Problem Set 6
Reliability (Go-Back-N Protocol, Sliding Window Protocol), TCP

Due: Start of class, Wednesday, April 17th

No handwritten solutions accepted. Show your work for full credit and submit on Gradescope.

1. Sequence Number Space
Consider the Go-Back-N protocol with a send window size of 30 and a sequence number range of 8192. Suppose that at time $t$, the next in-order packet that the receiver is expecting has a sequence number of $k$. Assume that, the medium may drop packets but does not reorder messages.

a. What are the possible sets of sequence number inside the sender’s window at time $t$? Justify your answer.
Here we have a window size of $N=30$. Suppose the receiver has received packet $k-1$, and has ACKed that and all other preceding packets. If all of these ACK's have been received by the sender, then the sender's window is $[k, k+N-1]$.

Suppose next that none of the ACKs have been received at the sender. In this second case, the sender's window contains $k-1$ and the $N$ packets up to and including $k-1$. The sender's window is thus $[k-N,k-1]$. By these arguments, the senders window is of size 15 and begins somewhere in the range $[k-N,k]$.

b. What are all possible values of the ACK field in the message currently propagating back to the sender at time $t$? Justify your answer.
If the receiver is waiting for packet $k$, then it has received (and ACKed) packet $k-1$ and the $N-1$ packets before that. If none of those $N$ ACKs have been yet received by the sender, then ACK messages with values of $[k-N,k-1]$ may still be propagating back.

Because the sender has sent packets $[k-N,k-1]$, it must be the case that the sender has already received an ACK for $k-N-1$. Once the receiver has sent an ACK for $k-N-1$ it will never send an ACK that is less that $k-N-1$. Thus the range of in-flight ACK values can range from $k-N$ to $k-1$.

c. With the Go-Back-N protocol, is it possible for the sender to receive an ACK for a packet that falls outside of its current window? Justify your answer with an example.
Yes. Suppose the sender has a window size of 3 and sends packets 1, 2, 3 at $t_0$. At $t1$ ($t1 > t_0$) the receiver ACKS 1, 2, 3. At $t2$ ($t2 > t1$) the sender times out and resends 1, 2, 3. At $t3$ the receiver receives the duplicates and re-acknowledges 1, 2, 3. At $t4$ the sender receives the ACKs that the receiver sent at $t1$ and advances its window to 4, 5,
6. At \( t5 \) the sender receives the ACKs 1, 2, 3 the receiver sent at \( t2 \). These ACKs are outside its window.

2. Sliding Window Protocols – Part 1

This question concerns the different trade-offs with sliding window protocols parameters.

a. Draw a timeline for the sliding window algorithm with \( SWS = 5, RWS = 4 \) frames for the following two situations. Use a timeout interval of about \( 2 \times RTT \).
   i. Frame 5 is lost.
   ii. Frames 5-8 are lost.

i) Possible solutions:

**Cumulative ACKs**

<table>
<thead>
<tr>
<th>Frame</th>
<th>Ack</th>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Selective Repeat ACKs**

<table>
<thead>
<tr>
<th>Frame</th>
<th>Ack</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Ack5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ii)
b. Consider the sliding window algorithm with SWS = RWS = 4 and NumSeqNumbers = 7. The Nth packet DATA[N] contains N mod 7 in its sequence number field. Show an example in which the algorithm becomes confused. No packets may arrive out of order. Note, this implies that NumSeqNumbers >= 8 is necessary and sufficient for correction operation of a sliding window protocol.

Sender transmits DATA[0 .. 3] first with sequence numbers 0 .. 3. Receiver receives all frames and sends a cumulative ACK which is lost. So, the sender retransmits DATA[0..3] with sequence numbers 0 .. 3. Now receiver is waiting for frames with sequence numbers 4, 5, 6, 0. So, the receiver thinks it received frame 0 (DATA[7]) while it has received DATA[0]. So, the receiver makes an error.

3. Sliding Window Protocols – Part 2

You want to transfer a file from Champaign to Chicago. For this problem, assume:

- The size of file is 160,000 bytes. It will be transferred in 1,500-byte data packets, for which 100 bytes are taken up with headers. The size of acknowledgement packets is 150 bytes including header. Every packet is acknowledged.
- The communication is bidirectional and the bandwidth is 200 Mbps (megabits/sec) in each direction. Hosts can send and receive at the same time.
- The propagation time from Champaign to Chicago is 10 ms, as is the propagation time in the opposite direction.
- The time to process a packet is very small, so the receiver can send an acknowledgement as soon as it receives a data packet, but not before it receives the entire data packet.
- Likewise, for sliding window, the sender can send packets back-to-back (to the degree that the window size permits).
- Assume there is no packet loss, corruption or reordering.

Each data packet holds 1,400 bytes of payload, so to transfer 160,000 bytes of payload will require \(\lceil \frac{160,000}{1,400} \rceil = \lceil 114.28 \rceil = 115 \) full-sized packets.

For a full-sized packet, the time to send it from Champaign to Chicago is:
\[ T_{data} = 10\text{ms} + \left( 1500 \text{ Bytes} \times 8 \text{ bits/Byte} \right) / 200 \text{ Mbps} = 10.06 \text{ ms} \]

Similarly, the time to send back a 150-byte acknowledgement is
\[ T_{ack} = 10\text{ms} + \left( 150 \text{ Bytes} \times 8 \text{ bits/Byte} \right) / 200 \text{ Mbps} = 10.006 \text{ ms} \]

The round-trip time (RTT) for a full-sized packet is therefore:
\[ RTT = T_{data} + T_{ack} = 10.06 \text{ ms} + 10.006 \text{ ms} = 20.066 \text{ ms} \]
a. How long will it take to transfer the file using Stop-and-Wait?

To send the file with Stop-and-Wait, each full-sized round requires time $RTT = T_{data} + T_{ack}$. We include the final partial packet to this to get:

$$T_{stop-and-wait} = 115 \times 20.066 \text{ ms} = 2307.59 \text{ ms}$$

b. What is the corresponding throughput (total amount of user data transferred = 160,000 bytes over the total time required to transfer it).

The throughput is the amount of data transferred divided by the time it took to send it, so:

$$\text{Throughput} = \frac{160000 \text{ Bytes} \times (8 \text{ bits/Byte})}{2.30759 \text{ sec}} = 0.5546913 \text{ Mbps}$$

c. How long will it take to transfer the file using a sliding window protocol with $SWS=RWS=20$. What is the throughput?

The first twenty packets will be sent back-to-back. The first ack comes back at a $RTT$ which is 20.066ms. The second ack is sent at $T_2 = 1500 \times 8 \text{ bits} / 200 \text{ Mbps} = 0.06\text{ms}$. Therefore, its ack is coming back at

$$T_2 + RTT = 0.06 \text{ ms} + 20.066 \text{ ms} = 20.126 \text{ ms}.$$  

After the 1st ack comes back, the sender starts to send another 20 packets back-to-back, the $i^{th}$ packet in this 20 packet is sent 1 $RTT$ time away from the ack of the $i^{th}$ packet in the first 20 packets. E.g., packet #22 is sent at $0.06 + 20.126 = 20.186 \text{ ms}$. 

Since there are 115 full packets, we know that #115 packet is sent at $6 \times RTT = 6 \times 20.066 \text{ ms} = 120.396 \text{ ms}$.

The ACKs for the last 19 packets will arrive the sender at $120.396\text{ms} + (1500 \text{ bytes} x 8 / 200 \text{ Mbps}) \times 19 = 120.396\text{ms} + 1.14\text{ms} = 121.536\text{ms}$

The throughput is $160000 \times 8 \text{ bits} / 121.536 \text{ ms} = 10.531 \text{ Mbps}$

This reflects a gain of nearly 20 times - not quite a full factor of 20 because the window doesn't allow us to fully parallelize transmission, but rather to pipeline it.
d. If SWS and RWS are set to the 2 times the bandwidth-delay product (bandwidth of the path times the roundtrip propagation time of the path), then how long does it take to transfer the file, and with what throughput?

The bandwidth-delay product of this path is:

\[ 200 \text{ Mbps} \times 20 \text{ ms} = 512 \text{ KB} \]

A window size of 600 KB corresponds to sending the entire file in a single burst. We can double-check this as follows:

The total time to receive the last ACK from the receiver will take:

\[ 115 \times 0.06 + 20.066 \text{ ms} = 27.026 \text{ ms} \]

So, the throughput is:

\[ \frac{160000 \times 8 \text{ bits}}{27.066 \text{ ms}} = 47.292 \text{ Mbps} \]

e. If SWS and RWS are set to three times the bandwidth-delay product, how long does the transfer take, with what throughput, and why?

Setting the window any higher won't change the throughput we can achieve, since the window computed using the bandwidth-delay product already allows us to send all our data in a single flight. More generally, using the bandwidth-delay product for the window allows us to completely “fill the pipe, namely to keep the forward transmission path continuously occupied. Any larger window can't enable us to go any faster, since we're already going as fast as the network can possibly let us go. (The larger window can lead to larger queues inside the network, since we give the network more load to process in a given amount of time. Ironically, this can lead to some of our packets being dropped due to exhausted queue space, which, as we'll see when we study congestion control, can actually cause the transfer to go much slower.)
4. Delay-Bandwidth Product for Links in Series

Consider three nodes A, B, C, D connected in series. Node A is connected to node B via a 4Gbps fiber optic link, 1000km in length. Node B is connected to node C via 50 Mbps link, 20 km in length. Node C is connected to node D via a 100 Mbps 15 km in length. The links are full duplex. The rate of transmission errors on the links, the switching time at B, and the time to transmit an ACK are all negligible and can be ignored. A large file is to be sent from node A to node D, and there is no other traffic on the links. Packets are 1024B, including headers.

a. Ignoring reliability and packet headers, what is the maximum throughput that can be achieved? Explain.

\[ \text{min}\{4 \text{ Gbps}, 50 \text{ Mbps}, 60 \text{ Mbps}\} = 50\text{Mbps} \]

The maximum throughput is determined by the bottleneck link speed.

b. What is the round-trip time from A to D?

*When using speed of light as \(3 \times 10^8\):*

Since the processing time and ACK transmit time are negligible, we have

\[ \text{RTT} = \text{transmit time} + 2 \times \text{propagation delay} \]

\[ = 8192/(4 \times 10^9) + 8192/(100 \times 10^6) + 8192/(50 \times 10^6) + 2 \times ((1000 \times 10^3)/(3 \times 10^8) + (35 \times 10^3)/(3 \times 10^8)) \]

\[ = 0.002048 \text{ ms} + 0.08192 \text{ ms} + 0.16384 \text{ ms} + 2 \times (3.333 + 0.117) \text{ ms} \]

\[ = 7.147 \text{ ms} \]

The propagation delay on the first link is the dominant component. Both transmit times are included. However, the transmit time on the second link is more significant.

*OR*

*When using speed of light as \(2 \times 10^8\):*

Since the processing time and ACK transmit time are negligible, we have

\[ \text{RTT} = \text{transmit time} + 2 \times \text{propagation delay} \]

\[ = 8192/(4 \times 10^9) + 8192/(100 \times 10^6) + 8192/(50 \times 10^6) + 2 \times ((1000 \times 10^3)/(2 \times 10^8) + (35 \times 10^3)/(2 \times 10^8)) \]

\[ = 0.002048 \text{ ms} + 0.08192 \text{ ms} + 0.16384 \text{ ms} + 2 \times (5 + 0.175) \text{ ms} \]

\[ = 10.597808 \text{ ms} \]

The propagation delay on the first link is the dominant component. Both transmit times are included. However, the transmit time on the second link is more significant.

c. What is the round-trip bandwidth delay product for the path from A to D? (Specify the units you use.)

\[ 50\text{Mbps} \times 7.147 \text{ ms} = 357350 \text{ bits} = 44668.75 \text{ B} = 43.62 \]

*OR* \[ 50\text{Mbps} \times 10.597808 \text{ ms} = 529890.4 \text{ bits} = 66236.3 \text{ B} = 64.68 \]
d. Suppose an end-to-end sliding window protocol is used with SWS = RWS. What is the optimal value for SWS?

Since the round-trip pipe holds 43.62 packets, SWS = RWS = 44 would be optimal. So 6 bit sequence numbers suffice.

OR

Since the round-trip pipe holds 64.68 packets, SWS = RWS = 65 would be optimal. So 7 bit sequence numbers suffice.

e. What would happen if you use an SWS value many times larger than the value you suggested in part (d)?

If SWS is very large, it can cause the queue at B to be very large. If the buffer space at B is limited, it could cause B to drop packets.

The solution which use $3 \times 10^8$ as the speed of light in fiber will also be considered as correct.

5. TCP RTT Estimation

One difficulty with the original TCP SRTT estimator is the choice of an initial value. In the absence of any special knowledge of network conditions, the typical approach is to pick an arbitrary value, such as 4 seconds, and hope this will converge quickly to an accurate value. If this estimate is too small, TCP will perform unnecessary retransmissions. If it is too large, TCP will wait a long time before retransmitting if the first segment is lost. Also, the convergence might be slow.

a. Choose $\alpha = 0.8$ and SRTT(0) = 1 seconds, and assume all measured RTT values = 0.6 second with no packet loss. What is SRTT(12)? Recall,

\[
SRTT(k + 1) = \alpha \cdot SRTT(k) + (1 - \alpha) \cdot RTT(k + 1).
\]

Describe your solution approach AND provide the numerical result (approximate to the 4th decimal place)

Based on the formula and $\alpha = 0.8$, we can calculate the nth SRTT value using:

\[
SRTT(n) = \alpha^n \cdot SRTT(0) + (1 - \alpha) \cdot RTT \sum_{j=0}^{n-1} \alpha^j.
\]

So with the given initial conditions, we can get SRTT(12):

\[
SRTT(12) = (0.8)^{12} \times 1 \times (1-0.8) \times 0.6 \times (0.8^0 + 0.8^1 + 0.8^2 + 0.8^3 + 0.8^4 + 0.8^5 + 0.8^6 + 0.8^7 + 0.8^8 + 0.8^9 + 0.8^{10} + 0.8^{11})
\]
\[ S_{RTT}(12) = 0.0687 \times 0.2 \times 0.6 (1+0.8+0.64 + 0.512 + 0.4096 + 0.3277 + 0.2621 + 0.2097 + 0.1668 + 0.1342 + 0.1074 + 0.0859) \]

\[ S_{RTT}(12) = 0.03838 \]

b. Using the same values as in part a), what happens if we use \( \alpha = 0.7 \) or \( \alpha = 0.9 \)? Provide a numerical result for \( S_{RTT}(12) \) in both cases, then describe the effect of a larger or smaller \( \alpha \) on the RTT estimation procedure.

Based on the formula and \( \alpha = 0.7 \), we can get \( S_{RTT}(12) \), and similarly, using \( \alpha = 0.9 \), \( S_{RTT}(12) \). From the calculation, we can conclude that a larger \( \alpha \) puts more weight put on the initial value and so results in a slower convergence time to the real RTT.

For \( \alpha = 0.7 \):

\[ S_{RTT}(12) = (0.7)^{12} \times 1 \times (1-0.7) \times 0.6 \times (0.7^0 + 0.7^1 + 0.7^2 + 0.7^3 + 0.7^4 + 0.7^5 + 0.7^6 + 0.7^7 + 0.7^8 + 0.7^9 + 0.7^{10} + 0.7^{11}) \]

\[ S_{RTT}(12) = 0.0138 \times 0.3 \times 0.6 (1+0.7+0.49 + 0.343 + 0.2401 + 0.1681 + 0.1177 + 0.0824 + 0.0577 + 0.04035 + 0.0282 + 0.0198) \]

\[ S_{RTT}(12) = 0.0271 \]

For \( \alpha = 0.9 \):

\[ S_{RTT}(12) = (0.9)^{12} \times 1 \times (1-0.9) \times 0.6 \times (0.9^0 + 0.9^1 + 0.9^2 + 0.9^3 + 0.9^4 + 0.9^5 + 0.9^6 + 0.9^7 + 0.9^8 + 0.9^9 + 0.9^{10} + 0.9^{11}) \]

\[ S_{RTT}(12) = 0.2824 \times 0.1 \times 0.6 (1+0.9+0.81 + 0.729 + 0.6561 + 0.5905 + 0.5314 + 0.4783 + 0.4305 + 0.3874 + 0.3487 + 0.3138) \]

\[ S_{RTT}(12) = 0.1215 \]

c. What is the retransmission ambiguity problem addressed by the Karn-Partridge algorithm? How does the algorithm avoid the ambiguity?

An acknowledgement does not necessarily acknowledge the most recent transmission of a packet. Therefore, any ACKs that are received for a packet that was transmitted more than once are not used in calculating the estimated RTT.
6. **TCP Slow Start**

Although slow start with congestion avoidance is an effective technique for coping with congestion, it can result in long recovery times in high-speed networks.

a. Assume a RTT delay of 120 ms and a link with an available bandwidth of 2 Gbps and a segment size of 1024 bytes. Determine the window size needed to keep the pipe full and the time it will take to reach that window size after a timeout using the Jacobson Algorithm, assuming that the sender was transmitting at the full window before the timeout.

The size of the window (in bits) is $100 \text{ms} \times 2 \text{Gbps} = 200 \times 10^6 \text{bits}$. Each packet has 8192 bits, so the window size is:

$$\left\lceil \frac{240 \times 10^6 \text{bits}}{8192 \text{ bits/packet}} \right\rceil = 29296.875 \text{ packets}$$

After the timeout, our congestion threshold is half the full window size (14,648.44 packets). TCP will use exponential growth until the congestion threshold is passed, at which point additive increase begins. Therefore, we can use exponential increase to a window size of $2^{14} = 16384$. In the 14th RTT, we will transmit 14,649 packets out. Now we do additive increase to get to the full window. This will take $29297 - 14,469 = 14,828$ RTTs using additive increase (one packet increase in the window per RTT). So, the total time to reach the full window is $(14 + 14828) \times \text{RTT} = 12,220 \times 120 \text{ms} = 1781.04 \text{ s}$.

b. **Repeat for a segment size of 32 Kbytes.** (Assume 1K = 1024)

The window size (in bits) does not change, but we need to refresh the window size for packets of 16KB. Each packet is 131’072 bits, so the window is:

$$\left\lceil \frac{240 \times 10^6 \text{bits}}{262144 \text{ bits/packet}} \right\rceil = 915.53 \text{ packets}$$

This makes the congestion threshold 916 packets, as what we described in a, there should be $(9+458) \times 0.1 = 56.04 \text{ s}$.