Roann Yanes (NetID: ryanes)
CSE 30264 Computer Networks
Professor Dong Wang
December 5, 2018

Homework 5

Problem 1:

The end-to-end argument in network system design asserts that a function (for example, reliable data delivery) should not be provided in the lower levels of the system unless it can be completely and concretely/correctly implemented at that level (*Lecture 21: End-to-End Protocols*). While the end-to-end argument argues in favor of the TCP/IP approach, it does allow for functions to be incompletely provided at a low level as a performance optimization technique. For example, it is consistent with the end-to-end argument to perform error detection (e.g., CRC) on a hop-by-hop basis (i.e. CRC is implemented at the data link (connectivity) level while we still have check-sum at end-to-end level); detecting and retransmitting a single corrupt packet across one hop is easier and more efficient than having to retransmit an entire file end-to-end (*Lecture 21: End-to-End Protocols* and *Computer Networks: A Systems Approach* (p. 399)).

Problem 2:

If the advertised window size in TCP goes to zero, potential deadlock between the sender and receiver could happen because an advertised window of 0 implies that the sending side cannot transmit any data, even though data it has previously sent has been successfully acknowledged. Furthermore, not being able to transmit any data means that the send buffer fills up, which ultimately causes TCP to block the sending process. As soon as the receiving process starts to read data again, the receive-side TCP is able to open its window back up, which allows the send-side TCP to transmit data out of its buffer. When this data is eventually acknowledged, LastByteAcked is incremented, the buffer space holding this acknowledged data becomes free, and the sending process is unblocked and allowed to proceed. Once the receive side has advertised a window size of 0, the sender is not permitted to send any more data, which means it has no way to discover that the advertised window is no longer 0 at some time in the future. TCP on the receive side does not spontaneously send non-data segments; it only sends them in response to an arriving data segment. To solve this problem, in TCP, whenever the other side advertises a window size of 0, the sending side persists in sending a segment with 1 byte of data every so often. It knows that this data will probably not be accepted, but it tries anyway, because each of these 1-byte segments triggers a response that contains the current advertised window. Eventually, one of these 1-byte probes triggers a response that reports a nonzero advertised window. The reason the sending side periodically sends this probe segment is that TCP is designed to make the receive side as simple as possible—it simply responds to segments from the sender, and it never initiates any activity on its own. This is an example of the smart sender/ dumb receiver rule (*Lecture 21: End-to-End Protocols* and *Computer Networks: A Systems Approach* (p. 411).

Problem 3:

(a) R1 cannot be congested because even in the worst-case scenario (which is when H1, H2 H3, and H4 all transmit 1MB/sec each to H8, and when H5, H6, H7, and H8 all transmit 1MB/sec each to H1), the aggregated incoming data speed for router R1 is not greater than the aggregated outcoming data speed for router R1 (as 4 MB/sec of data comes from both the left and right of R1, which causes the link to reach maximum capacity but not to become congested). For the link R2 → R1, we have incoming 4 MB/s worth of data and outgoing 4 MB/sec worth of data (this is also the case for the link R1 → R3). R1 cannot become congested because the aggregated incoming data speed will never exceed the aggregated outgoing data speed.

(b) A traffic pattern that congests R2 alone is as follows: H5 and H6 send 1MB/sec each to H4, and H7 and H8 send 1MB/sec each to H3. With this traffic pattern, R2 becomes congested because it needs to transmit 4 MB/sec to R5 but is unable to because the link bandwidth from R2 → R5 is 2 MB/sec (R5 is not congested as it receives data from R2, and this traffic is evenly divided between H3 and H4).

(c) A traffic pattern that congests R7 alone is as follows: H5 and H6 send 1MB/sec each to H8. With this traffic pattern, R7 receives 2 MB/sec worth of data and needs to transmit to H8; it cannot forward this data to H8 because the link R7 → H8 has a link bandwidth of 1 MB/sec, therefore, R7's aggregated incoming data speed is greater than its outgoing data speed, and R7 alone is congested.

Problem 4:

(a) Round robin

Packet	Size	Flow	Weighted F_i
1	**180**	**1**	**90**
2	**30**	**1**	**105**
3	**200**	**1**	**205**
4	**70**	**1**	**240**
5	**60**	**2**	**20**
6	**150**	**2**	**70**
7	**240**	**2**	**150**
8	**160**	**3**	**160**
9	**80**	**3**	**240**
10	**90**	**3**	**330**

Packet Order: **Packet 1, Packet 5, Packet 8, Packet 2, Packet 6, Packet 9, Packet 3, Packet 7, Packet 10, Packet 4**

(b) Fair queuing

$F_i = \max(F_{i-1}, A_i) + P_i$

Packet	Size	Flow	F_i
1	**180**	**1**	**180**
2	**30**	**1**	**210**
3	**200**	**1**	**410**
4	**70**	**1**	**480**
5	**60**	**2**	**60**
6	**150**	**2**	**210**
7	**240**	**2**	**450**
8	**160**	**3**	**160**
9	**80**	**3**	**240**
10	**90**	**3**	**330**

Packet Order: **Packet 5, Packet 8, Packet 1, Packet 2, Packet 6, Packet 9, Packet 10, Packet 3, Packet 7, Packet 4**

(c) Weighted queuing with flow 1 having weight 2, flow 2 having weight 3 and flow 3 having weight 1

$F_i = \max(F_{i-1}, A_i) + P_i/weight$

Packet	Size	Flow	Weighted F_i
1	**180**	**1**	**90**
2	**30**	**1**	**105**
3	**200**	**1**	**205**
4	**70**	**1**	**240**
5	**60**	**2**	**20**
6	**150**	**2**	**70**
7	**240**	**2**	**150**
8	**160**	**3**	**160**
9	**80**	**3**	**240**
10	**90**	**3**	**330**

Packet Order: **Packet 5, Packet 6, Packet 1, Packet 2, Packet 7, Packet 8, Packet 3, Packet 4, Packet 9, Packet 10**