CS 716: Introduction to communication networks

- 19th class; 12th Oct 2011

Instructor: Sridhar Iyer
IIT Bombay

Recap of last class: TCP basics

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

Byte-stream based protocol

- Ignores message boundaries. ACK indicates next byte expected.

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

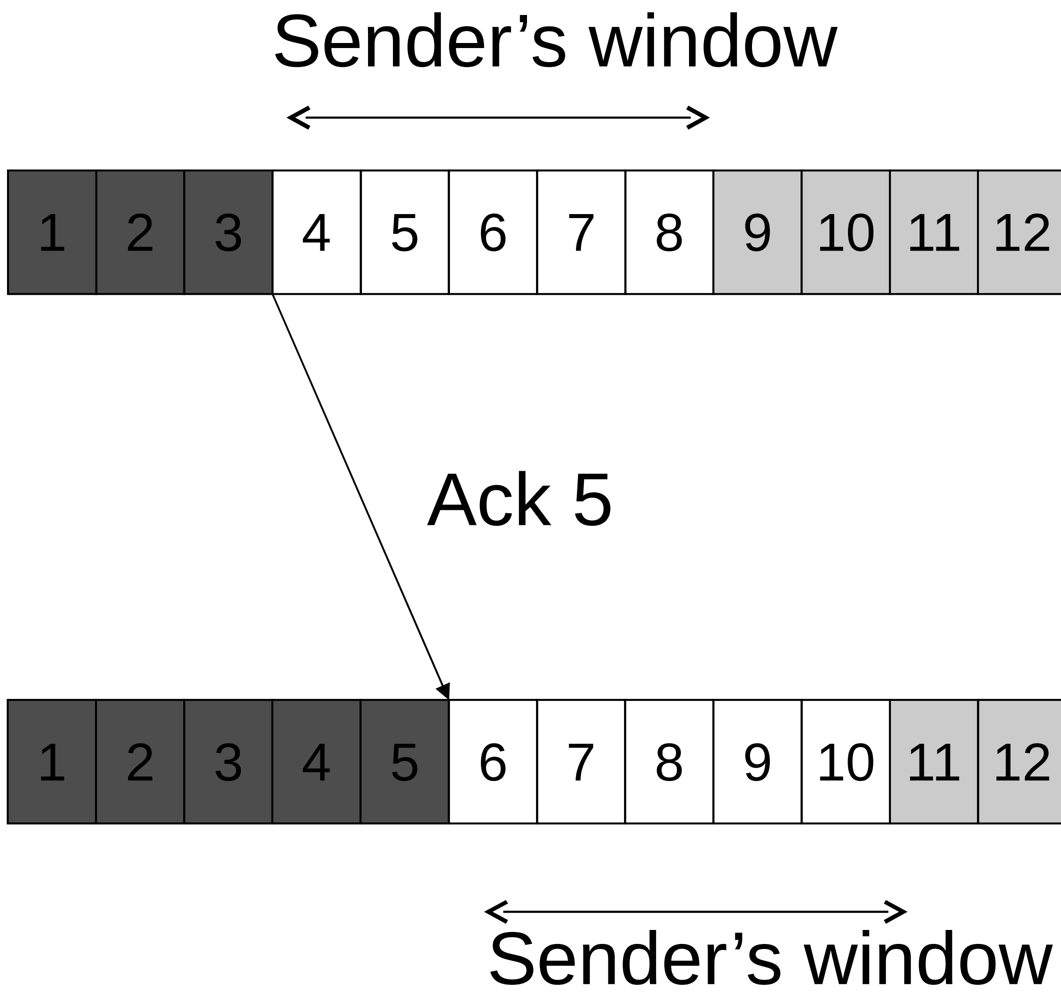
- Sends ACK
- Re-sequences the data
- Discards duplicates

•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

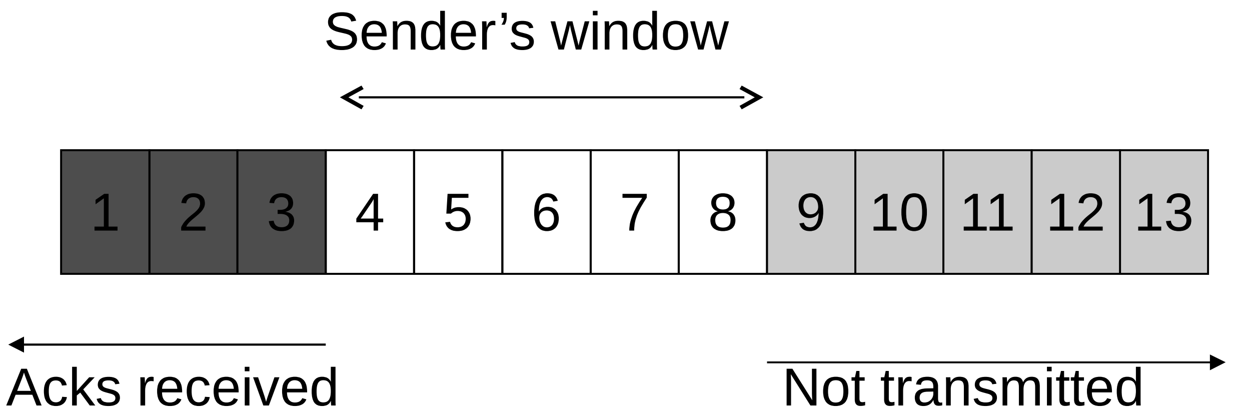
- TCP uses the sliding window protocol for flow control
  - Allows sender to transmit multiple segments without waiting for ACKs
  - Sender's window size is upper limit on un-ACKed segments
  - Similarly, receiver has a window for buffering (not necessarily the same size as the senders')
- Window size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

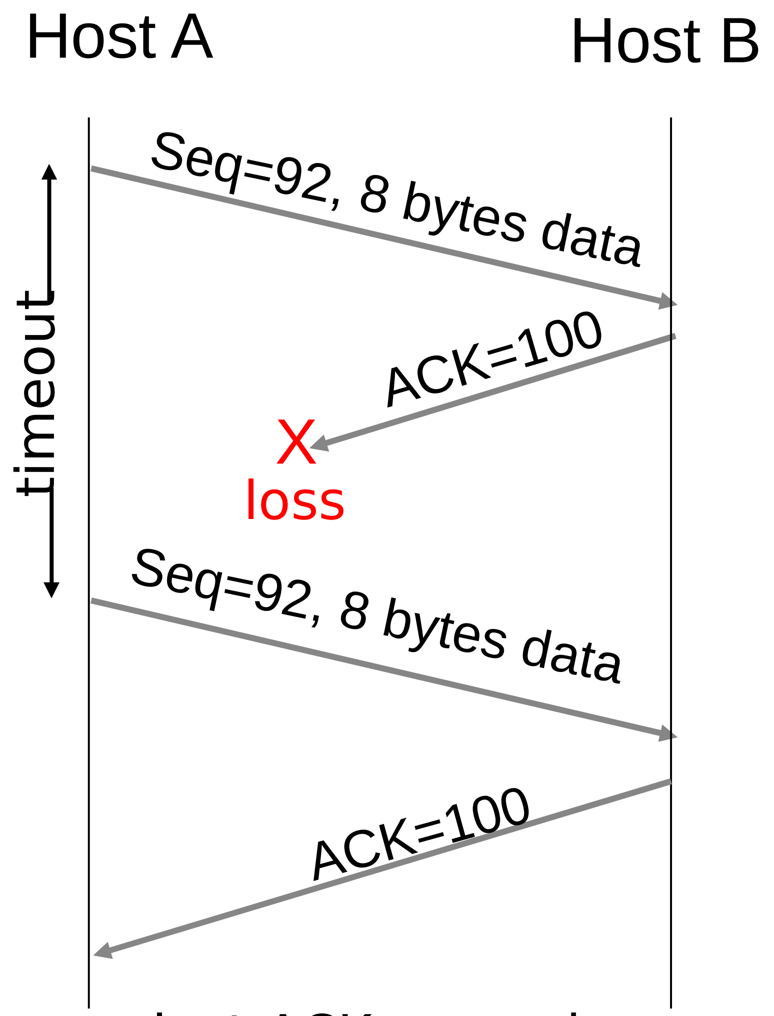
- Window size minimum of
  - receiver's advertised window - determined by available buffer space at the receiver
  - congestion window - determined by sender, based on network feedback



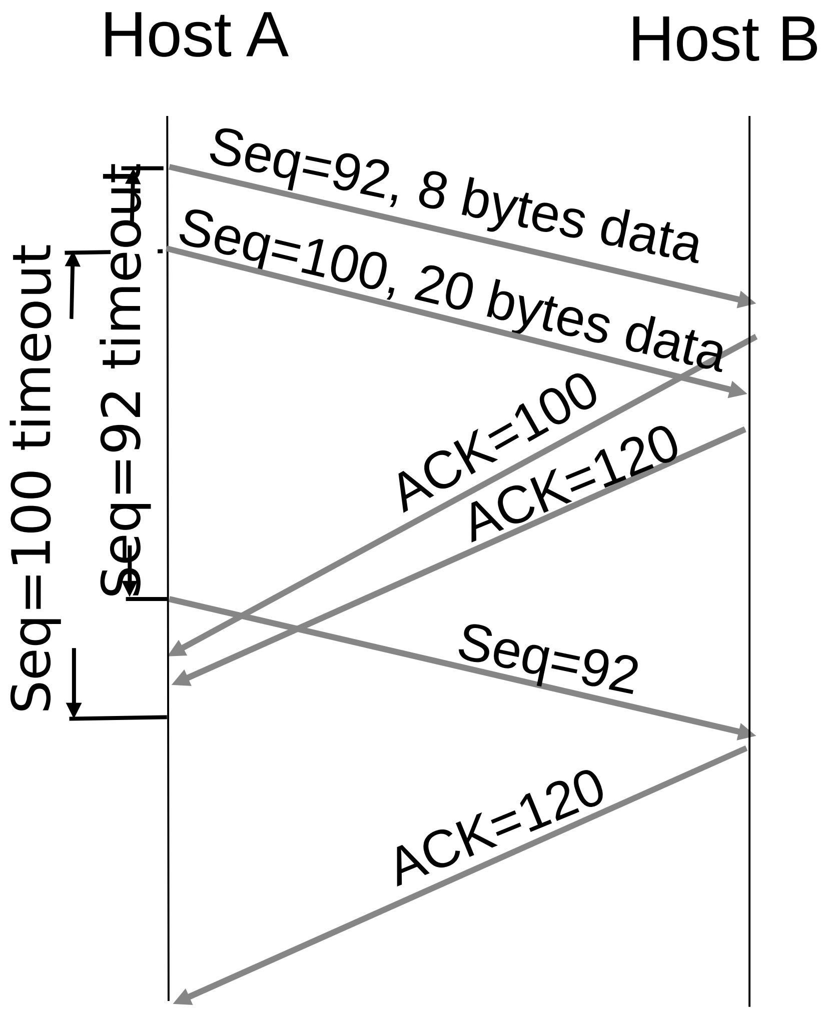
TCP ACK generation

- 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

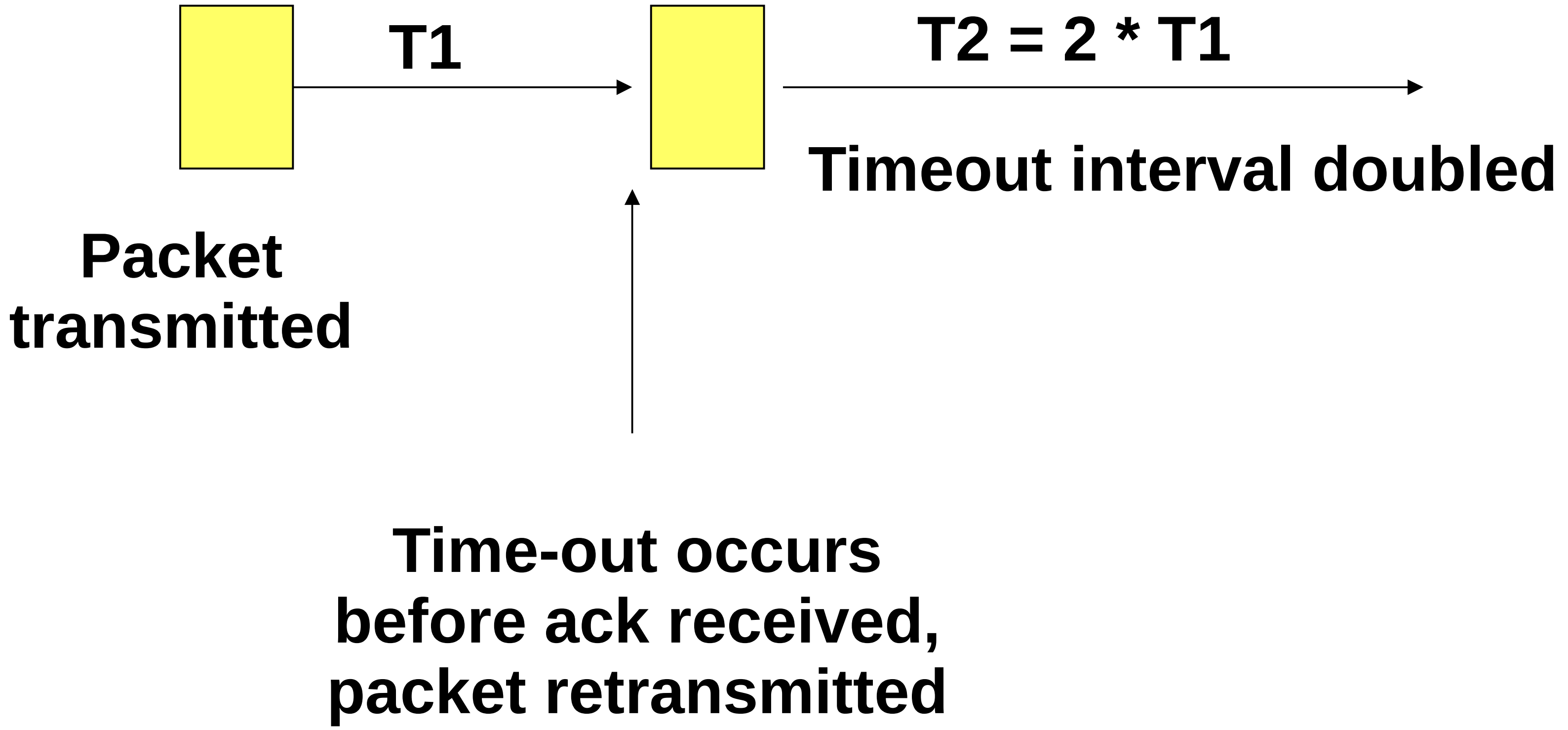
- Accurate timeout mechanism is important for congestion control
  - Too long: Under-utilization
  - Too short: Wasteful retransmission
- Fixed: Choose a timer interval apriori;
  - useful if system is well understood and variation in packet-service time is small
- Adaptive: Choose interval based on past measurements of RTT

Exponential averaging filter

- Measure SampleRTT for segment/ACK pair
- Compute weighted average of RTT
    - EstimatedRTT = α PrevEstimatedRTT + $(1 - \alpha)$ SampleRTT
  - RTO = β * EstimatedRTT
- Typically $\alpha = 0.9$; $\beta = 2$

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

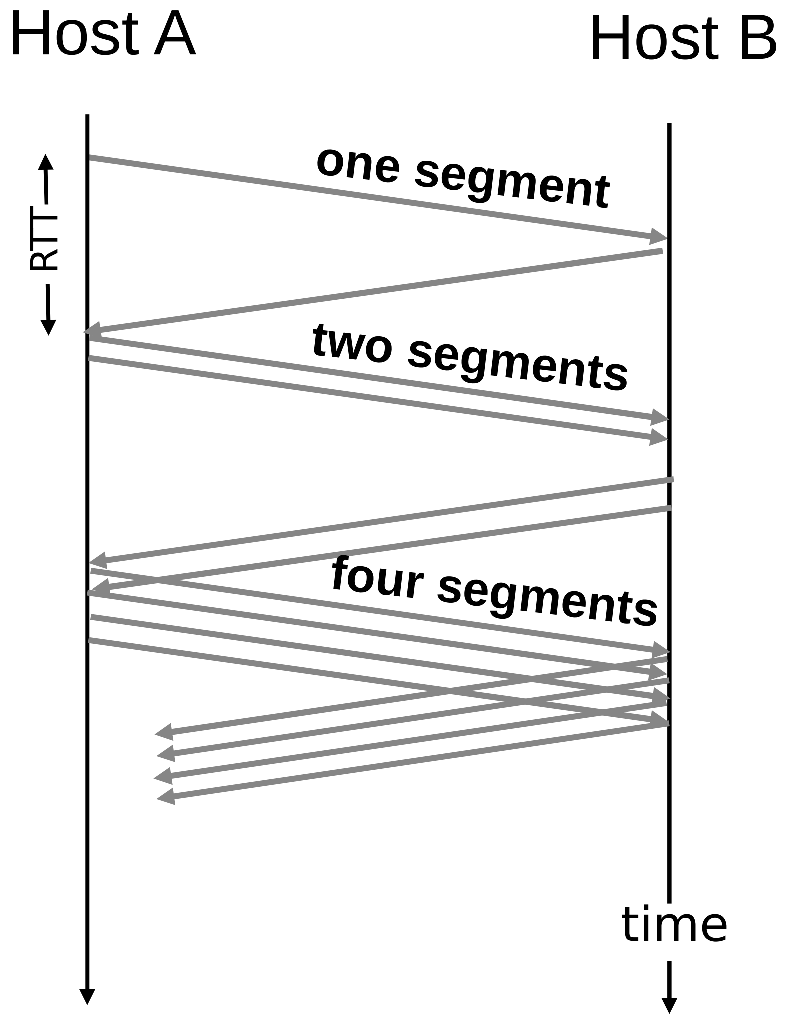
- CongestionWindow: cwnd

- 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**

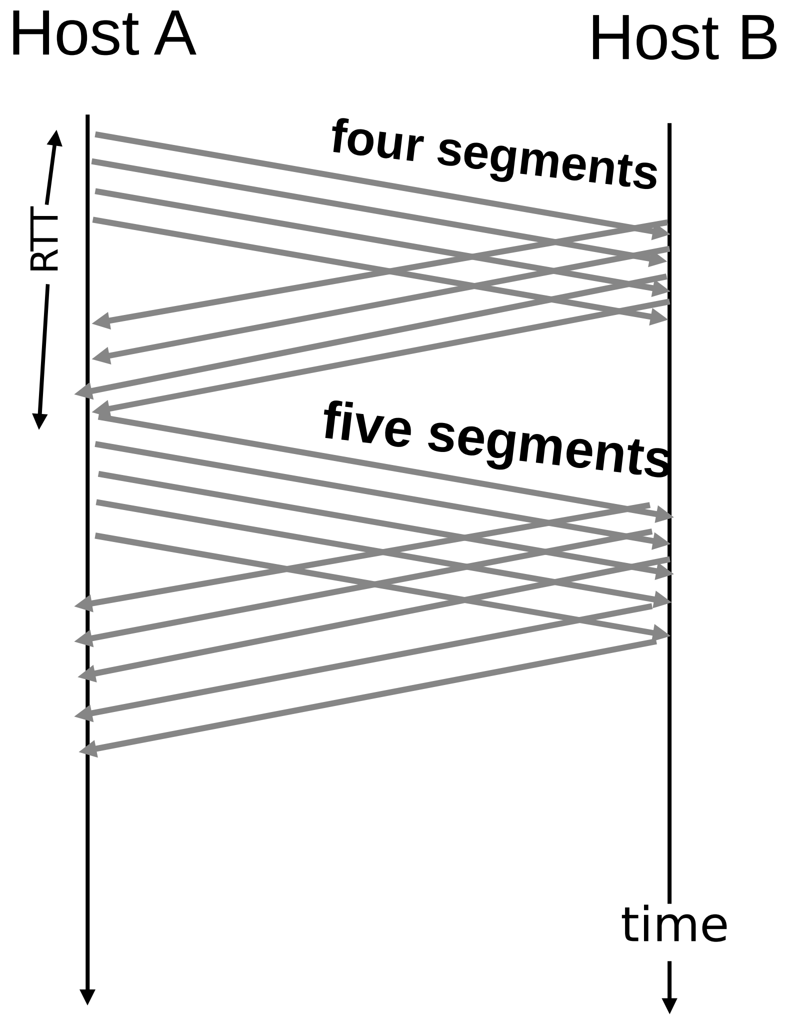


Congestion avoidance phase

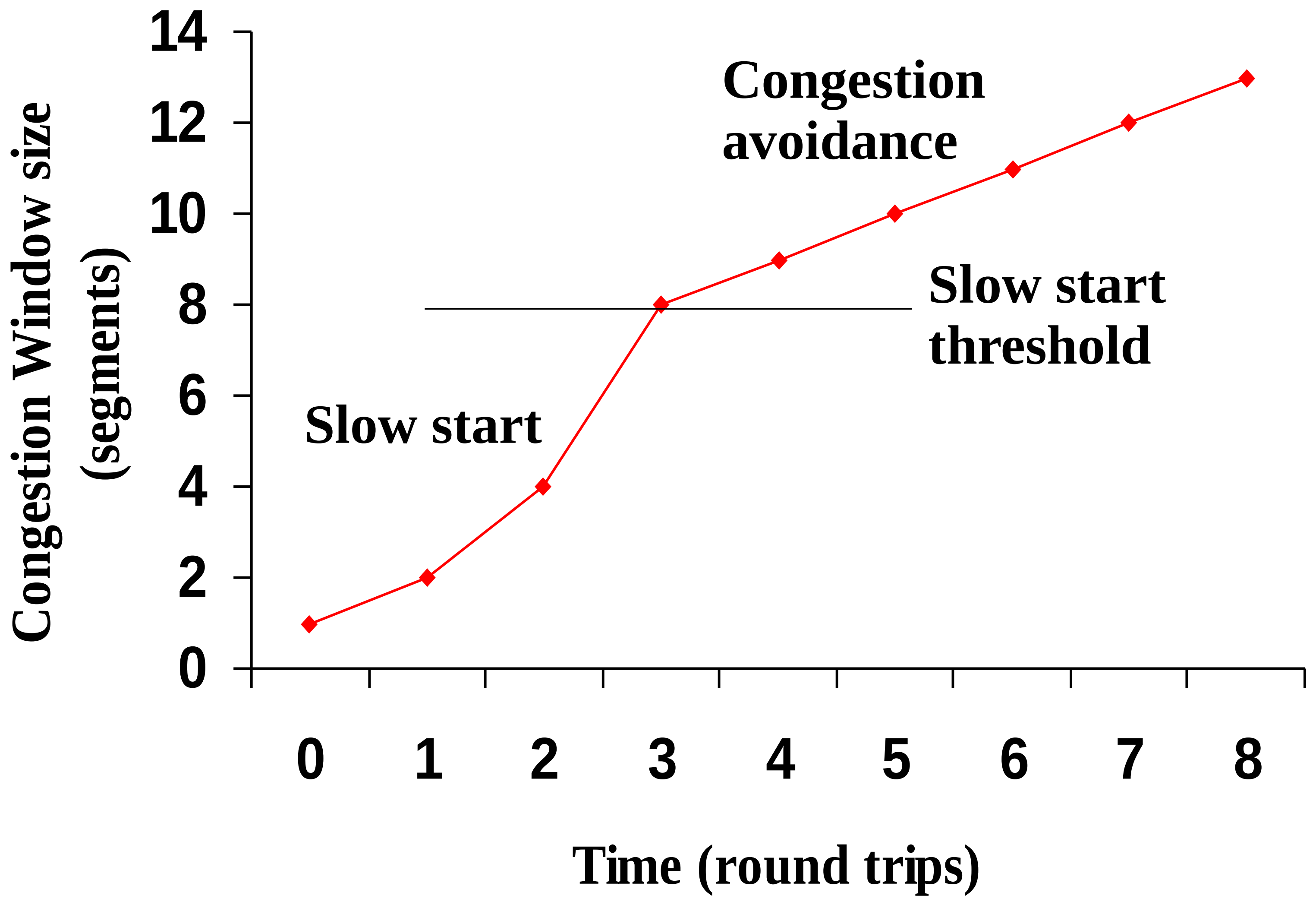
/* Cwnd > threshold */

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

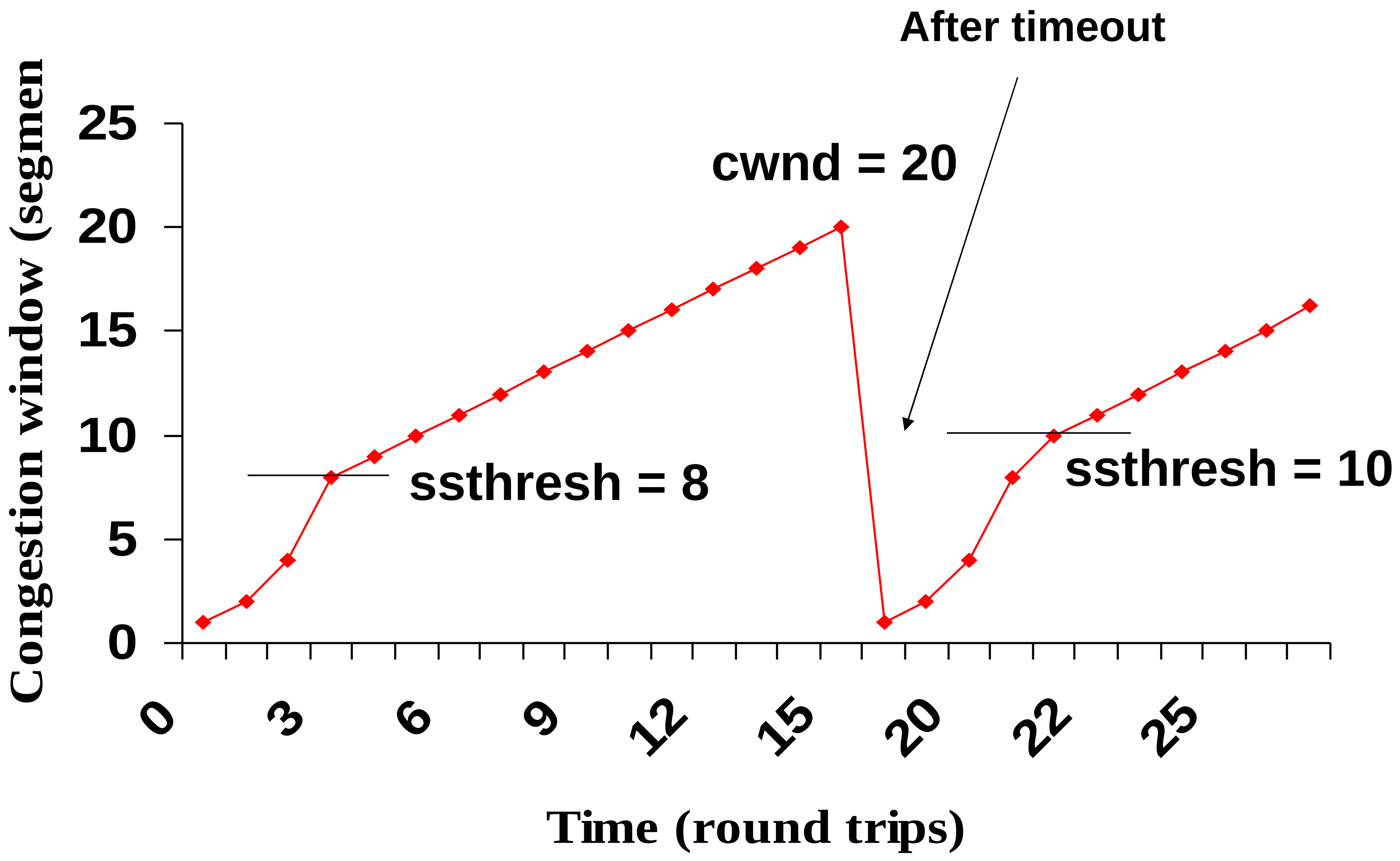
1



Typical TCP behaviour



Congestion control: Timeout

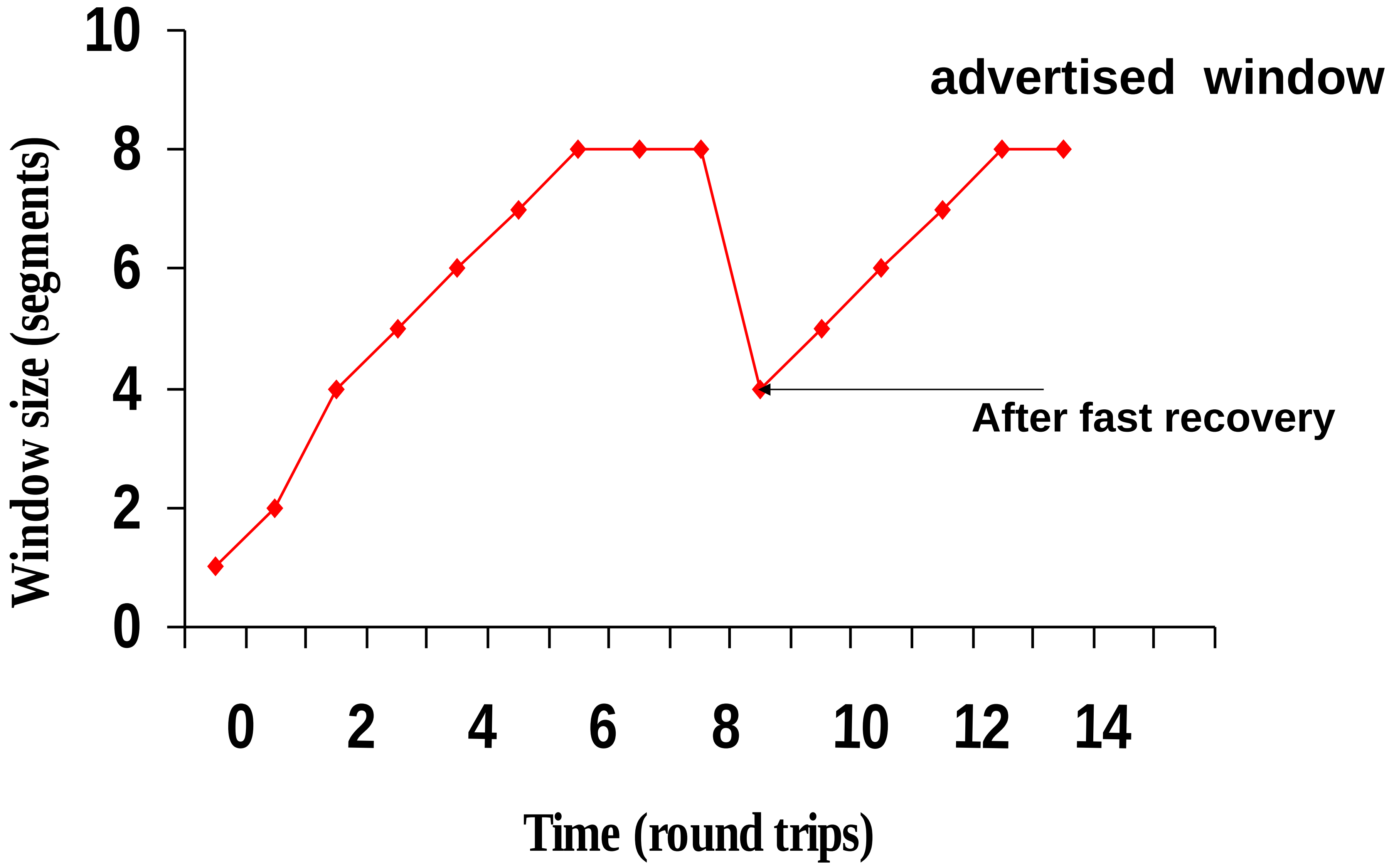


Fast retransmit and Fast recovery

- 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, ...