CS 348: Computer Networks

- TCP; 24th Sept – 4th Oct 2012

Instructor: Sridhar Iyer
IIT Bombay

Reliable Transport

- We have already designed a reliable communication protocol for an analogy scenario.
  - Recall the functioning of secretaries in the CEO example. What did they have to do to reliably transfer the document using weak and unreliable messenger boys?
- From the analogy, we have learnt some ideas about how to ensure reliable transport.
  - Use buffers, timeouts, acknowledgments, retransmission ..
- We need to apply these ideas to an IP network to get details of the protocol between a source (S) and destination (D).

More specifically ...

The actions of the transport layer:

- Before beginning the data transfer:
  - What actions does (transport layer at) S need to do?
  - What messages does S need to send to D?
  - What responses does D need to give to S?
  - What actions does D need to do at its end?
- During the data transfer:
  - Consider above questions again in this context.
- Upon completion of the data transfer:
  - Consider above questions again in this context.

TCP: Transmission Control Protocol [RFC 793, …]

Guaranteed service protocol

- Ensures that a packet has been received by the destination by using timeouts, acknowledgements and retransmission

Connection-oriented protocol

- Applications need to establish a TCP connection prior to transfer, to fix initial sequence numbers
  - Done using a 3-way handshake

Full duplex protocol

- Both ends can simultaneously read and write

More TCP features

Flow and congestion control:

- Source uses feedback (ack) to adjust transmission rate.

Byte stream:

- Ignores message boundaries.
- Source may send two messages of length 20 and 50 bytes, but destination may simply receive 70 bytes.

Multiplexed:

- many applications can share access to a single TCP layer.

TCP functioning

Application data is broken into Segments (what TCP considers the best sized units to send)

- Segment: unit of data passed from TCP to IP
- MSS: Maximum segment size

TCP sequences data by associating a sequence number with every **byte** it sends

Sending TCP maintains a timer for each segment sent

- waiting for acknowledgement (ACK)
- If ACK doesn't come in time, segment is retransmitted

TCP functioning

Receiving TCP

- Sends ACK: ACK number is the sequence number of the next byte expected
- Re-sequences the data
- Discards duplicates

•Congestion and Flow control

- Sending TCP regulates amount of data to avoid network congestion
- Receiving TCP prevents fast senders from swamping it

TCP connections and sockets

connections are the fundamental abstraction of communication.

- Port: A number on a host assigned to an application to allow multiple destinations.
- Endpoint: A pair, a destination host and a port number on that host.
- Connection: A pair of end points.
- Socket: An abstract address formed by the IP address and port number (characterizes an endpoint)

Unique identifier for a connection: [<Source IP address, port number>, <Destination IP address, port number>]

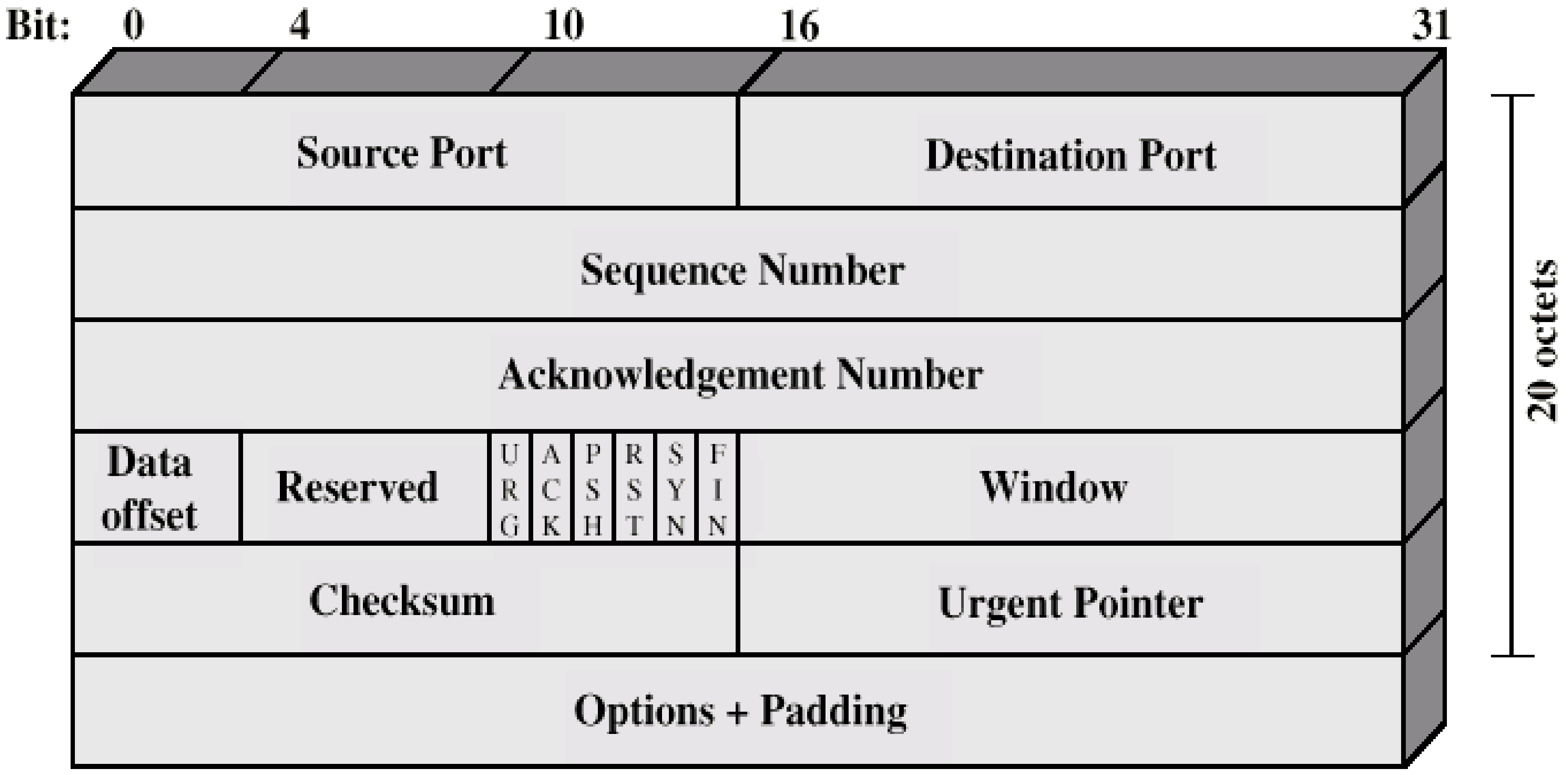
There are exactly two **end-points** communicating with each other in a TCP connection; Broadcast and multicast aren’t applicable to TCP.

MSS: Maximum segment size

- Largest size of segment that TCP will send to the other end
  - Each end announces its MSS at connection establishment time
  - default size is 536 bytes (576 – 40)

- Done to avoid fragmentation
  - MSS related to outgoing link’s MTU

TCP header


Bit: 0 4 10 16 31
Source Port
Destination Port
Sequence Number
Acknowledgement Number
Data offset
Reserved
URG
ACK
PSH
RST
SYN
FIN
Window
Checksum
Urgent Pointer
Options + Padding
20 octets

Port numbers

- 16-bit port numbers- 0 to 65535

- 0 to 1023 are *well-known* ports
  - assigned to common applications
  - telnet uses 23, SMTP 25, HTTP 80 etc.

- 1024 to 49151 are registered ports
  - 6000 through 6063 for X-win server

- 49152 to 65535 are *dynamic* or *private* ports.

More questions ...

- How does the transport layer at S distinguish between packets coming down to it from multiple applications (ex: http and ssh)?
  - Their destination (D) may be the same or different.
- How to determine the number bits to allocate for the sequence number (unique packet id)?
  - Suppose you use 3 bits, the 1st packet and 8th packet will both have [000] as the sequence number.
- How to decide the number of packets that S could transmit before it waits for the 1st acknowledgment?
  - Suppose you transmit one packet, wait for its ack, then the next packet and so on, what is the drawback?

Sequence number size

Sequence number identifies the byte in the stream between sender & receiver; Sequence number wraps around to 0 after reaching $2^{32} – 1$

Should be long enough so that sender does not confuse the sequence numbers on acks

- sending at < 100 packets/sec (R)
- wait for 200 sec before giving up (T)
- receiver may delay up to 100 sec (A)
- packet can be in network up to 300 sec (MSL: Max Segment Lifetime)
- Sender may send 900*100 packets before ack

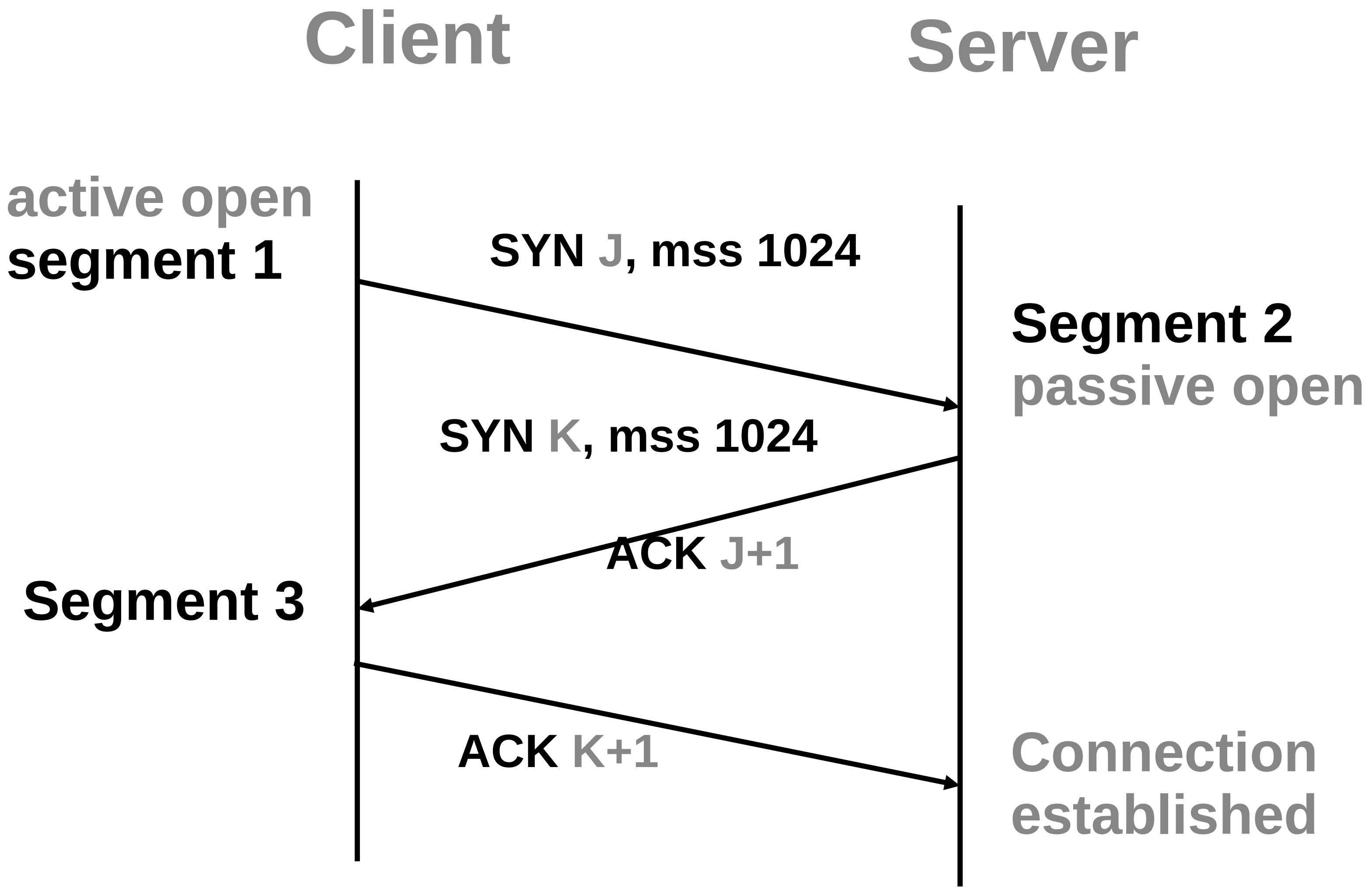
2^seq_size > R (2 MSL + T + A)

Initial sequence number

- Sequence numbers: another reason
  - host A opens connection to B, source port 123, destination port 456
  - Suppose connection terminates, a new connection opens, A and B assign the same port numbers
  - delayed pkt arrives from old connection
- New connection will have different initial sequence number (ISN)

- Tutorial Question: Confirm that Sequence Number Wrap Around Time is around 57 minutes for 10 Mbps Ethernet, while using 32 bits for Sequence Number.

TCP connection establishment



TCP 3-way handshake

1: Client sends SYN segment specifying the port number of *Server* and its initial sequence number (ISN)

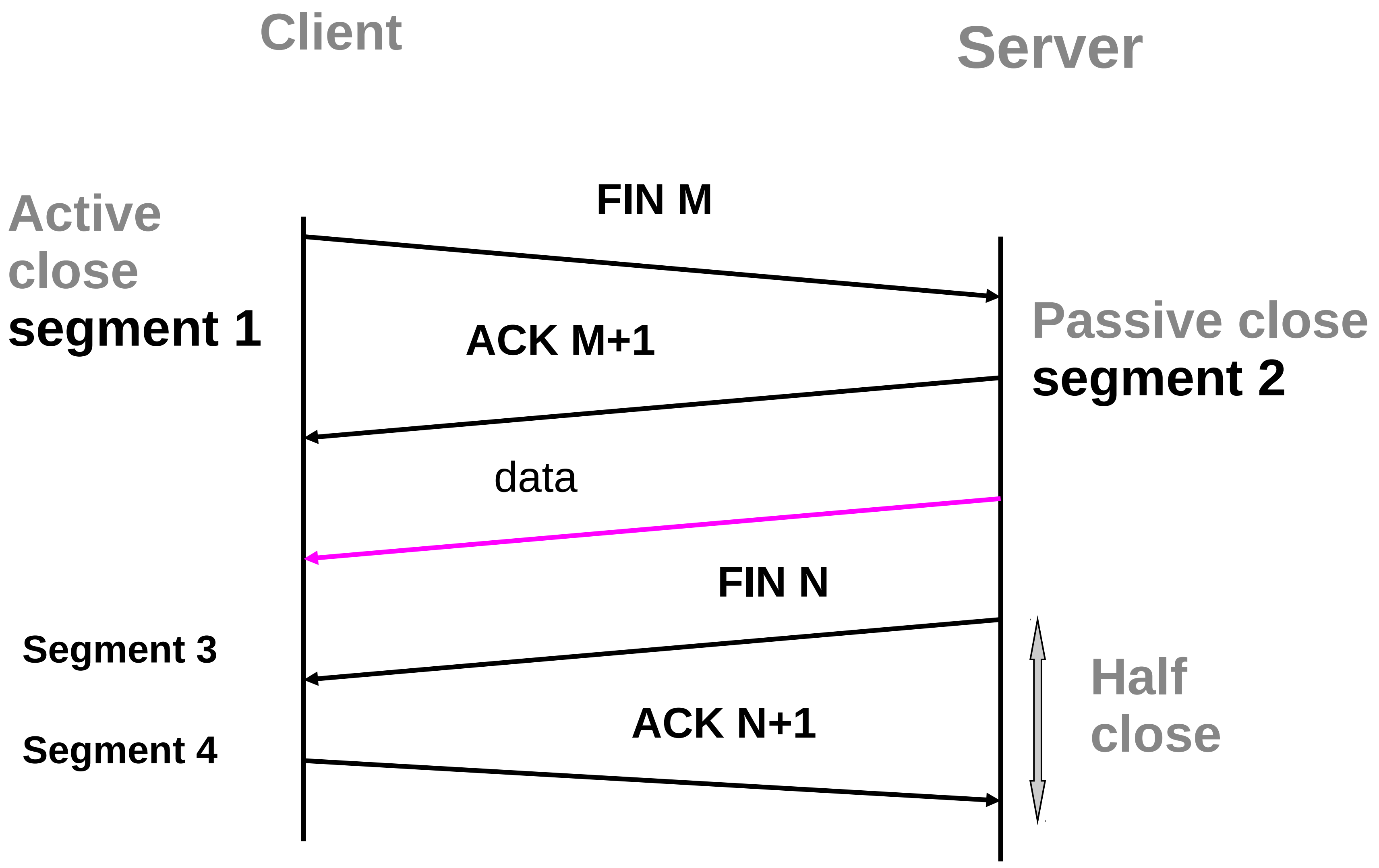
2: Server responds with its own SYN containing its ISN and also acknowledges the client’s SYN

3: client responds to the SYN from the server by ACKing its ISN plus one

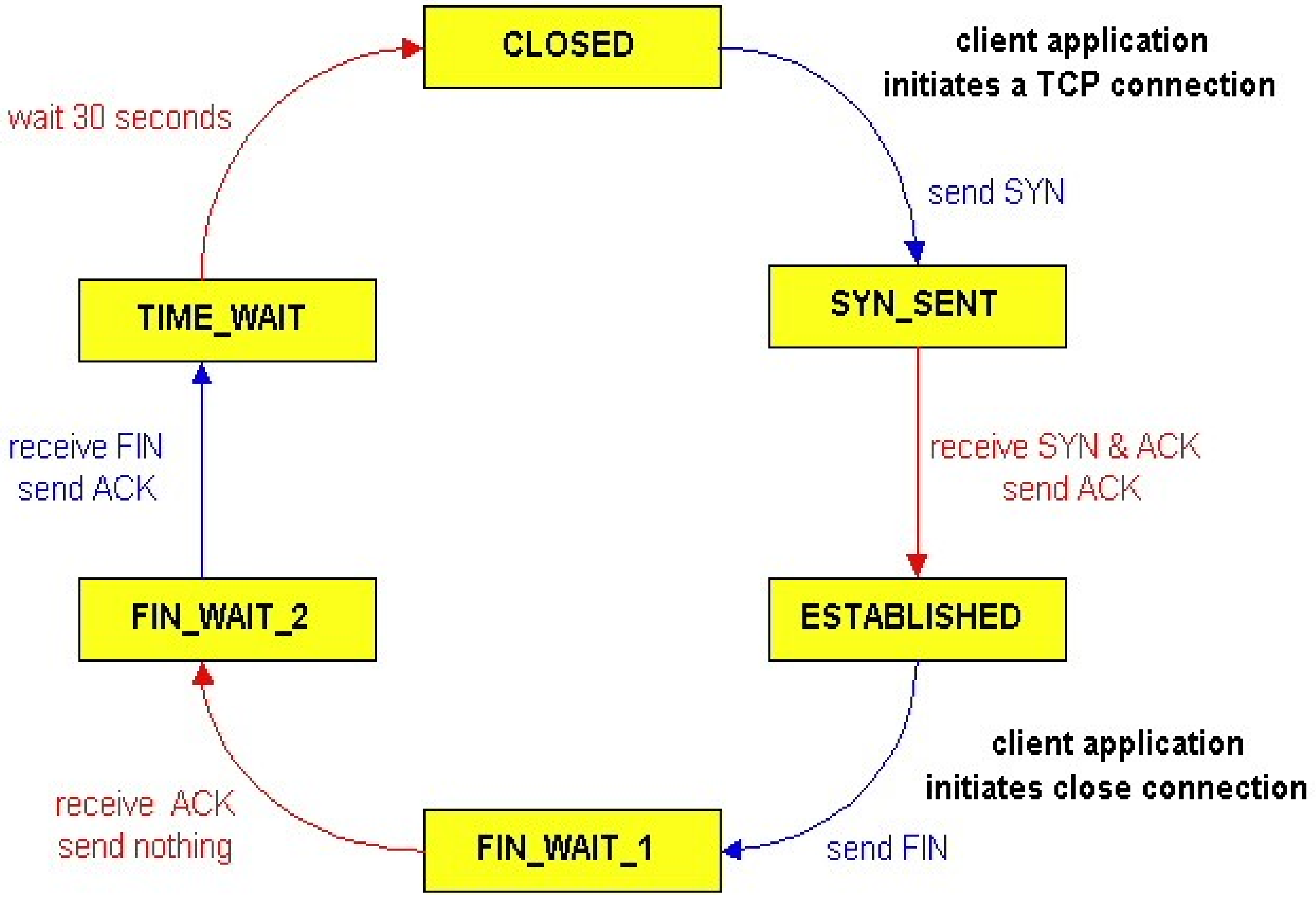
•Why 3-way handshake?
Problem with 2-way handshake is that SYNs themselves are not protected with sequence numbers
3-way handshake protects against delayed SYNs

Wait for 1 MSL (30s to 2 min) upon boot before initiating connection

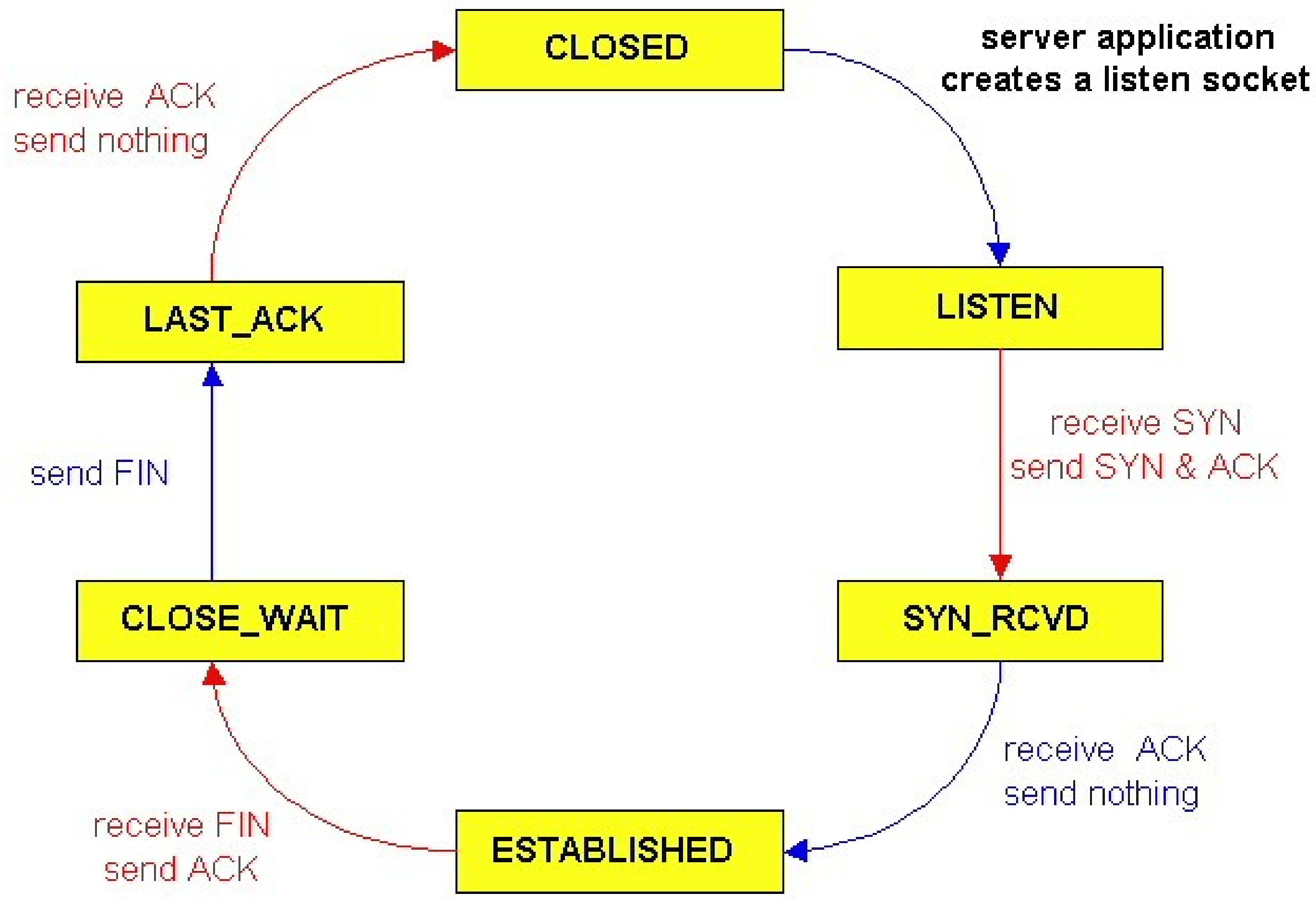
TCP connection termination



TCP client states



TCP server states



Recap: TCP functioning

Socket: [<Source IP address, port no>, <Destination IP address, port no >]

Sending TCP maintains a timer for each segment sent

- waiting for acknowledgement (ACK)
- If ACK doesn’t come in time, segment is retransmitted

Receiving TCP

- Sends ACK
- Re-sequences the data
- Discards duplicates

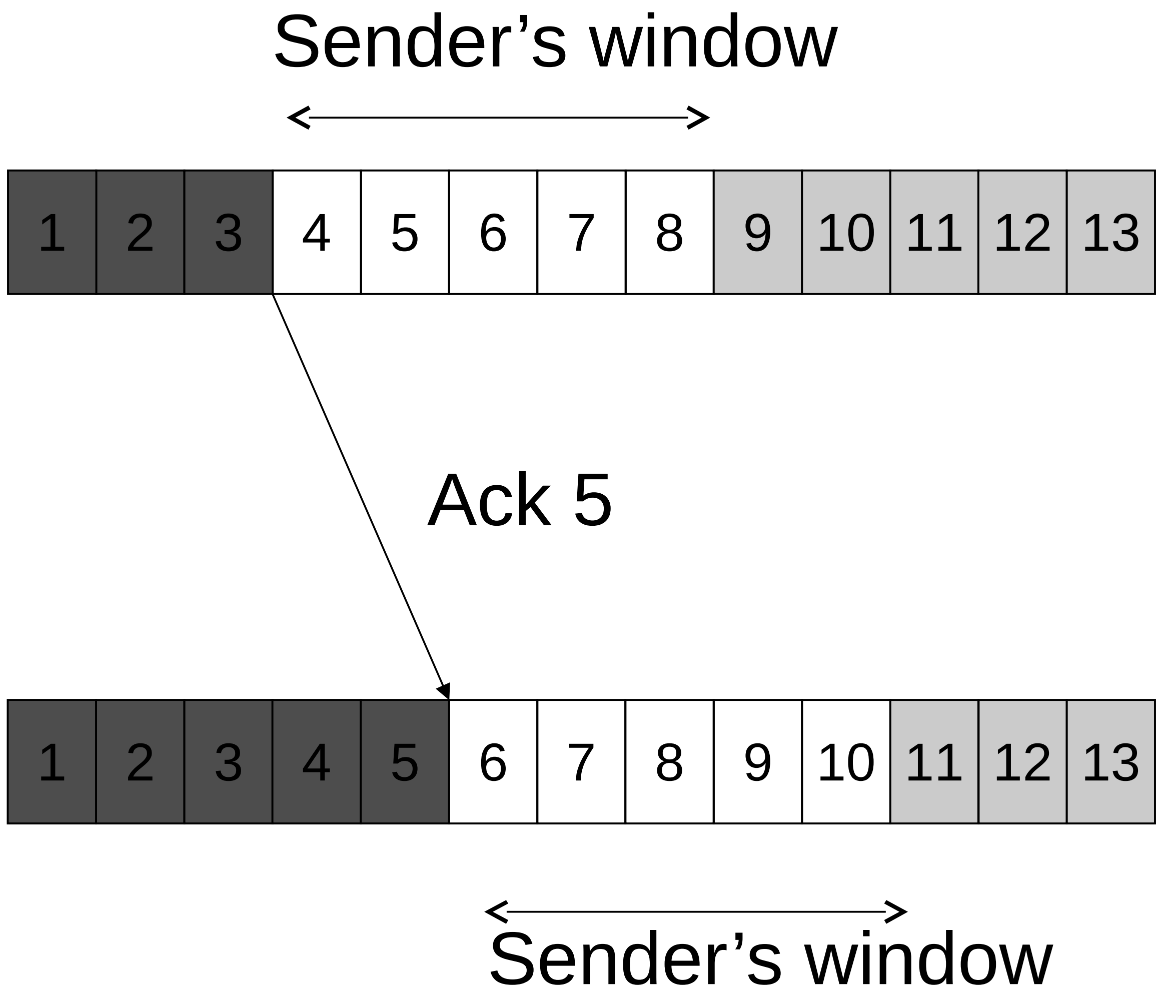
•Sequence number identifies the byte in the stream between sender & receiver; Sequence number wraps around to 0 after reaching $2^{32} – 1$

TCP flow control

- TCP uses the sliding window protocol for flow control
  - Allows sender to transmit multiple segments without waiting for ACKs
  - Sender’s window size is upper limit on un-ACKed segments
  - Similarly, receiver has a window for buffering (not necessarily the same size as the senders’)

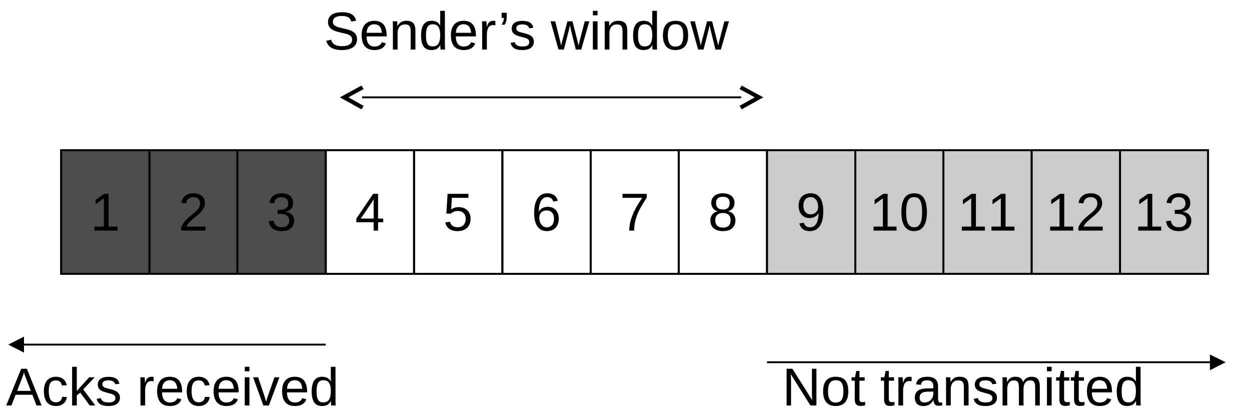
- Window size may grow and shrink
  - Window size controls how much data (bytes), starting with the one specified by the Ack number, that the receiver can accept
  - 16-bit field limits window to 65535 bytes

Window advancement



Window based flow control

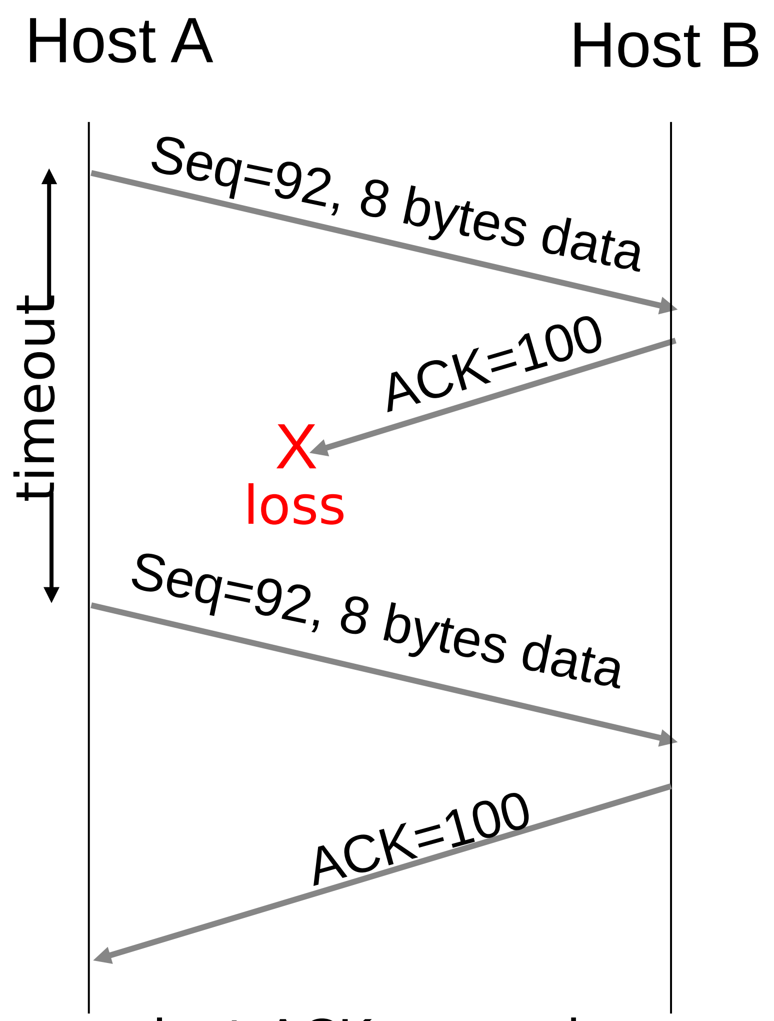
- Window size minimum of
  - receiver’s advertised window - determined by available buffer space at the receiver
  - congestion window - determined by sender, based on network feedback



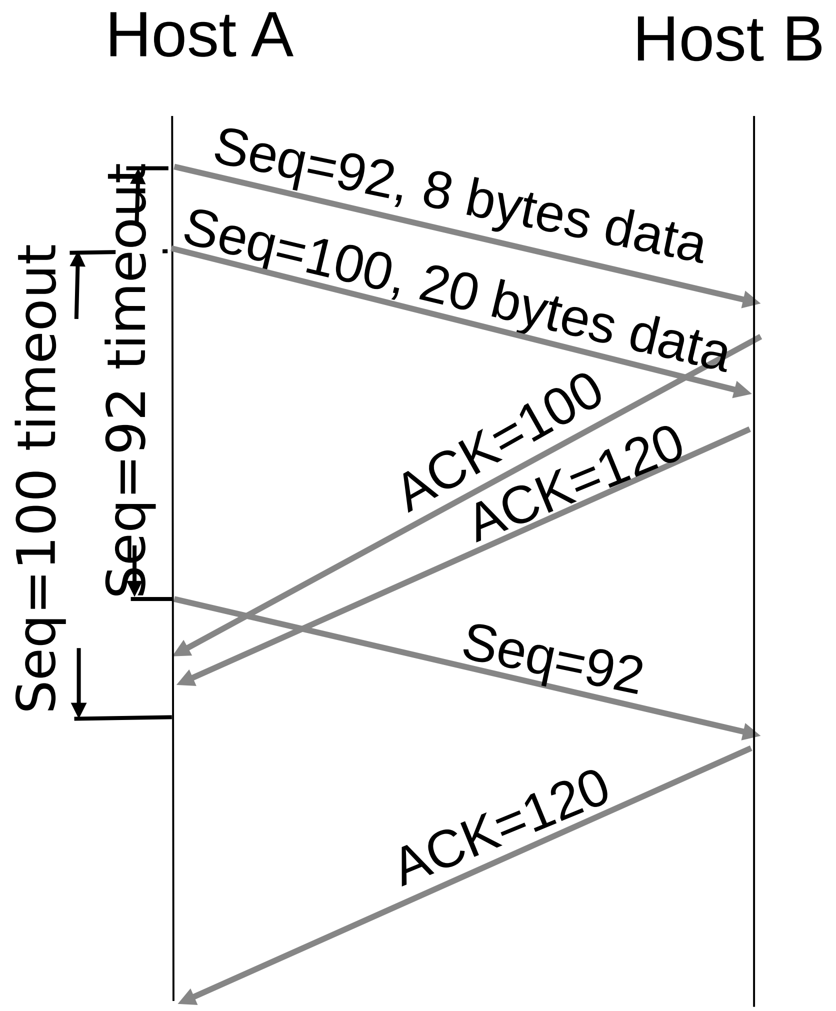
TCP ACK generation

- in-order segment arrival, everything else already ACKed: Delayed ACK (500 ms)
- in-order segment arrival, one delayed ACK pending: Cumulative ACK
- out-of-order segment arrival, higher-than-expect seq no.: Duplicate ACK
- arrival of segment that partially or completely fills gap: Cumulative ACK

TCP: retransmission scenarios



lost ACK scenario



premature timeout,
cumulative ACKs

Activity - Retransmission

- What will happen if we choose too small a value for the value of Retransmission Timeout (RTO)?
- What will happen if we choose too large a value?
- How should we choose an appropriate value of RTO?
  - If you feel that the value of RTO should be adaptive (vary dynamically depending on the “situation”), on what basis should the initial value be chosen? How should the adaptation be performed?
  - When does it make sense to have a fixed value of RTO?

RTT estimation

- Accurate timeout mechanism is important for congestion control
  - Too long: Under-utilization
  - Too short: Wasteful retransmission

- Fixed: Choose a timer interval apriori;
  - useful if system is well understood and variation in packet-service time is small

- Adaptive: Choose interval based on past measurements of RTT

Exponential averaging filter

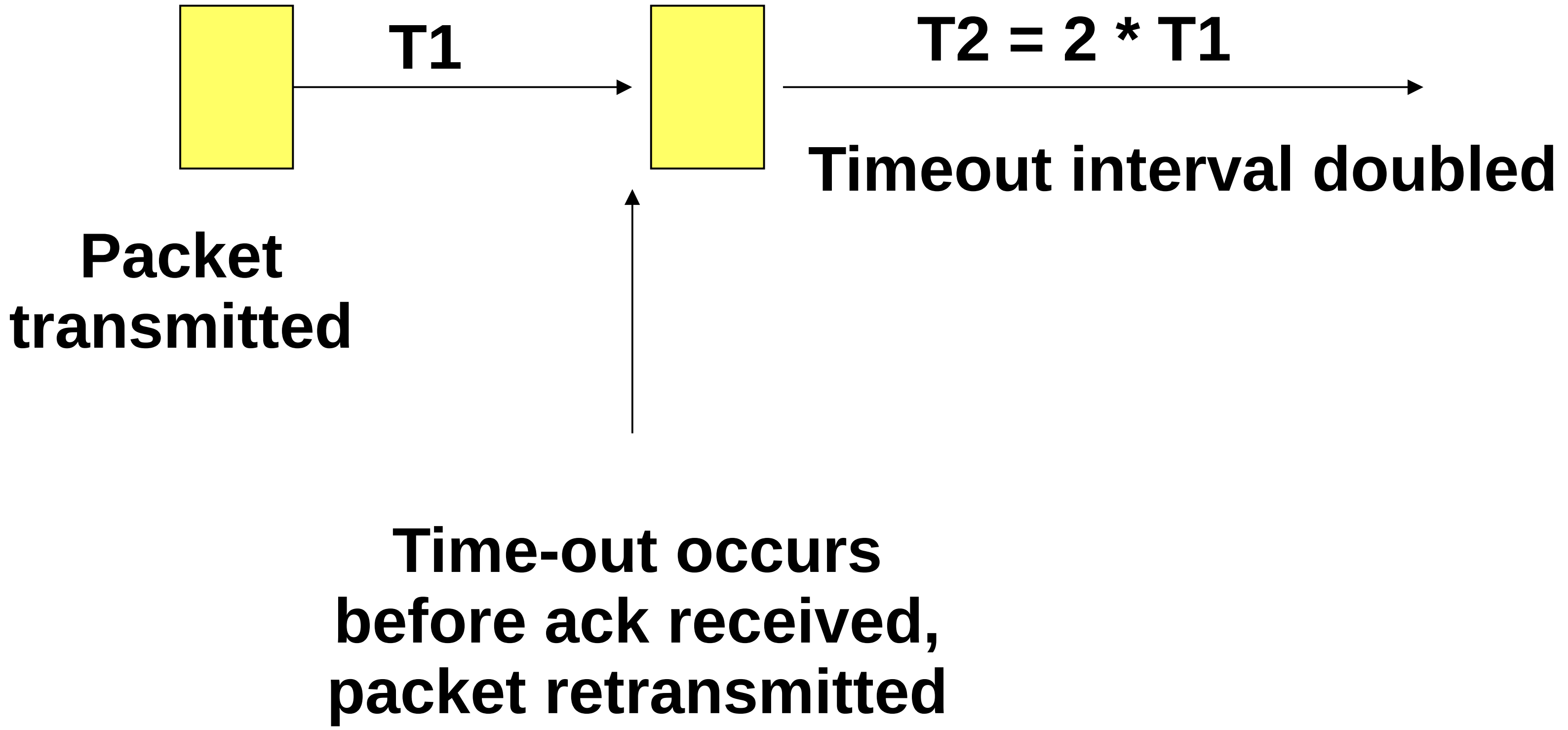
- Measure SampleRTT for segment/ACK pair

- Compute weighted average of RTT
    - EstimatedRTT = α PrevEstimatedRTT + (1 – α) SampleRTT
  - RTO = β * EstimatedRTT

- Typically α = 0.9; β = 2

Exponential backoff

- Double RTO on each timeout
- Reset RTO when timely ACK is received for non-retx segment.



TCP Retransmission

- Default:
  - Cumulative ACKs, Duplicate ACKs
  - go-back-N retransmission ← Recall this?

- Optimization:
  - Selective ACK (SACK)
    - need to specify ranges of bytes received (requires large overhead)
  - Selective retransmission

Window Size

- TCP uses sliding window protocol for efficient transfer
  - Default buffer: 4096 to 16384 bytes
  - Ideal: window (and Rx buffer) = bandwidth-delay product

- AdvertisedWindow = MaxRcvBuffer - (LastByteRcvd – NextByteRead)
  - MaxSendWin = MIN (CongestionWindow, AdvertisedWindow)
  - CongestionWindow limits amount of data in transit
- SendingWindow = MaxSendWin - (LastByteSent – LastByteAcked)

- What happens if the AdvertisedWindow is small (few bytes)?

Silly-Window syndrome

- Sender's window <= AdvertisedWindow of Receiver
- Slow application receiver
  - TCP advertises small windows.
  - Sender sends small segments.

- Solution:
  - Receiver advertises window only if size is MSS or half the buffer.
  - Sender typically sends data only if a full segment can be sent.

What happens if AdvertisedWindow = 0?

Activity - Congestion control

- On detecting a packet loss, TCP sender assumes that network congestion has occurred and reduces the CongestionWindow, which in turn reduces amount of data that can be sent per RTT.

- How should we choose value of CongestionWindow?
  - Given that it should be adaptive, what should be its initial value?
  - How should the adaptation (increase/decrease) be performed?

AIMD: Additive increase Multiplicative decrease

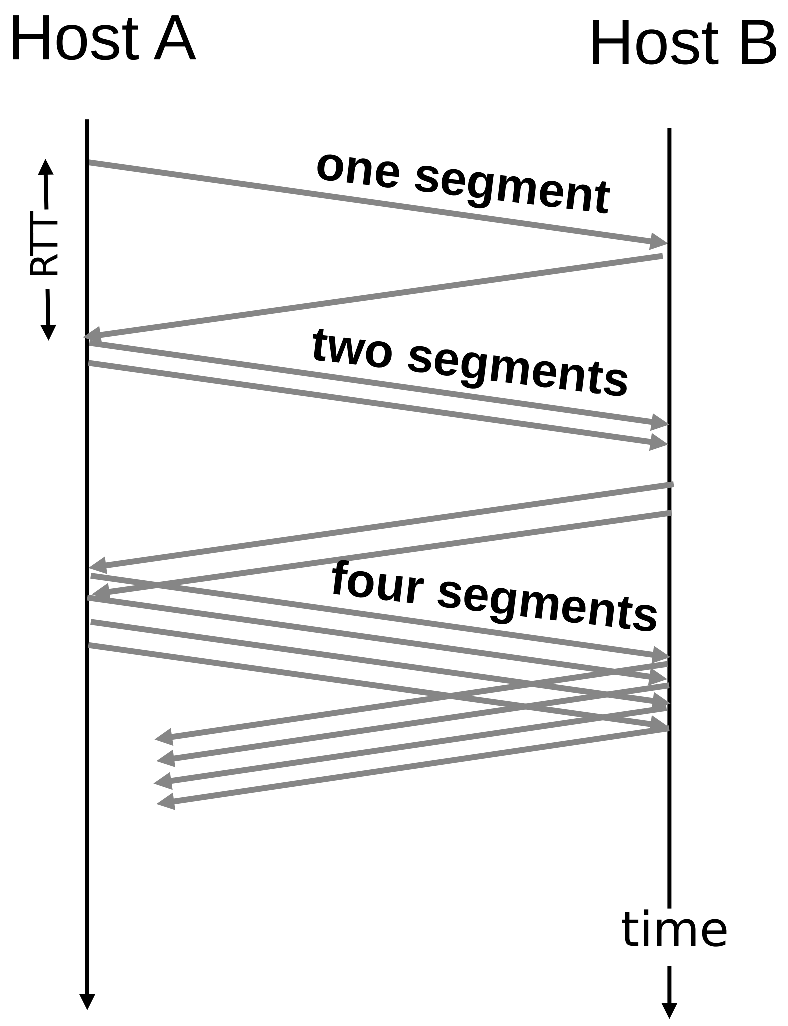
- Source infers congestion upon RTO.
- Increase CongestionWindow (linearly, by 1 per RTT) when congestion goes down.
- Decrease CongestionWindow (multiplicatively, by factor of 2) when congestion goes up.
-
- Provides fair sharing of links

Slow start and Congestion avoidance

- AIMD may be too conservative
- CongestionWindow: cwnd
- Slow Start
  - Increase cwnd exponentially upto a threshold (ssthresh)
- Congestion Avoidance
  - Increase cwnd linearly after ssthresh

Slow start phase

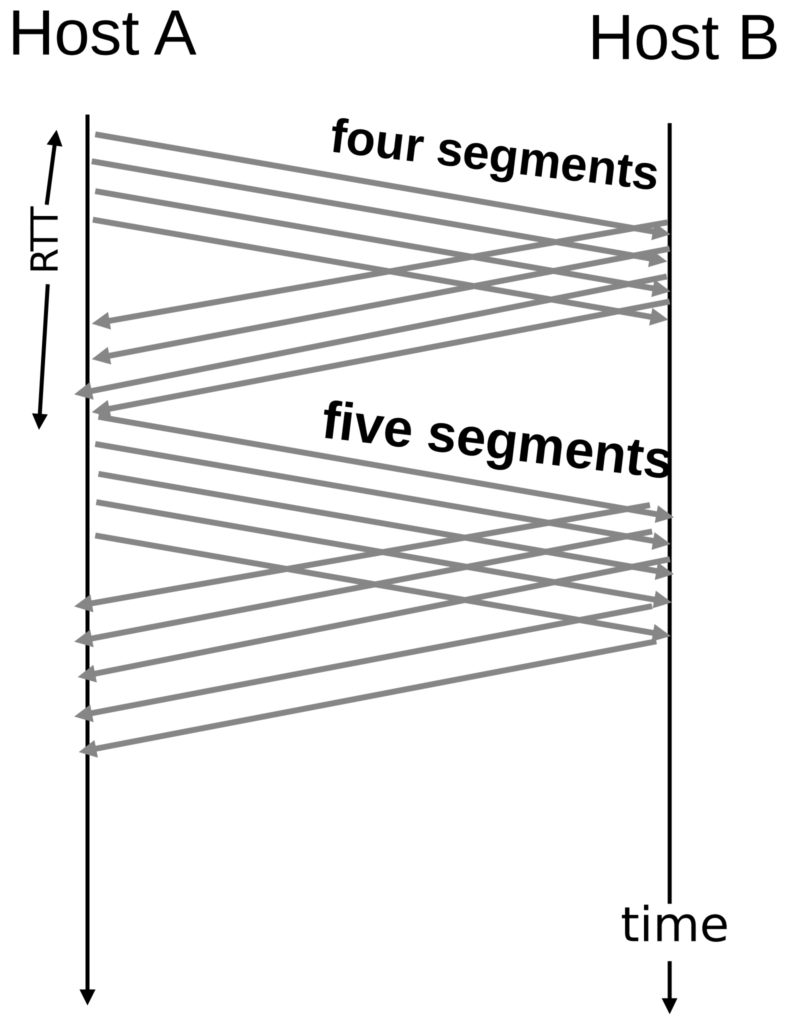
- **initialize:**
  - **Cwnd = 1**
- **for (each ACK)**
  - **Cwnd++**
- **until**
  - **loss detection OR**
  - **Cwnd > ssthresh**



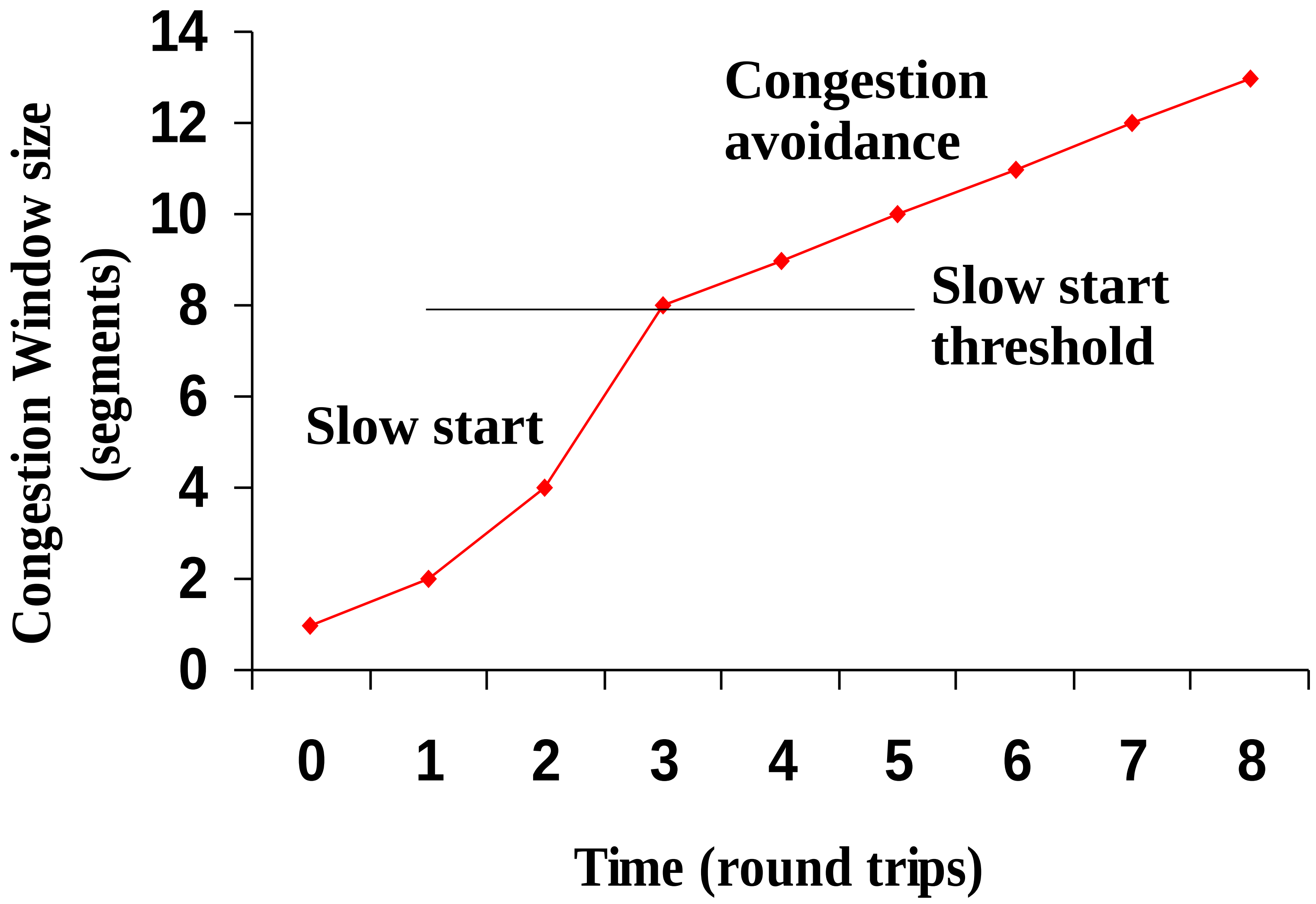
Congestion avoidance phase

```
/* Cwnd > threshold */
• Until (loss detection)  {
  every w ACKs:
      Cwnd++
  }
• ssthresh = Cwnd/2
• Cwnd = 1
• perform slow start
```

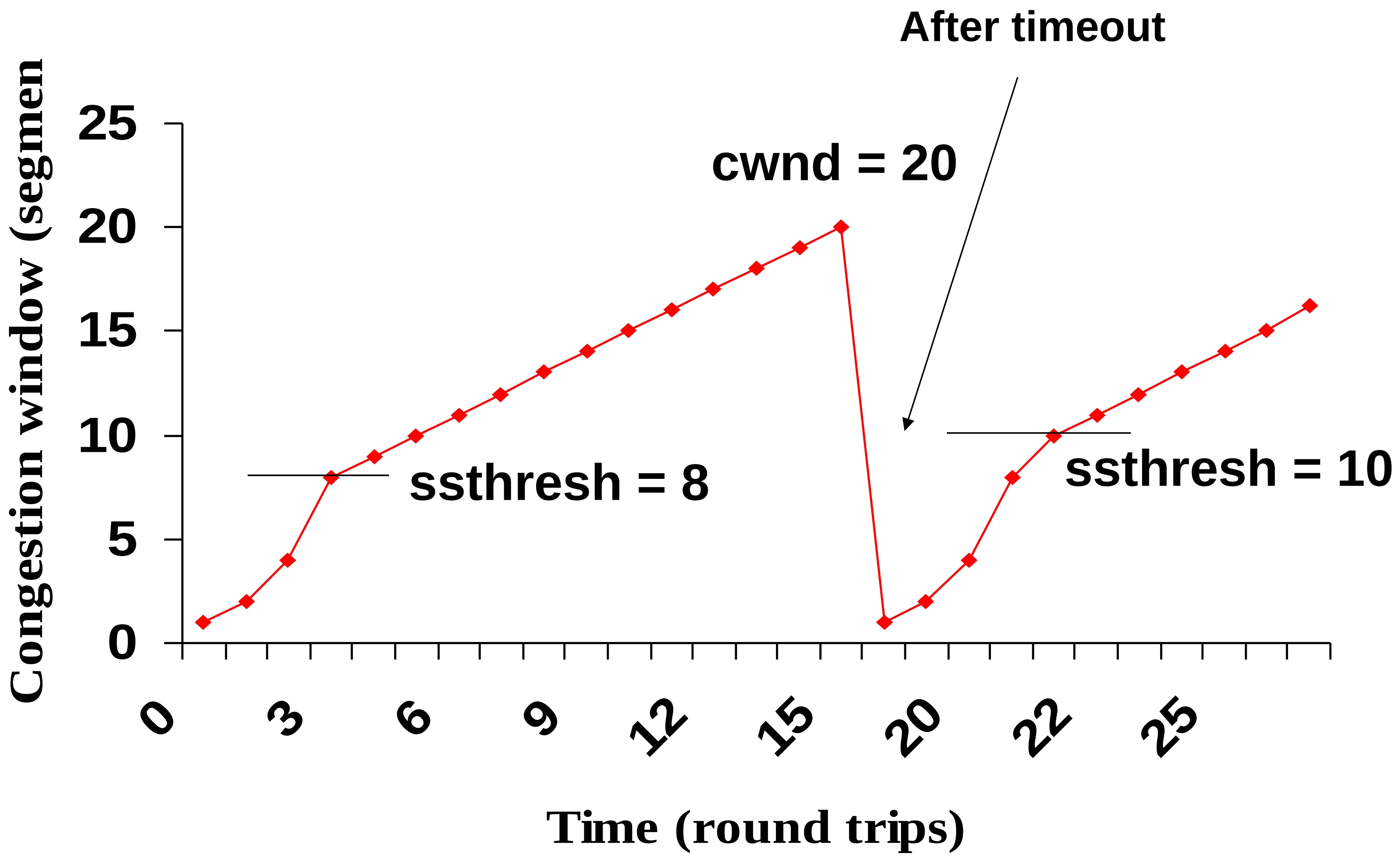
1



Typical TCP behaviour



Congestion control: Timeout

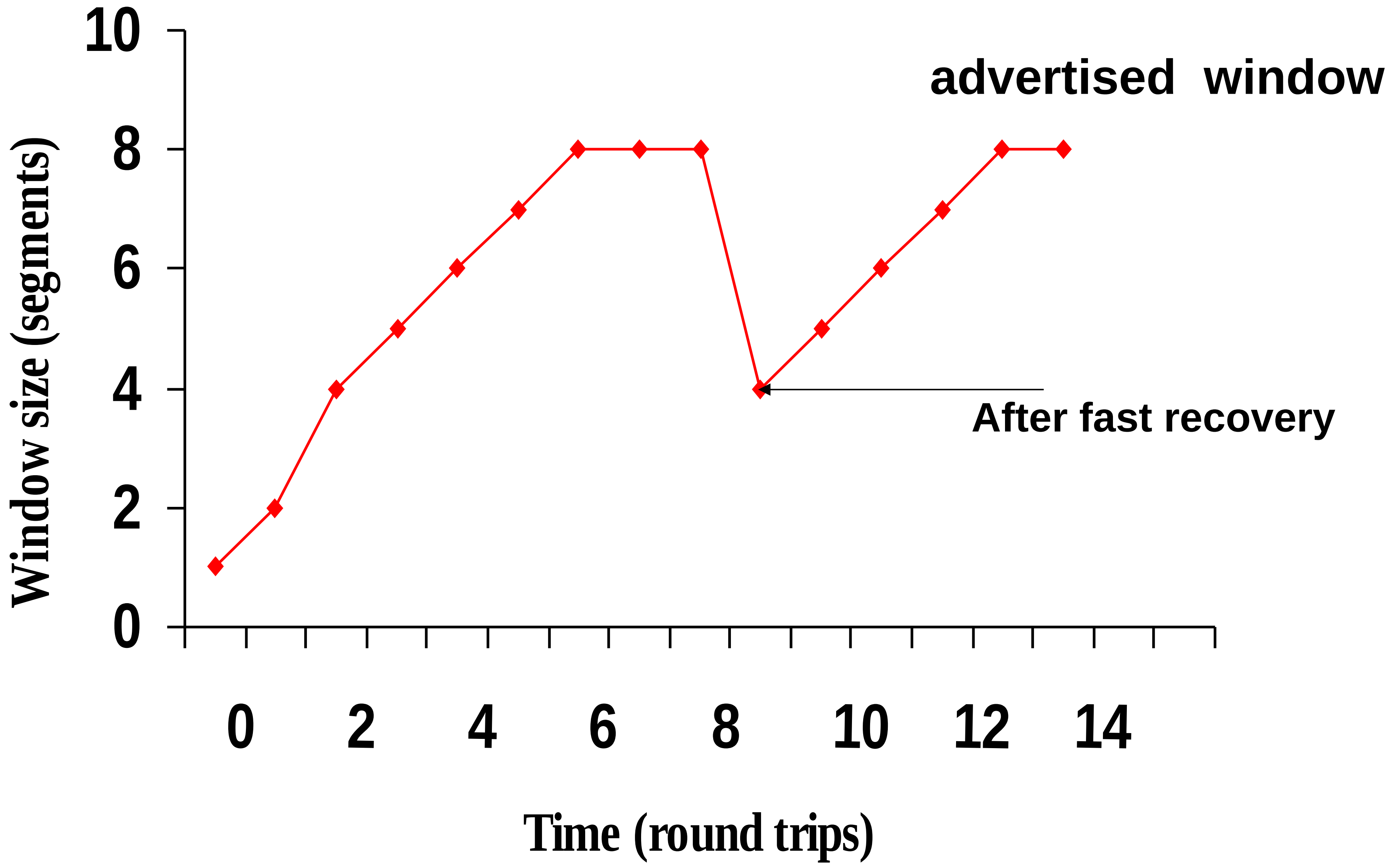


Fast retransmit and Fast recovery

- Slow start follows timeout
  - timeout occurs when no more packets are getting across.
- IF one packet got dropped but others got through?

- Fast retransmit follows duplicate ACKs
  - multiple (>= 3) dupacks come back when a packet is lost, but latter packets get through.
- Fast recovery follows fast retransmit
- ssthresh = min(cwnd, advertised window)/2
- New cwnd = ssthresh

Congestion control: Fast retransmit and Fast recovery



TCP Tahoe

- Detects congestion using timeouts
- Initialization
  - cwnd initialized to 1;
  - ssthresh initialized to 1/2 MaxWin
- Upon timeout
  - ssthresh = 1/2 cwnd, cwnd = 1
  - enter slow start

TCP Reno

- Detects congestion loss using timeouts as well as duplicate ACKs
- On timeout, TCP Reno behaves same as TCP Tahoe
- On fast retransmit
  - skips slow start and goes directly into congestion avoidance phase
  - ssthresh = 1/2 cwnd; cwnd = ssthresh

Self study: UDP (User Datagram Protocol)

- Datagram oriented Internet transport
- “best effort” service; doesn't guarantee any reliability. UDP segments may be:
  - lost
  - delivered out of order to application
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

  - What applications is UDP transport layer suitable for?
  -

Closure

- TCP animations
  - http://oscar.iitb.ac.in/onsiteDocumentsDirectory/tcp/tcp/index.html
  - http://www.net-seal.net/animations.php
- Tutorial Questions:
  - What are the pros and cons of TCP versus UDP?
  - Plot graph of TCP Reno sender when every 7th packet is delivered out-of-sequence.
- Topics NOT covered
  - TCP Optimizations: SACK, ECN, ...
  - Throughput analysis, QoS guarantees, ...