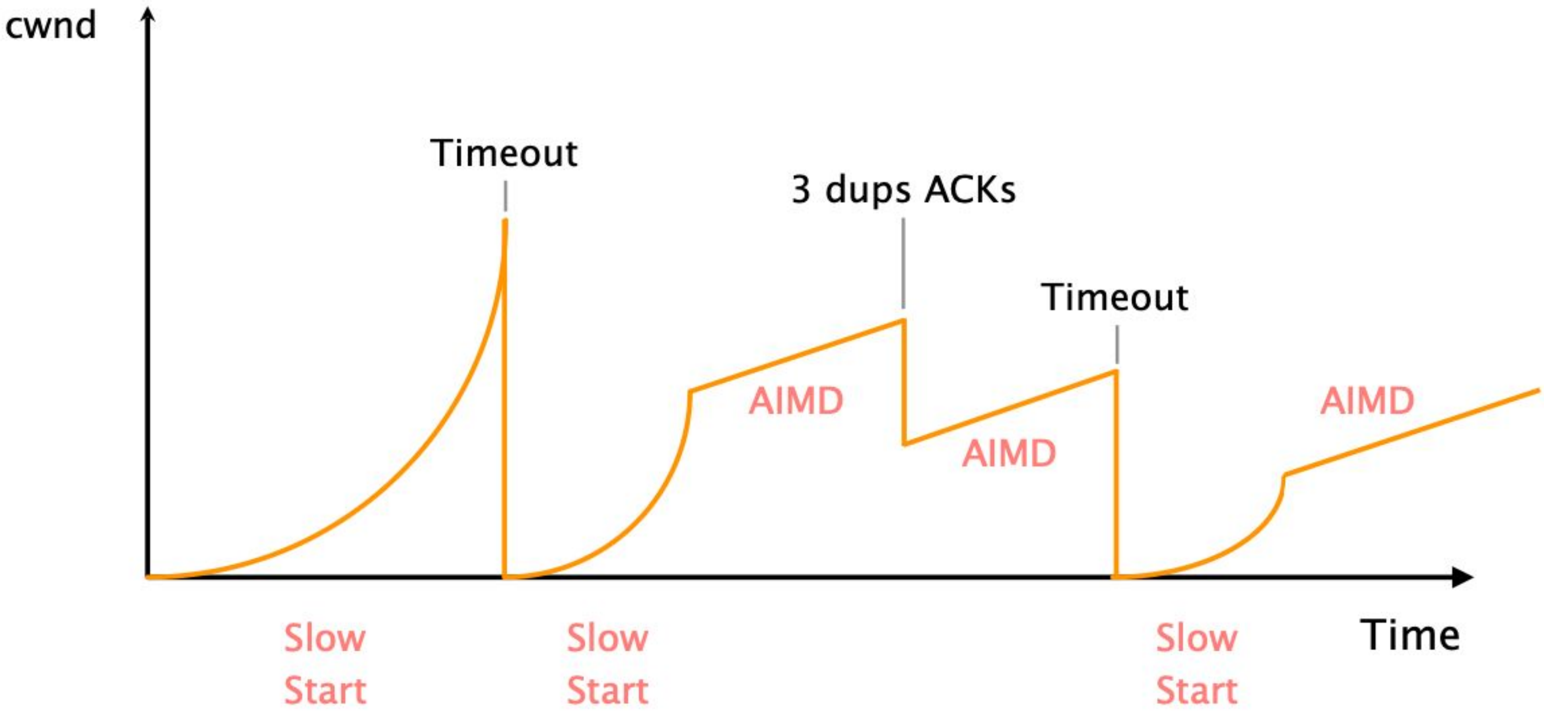
Congestion control makes TCP throughput look like a “sawtooth”



Questions

When the retransmission timer expires at the sender, the value of ssthreshold is set to what?

Where is the CWND decided? Is it negotiated?

Questions

When the retransmission timer expires at the sender, the value of ssthreshold is set to what?

½ the current CWND

Where is the CWND decided? Is it negotiated?

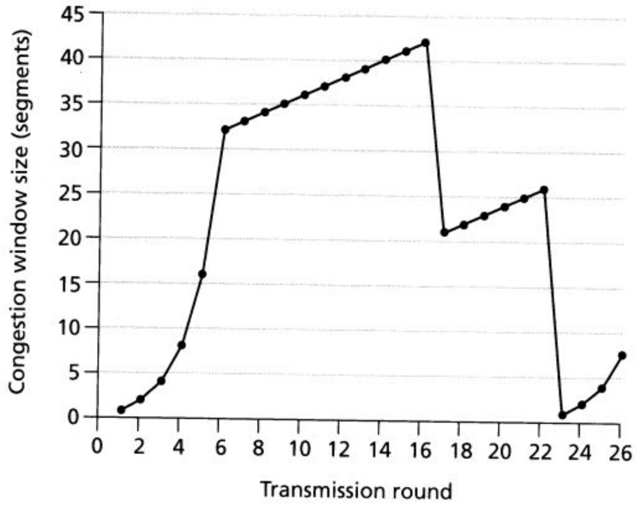
NOT a header, calculated solely by the sender

Questions

Identify time intervals where TCP slow-start is operating.

Identify time intervals where TCP congestion-avoidance is operating

After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout event?

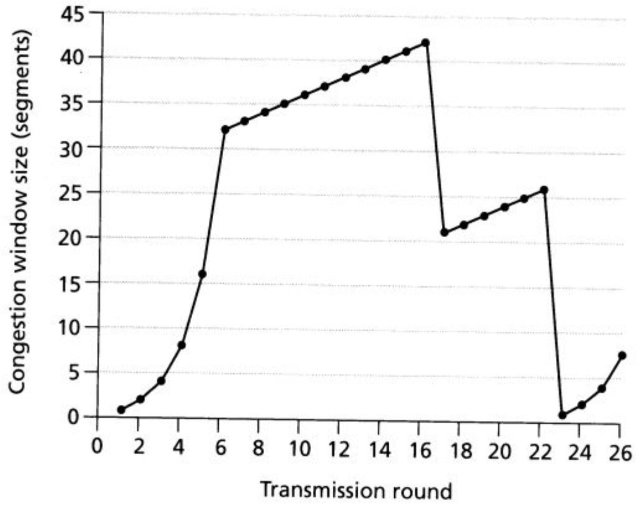


Questions

Identify time intervals where TCP slow-start is operating.
[1,6] and [23,26]

Identify time intervals where TCP congestion-avoidance is operating
[6,16] and [17,22]

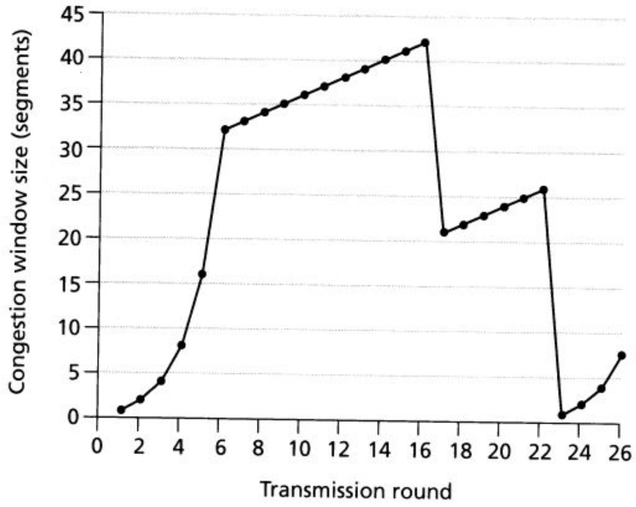
After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout event?
At the 16th transmission round, packet loss is recognized by a triple duplicate ACK. If there was a timeout, the congestion window size would have dropped to 1.



Questions

After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout event?

What is the ssthreshold value at the first transmission round?



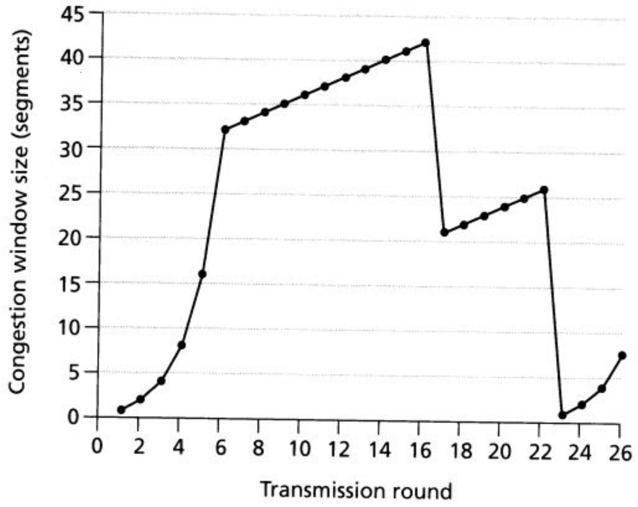
Questions

After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout event?

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

What is the ssthreshold value at the first transmission round?

The threshold is initially 32, since it is at this window size that slow start stops and congestion avoidance begins.

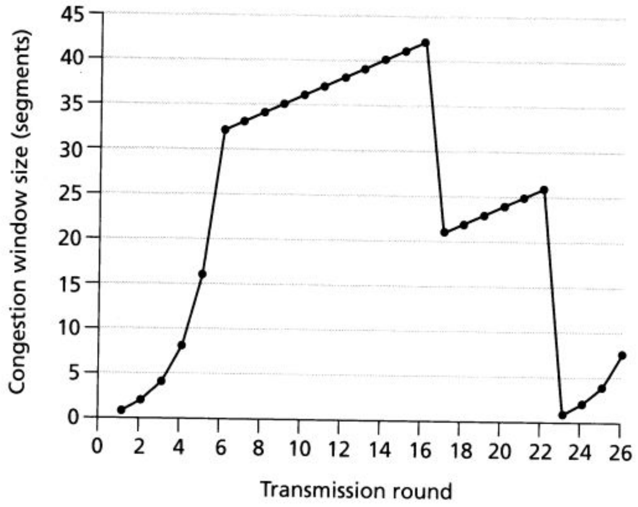


Questions

What is the ssthreshold value at the 18th transmission round?

What is the ssthreshold value at the 24th transmission round?

What will be the values of CWND and ssthreshold if packet loss is detected after the 26th round by receipt of triple duplicate ACKs?

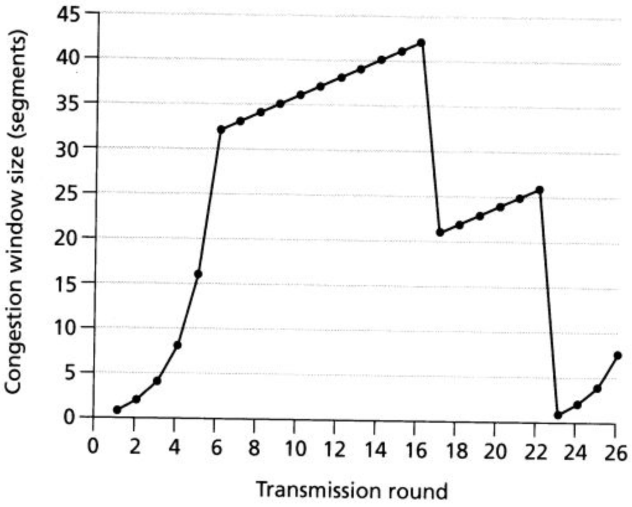


Questions

What is the ssthreshold value at the 18th transmission round?
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.

What is the ssthreshold value at the 24th transmission round?
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 26. Hence the threshold is 13 during the 24th transmission round.

What will be the values of CWND and ssthreshold if packet loss is detected after the 26th round by receipt of triple duplicate ACKs?
The congestion window and threshold will be set to half the current value of the congestion window (8) when the loss occurred. Thus the new values of the threshold and window will be 4.



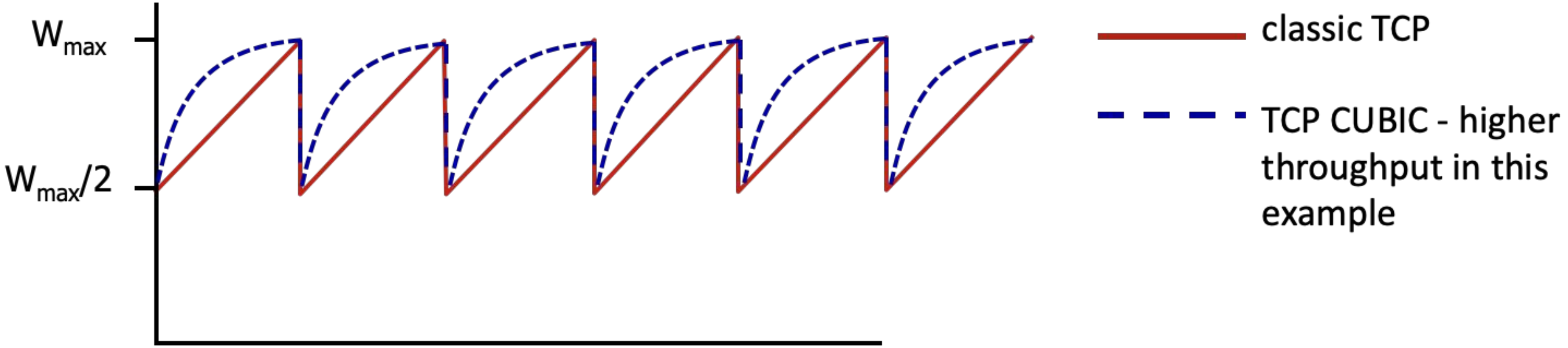
Wireshark example

Is there a better way than AIMD to probe for usable bandwidth?

TCP CUBIC

Insight/intuition:

- Wmax: sending rate at which congestion loss was detected
- congestion state of bottleneck link probably (?) hasn't changed much
- after cutting rate/window in half on loss, initially ramp to to Wmax faster, but then approach Wmax more *slowly*



TCP CUBIC

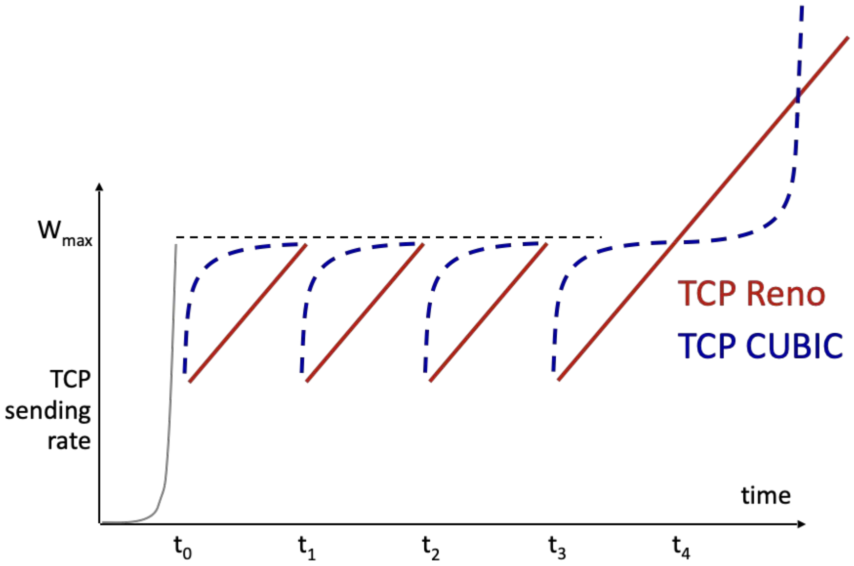
K: point in time when TCP window size will reach Wmax

- K itself is tuneable

increase W as a function of the cube of the distance between current time and K

- larger increases when further away from K
- smaller increases (cautious) when nearer K

TCP CUBIC default in Linux, most popular TCP for popular Web servers



TCP and congested bottleneck links

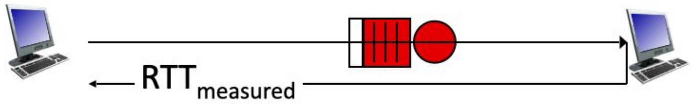
TCP (classic, CUBIC) increase TCP’s sending rate until packet loss occurs at some router’s output: the bottleneck link

understanding congestion: useful to focus on congested bottleneck link

Goal: “keep the end-to-end pipe just full, but not fuller”

Delay based congestion control

Goal: “keep the end-to-end pipe just full, but not fuller”



$$\text{measured throughput} = \frac{\text{\# bytes sent in last RTT interval}}{RTT_{measured}}$$

Delay-based approach:

- RTT_{min} - minimum observed RTT (uncongested path)
- uncongested throughput with congestion window `cwnd` is $cwnd/RTT_{min}$

if measured throughput “very close” to uncongested throughput
    increase `cwnd` linearly /* since path not congested */
else if measured throughput “far below” uncongested throughout
    decrease `cwnd` linearly /* since path is congested */

Delay based congestion control

congestion control without inducing/forcing loss

maximizing throughput (“keeping the just pipe full… ”) while keeping delay low (“…but not fuller”)

a number of deployed TCPs take a delay-based approach

- BBR deployed on Google’s (internal) backbone network

TCP makes assumptions

Will it ever actually fill the available bandwidth?

Most TCP flavors base their congestion control entirely on loss - can you think of any issue with that?

TCP over “long, fat pipes”

example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput

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 = $2 \cdot 10^{-10}$ **YIKES**

versions of TCP for long, high-speed scenarios

TCP Throughput (Mbps)	RTTs between losses	cwnd	Packet Loss Rate P
1	5.5	8.3	0.02
10	55	83	0.0002
100	555	833	2×10^{-6}
1000	5555	8333	2×10^{-8}
10,000	55555	83333	2×10^{-10}

Is TCP Fair?

Is TCP Fair?

A: Yes, under idealized assumptions:

- same RTT
- fixed number of sessions

Is TCP Fair?

Fairness and UDP

- multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- instead use UDP:
  - send audio/video at constant rate, tolerate packet loss
- there is no “Internet police” policing use of congestion control

Fairness, parallel TCP connections

- application can open multiple parallel connections between two hosts
- web browsers do this , e.g., link of rate R with 9 existing connections:
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2

Is TCP Ideal?

- TCP adds a lot of complexity in exchange for reliable, in-order byte delivery

- UDP is much faster / simpler
    - If you aren't dealing with a lot of loss, UDP could be better

- TCP flows have a fundamental feature that must be considered / engineered around:
    - Head of line (HOL) blocking
        - One lost packet in the TCP stream makes all others wait until that packet is re-transmitted and received.