Final Review

CONGWIN

A host has started a TCP transmission with **MSS = 1460 bytes. It uses "slow-start", with a **limit of 11680 bytes.**
--What is the congestion window size after sending 15 packets? Assume all packets have length = MSS.
→Slow-start begins with **1 MSS**, then doubles each round until the limit is reached. Then it increases linearly until loss or **3 duplicate ACK's**.

1) CW = 1 MSS (1460 bytes), send packet #1

2) CW = 2 MSS (2920 bytes), send packets #2 and #3

3) CW = 4 MSS (5840 bytes), send packets #4 - #7

4) CW = 8 MSS (11680 bytes), send packets #8 - #15

Since limit is reached, next increase is linear: **11680 + 1460 = 13140**

Suppose that after sending packet #8, we get 3 duplicate ACK's. What is the new congestion window size?
→7 MSS (10220 bytes)

Suppose that after sending packet #7, we get a timeout on packet #4. What is the new congestion window size?
→1 MSS (1460 bytes)

What is the Maximum Segment Size (MSS)?

- This is the maximum amount of transport-layer data which can be sent, such that it will fit within one link-layer data frame.

Goals of Congestion Control:

- Reliably detect network congestion
- Avoid congestion (when possible)
- React to lessen the problem (when necessary)

How can TCP react to congestion?

- TCP will try to match its send rate to network conditions.
- To do this, TCP must do the following:

    - Define a "congestion window" (CongWin), to limit the send rate
    - Continually probe for available bandwidth and update CongWin
    - React to segment delay/loss and update CongWin
- Reacting to congestion is on the top-ten list of networking research topics!

To probe for available bandwidth, TCP first will enter something called the "slow-start phase". In this phase, CongWin is initially set to 1 maximum segment size (MSS)**. TCP will then increase CongWin exponentially until either a Limit is reached, or a loss event occurs.

After slow-start concludes, it is assumed that TCP is close to the ideal sending rate. In the next phase, TCP will increase CongWin additively, and decrease CongWin multiplicatively. This is known as "additive increase, multiplicative decrease" (AIMD). In the phase, known as the "congestion avoidance" phase, a graph of CongWin size produces the following sawtooth pattern

How does the network core help with congestion control?
→Network core assistance: The network core senses when congestion is an issue – directly – and sends messages (either to the destination host, which bounces it back to the source host, or directly back to the source host) indicating congestion in the core, and possibly advising on a course of action.

How does the source host help with congestion control?
→Host inference: A source host attempts to infer congestion in the network core based on observables (e.g. round trip time, dropped packets)

In basic terms, how is congestion control implemented in TCP? What is this method called?
→When no congestion is perceived, the sliding window size gradually increases (additive increase). When there is perceived congestion, the sliding window size is drastically reduced (multiplicative decrease). Together this makes up an additive-increase multiplicative-decrease (AIMD) scheme.

**TCP's "Slow Start" mode allows for a very slow start, but a rapid increase (exponential), in the size of the congestion window. What is its purpose?

- This allows TCP to start sending amounts of data that do not significantly impact congestion, if it exists.
- This allows TCP to rapidly adjust its CongWin to a size that will not impact congestion.
- This allows TCP to rapidly probe for available bandwidth in the core.
- This allows TCP to rapidly infer the level of congestion in the core.

TCP connections (socket connections) are set up and torn down by exchanging messages between hosts. To setup a connection requires 3 messages to be sent between a client and server. First, the client sends a packet with the SYN bit (synchronize flag) set. The server accepts the packet, and sends an ack packet with SYN and ACK bits set. Finally, the client sends a final ack packet back to the server. Note that the final ack may also include client data. It is important to note, that both the client and server maintain state machines during this process, which completes with the socket connection fully initialized on both hosts.

The process to tear-down a TCP socket connect is similar. In this process, the server receives a packet with the FIN bit set. It responds with an ack, but because there may still be data left to send, it may continue to do so. Once all of the data is sent, the server will send a final ack with the FIN bit set, and the client will respond. Note that both the server and client maintain timers through this process, mainly because TCP connection teardown is somewhat unpredictable. It is unpredictable, because either host may terminate the communicating process before the teardown has a chance to complete.

If n TCP connections share the same bottleneck link of bandwidth R, then each should have an average rate of R/n.

What this means in practice, is that TCP is fair to each connection. While this is true, you may have already spotted the loophole: All a process has to do, is open multiple TCP connections, and Voilà! More bandwidth! In fact, your internet browser will do this to speed up web page downloads.

**3 Steps of TCP Connection Setup

1) SYN: Client host sends a segment to the Server host with the SYN bit set, the ACK bit cleared, and some pre-generated sequence number.

2) SYN ACK: Server host sends a segment to the Client host with the SYN and ACK bits set, a locally generated sequence number, and an ACK number equal to the first segment's sequence number plus one.

3) ACK: Client host sends a segment to the Server host with the ACK bit set, the SYN bit cleared, and the ACK number is the previous segment's sequence number plus one. Note that this segment may contain actual application data in the payload.

**4 Steps of TCP Connection Takedown:

1) FIN: Client host sends a segment to the Server host with the FIN bit set, the ACK bit cleared, and its next sequence number.

2) FIN ACK: Server host sends a segment to the Client host with the ACK bit set and Fin bit cleared. The ACK number is the next expected sequence number. (This is a normal TCP ACK segment).

3) FIN: Server host sends a segment to the Client host with the FIN bit set, the ACK bit cleared, and its next sequence number.

4) FIN ACK: Client host sends a segment to the Server host with the ACK bit set and the FIN bit cleared, and the ACK number is the previous segment's sequence number plus one. (This is a normal TCP ACK segment)

**What is Fairness in TCP

→The goal is to share bandwidth evenly amongst all connections through a router. TCP works toward fairness due entirely to its congestion window size algorithm. This is on a per-connection basis, so for example a web browser which spins off a connection for every image it has to download (in parallel) would not be "fair" to a single-connection FTP transfer.

**Given a 1 Gbps link with TCP applications A, B, and C. Application A has 3 TCP connections to a remote web server; application B has 1 TCP connection to a mail server; application C has 4 connections to a remote web server. According to TCP "fairness" … during times when all connections are transmitting, how much bandwidth should each application have?

→A gets 375 Mbps.
B gets 125 Mbps.
C gets 500 Mbps

**Min. TCP header size? 20 bytes

**In a Selective acknowledgement scheme, a received ACK indicates only that the ACK'd segment was received.

**In a Cumulative acknowledgement scheme, a received ACK indicates all segments prior to the ACK'd segment were received.

**Max TCP header size? 60 *bytes*

**Calculating EstimatedRTT:

$$EstimatedRTT_{New} = (1-\alpha)EstimatedRTT_{Prev} + \alpha \times SampleRTT_{Recent}$$

**HostA has established a TCP connection with HostB in a remote network. HostA is sending packets to HostB. Assume we have configured TCP, somehow, to ACK every segment (no ACKing every other segment). Assume that the timeout is the same for all packets. HostB's "window size" is 20000 bytes. HostB has already received and acknowledged everything sent by HostA's application up to and including byte #1,787. HostA now sends segments of the same application data stream in order:

P: 232 bytes
Q: 430 bytes
R: 405 bytes

Suppose the segments arrive at Host B in the order Q, P, and R. What is the acknowledgment number on the segment sent in response to segment R?

→2855

**HostA has established a TCP connection with HostB in a remote network. HostA is sending packets to HostB. Assume we have configured TCP, somehow, to ACK every segment (no ACKing every other segment). Assume that the timeout is the same for all packets. HostB's "window size" is 20000 bytes. HostB has already received and acknowledged everything sent by HostA's application up to and including byte #4,197. HostA now sends segments of the same application data stream in order:

P: 411 bytes
Q: 142 bytes
R: 271 bytes

What is the sequence number on segment P?

→4198

**Assume a TCP sender is continuously sending 1,323-byte segments. If a TCP receiver advertises a window size of 7,669 bytes, and with a link transmission rate 14 Mbps an end-to-end propagation delay of 37.3 ms, what is the utilization? Assume no errors, no processing or queueing delay, and ACKs transmit instantly. Also assume the sender will not transmit a non-full segment. Give answer in percentages, rounded to one decimal place, without units (e.g. for an answer of 10.43% you would enter "10.4" without the quotes).
→5 (margin of error +/- 0.1)

**TCP Timeout Interval:
→$TimeoutInterval = EstimatedRTT_n + 4 \cdot DevRTT_n$

**DevRTT:
→$DevRTT_n = (1-\beta) \cdot DevRTT_{n-1} + \beta \cdot |SampleRTT_{new} - EstimatedRTT_{n-1}|$

**Given a nodal delay of 68.6ms when there is no traffic on the network (i.e. usage = 0%), what is the effective delay when network usage = 10.8% ? (Give answer is miliseconds, rounded to one decimal place, without units. So for an answer of 0.10423 seconds you would enter "104.2" without the quotes).
→76.9

**A host starts a TCP transmission with an EstimatedRTT of 44.7ms (from the "handshake"). The host then sends 3 packets and records the RTT for each:

SampleRTT1 = 26.5 ms

SampleRTT2 = 39.1 ms

SampleRTT3 = 41 ms

(NOTE: SampleRTT1 is the "oldest"; SampleRTT3 is the most recent.)

Using an exponential weighted moving average with a weight of 0.4 given to the most recent sample, what is the EstimatedRTT for packet #4? Give answer in miliseconds, rounded to one decimal place, without units, so for an answer of 0.01146 seconds, you would enter "11.5" without the quotes.

→39.3

**Given a 2 Gbps link with TCP applications A, B, and C.

- Application A has 37 TCP connections to a remote web server
- Application B has 1 TCP connection to a mail server
- Application C has 19 TCP connections to a remote web server.

According to TCP "fairness", during times when all connections are transmitting, how much bandwidth should Application C have? (Give answer in Mbps, rounded to one decimal place, without units. So for an answer of 1234,567,890 bps you would enter "1234.6" without the quotes.)

→666.7

**Imagine a mythical set of protocols with the following details.

Maximum Link-Layer data frame: 1,255 bytes

Network-Layer header size: 29 bytes

Transport-Layer header size: 22 bytes

What is the size, in bytes, of the MSS? (Give answer without units)

→1204

**The rate of CongWin size increase (in terms of MSS) while in TCP's Slow-Start phase is Exponential .

**In host-inferred congestion control, congestion is detected based on delayed and/or dropped packets.

**TCP has a congestion control mechanism.

**UDP DOES NOT implement network fairness.

**TCP DOES NOT implement network fairness directly.

**UDP DOES NOT use an additive-increase multiplicative-decrease (AIMD) system to manage flows.

**Where do Network-layer protocols run?

Everywhere in the internet

The “first hop” router is the first router a datagram is passed to from the source host on the path to the destination host.

NetWork Layer→At a very high level, the purpose is to ensure logical communication from host to host. In a more nuts-and-bolts view, the purpose is to determine a path from source to destination that a datagram can take, and to move it on that path (within the network layer).

Routing is the process of determining the path a datagram will take from source to destination in the internet, whereas forwarding is the process within a router of moving a datagram from an input port to the appropriate output port, so that it may take the next step in its journey from source to destination.

A connection-oriented network layer implies a virtual-circuit network. This type of network has a call setup at the beginning of a host-to-host connection, and from that point the state of the connection is preserved in all routers from source to destination, until the call is taken down. Some primary benefits of VC networks are guaranteed bandwidth and timing (jitter), which makes them ideal for streaming audio/video.

A connectionless network layer implies a datagram network. The internet is a datagram network. The advanatages are similar to those of UDP – there is FAR less overhead. Each network need not preserve the state for every host-to-host communication passing through it. With the billions of devices on the internet, this would be terribly implausible regardless of the rapid development of storage space and access times.

The forwarding table in a datagram network's router matches ranges of addresses to output ports, rather than matching each address to its own output port. By doing this, it saves tremendously on time and required storage/access capability.

Routing Algorithm, and where it is stored→Finds a path from a router to destination, and selecting it appropriately. The result of the algorithm is used to construct the router forwarding table (or routing table), which is stored in the router.
**TRANS DELAY at output: Yes, due to transmission rate limitations on the output port link: if several datagrams were switched to the same output port, they will have to wait for access to the transmission medium. Packet loss can also occur, if the buffer overflows.
**At input?--> Yes, due to head-of-line blocking or output port contention: if the datagram at the front of the line in the input port queue cannot be transferred to the output port because there is already a transfer occurring to, or a full queue at, the desired port. This would cause a delay in transferring the HOL datagram to its output port. Packet loss can occur here as well, if the input buffer overflows.

**Router with routing table:
A datagram with the destination IP address 158.30.143.30 would be routed to:
→Divide each number by 2
→→ 158/2 = 79, remainder is 0
> 79/2 = 39, remainder is 1
> 39/2 = 19, remainder is 1
> 19/2 = 9, remainder is 1
> 9/2 = 4, remainder is 1
> 4/2 = 2, remainder is 0
> 2/2 = 1, remainder is 0
> 1/2 = 0, remainder is 1
Do same fo 30, 143 and 30

TCP/IP Datagram, what is altered?
→Dest Port, Header Checksum, Dest IPAdress

**Suppose that a 2200-byte datagram (identification #40) must transit a network which has a 660-byte MTU. Assume the minimum IP and TCP header sizes, i.e., the IP header is 20 bytes and the TCP header is 20 bytes.

1. How many fragments are created?
2. How many bytes of application data are carried in the first fragment?
3. How many bytes of application data are carried in the second fragment?
4. How many bytes of application data are carried in the last fragment?
5. What is the identification number of the second fragment?
6. What is the fragment offset in the last fragment?

p.s. I have the answers, but I need to know how to do it. Thanks.

→1.) Num of Frags = Size of dg/MTU
→2.) Frag1: Total bytes = 640, Header Bytes = 20, Total Data: 620 Bytes
→3.) Frag2: Total: 660, Header = 20, Total = 640 Bytes
→4.) Last Frag: (total dg size – (MTU * 3))
→5.) ID is 40 for all except first which is 0
→6.) Offset = MTU data size/8 bytes

NAPT translate Ip Addreess AND Port #s
→False
→The "time to live" field in a modern IPv4 datagram header specifies the number of remaining hops before the datagram is dropped.
→The largest amount of data, in bytes, which can be accomodated throughout a datagram's route from sender to receiver is called the PATH MTU
→IP Header NOT encapsulated in TCP data frags
**queueing at a router's input ports? (Check all that apply)
→Slow outbound link transmission rate.
Head of Line blocking.
Output port contention.
→Windows uses ICMP
→ICMP can carry messages from:
Router to Router
Destination Host to Source Host
Router to Sender Host
Source Host to Destination Host
Transport Layer Header IS encapsulated in every datag frag
Nodes represent routers
Explicit in 6 but not 4
→128 bit address, Extension headers, flow labeling, expliciat payload length
→In IPv6, fragmentation handled at NETWORK EDGE
→When encountering an IPv4-only router, an IPv6 datagram is encapsulated in an IPv4 datagram, with the next in-line IPv6 router as its destination.
→Transition: IPv4 routers still in use must "tunnel" IPv6 datagrams, by fragmenting/encapsulating them in IPv4 datagrams
→IPv6 does NOT have checksum
→::ffff:ffff:fffff
NOT a valid address
→1234::a03:abcd
VALID

→For an machine using 2-dimensional even parity for error detection/correction, and the following received bytes, where is the error? If there is no error, select "No Error" for both boxes. Bits are numbered left-to-right and top-down, indexed 1 => 7 then Parity.

Byte # 2

Bit # 4

Check rows(byte) and columns(bit), if ODD # of 1's, ERROR present
Star Ethernet uses the same multiple access control as Bus Ethernet.
→Chunks→Channel partitioning
→Two adjacent nodes, point to point
→Most modern Ethernet LAns sue STAR
→Link-layer deice at center: switch

D's IP address

RouterA's NIC#1 IP address

RouterA's NIC#1 MAC address

RouterA's NIC#1 MAC address

D's IP address

D's IP address

D's MAC address

D's MAC address

D's IP address

Invalid: ::ffff:ABCD:DBCA

Wired networks→easy

When using an *RSA* algorithm to construct private and public keys for a public key encryption system, choose prime numbers p and q, and then calculate $n = pq$, $z = (p-1)(q-1)$. Then choose e and d to create the public key and the private key . Suppose that $p = 5$, and $q = 11$. Which of the following values will work for d and e? Check all that apply.
→n = 5*11 = 55
→z = (5 – 1) * (11-1) = 4 * 10 = 40
→55 and 40 are true nums
→Insert different choices for P and Q, check if you get 55 + 40. If yes, good. If not, false.

Application
→Supprt network apps, determine dest IP addres
Transport
→Manage comms between processes
Network
→Comms between HOSTS
Link
→Called ‘frame’, encapsulate, detect/correct errors
Physical