National University of Computer & Emerging Sciences

CS 3001 - COMPUTER NETWORKS

Lecture 14

Chapter 3

13th March, 2025

Nauman Moazzam Hayat

nauman.moazzam@lhr.nu.edu.pk

Office Hours: 11:30 am till 01:00 pm (Every Tuesday & Thursday)

Chapter 3
Transport Layer

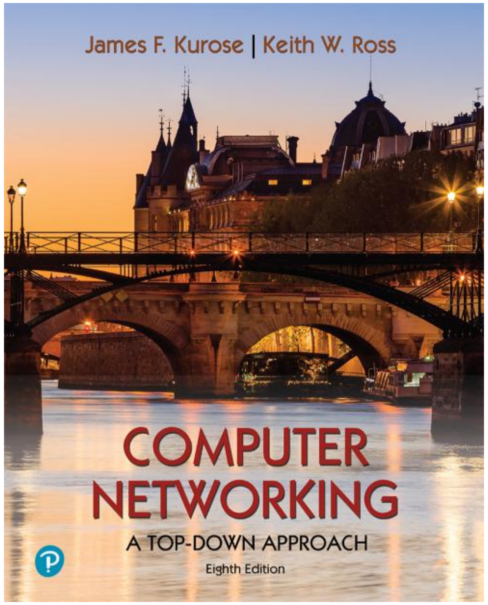
A note on the use of these PowerPoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR



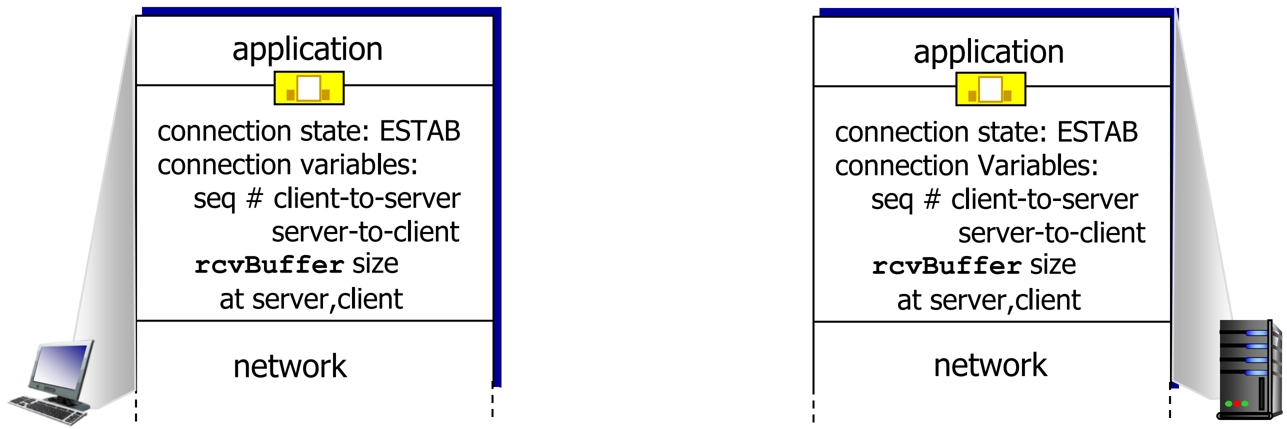
Computer Networking: A Top-Down Approach
8^{th} edition
Jim Kurose, Keith Ross
Pearson, 2020

Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- Principles of congestion control
- TCP congestion control

TCP connection management

before exchanging data, sender/receiver "handshake":
- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters (e.g., starting seq #s)

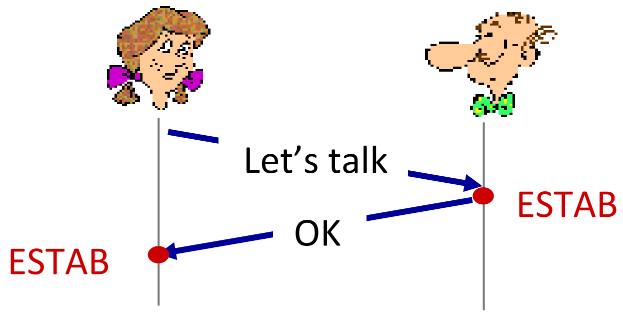


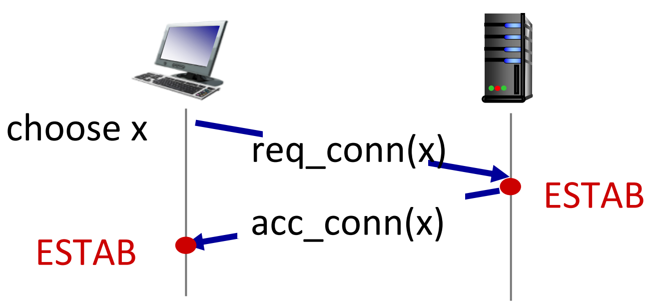

```
Socket clientSocket =
  newSocket("hostname","port number");
```

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

Agreeing to establish a connection

2-way handshake:

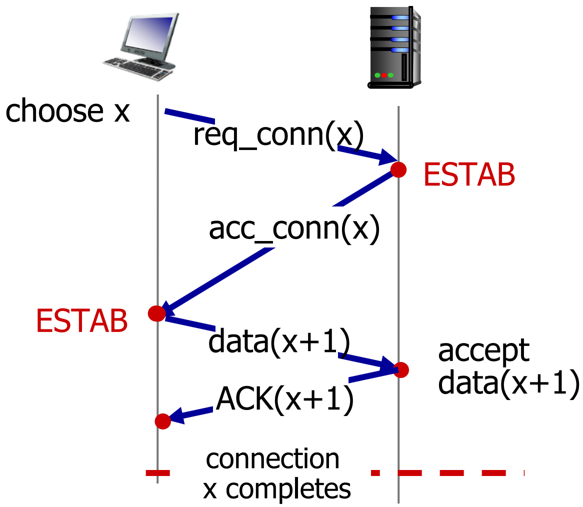




Q: will 2-way handshake always work in network?

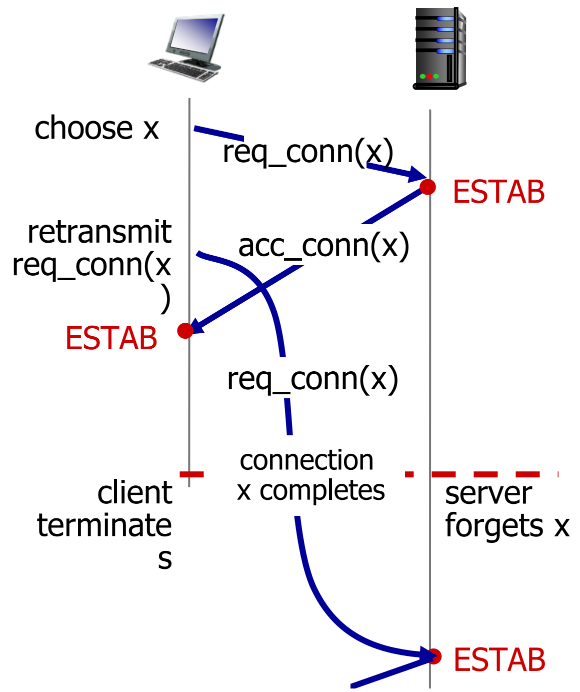
- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can’t “see” other side

2-way handshake scenarios



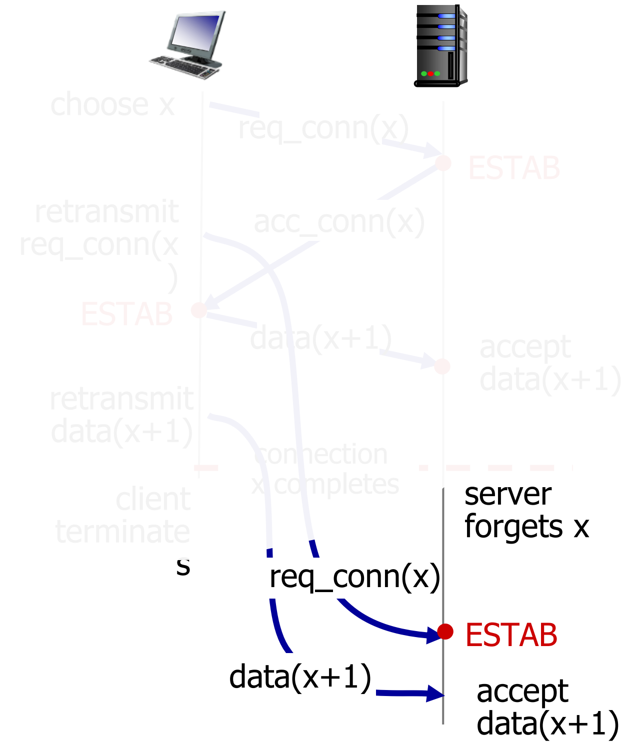
No problem!

2-way handshake scenarios



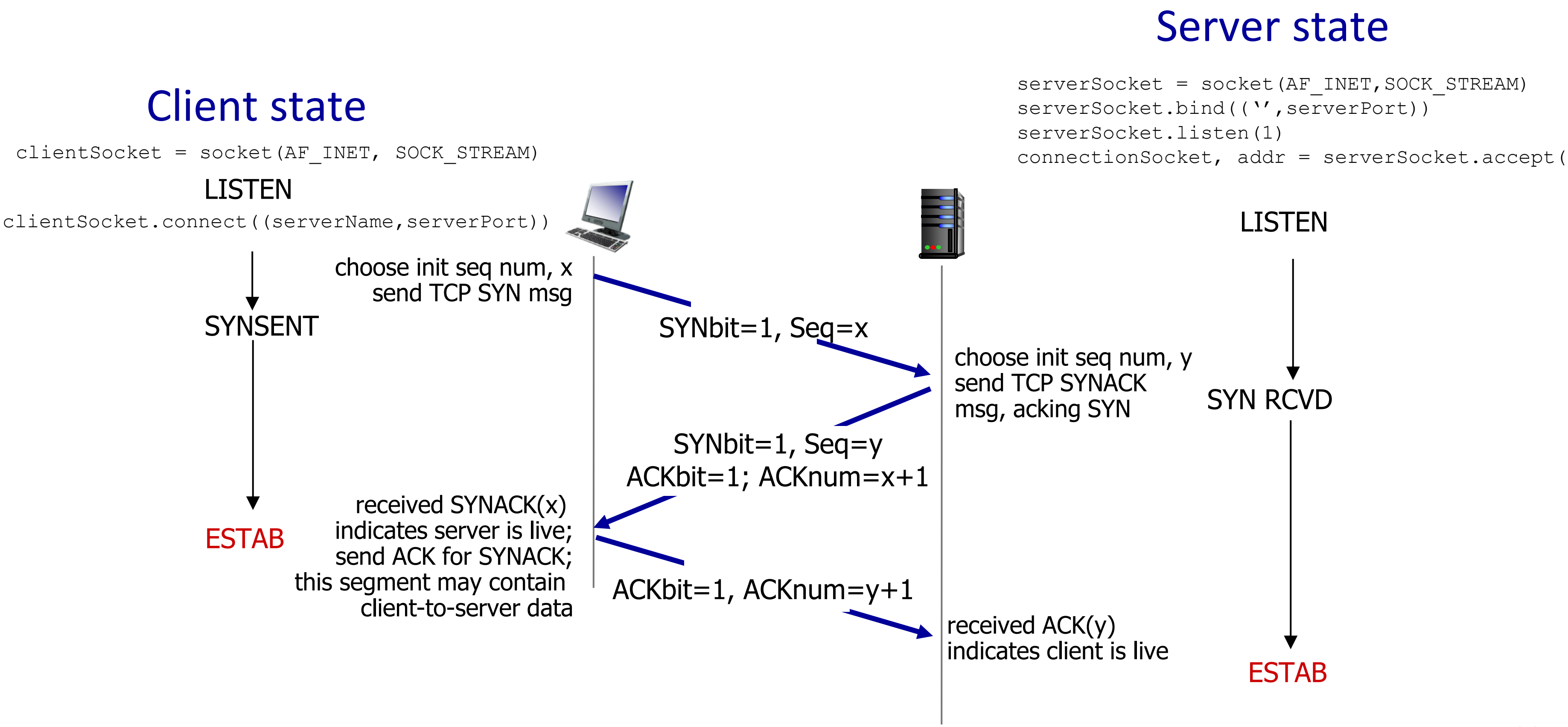
Problem: half open connection! (no client)

2-way handshake scenarios



Problem: dup data accepted!

TCP 3-way handshake



A human 3-way handshake protocol

Closing a TCP 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

Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

Principles of congestion control

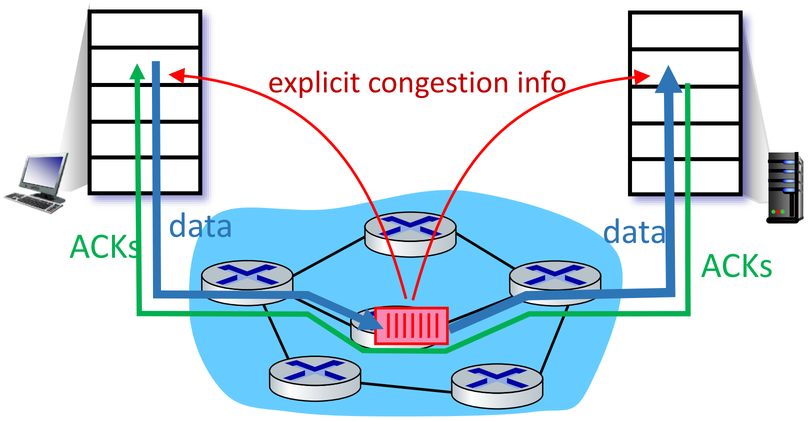
Congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- manifestations:
  - long delays (queueing in router buffers)
  - packet loss (buffer overflow at routers)
- different from flow control!
- a top-10 problem!

congestion control: too many senders, sending too fast

flow control: one sender too fast for one receiver

Approaches towards congestion control



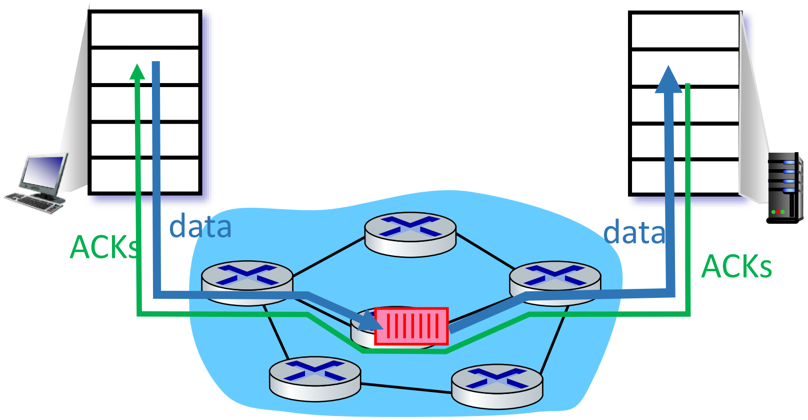
Network-assisted congestion control:

- routers provide *direct* feedback to sending/receiving hosts with flows passing through congested router
- may indicate congestion level or explicitly set sending rate
- TCP ECN, ATM, DECbit protocols

Approaches towards congestion control

End-end congestion control:

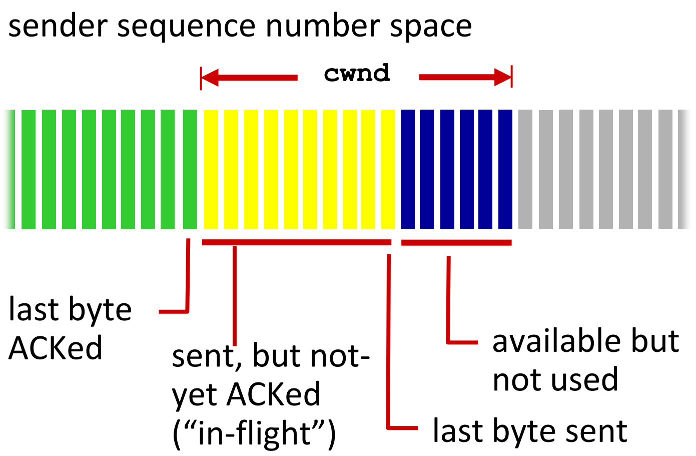
- no explicit feedback from network
- congestion *inferred* from observed loss, delay
- approach taken by TCP



Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

TCP congestion control: details



TCP sending behavior:

- *roughly:* send `cwnd` bytes, wait RTT for ACKS, then send more bytes

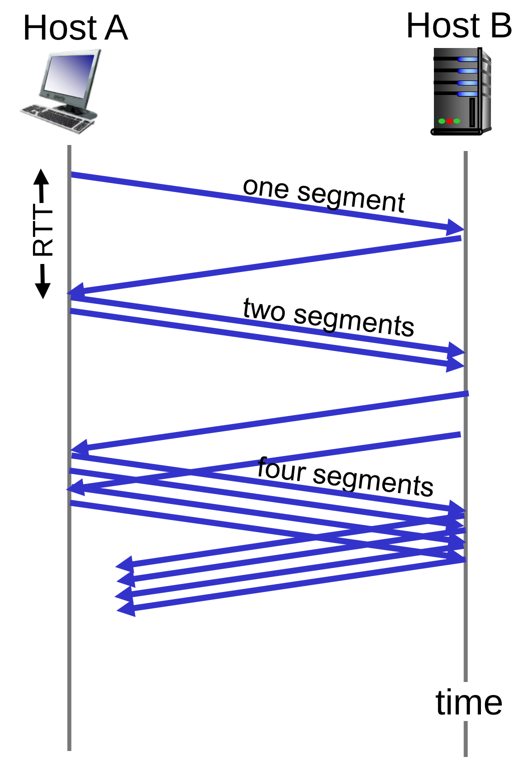
$$\text{TCP rate} \approx \frac{\texttt{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

- TCP sender limits transmission: **`LastByteSent- LastByteAcked` ≤ `cwnd`**
- `cwnd` is dynamically adjusted in response to observed network congestion (implementing TCP congestion control)

`(LastByteSent - LastByteAcked` ≤ `min{cwnd, rwnd}`
`but ignore rwnd for this congestion control discussion`

TCP slow start

- when connection begins, increase rate exponentially until first loss event:
  - initially **`cwnd`** = 1 MSS
  - double **`cwnd`** every RTT
  - done by incrementing **`cwnd`** for every ACK received
- *summary:* initial rate is slow, but ramps up exponentially fast



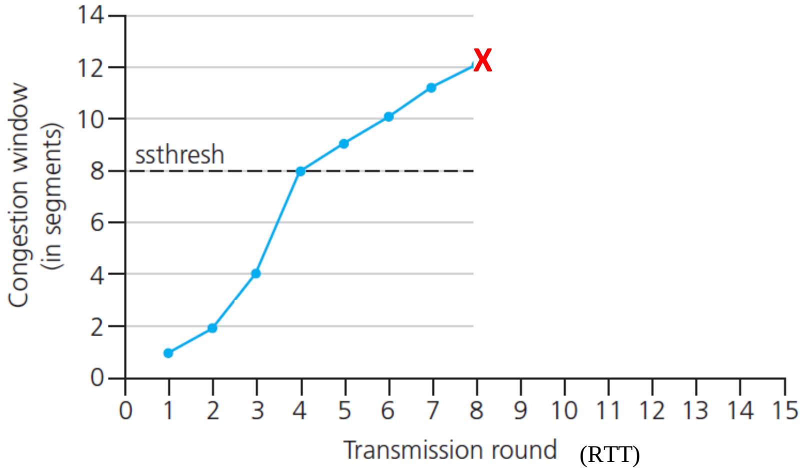
TCP: from slow start to congestion avoidance

Q: when should the exponential increase switch to linear?

A: when **`cwnd`** gets to 1/2 of its value before timeout. (ssthresh)

Implementation:

- variable **`ssthresh`**
- on loss event, **`ssthresh`** is set to 1/2 of **`cwnd`** just before loss event



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

TCP congestion control: AIMD (used in Congestion Avoidance mode)

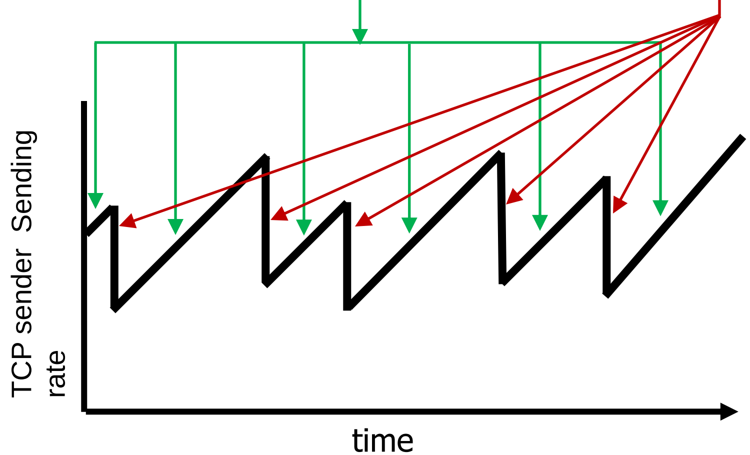
- *approach:* senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event

Additive Increase

increase sending rate by 1 maximum segment size every RTT until loss detected

Multiplicative Decrease

cut sending rate in half at each loss event



AIMD sawtooth behavior: *probing* for bandwidth

TCP AIMD: more

Multiplicative decrease detail - sending rate is:

TCP Tahoe

- Cut cwnd to 1 MSS (maximum segment size) when loss detected by timeout
- Cut cwnd to 1 MSS (maximum segment size) when loss detected by triple duplicate ACK

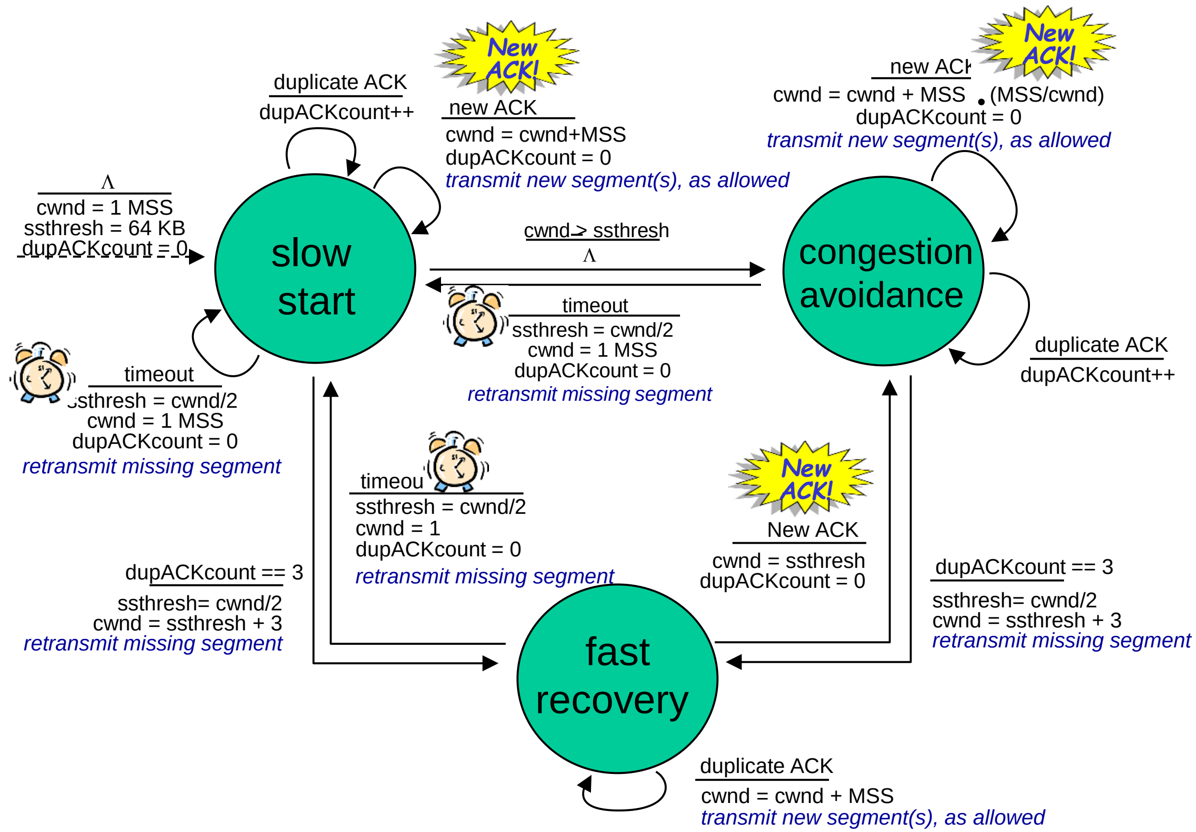
TCP Reno

- Cut cwnd to 1 MSS (maximum segment size) when loss detected by timeout
- Cut cwnd in half on loss detected by triple duplicate ACK, then grows linearly

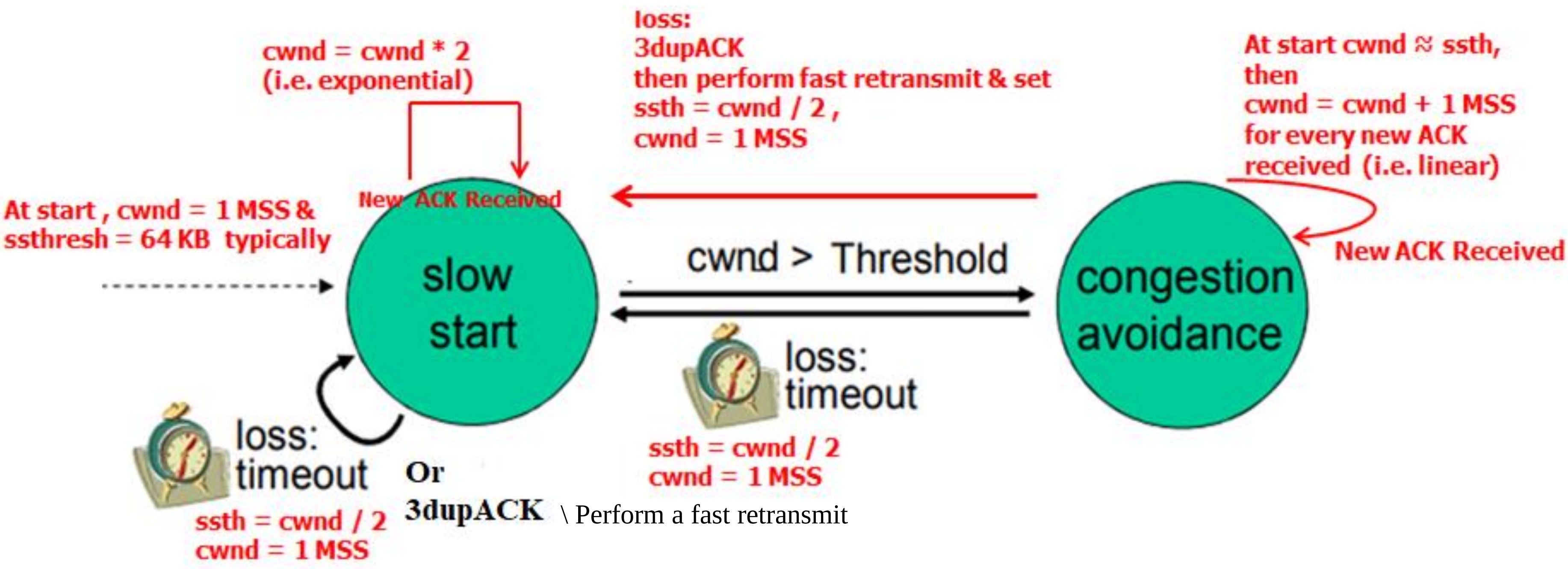
Why AIMD?

- AIMD – a distributed, asynchronous algorithm – has been shown to:
  - optimize congested flow rates network wide!
  - have desirable stability properties

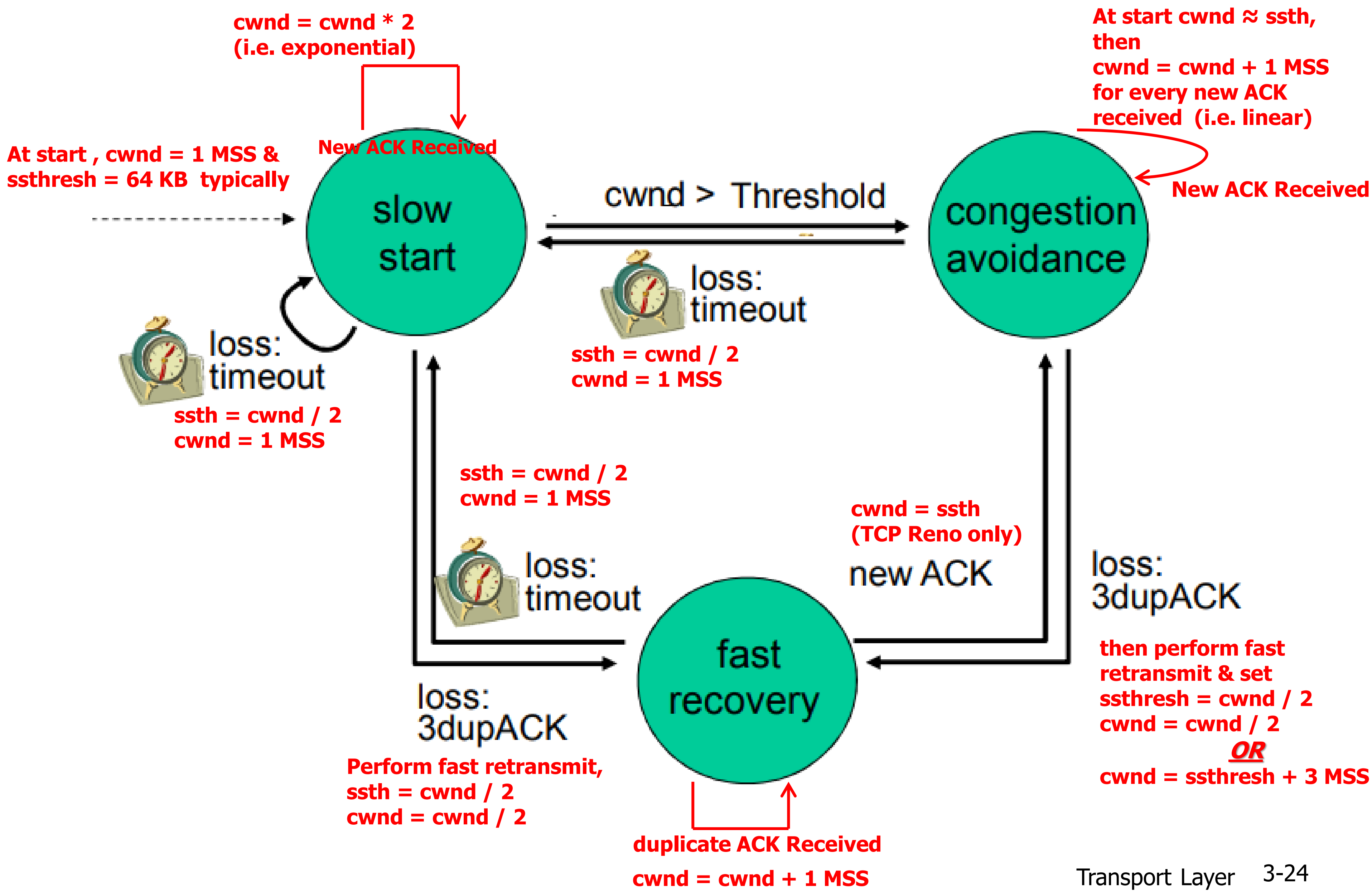
Summary: TCP congestion control



Summary: TCP Congestion Control (TCP Tahoe)



Summary: TCP Congestion Control (TCP Reno)



Chapter 3: summary

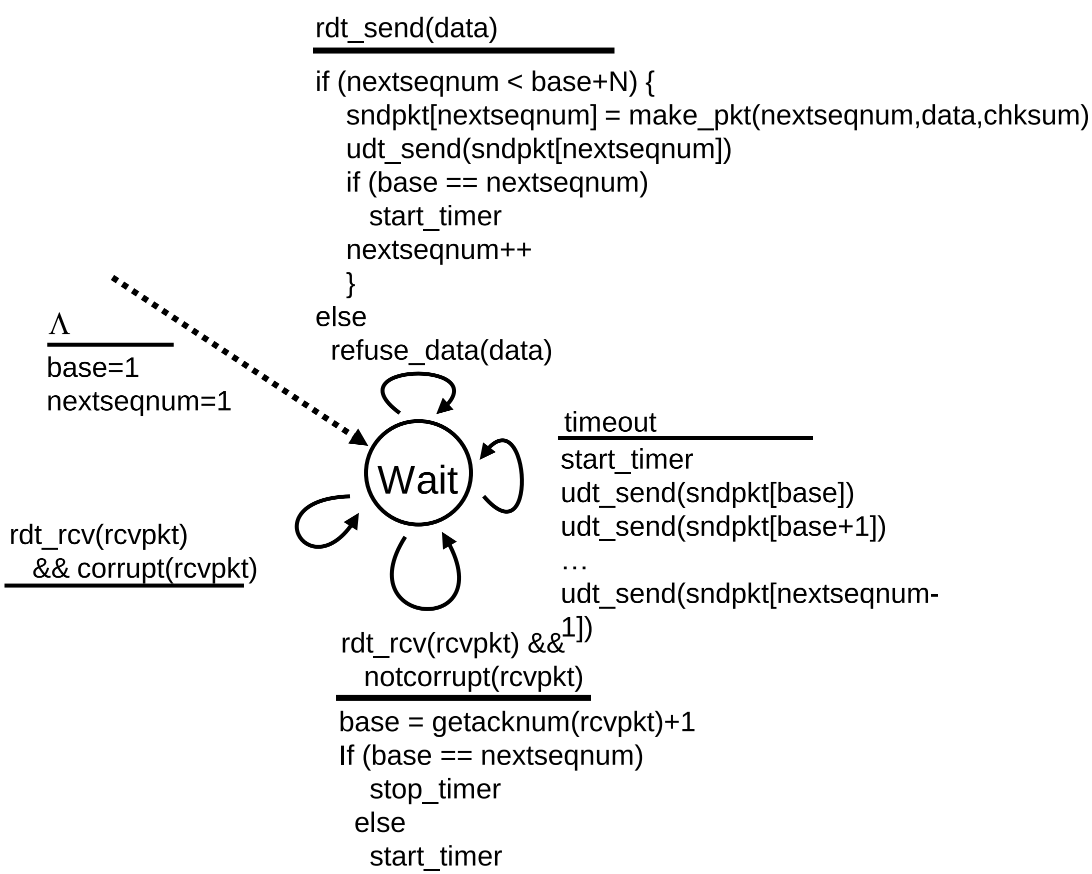
- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- instantiation, implementation in the Internet
  - UDP
  - TCP

Up next:

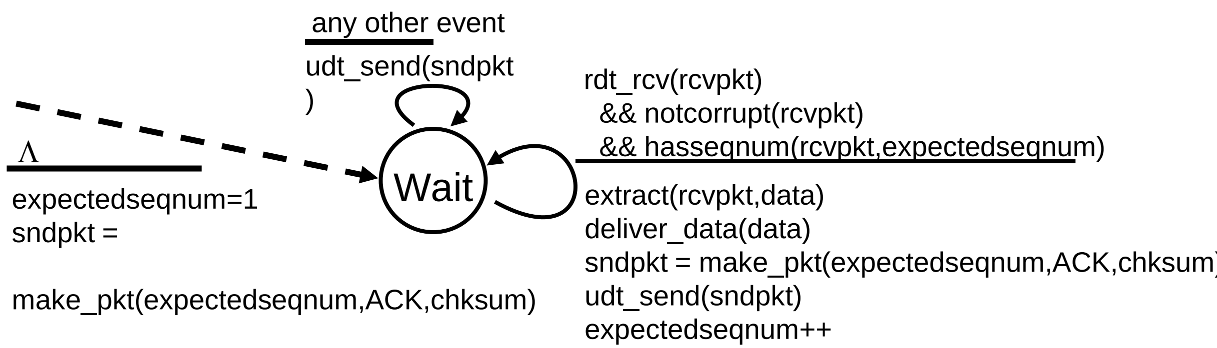
- leaving the network "edge" (application, transport layers)
- into the network "core"
- two network-layer chapters:
  - data plane
  - control plane

Additional Chapter 3 slides

Go-Back-N: sender extended FSM



Go-Back-N: receiver extended FSM



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

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

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

TCP sender (simplified)

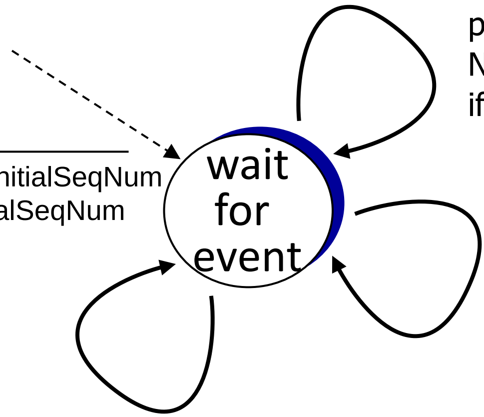
data received from application above

---

create segment, seq. #: NextSeqNum
pass segment to IP (i.e., “send”)
NextSeqNum = NextSeqNum + length(data)
if (timer currently not running)
    start timer

Λ

NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum



timeout

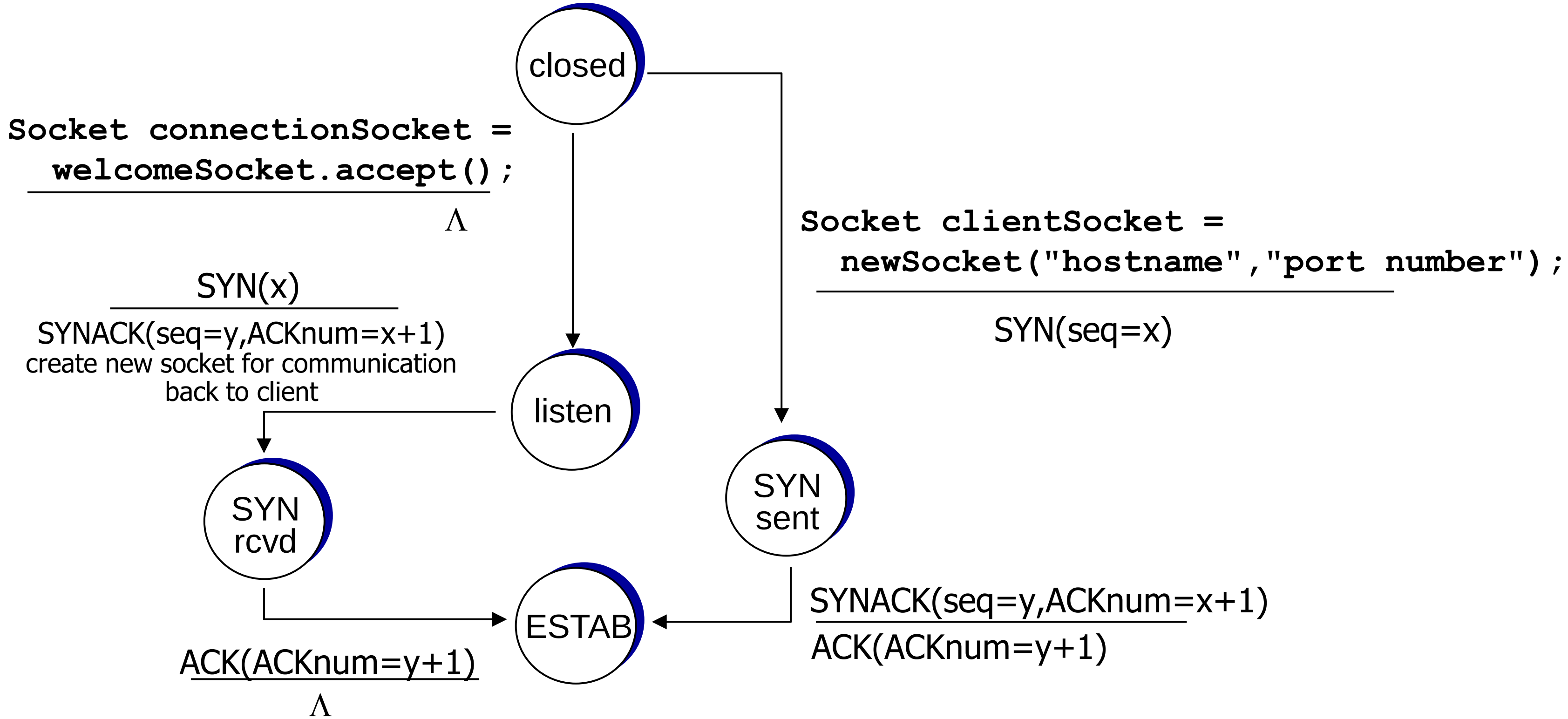
---

retransmit not-yet-acked segment with smallest seq. #
start timer

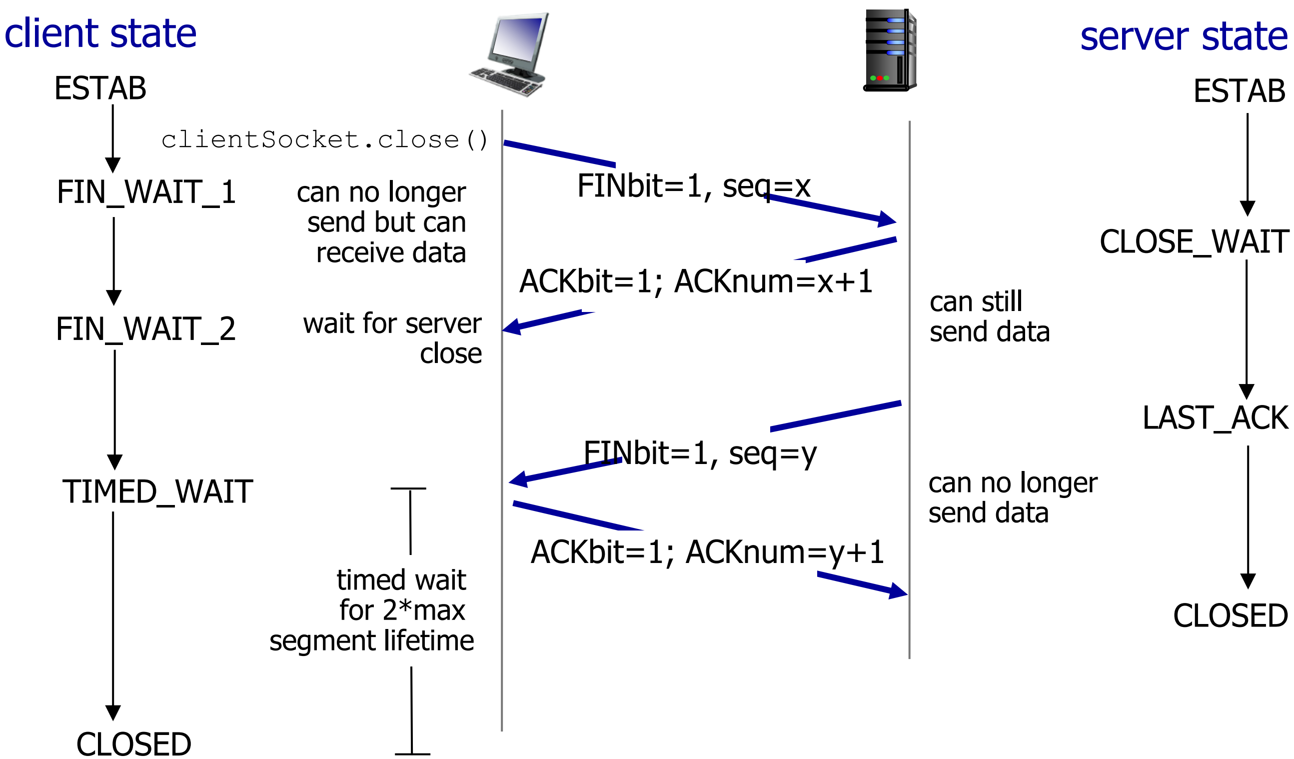
ACK received, with ACK field value y

```
if (y > SendBase) {
   SendBase = y
   /* SendBase–1: last cumulatively ACKed byte */
   if (there are currently not-yet-acked segments)
        start timer
      else stop timer
  }
```

TCP 3-way handshake FSM



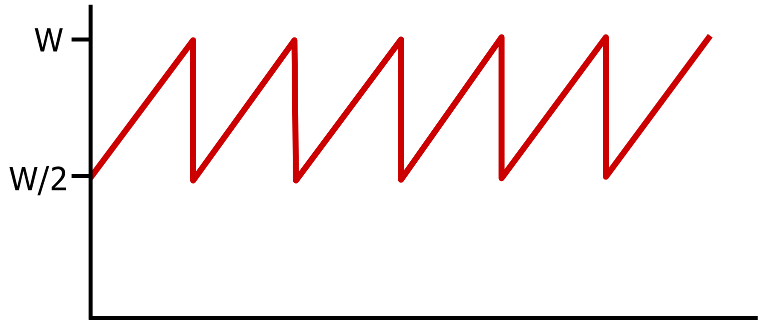
Closing a TCP connection



TCP throughput

- avg. TCP thruput as function of window size, RTT?
  - ignore slow start, assume there is always data to send
- W: window size (measured in bytes) where loss occurs
  - avg. window size (# in-flight bytes) is ¾ W
  - avg. thruput is 3/4W per RTT

$$\text{avg TCP thruput} = \frac{3}{4}\frac{W}{RTT} \text{ bytes/sec}$$



TCP over "long, fat pipes"

- example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- requires W = 83,333 in-flight segments
- throughput in terms of segment loss probability, L [Mathis 1997]:

$$\text{TCP throughput} = \frac{1.22 \cdot \text{MSS}}{\text{RTT}\sqrt{L}}$$

➜ to achieve 10 Gbps throughput, need a loss rate of $L = 2 \cdot 10^{-10}$ *– a very small loss rate!*

- versions of TCP for long, high-speed scenarios

Assignment # 3 (Chapter – 3)

- *3rd Assignment will be uploaded on Google Classroom on Thursday, 13th March, 2025, in the Stream - Announcement Section*

- *Due Date: ~~Thursday, 20th March~~ Tuesday, 25th March, 2025 (Handwritten solutions to be submitted during the lecture; deadline extended due to LAB midterms next week)*

- *Please read all the instructions carefully in the uploaded Assignment document, follow & submit accordingly*

Quiz # 3 (Chapter – 3)

- *On: ~~Thursday, 20th March, 2025~~ , Tuesday, 25th March, 2025(During the lecture; deadline extended due to LAB midterms next week)*

- *Quiz to be taken during <u>own section class only</u>*