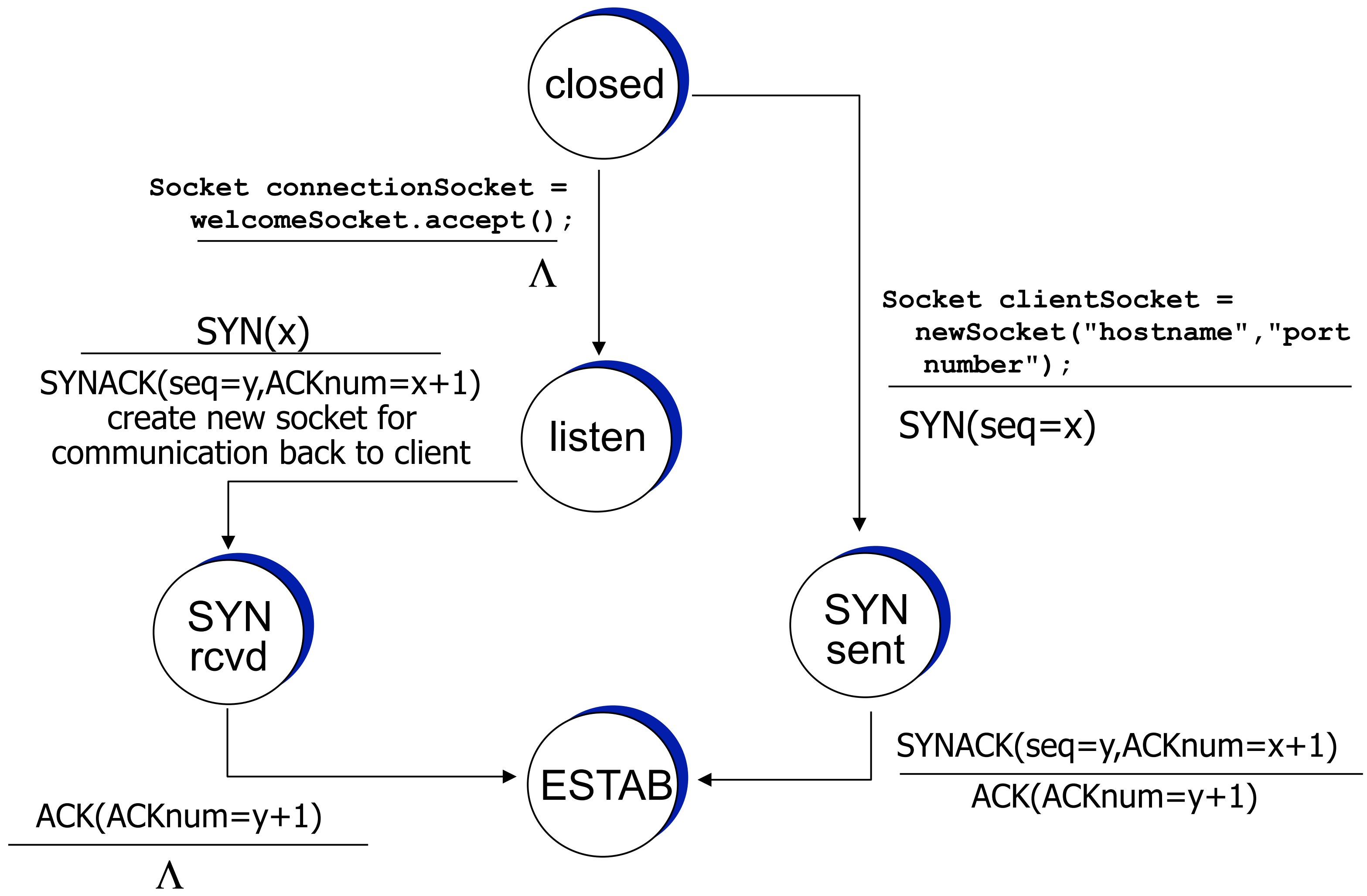




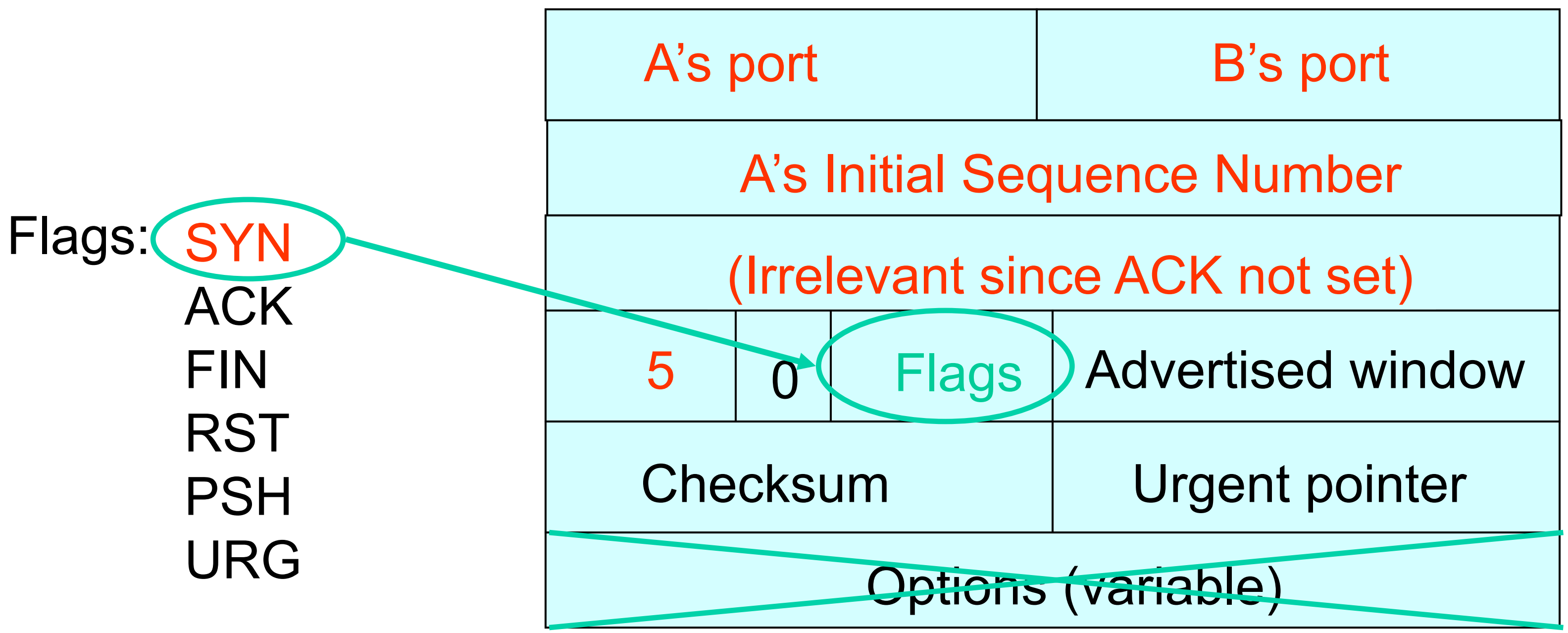
TCP 3-way handshake



TCP 3-way handshake: FSM

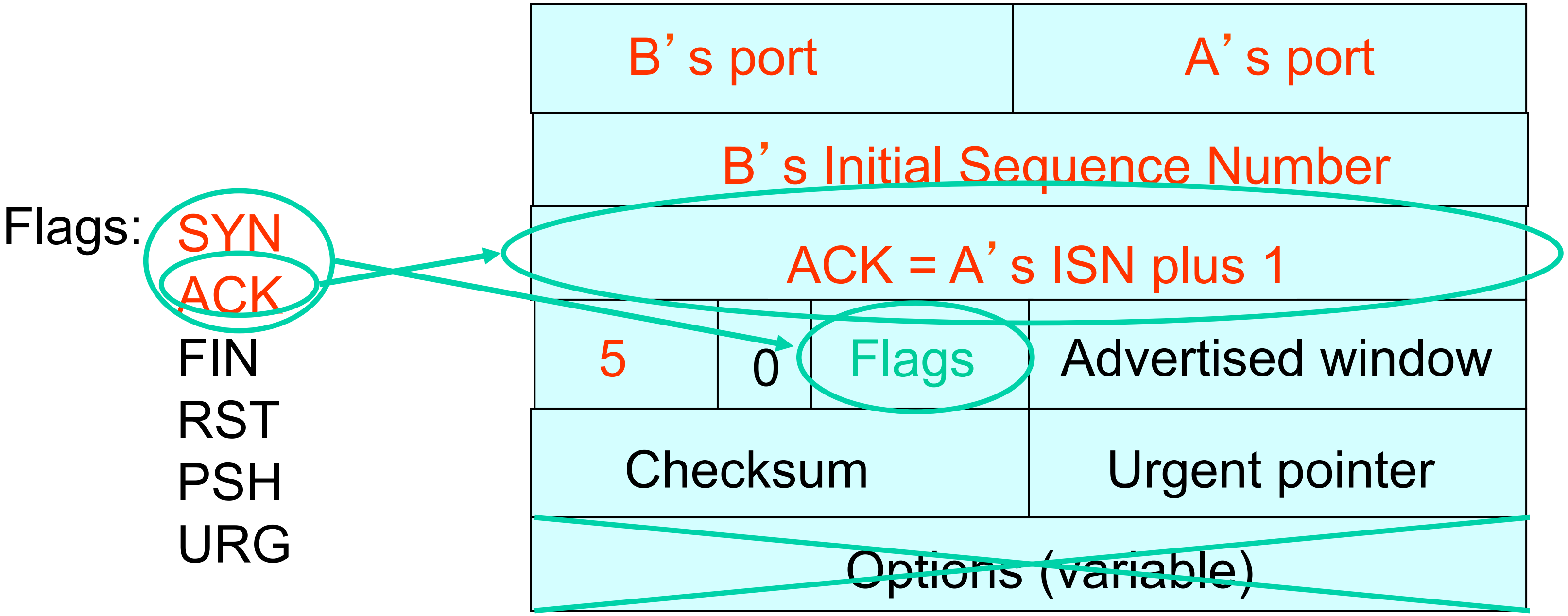


Step 1: A’s Initial SYN Packet



A tells B it wants to open a connection…

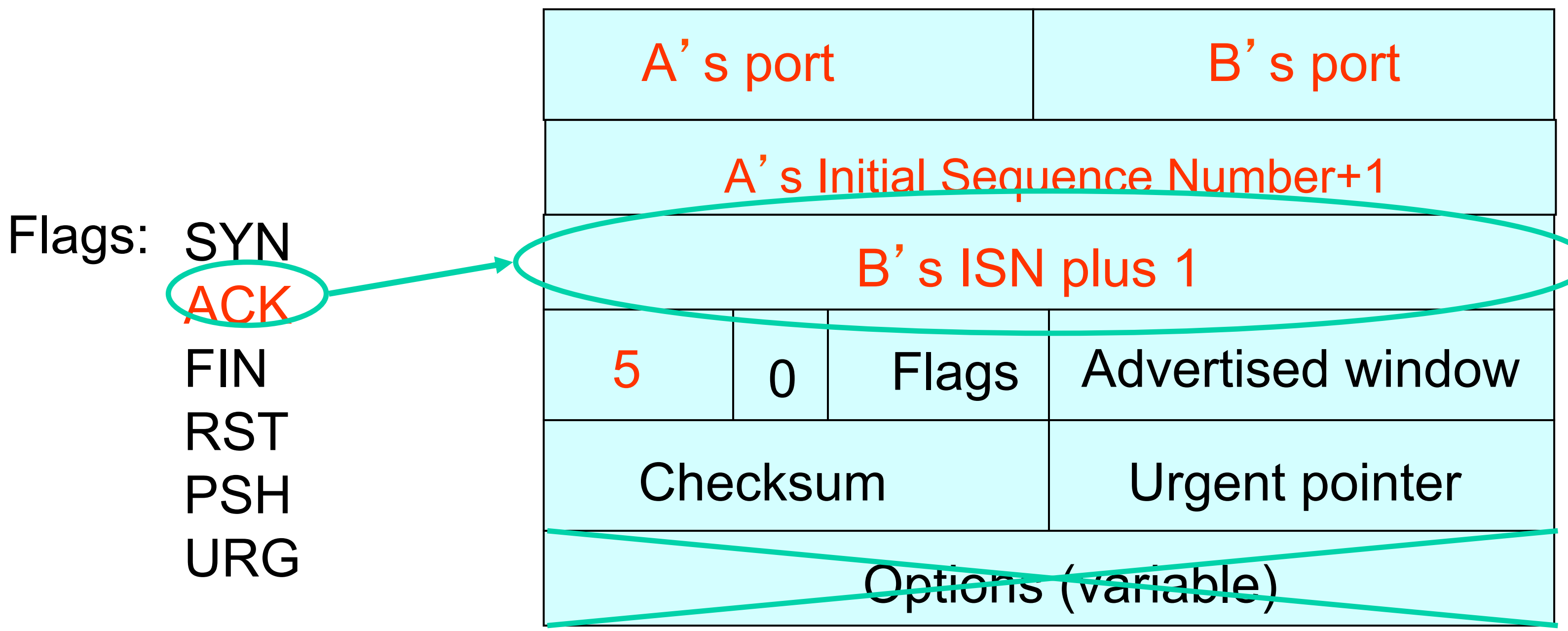
Step 2: B's SYN-ACK Packet



B tells A it accepts, and is ready to hear the next byte…

… upon receiving this packet, A can start sending data

Step 3: A’s ACK of the SYN-ACK



A tells B it’s likewise okay to start sending

… upon receiving this packet, B can start sending data

What if the SYN Packet Gets Lost?

- Suppose the SYN packet gets lost
  - Packet is lost inside the network, or:
  - Server discards the packet (e.g., it's too busy)
- Eventually, no SYN-ACK arrives
  - Sender sets a timer and waits for the SYN-ACK
  - … and retransmits the SYN if needed
- How should the TCP sender set the timer?
  - Sender has no idea how far away the receiver is
  - Hard to guess a reasonable length of time to wait
  - **SHOULD** (RFCs 1122 & 2988) use default of 3 seconds
    - Some implementations instead use 6 seconds

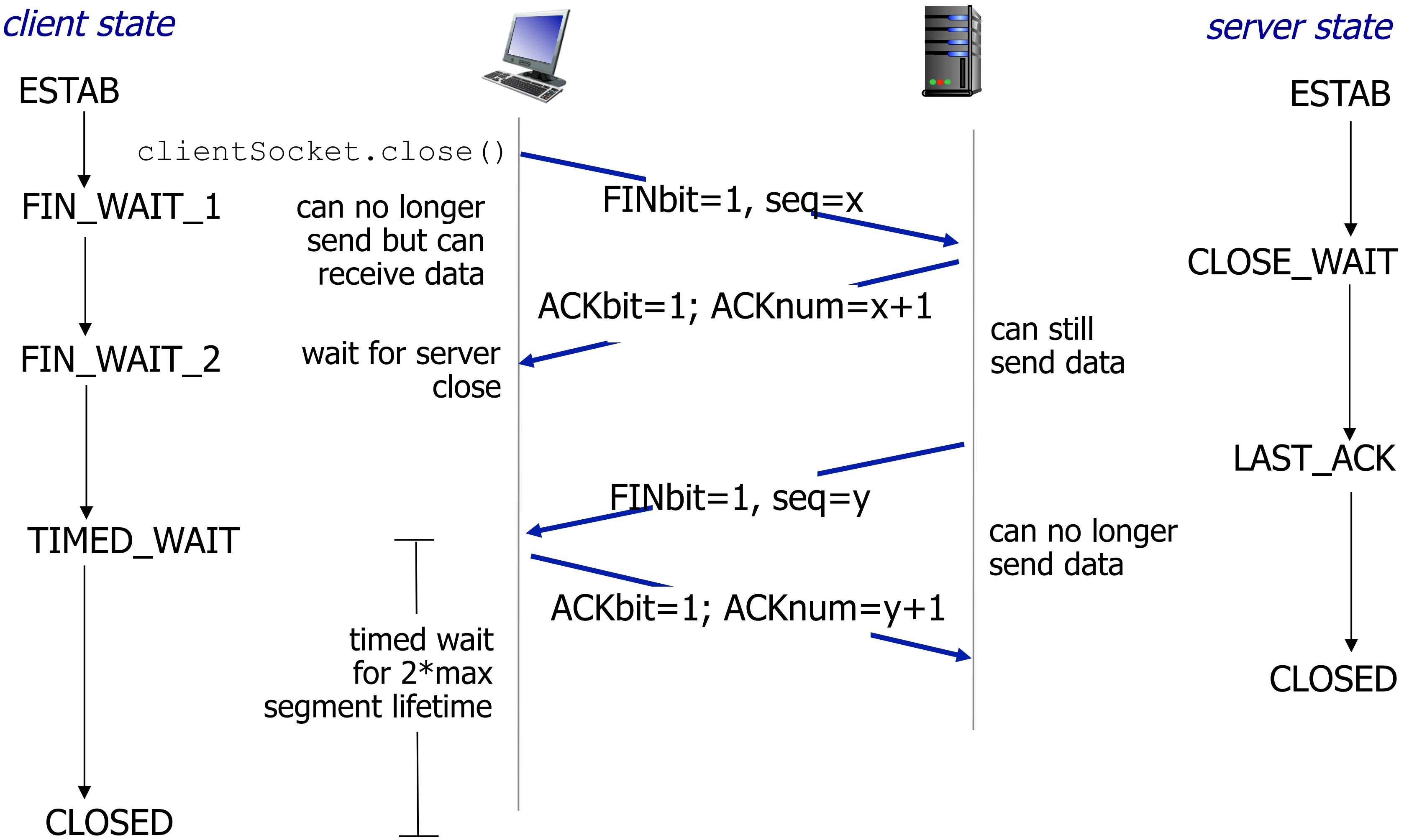
SYN Loss and Web Downloads

- User clicks on a hypertext link
  - Browser creates a socket and does a "connect"
  - The "connect" triggers the OS to transmit a SYN
- If the SYN is lost…
  - 3-6 seconds of delay: can be very long
  - User may become impatient
  - … and click the hyperlink again, or click "reload"
- User triggers an "abort" of the "connect"
  - Browser creates a new socket and another "connect"
  - Essentially, forces a faster send of a new SYN packet!
  - Sometimes very effective, and the page comes quickly

TCP: closing a connection

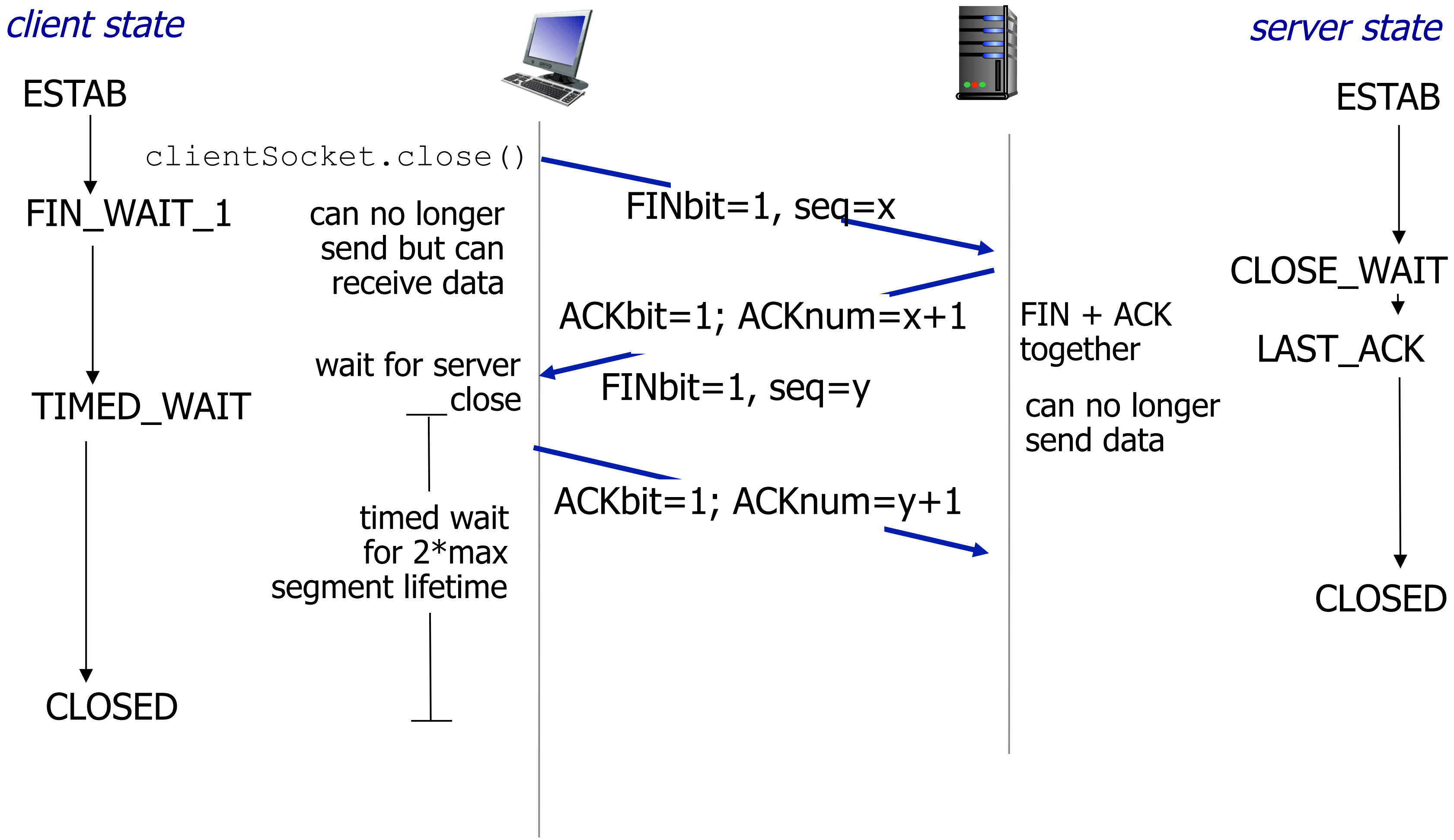
- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled

Normal Termination, One at a Time

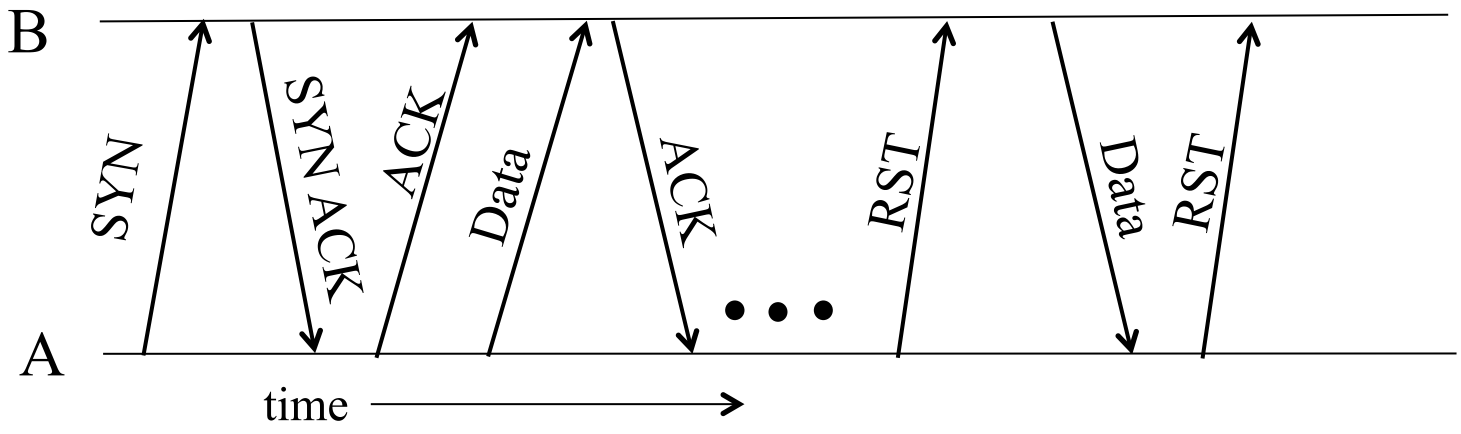


TIMED_WAIT: Can retransmit ACK if ACK is lost

Normal Termination, Both Together

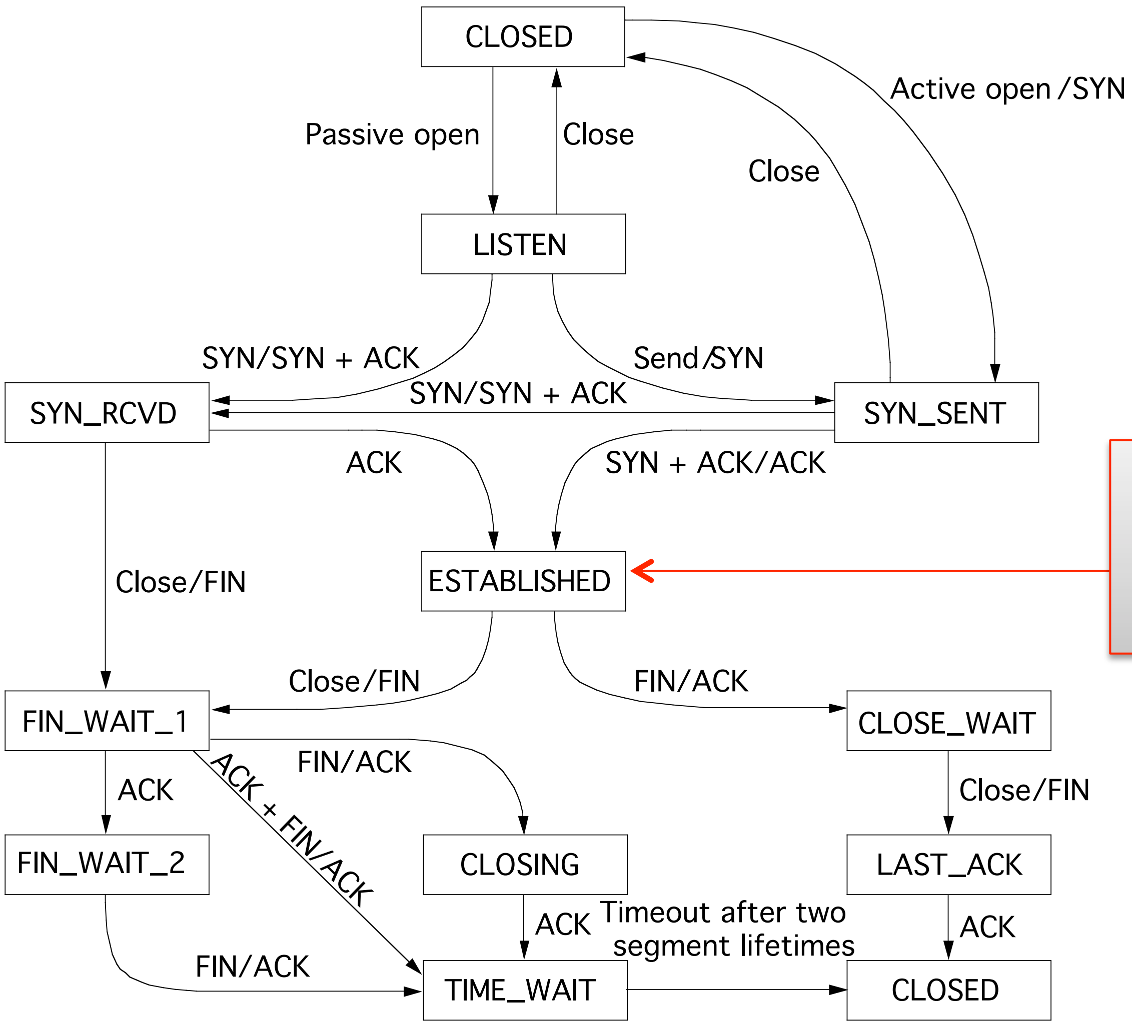


Abrupt Termination



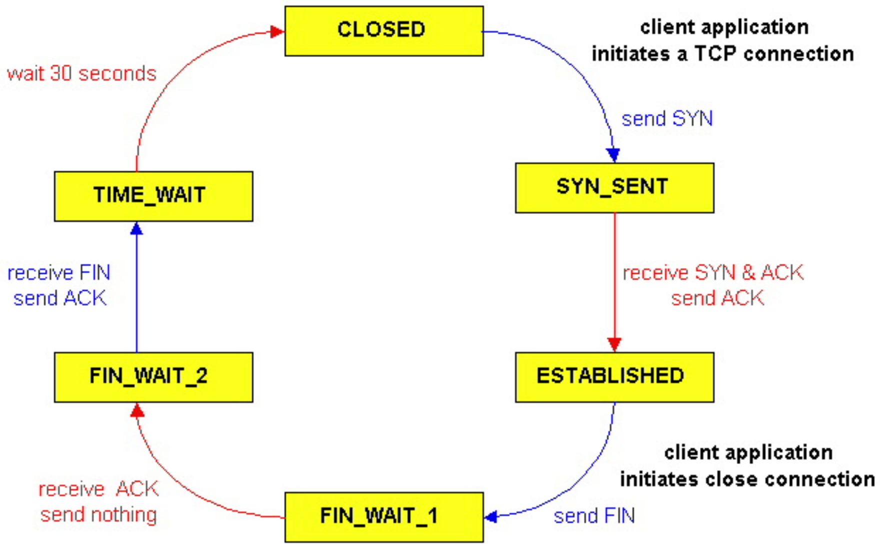
- A sends a RESET (**RST**) to B
  - E.g., because application process on A crashed
- That's it
  - B does not ack the **RST**
  - Thus, **RST** is not delivered reliably
  - And: any data in flight is lost
  - But: if B sends anything more, will elicit another **RST**

TCP Finite State Machine



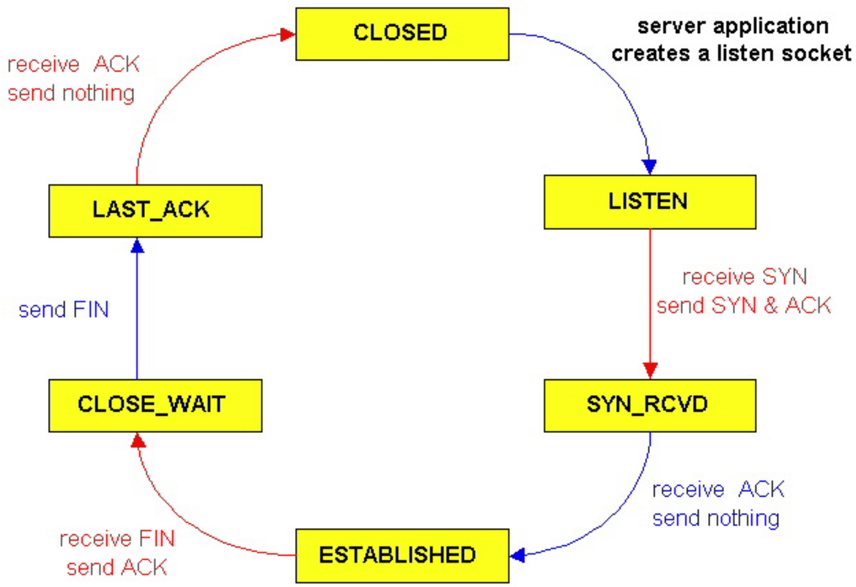
Data, ACK exchanges are in here

TCP Connection Management (cont)



TCP client lifecycle

TCP server lifecycle



TCP SYN Attack (SYN flooding)

- Miscreant creates a fake SYN packet
  - Destination is IP address of victim host (usually some server)
  - Source is some spoofed IP address
- Victim host on receiving creates a TCP connection state i.e allocates buffers, creates variables, etc and sends SYN ACK to the spoofed address (half-open connection)
- ACK never comes back
- After a timeout connection state is freed
- However for this duration the connection state is unnecessarily created
- Further miscreant sends large number of fake SYNs
  - Can easily overwhelm the victim
- Solutions:
  - Increase size of connection queue
  - Decrease timeout wait for the 3-way handshake
  - Firewalls: list of known bad source IP addresses
  - TCP SYN Cookies (explained on next slide)

TCP SYN Cookie

- On receipt of SYN, server does not create connection state
- It creates an initial sequence number (*init_seq*) that is a hash of source & dest IP address and port number of SYN packet (secret key used for hash)
  - Replies back with SYN ACK containing *init_seq*
  - Server does not need to store this sequence number
- If original SYN is genuine, an ACK will come back
  - Same hash function run on the same header fields to get the initial sequence number (*init_seq*)
  - Checks if the ACK is equal to (*init_seq*+1)
  - Only create connection state if above is true
- If fake SYN, no harm done since no state was created

http://etherealmind.com/tcp-syn-cookies-ddos-defence/

Taking Stock (1)

- The concepts underlying TCP are simple
  - acknowledgments (feedback)
  - timers
  - sliding windows
  - buffer management
  - sequence numbers

Taking Stock (2)

- The concepts underlying TCP are simple
- But tricky in the details
  - How do we set timers?
  - What is the seqno for an ACK-only packet?
  - What happens if advertised window = 0?
  - What if the advertised window is ½ an MSS?
  - Should receiver acknowledge packets right away?
  - What if the application generates data in units of 0.1 MSS?
  - What happens if I get a duplicate SYN? Or a RST while I'm in FIN_WAIT, *etc., etc., etc.*

Transport: summary (so far)

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - **congestion control (next week)**
- instantiation, implementation in the Internet
  - UDP
  - TCP

next:

- leaving the network "edge" (application, transport layers)
- into the network "core"

Mid-semester Exam

- **29th Aug (Mon, Week 6), regular lecture hours (4-6pm)**
- **Various rooms – check webpage for your room**
- **Exam will run for 90 minutes**
- Check dedicated page on the course website
- Sample exam provided
- Content
  - Topics covered in Week 1 -5 Lectures
  - Chapter 1, 2 and 3 (3.1-3.5) from textbook
  - All self-study sections are included
  - The external references (papers, links, etc.) are NOT included
- Closed book
- No laptops, tablets, phone, electronic devices, …
- BYO Calculator
- Discussions on forum encouraged
- Good luck