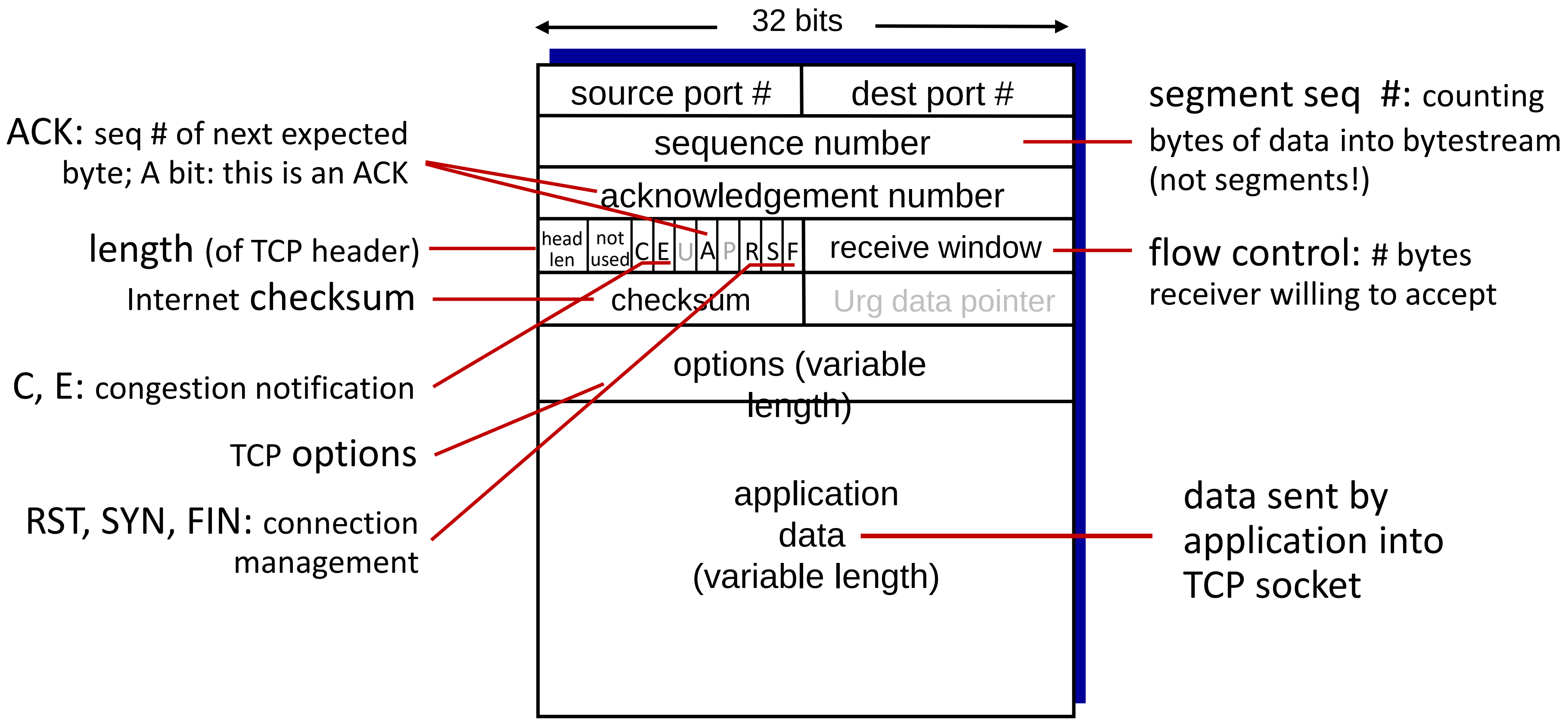
Chapter 3: roadmap

- 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 segment structure



MSS

- Maximum Segment Size
- The default TCP Maximum Segment Size is 536
- For most computer users, the MSS option is established by the operating system
- The maximum segment size is specified as a TCP option, initially in the TCP SYN packet during the TCP handshake. The value cannot be changed after the connection is established

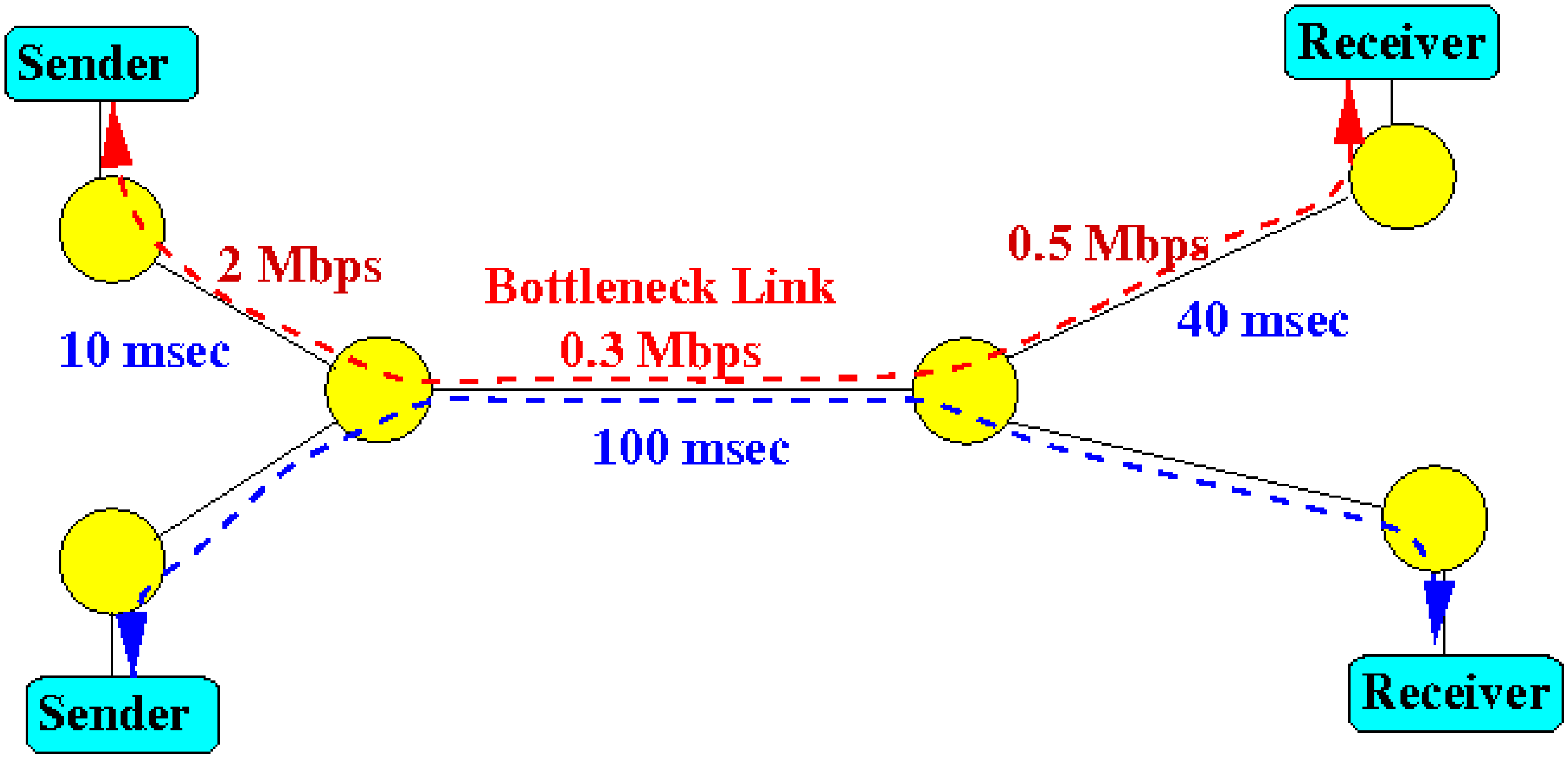
MSS

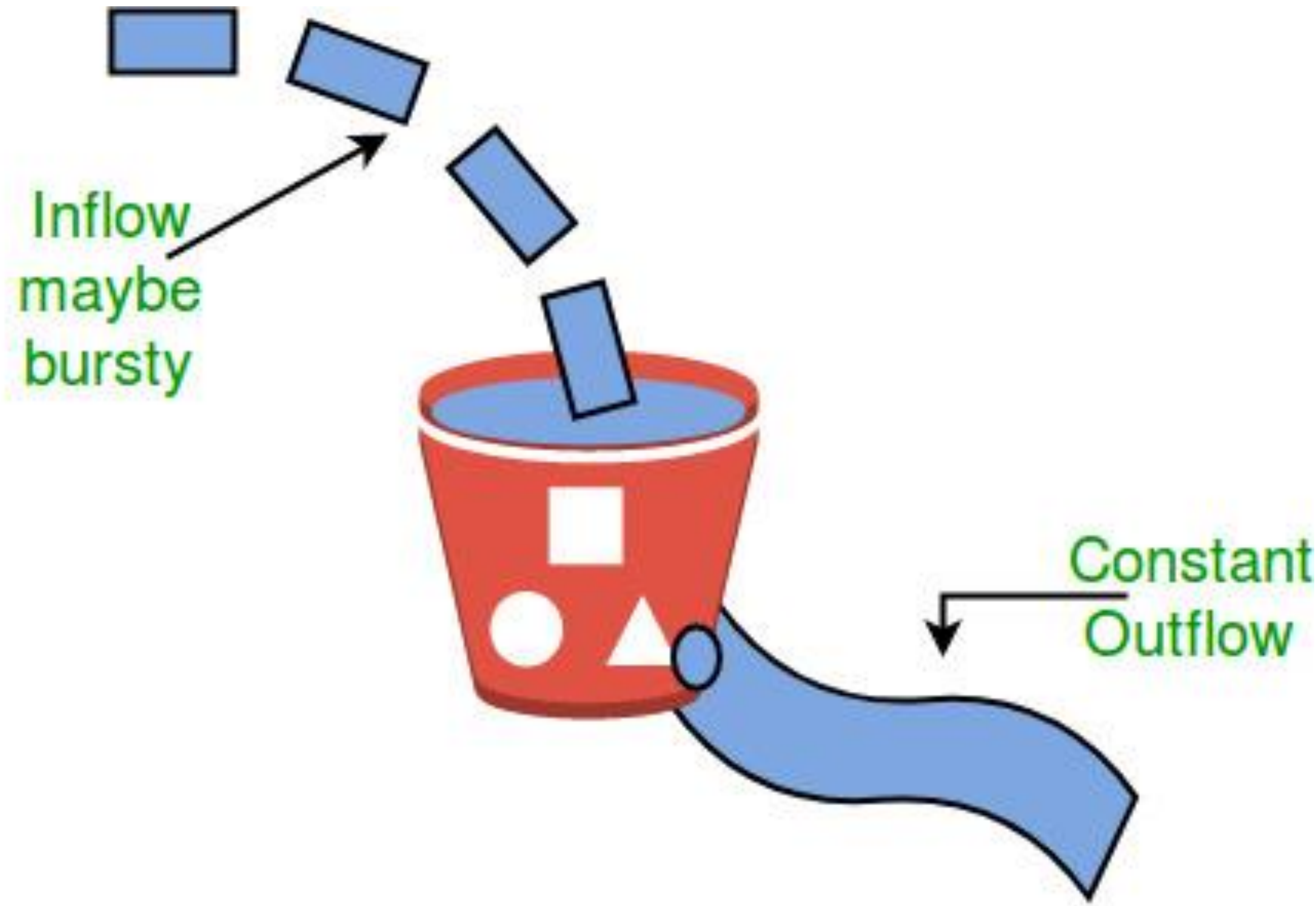
- To avoid fragmentation in the IP layer, a host must specify the maximum segment size as equal to the largest IP datagram that the host can handle minus the IP header size and TCP header sizes
- Therefore IPv4 hosts are required to be able to handle an MSS of 536 octets (= 576 - 20 - 20)

The TCP Maximum Segment Size and Related Topics (RFC 879, 1983)

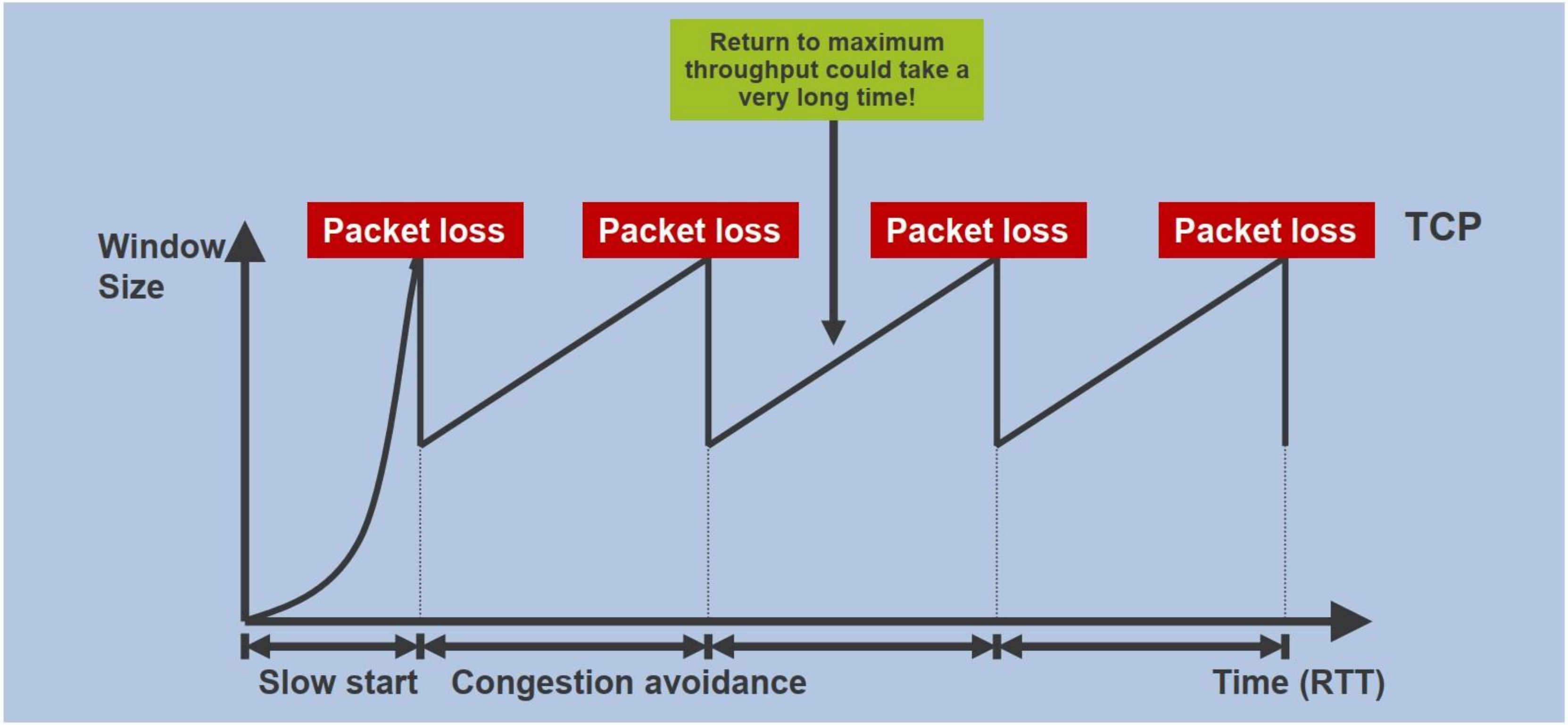
The default IP Maximum Datagram Size is 576

The default TCP Maximum Segment Size is 536


Sender
Sender
Receiver
Receiver
2 Mbps
10 msec
Bottleneck Link
0.3 Mbps
100 msec
0.5 Mbps
40 msec


Inflow
maybe
bursty
Constant
Outflow

TCP Behavior



RFC1323 - TCP Extensions for High Performance

TCP congestion control:
additive increase, multiplicative decrease

- *AIMD*
- *Approach:* increase transmission rate (***congestion window size***), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **cwnd** by 1 MSS every RTT until loss detected
  - *multiplicative decrease*: cut **cwnd** in half after loss

TCP congestion control: AIMD

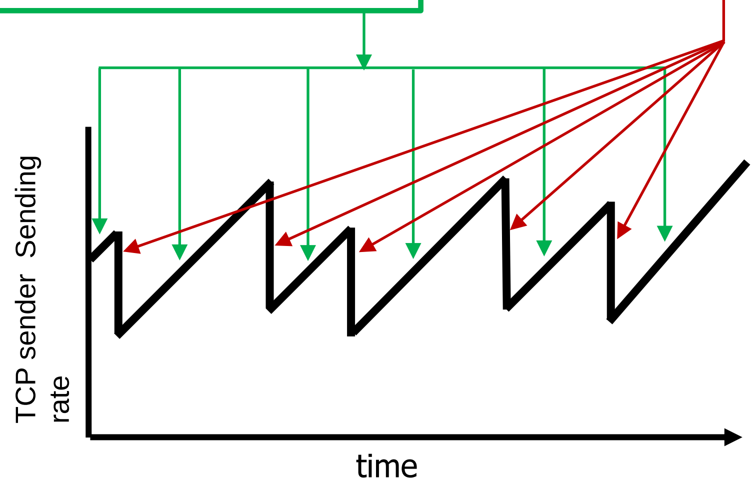
- *approach:* senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event

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

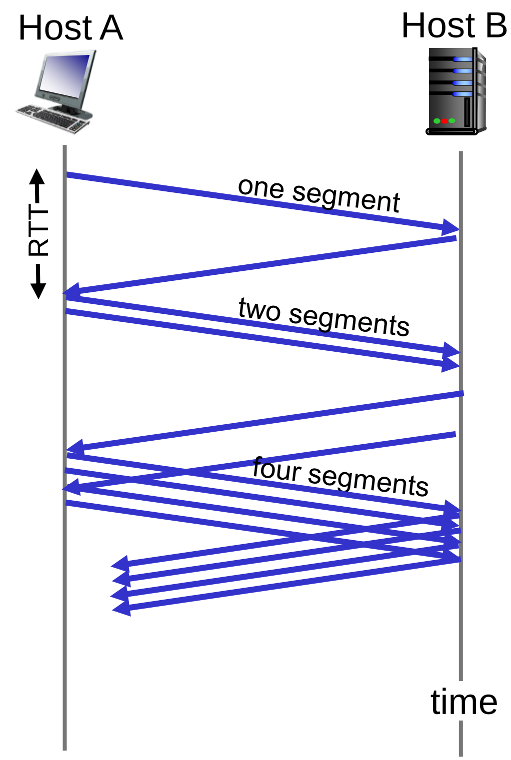
- Cut in half on loss detected by triple duplicate ACK (TCP Reno)
- Cut to 1 MSS (maximum segment size) when loss detected by timeout (TCP Tahoe)

Why AIMD?

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

TCP slow start

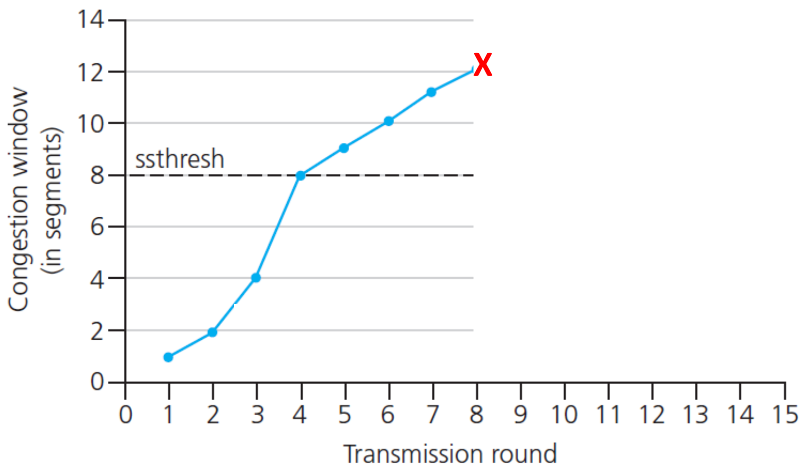
- when connection begins, increase rate exponentially until first loss event:
  - initially **`cwnd`** = 1 MSS
  - double **`cwnd`** every RTT
  - done by incrementing **`cwnd`** for every ACK received
- *summary:* initial rate is slow, but ramps up exponentially fast



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.



Implementation:

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