CS 168
Spring 2025

Congestion Control

Discussion 7

1 True/False

1.1 UDP uses congestion control.

F

1.2 Flow control slows down the sender when the network is congested.

F

1.3 For TCP timer implementations, every time the sender receives an ACK for a previously unACKed packet, it will recalculate ETO.

F. Only clean sample

1.4 CWND (congestion window) is usually smaller than RWND (receiver window).

T

1.5 AIMD is the only "fair" option among MIMD, AIAD, MIAD, and AIMD.

T

2 Impact of Fast Recovery

Consider a TCP connection, which is currently in Congestion Avoidance (AIMD):

- The last ACK sequence number was 101.
- The CWND size is 10 (in packets).
- The packets 101–110 were sent at t = 0, 0.1, ..., 0.9 (sec), respectively.
- The packet 102 is lost only for its first transmission.
- RTT is 1 second.

2.1 Without fast recovery:

- On new ACK, CWND += $\frac{1}{\lfloor \text{CWND} \rfloor}$
- On triple dupACKs, SSTHRESH = $\lfloor \frac{\text{CWND}}{2} \rfloor$, then CWND = SSTHRESH.

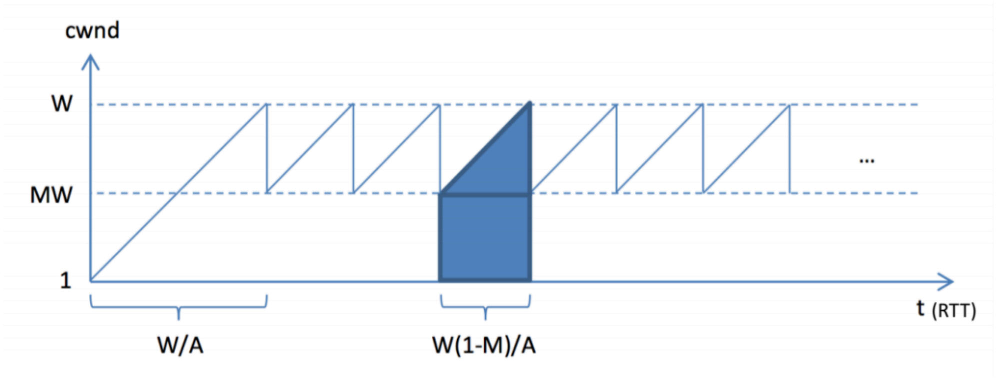
Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (mark retransmits)
1.0	102 (101)	$10 + \frac{1}{10} = 10.1$	111
1.2	102 (103)	10.1	/
1.3	102 (104)	10.1	/
1.4	102(105)	5	102(RX)
1.5	102(106)	5	
1.6	102(107)	5	
1.7	102(108)	5	
1.8	102(109)	5	
1.9	102(110)	5	
2.0	102(111)	5	
2.4	112(102)	5 + 0.2 = 5.2	112-116

$\boxed{2.2}$ With fast recovery:

- On triple dupACKs, SSTHRESH = $\lfloor \frac{\text{CWND}}{2} \rfloor$, then CWND = SSTHRESH + 3, enter fast recovery.
- In fast recovery, CWND += 1 on every dupACK.
- On new ACK, exit fast recovery, CWND = SSTHRESH.

Time (sec)	**Receive ACK (due to)**	**CWND**	**Transmit Seq # (mark retransmits)**
1.0	102 (101)	$10 + \frac{1}{10} = 10.1$	111
1.2	102 (103)	10.1	/
1.3	102 (104)	10.1	/
1.4	102(105)	5+3 = 8	102(RX)
1.5	102(106)	9	
1.6	102(107)	10	
1.7	102(108)	11	112
1.8	102(109)	12	113
1.9	102(110)	13	114
2.0	102(111)	14	115
2.4	112	5	116

3 AIMD Throughput



Consider a generalized version of AIMD, where:

- For every window of data ACK_ed, the window size increases by a constant A.
- When the window size reaches W, a loss occurs, and the window size is multiplied by a constant $M < 1$.

For simplicity, *assume that* $W(1-M)$ *is divisible by* A. Thus, the window sizes will cycle through the following: WM, WM + A, WM + 2A, ... W. Let the RTT denote the packet round trip time. A graph of window size versus time is referenced in the figure above.

$\boxed{3.1}$ What is the average throughput? *Express your answers in the number of packets, so we do not need to consider MSS.*

throughput =
(MW + W)/
(2*RTT)

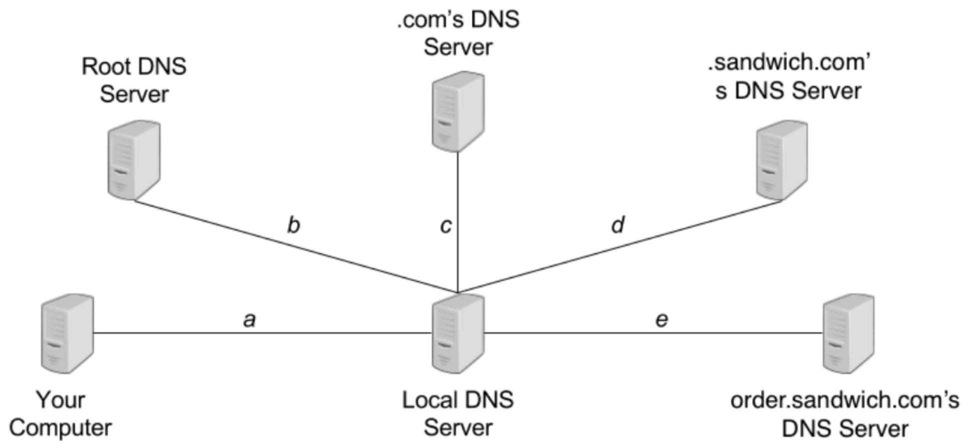
$\boxed{3.2}$ Calculate the loss probability p, using W and M.

drop 1 every W(1-M)/A * (MW+W)/2
packets, so p = 1 / (W(1-M)/A * (MW
+W)/2)

$\boxed{3.3}$ Derive the formula for throughput in part (a) when $M = 0.5$ and $A = 1$, using only p and RTT.

throughput = W(M + 1) / 2 * RTT, so we
should calculate W. Since p = 2A /
(W^2(1-M^2), W = sqrt(8 /3p).
so tpt = 1/RTT * sqrt(3/2p)

4 Domain Name System



A sandwich ordering website **`www.order.sandwich.com`** is accepting online orders for the next T minutes. Consider the following setup of DNS servers, with annotated latencies between servers.

Assume that:

- The latency between your computer and the website's server is t.
- Once you send an order for a sandwich, you must wait for a confirmation response from the website before issuing another.
- Your computer **does not cache** the website's IP address.

$\boxed{4.1}$ Your local DNS server doesn't cache any information.

time per sandwich order: 2b+2c +2d + 2a + 2e + 2t.

$\boxed{4.2}$ Your local DNS server caches responses, with a time-to-live $L \geq T$.

first order takes 2a + 2b + 2c + 2d + 2e + 2t. subsequent order takes 2a + 2t.

$\boxed{4.3}$ Let $T = 600$ seconds and $a = b = c = d = e = t = 1$ second. Your local DNS server caches responses with a finite TTL of **30 seconds**.

120.