CS 168
Spring 2025

Congestion Control
Discussion 7

1 True/False

1.1 UDP uses congestion control.

False, TCP uses congestion control.

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

False, flow control ensures the sender doesn't overflow the receiver's buffer.

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

False, only clean samples are used. For example, ACKs on a packet that have been retransmitted are not used since the sender cannot be sure which version the ACK is from.

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

True.

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

True.

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	/

Time (sec)	Receive ACK (due to)	CWND	Transmit Seq # (mark retransmits)
1.4	102 (105)	$\lfloor \frac{10.1}{2} \rfloor = 5$	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)	$5 + \frac{1}{5} = 5.2$	112 - 116

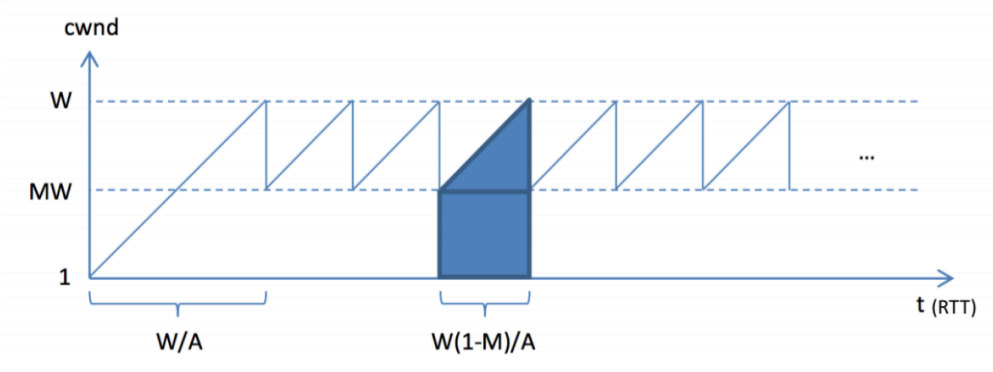
2.2 With fast recovery:

- 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)	$\lfloor \frac{10.1}{2} \rfloor + 3 = 8$	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 (102)	SSTHRESH = 5	116

Note: *Three dupACKs = 1 regular ACK + 3 ACKs of a sequence number that have been ACKed before already, so we retransmit on the 4th ACK of seq 102.*

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.

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

Throughput = $\frac{\text{Average number of packets in flight}}{\text{RTT}} = \frac{(\text{MW}+W)}{2\times \text{RTT}} = \frac{W(M+1)}{2\times \text{RTT}}$

3.2 Calculate the loss probability p, using W and M.

We have one drop out of every $\frac{W(1-M)}{A} \times \frac{\text{MW}+W}{2}$ packets sent (the area of the shaded trapezoid in the plot). Thus, the loss probability is:

$p = \frac{1}{\frac{W(1-M)}{A}\times\frac{\text{MW}+W}{2}} = \frac{2A}{W^2(1-M^2)}$

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

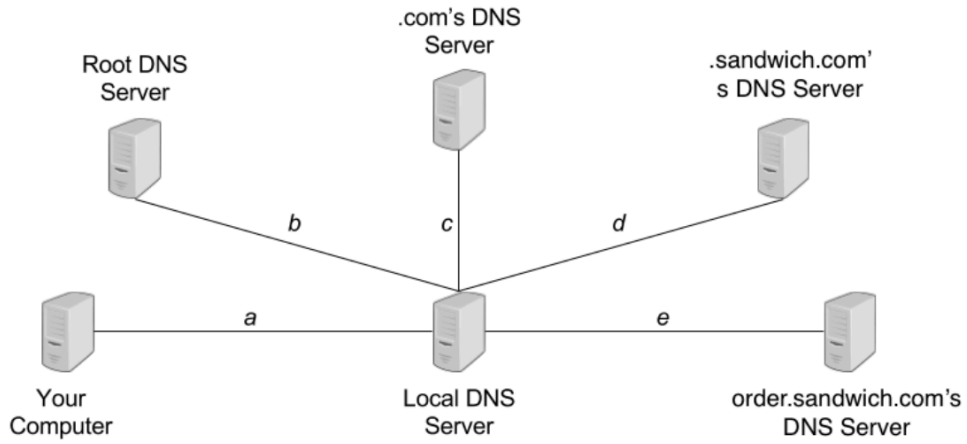
Throughput = $\frac{W(M+1)}{2\times \text{RTT}} = \frac{3W}{4\times \text{RTT}}$

p = $\frac{2A}{W^2(1-M^2)} = \frac{2}{0.75W^2} = \frac{8}{3W^2}$

We get $W = \sqrt{\frac{8}{3p}}$ from the loss probability p, and plug this into the throughput equation:

Throughput = $\frac{3W}{4\times \text{RTT}} = \frac{3\times\sqrt{\frac{8}{3p}}}{4\times \text{RTT}} = \sqrt{\frac{9\times 8}{16\times 3\times p}}\frac{1}{\text{RTT}} = \frac{1}{\text{RTT}}\sqrt{\frac{3}{2p}}$

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.

4.1 Your local DNS server doesn't cache any information.

Your computer begins by issuing a DNS query for `www.order.sandwich.com` to its local DNS server, which takes time a. Your local DNS server then **iteratively queries** the root DNS server, `.com`'s DNS server, `.sandwich.com`'s DNS server, and `order.sandwich.com`'s DNS server, which takes time $2b + 2c + 2d + 2e$. It then returns the result of the query to you, which takes time a.

After obtaining the IP address, your computer **issues a sandwich request** to `www.order.sandwich.com`, which responds with an order confirmation, taking time $2t$.

The **total time per sandwich order** is: $\mathbf{2a + 2b + 2c + 2d + 2e + 2t}$

Thus, the **total number of sandwiches** that can be ordered in time T is: $\left\lfloor \frac{T}{2a+2b+2c+2d+2e+2t} \right\rfloor$

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

If $L \geq T$, once the result of the DNS query for `www.order.sandwich.com` is cached, it remains cached in our local DNS server until the website's server ultimately goes down.

- The **first query** takes the same amount of time as before: $2a + 2b + 2c + 2d + 2e + 2t$
- **Subsequent queries** take only $\mathbf{2a + 2t}$, since the local DNS server can now provide the IP immediately.

Thus, the **total number of sandwiches** we can order is:

$$\begin{cases} 1 + \left\lfloor \frac{T-(2a+2b+2c+2d+2e+2t)}{2a+2t} \right\rfloor & \text{if } T \geq 2a + 2b + 2c + 2d + 2e + 2t \\ 0 \text{ otherwise} \end{cases}$$

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**.

When the first DNS query is made, the response gets cached in the **local DNS server** at:

$a + 2b + 2c + 2d + 2e = 9$ seconds

This response remains cached **for 30 seconds**, expiring at **time 39 seconds**.

- The **first order is completed at time**: 12 seconds $(9s + a + t)$
- **Additional orders** can be placed from time **12 to 39**, each taking **4 seconds** $(2a + 2t)$.
- The number of orders made during this cached period is: $\left\lceil \frac{39-12}{4} \right\rceil = 7$
- The **pattern repeats every 40 seconds** ($T = 600$ allows for **15 cycles**): $\frac{600}{40} = 15$
- Total sandwiches ordered: 15 × (7 + 1) = **120**

Not bad!