CSC458 Problem Set #2

Filip Tomin
1001329984

1. If a part of an IP address was dedicated to geographical position, there would be even less available possible addresses for use in heavily populated areas. It may also add confusion for providers who covered multiple areas, as they would then have two sets of prefixes depending on the area, rather than one central address.

2. X should not advertise the path to B. Other than the fact that a consumer should not share its providers with its providers, there might also be the chance that ISP B doesn't have any path to that prefix and isn't connected to ISP A. In that case, ISP B might try to record the path going through ISP X, and in this provider-consumer scenario, X should not be handling any traffic from A or B.

3. A mobile host might be on a foreign network, not directly connected to its host network. In this case, DHCP may be able to give the mobile host an IP for use while on the foreign network, but any packets destined for that mobile host would still need to route through the host network, thus some more setup than just a new IP from DHCP would be required.

4. The main downside of the host agent system is distance. A mobile host would be, by definition, mobile, and could travel very far from the host. In this case, packets that might coincidentally have a very optimal path to the mobile host would have to first travel to the host agent which would then send it to the mobile host, increasing the travel time of the packet, leading to delays and more potential errors.

5. (a) The sequence number is an unsigned 32-bit integer, and would wrap around after it reaches FFFFFFFF or 4294967295 bytes are transferred. So: $4294967295 \times 8 = 34359738360$ bits. At 1-Gbps, with that many bits, it would take about 34.36 seconds for the sequence number to wrap.

   (b) The timestamp would have to increment 4294967295 times to wrap, dividing by 1000 (per each sequence wrap) means 4294967.295 sequence wraps, multiplying that by the time for a sequence wrap (34.36 seconds) gives us approximately 147573952.52 seconds or about 4.68 years.

6. (a) When A receives the data, it will send B an ACK with the new sequence number. Since the connection is idle, the new sequence number will be higher than B's

next send sequence number, and the ACK is unacceptable. B will send another ACK stating its actual sequence number and A will then reject the forged data.

(b) Each end would send ACKs for the forged data, these would be unacceptable, and new ACKs would be sent. These would also turn out to be unacceptable, and new ACKs will be sent. This process would repeat indefinitely until one ACK is lost. If A later sent 200 bytes to B, B would disregard the first 100 as duplicate, and only deliver the second 100, then send a valid ACK back to A.

7. For a reset, if it is outside the receive window then it is ignored, otherwise the connection is reset. For a finish, it is also ignored if it is outside the receive window, but if the sequence number is not NextByteExpected then the segment is buffered until preceding segments are received.

8. It is important for SYNs and FINs to each take up one sequence number because otherwise there is a possibility for ambiguity to arise with acknowledgments for FIN, where the sender would be unsure of whether the ACK was for the FIN or for the data before it.

9. The graphs could not be drawn as the method of measuring load is useless. With the current method of packets per second from A to B, because of the R-B link, the load will always be 1. Another way to measure load is necessary, and a more appropriate measurement would be to consider the amount of packets within R's queue.

10. (a) The fairness index function is:

$$f(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2} = \frac{511225}{546125} \approx 0.936$$

(b) Same as above but with an additional 1000 KBps flow:

$$f(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} x_i^2} = \frac{2941225}{6655350} \approx 0.442$$

11. (a) The window size is doubled for each ACK received. Starting at 1 KB, 10 ACKs need to be received to reach 1024 KB (1 MB), thus it would take 10 RTTs to reach this point.

(b) Assuming the window never stops multiplying, reaches the receive window and never decreases, then it would take 19 RTTs to send the 10MB file:

RTTs	Current Window Size MAX:1024KB	Total Data Sent
0	1KB	1KB
1	2KB	3KB
2	4KB	7KB
3	8KB	15KB
4	16KB	31KB
5	32KB	63KB
6	64KB	127KB
7	128KB	255KB
8	256KB	511KB
9	512KB	1023KB
10	1024KB (MAX)	2047KB
11	MAX	3071KB
12	MAX	4095KB
13	MAX	5119KB
14	MAX	6143KB
15	MAX	7167KB
16	MAX	8191KB
17	MAX	9215KB
18	MAX	10239KB
19	MAX	**10MB**

(c) Time to send file = $19 \times 50 = 950$ms.
Bits in file = $10 \times 8 = 80$Mb.
Throughput = $\frac{80}{0.95} = 84.21$Mbps.
Percentage used = $\frac{84.21}{1000} = 0.0842 = 8.42\%$ utilization.

12. From 0.2 to 0.5 is slow start on startup. After the timeout at approximately 1.9, from 1.9 to 2.5 is slow start after timeout and from 2.5 to 5.5 is linear-increase congestion avoidance. It seems like a reset occurs at 5.5, so from then on would also be slow start on startup, with 5.7 on being linear-increase congestion avoidance.
From T = 0.5 to T = 1.9, the sender is not receiving any ACKs and so the congestion window does not change size. At 1.9, there is a timeout and the window is reset.
The feature included in 6.28 but not in 6.11 is fast retransmit, shown in use at T = 5.5 (for 6.28), there is no long section where the congestion window is flat.
Both 6.13 and 6.28 are lacking the fast recovery feature, shown by the drop in the congestion window at T = 5.5 (for 6.28)