

Internet Networking - Homework 5

Yosef Goren & Ori Evron

June 30, 2023

Contents

1	Max/Min Fairness	1
1.1	Splitting the problem	1
1.2	Clockwise Flows	1
1.3	Counter-Clockwise Flows	2
2	TCP Reno Congestion Control	2
2.1	2
2.2	2
2.3	2
2.4	2
2.5	3
2.6	3
2.7	3
3	TCP Congestion Control	3
4	TCP Cubic Congestion Control	3

1 Max/Min Fairness

1.1 Splitting the problem

First we note that we can significantly simplify the solution by splitting the problem into two distinct parts: Since there is no contention between the flows which go clockwise and those that go counter-clockwise, we can address each of these classes separately - and the combined results will be equivalent to if we were to run the algorithm separately on each one.

1.2 Clockwise Flows

In the first round, we see that the link between node 3 and 4 is the bottleneck. This happens because as seen in the graph, the number of flows that pass through it is maximal, so, all the passing flows have a value of $1/6$. In the second round,

all the flows that passes through the lin between 3 and 4 are removed, and the new bottleneck is the link between nodes 5 and 6, the flow between 5 to 6 and 4 to 6 each get $1/4$. In the thierd round, the only flow left is the flow between 4 to 5, it has $10 - 5/6 - 1/4 = 8 + 11/12$.

1.3 Counter-Clockwise Flows

In the first round, we see that the link with the minimal allocatable bandwidth per flow (a.k.a the bottleneck) is the link from 2 to 1 - which has 8 flows contending for it's bandwith, while it can only pass 1 unit - meaning each flow will only recive $\frac{1}{8}$ of a unit.

At this point - we can eliminate all flows involved in that link - which leaves us with a single flow from node 1 to node 6. Note that the bandwidth left on that link is now 9. This link is also the next bottleneck. Hence the $1 \rightarrow 6$ flow recived 9 bandwidth units in total - and all flows have been eliminated.

To sum up the final bandwidths recived by each flow:

- $1 \rightarrow 6$: 9 units of bandwidth.
- All ofther flows: $\frac{1}{8}$ units of bandwidth.

2 TCP Reno Congestion Control

2.1

The Transmission rounds where the **ssthresh** value changes are the ones where a drop is seen due to a duplicate ack: rounds 16 and 22. After round 16 - **ssthresh** is set to 21 segments and after round 22 it is set to about 13 segments.

2.2

The time intervals during which the algorithm operates in "slow start" are between 0 and 6, and from 23 until the end.

2.3

This would cause **cwnd** and **ssthresh** to drop to about 4 segments.

2.4

After the 16th round, the packet loss was detected through duplicate ACKs. This conclusion is supported by the fact that the congestion window (**cwnd**) was not set to 1.

If the case were a timeout, the **cwnd** would have been set to 1.

2.5

The round 22 packet loss was discovered by a timeout. We know this is the case due to the way `cwnd` has changed: since it drops directly to 1 instead of halving its value - we know it was a timeout.

2.6

The initial value for `sssthresh` was 32 or 33, we know this since this is the point where the congestion control policy changes from slow start to congestion avoidance.

2.7

By essentially doing an integral over the graph we can find how many segments have been sent up to a certain point. Since we want to know when the 100'th segment was sent - we apply this process up to the point where the accumulating integral is equal to 100. After transmission round the total is 63, and after transmission round the total is 126 - so the 100'th segment must have been sent at the 7'th round.

3 TCP Congestion Control

4 TCP Cubic Congestion Control