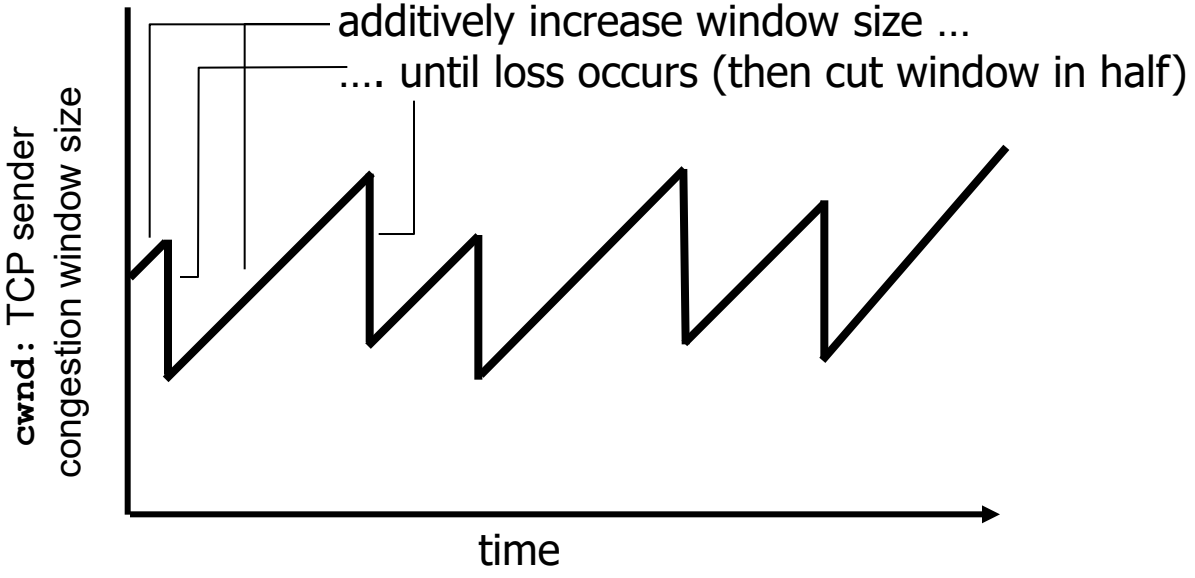
TCP congestion control: additive increase multiplicative decrease

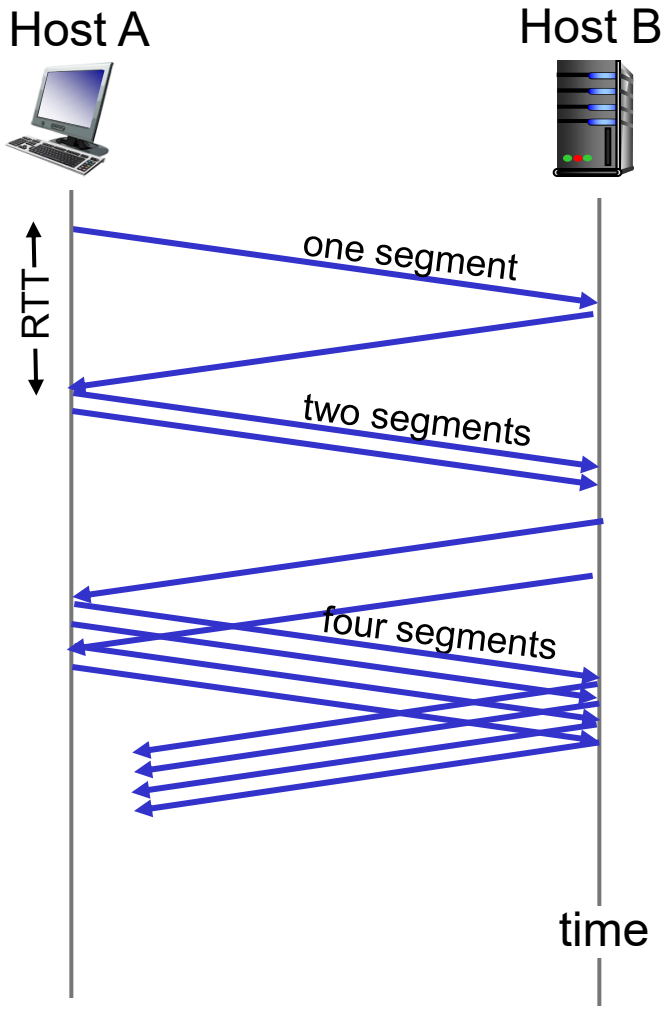
- *approach:* sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **`cwnd`** by 1 MSS every RTT until loss detected
  - *multiplicative decrease:* cut **`cwnd`** in half after loss

AIMD saw tooth behavior: probing for bandwidth



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



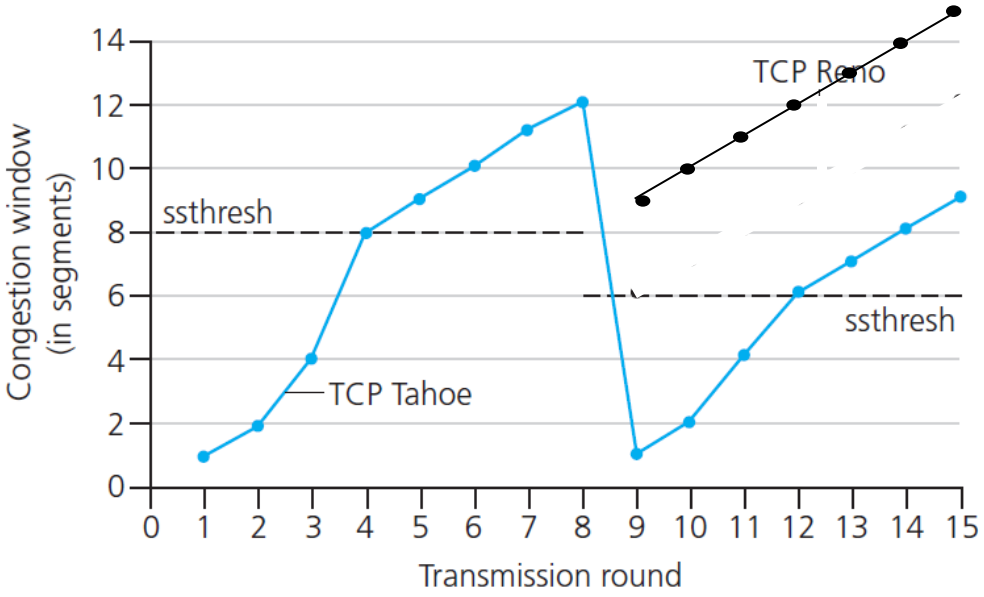
TCP: detecting, reacting to loss

- loss indicated by timeout:
  - **`cwnd`** set to 1 MSS;
  - window then grows exponentially (as in slow start) to threshold, then grows linearly
- loss indicated by 3 duplicate ACKs: TCP RENO
  - dup ACKs indicate network capable of delivering some segments
  - **`cwnd`** is cut in half window (adding in 3 **MSS** for good measure to account for the triple dup ACKs received) then grows linearly
- TCP Tahoe always sets **`cwnd`** to 1 (timeout or 3 duplicate acks)

TCP: switching from slow start to CA

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

Problem 1

a) TCP slowstart is operating in the intervals [1,6] and [23,26]

b) TCP congestion advoidance is operating in the intervals [6,16] and [17,22]

c) After the 16th transmission round, packet loss is recognized by a triple duplicate ACK. If there was a timeout, the congestion window size would have dropped to 1.

d) After the 22nd transmission round, segment loss is detected due to timeout, and hence the congestion window size is set to 1.

Problem 1

e) The threshold is initially 32, since it is at this window size that slowstart stops and congestion avoidance begins.

f) The threshold is set to half the value of the congestion window when packet loss is detected. When loss is detected during transmission round 16, the congestion windows size is 42. Hence the threshold is 21 during the 18th transmission round.

g) The threshold is set to half the value of the congestion window when packet loss is detected. When loss is detected during transmission round 22, the congestion windows size is 29. Hence the threshold is 29/2 ~= 14 during the 24th transmission round.

Note: cwnd = 21 + 3 =24 at round 17. It then increases 1 per round, so till round 22, cwnd = 24+5 = 29

Problem 1

h) During the 1st transmission round, segment 1 is sent; segment 2-3 are sent in the 2nd transmission round; segment 4-7 are sent in the 3rd transmission round; segment 8-15 are sent in the 4th transmission round; segment16-31 are sent in the 5th transmission round; segment 32-63 are sent in the 6th transmission round; segment 64 – 96 are sent in the 7th transmission round. Thus segment 70 is sent in the 7th transmission round.

i) The threshold will be set to half the current value of the congestion window (8) when the loss occurred, and the congestion window will be set to the new threshold value +3 MSS. Thus the new values of the threshold and window will be 4 and 7, respectively.

Problem 1

j) Recall that TCP Tahoe always sets cwnd to 1 (timeout or 3 duplicate acks). ssthresh is set to 1/2 of cwnd just before loss event.

So threshold is 21, and congestion window size is 4.

Problem 2

No. There is nothing to stop a TCP-based application from using multiple parallel connections. For example, Web browsers often use multiple parallel TCP connections to transfer the multiple objects within a Web page. When an application uses multiple parallel connections, it gets a larger fraction of the bandwidth in a congested link.

Problem 3

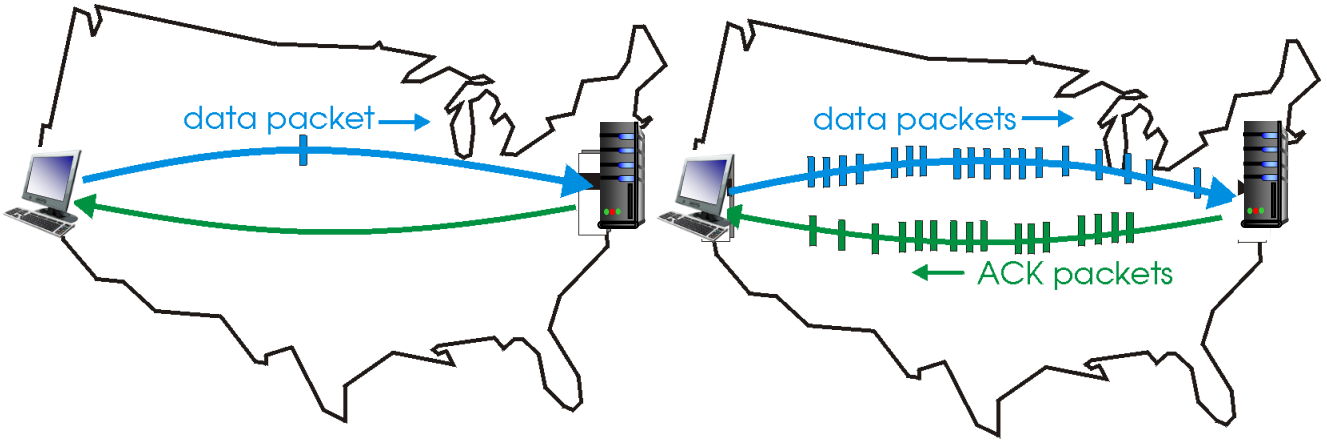
a) 27 bytes

b) 65.

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

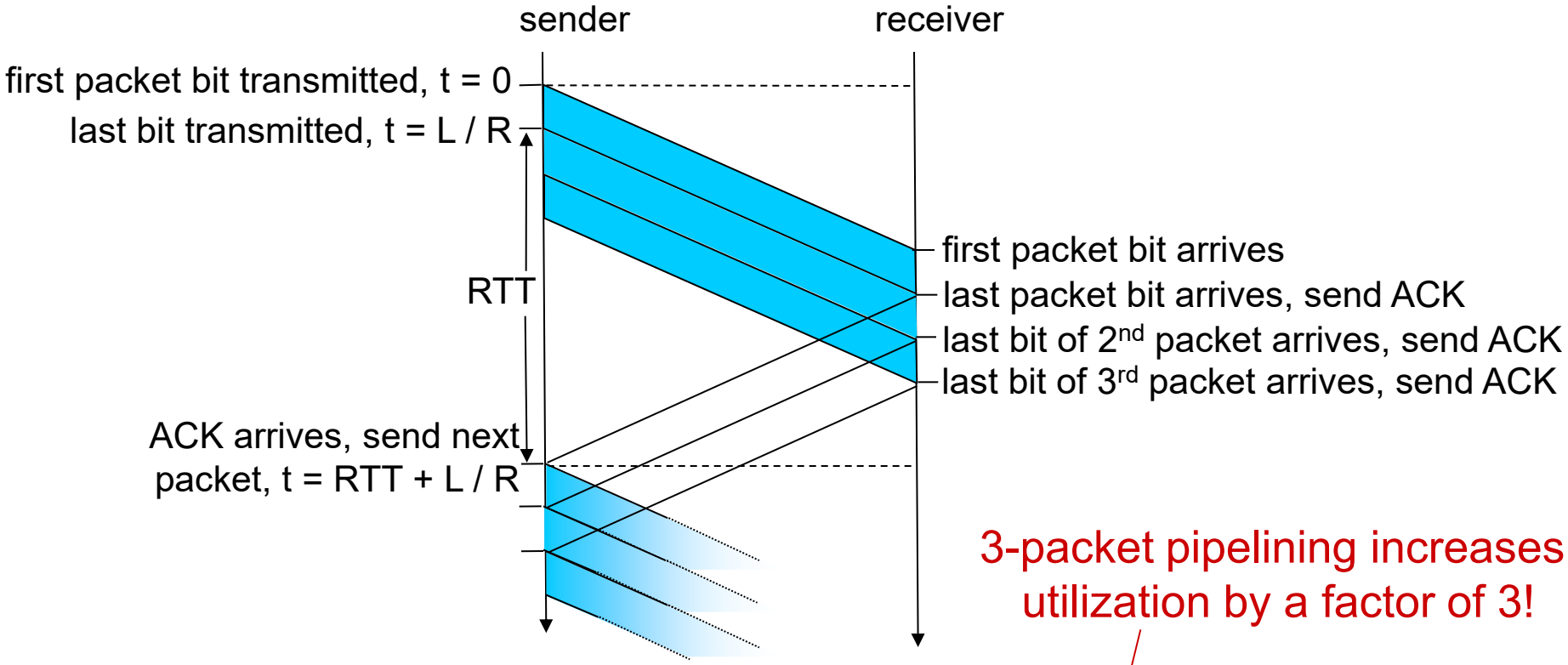


(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

- two generic forms of pipelined protocols: *go-Back-N, selective repeat*

Pipelining: increased utilization



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

$$U_{sender} = \frac{3L / R}{RTT + L / R} = \frac{.0024}{30.008} = 0.00081$$

Pipelining: increased utilization

- **Transmission (Serialization) Time**: L/R, the time to push all bits of a packet onto the wire at the sender's rate. This is explicitly shown in the diagram (e.g., the slanted blue lines represent packets being serialized one after another during pipelining).
- **Propagation Delay**: The physical time for electromagnetic signals (bits) to travel the distance to the receiver and back (round-trip). This is **not** ignored—it's the dominant part of **RTT** in most real networks.
- **Other Delays** (often omitted in ideal models): Queuing (waiting in router buffers) and processing (at endpoints), but the diagram assumes a clean pipe.

Problem 4

It takes 9.6 microseconds (or 0.0096 milliseconds) to send a packet, as $1200*8/10^9=9.6$ microseconds. In order for the sender to be busy 97 percent of the time, we must have

or approximately 3033 packets.

$$util = 0.97 = \frac{0.0096n}{30}$$