Congestion Control and Synchronization

Joe Hoffert and Jeremy Weatherford

Abstract

In this project, we investigated congestion control synchronization in TCP senders. When multiple TCP flows are sharing the same bottleneck link, congestion of the link in a synchronized situation can cause an overreaction, with all or most of the senders backing off simultaneously. A more appropriate response would be for the minimum number of senders to be affected by any given congestion event. Our approach of varying the RTTs for each sender shows conclusively that dissimilar RTTs reduce synchronization, and thus reduce the magnitude of oscillations in the router queue size.

Motivation

TCP congestion control, although it has effectively reduced the dangers of congestion collapse, can cause dramatic underutilization of network bottleneck links. This can have direct financial implications for the owners of those links, whether large or small. In this project, we investigate the causes of congestion control synchronization, and approaches for mitigating its effects.

Initial Directions

Our initial approach was two-fold: 1) simulate a router in software and modify parameters to notice and affect flow synchronization within this bottleneck router and 2) model flow synchronization in the network simulation tool ns-2 and modify parameters to notice and affect flow synchronization in the ns-2 bottleneck router.

The simulated router was designed to attach to the TCP senders and sinks from Assignment 2 in order to model a droptail queue for testing in a controlled network environment. Although we were able to control the network using this method, the simulation was highly unpredictable, being subject to the whims of the kernel scheduler. We discussed the possibility of reimplementing this configuration in a single process, and decided that ns-2 provided exactly the framework that we were looking for.

The initial focus with the ns-2 simulations was to reduce or eliminate the number and/or duration of empty queue events in the bottleneck router. This was motivated by wanting to reduce or eliminate underutilization of a bottleneck link. When a link's router queue is empty, the link is "quiet." Assuming that there are TCP senders who could have transmitted during that time, this represents underutilization of the critical resource.

The first goal with the ns-2 simulations was to create a scenario that caused empty bottleneck router queues. After trying a few different configurations we found the following scenario to produce over 10 empty router queue events in a 60 second period: a bottleneck router with a bandwidth of 0.3 Mb/s and latency of 200ms plus 10 FTP/TCP/NewReno flows going into and out of the bottleneck router with a latency of 50

ms on either side of the router. With this baseline configuration resolved, we began manipulating different parameters to see the effect on the number of empty queue events.

The table below shows some of the different configurations that were tried. In short, we tried separately modifying the number of flows, the RTTs of the flows (first by modifying the RTT of a single flow while keeping the others constant then having 2 groups with different RTTs), and the size of the bottleneck router queue. Relevant to the router queue size we tried setting the size based on the bit-rate delay product and the over-square-root formulae. Surprisingly, queue size based on these formulae did not decrease the number of empty router queue events. At a minimum this was conclusive evidence that these formulae are not necessarily good general purpose solutions even though there may be certain configurations for which they are well suited. Another surprising result was found using random early detection (RED) queues rather than droptail queues. We assumed using RED queues would decrease empty router queue events but this was not the case in the simulations we ran. Again, there may be certain configurations where RED queues reduce underutilization of bottleneck routers but this can not be said in all cases.

# of flows	**buffer size**	**empty queue events**	**Notes**
10	10	**10+**	Baseline case
5	15	**10+**	bit-rate delay product
10	4, 5, 6	**10+**	used over-square-root, i.e., (RTT*C)/sqrt(n), not promising
20	10	2	.
20	20	0	.
20	40	0	
15	15	2	
15	15(RED)	**10+**	Unexpected results w/ RED queue
10	10	6	plus 200ms latency on one link
10	10	**10+**	plus 300ms latency on one link
10	10	**10+**	plus 120ms latency on 1/2 links
10	10	3	plus 100ms latency on 1/2 source links
10	9	**10+**	.
10	15	3	up-front+short duration - promising
10	20	3	up-front+short duration - promising
10	25	1	up-front - promising
10	11	**10+**	.
10	12	8	up-front+short duration - markedly better than queue size of 11
10	13	**10+**	markedly worse than queue size of 12
10	14	4	almost as good as 15

As can be seen from the table above the only promising results came from modifying the size of the router queue. The empty queue events were fewer and of shorter duration as the router queue size was increased to around 15 packets. However, at this point we were encouraged to redirect our efforts not looking at empty bottleneck queue events but rather

focusing on router queue size oscillations. This lead to some promising results as outlined below.

Methods

All experiments described here were conducted using Network Simulator (ns-2). Initial experiments were tried using separate processes running over loopback or LAN connections, but were too uncontrolled to be useful (although this tack provided useful input and direction for ns-2 simulations). Similar problems arose with using PlanetLab as an experimental base. ns-2 has the great advantage of being entirely deterministic, allowing us to create an entirely isolated experimental environment.

The baseline network layout was as follows: the set of N senders each have individual links connecting them to router A, at one end of the bottleneck link, with bandwidth W. Router B, at the other end of the link, has individual links to each of the N receivers. All data transfers were one-way and continuous over the measurement period, with payload sizes of 1500 bytes (packetsize of 1460 bytes + header size of 40 bytes). The baseline situation had all RTTs identical, equal to 600ms (200 ms for the bottleneck router link and 50 ms on each side of the router, or 300ms each direction) and the size of the bottleneck router queue was 45 packets. This size is particularly large so that we would be better able to notice oscillations in the router's buffer size.

All senders and receivers modeled the TCP New Reno protocol. The router was modeled as a droptail queue, which is the most common policy among existing devices. ns-2 logs router and cwnd data with a granularity of 5ms.

Our first promising approach was to assign RTTs to each of the N flows in sequence, starting at 600ms, in steps of 5ms. Thus, no two flows would have an identical RTT. This is called the "stepped" approach below. We expected this to decrease the degree of synchronization of the senders by staggering the times at which they would be sending.

The second approach assigned RTTs at random from a linear distribution between 600 and 700ms. This is called the "random" approach below. It is similar to the "stepped" approach, but allows for the possibility of applying it to a large number of flows. In such a situation, always separating RTTs by 5ms would not be possible or desirable, but the linear distribution would ensure as much separation as possible within the given constraints.

Results

The data from our baseline case is shown in figures 1 and 2. The baseline case involved N=10 flows, with all RTTs equal to 600ms. Data was collected for 2 minutes using ns-2. The router buffer was 45 packets. Figure 1 shows the congestion window for each of the 10 senders. It is readily obvious that the senders are moving through the stages of TCP congestion control simultaneously, all reacting to congestion events at the same time. Figure 2 shows the router queue size over the same period of time. The impact of the synchronized congestion reactions can be seen in the magnitude of the drops in the router queue level.

The data from using the "stepped" approach are seen in figures 3 and 4. With N=10 flows, the RTTs were assigned from 600ms to 690ms. Figure 3 clearly shows that the congestion windows for the individual senders are no longer moving in tandem, and the impact of this can be seen on the router queue level fluctuations in figure 4. While the number of congestion events is the same, the magnitude of the oscillations has been significantly reduced.

We determined that the beneficial impact of the "stepped" approach significantly drops off when the step size is reduced to 1ms or below, which is the case seen in figure 7. However, a step size of 2ms produces results comparable to the step size of 5ms shown in figures 3 and 4.

The data from using the "random" approach is seen in figures 5 and 6. With N=10 flows, the RTTs were assigned from a linear distribution between 600ms and 700ms. Once again, the congestion control states of the senders have been desynchronized, reducing the magnitude of oscillations in the router queue. However, the impact is not significantly different from the stepped approach. The major benefit of the random approach is that it can be applied to any number of connections, whereas separating every RTT by 5ms is impractical for large numbers of connections.

Another improvement of the random approach over the stepped is in the area of fairness. Since all links have an equal chance of receiving a given RTT, no one connection is favored over another. Combined with a periodic reassignment of RTTs, fairness can be assured in the average case.

Conclusions

The degree of congestion control synchronization has a measurable impact on the utilization of the bottleneck link in a system. Modifying the RTTs in a synchronized system is sufficient to desynchronize the system, resulting in smaller oscillations in the router queue level. Doing so in a random fashion appears to be the best approach, since it can be scaled to any number of connections, and theoretically allows for a degree of fairness by averaging the additional delays across all senders. Further work would be necessary to analyze the scalability of this solution, however.

Figures

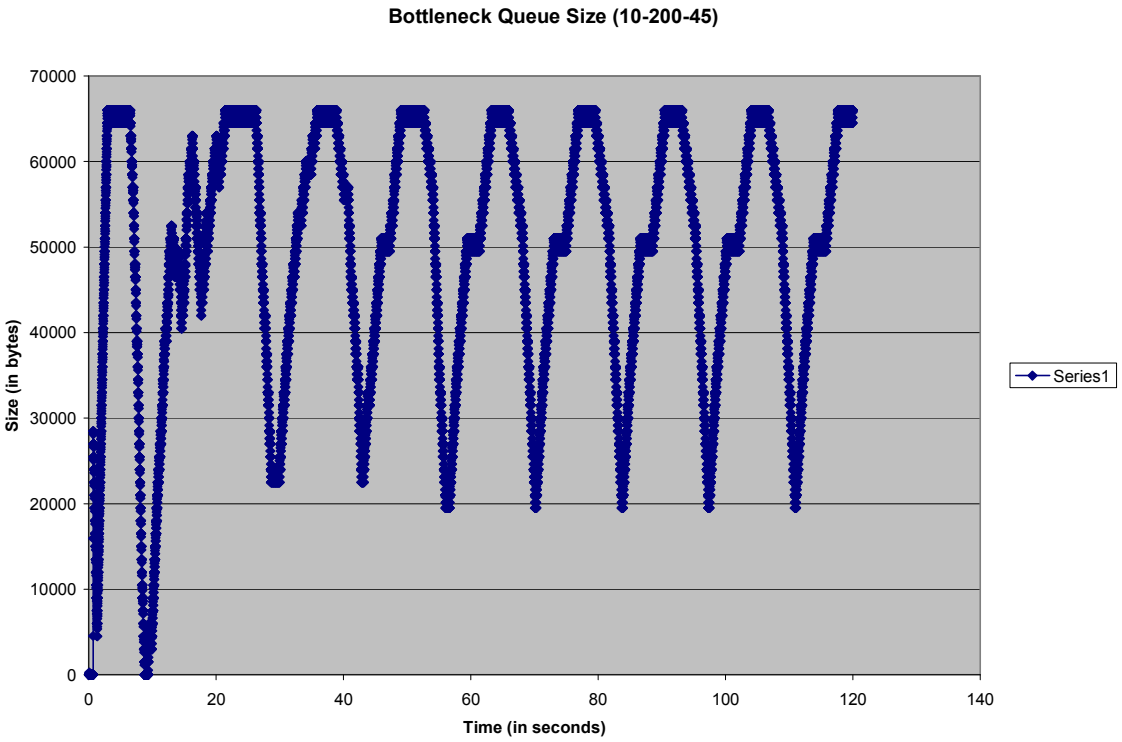

Figure 1: qm baseline

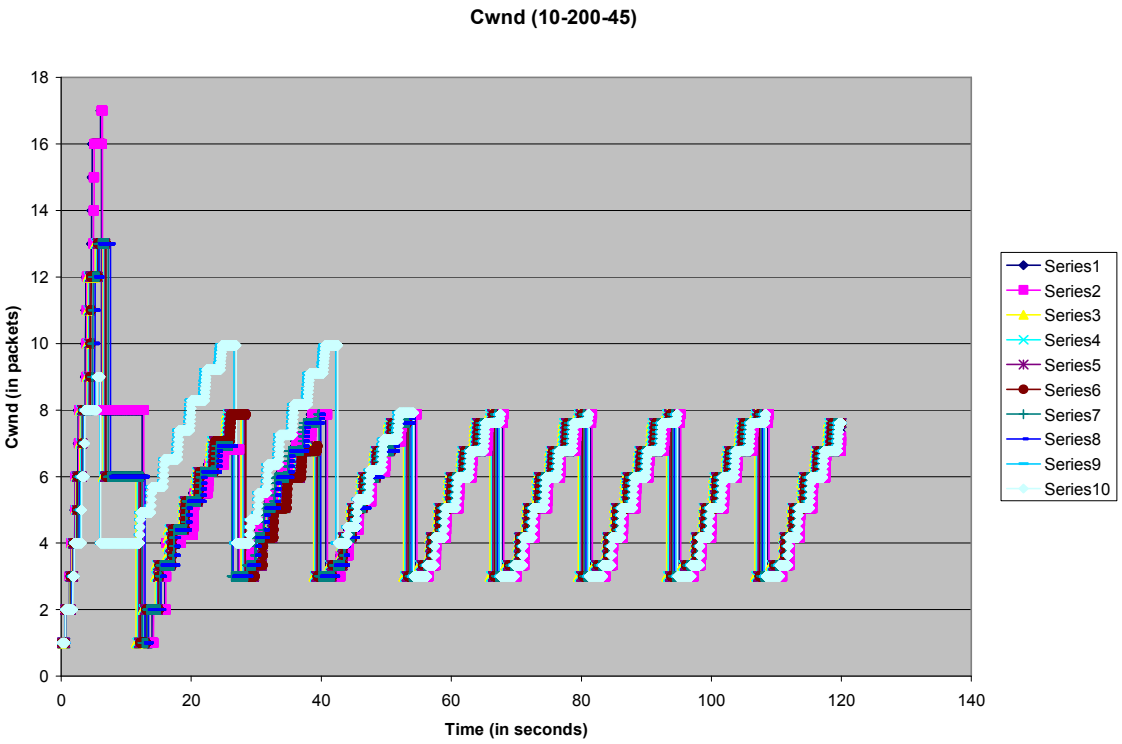

Figure 2: cwnd baseline

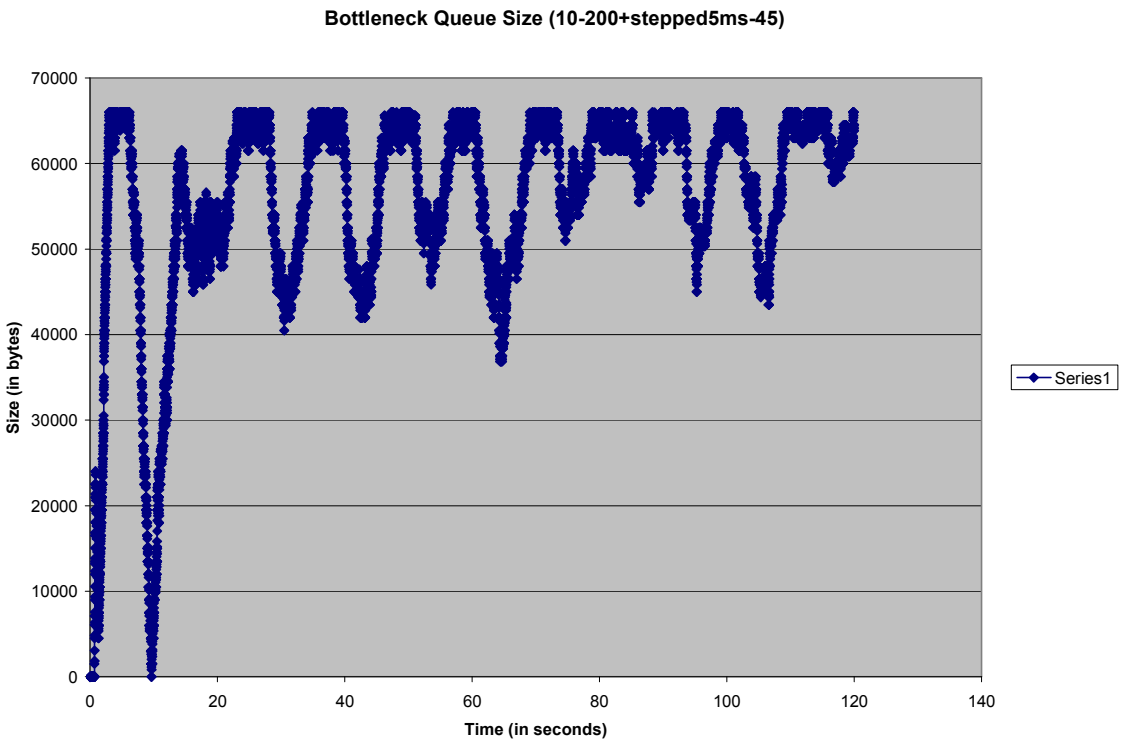

Figure 3: qm stepped5ms

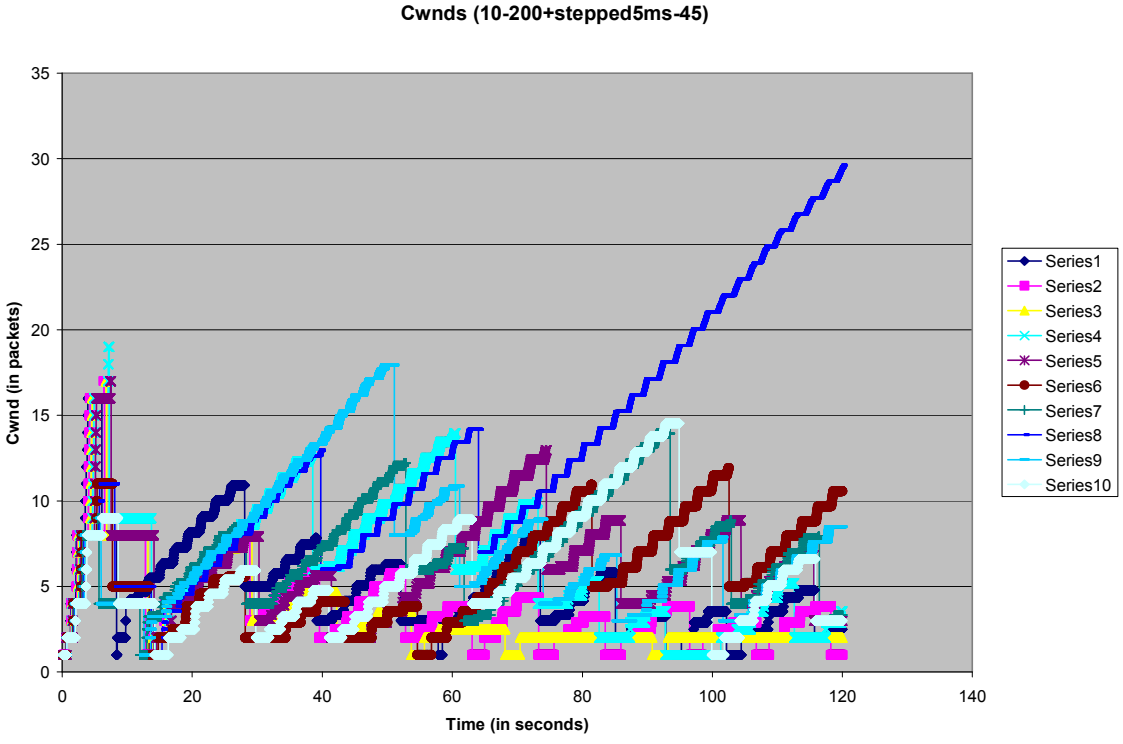

Figure 4: cwnd stepped5ms

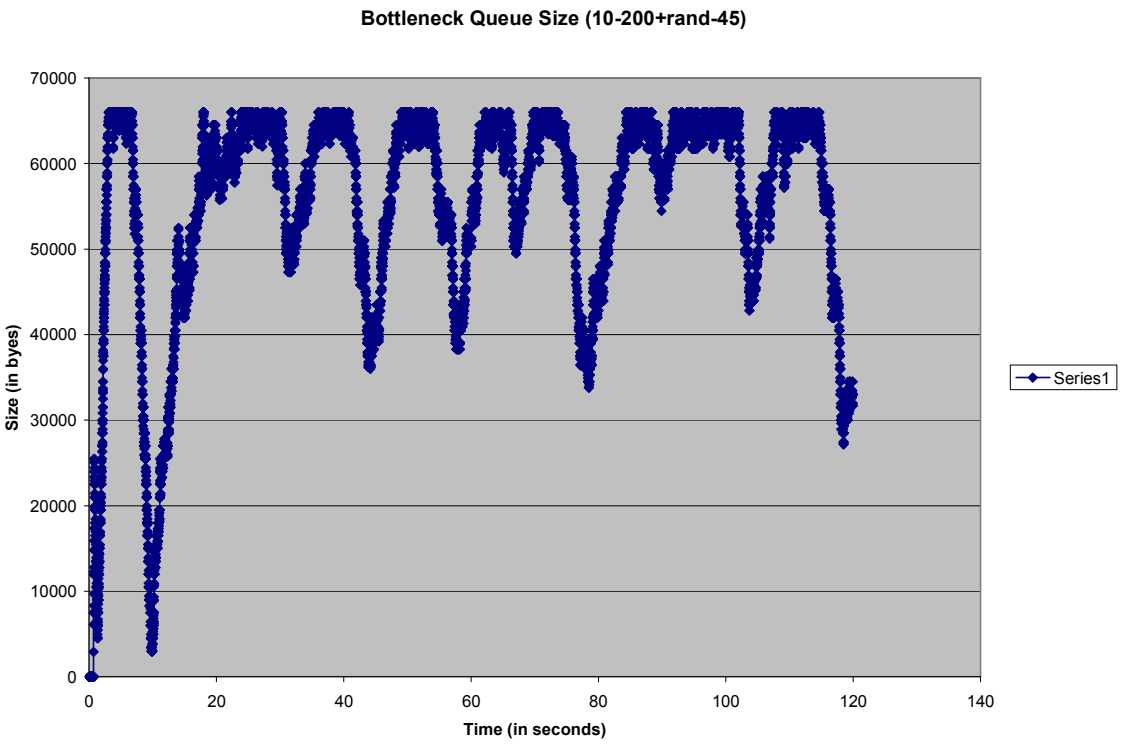

Figure 5: qm random

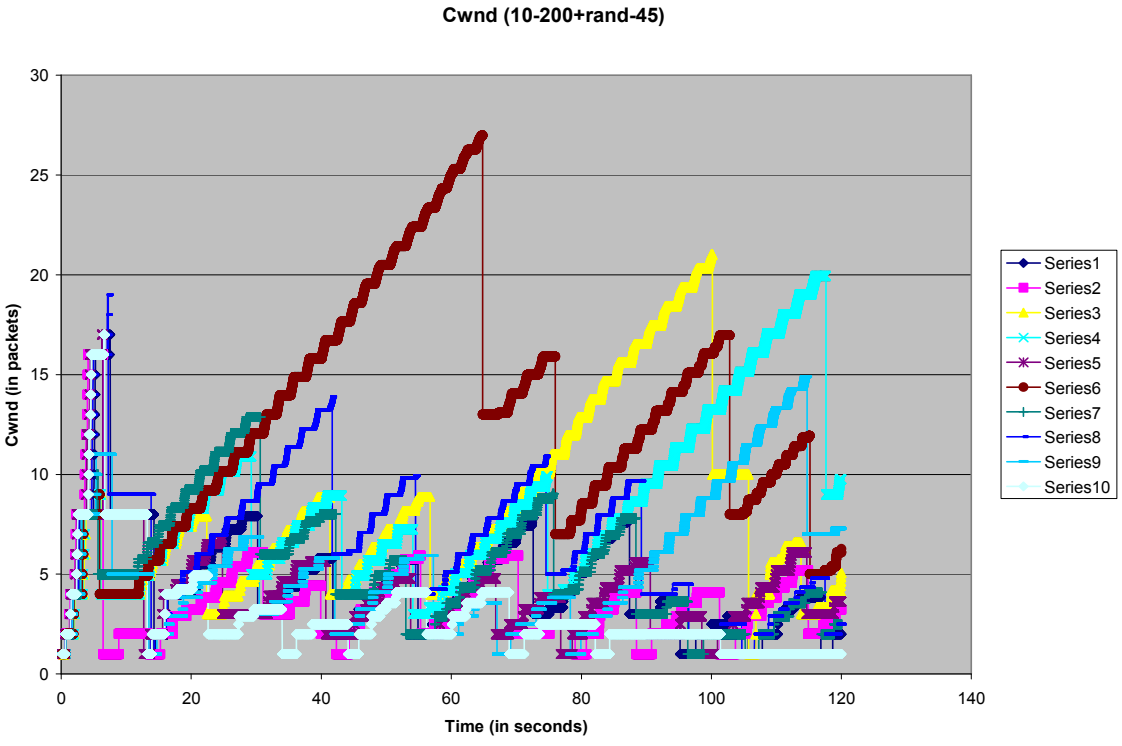

Figure 6: cwnd random

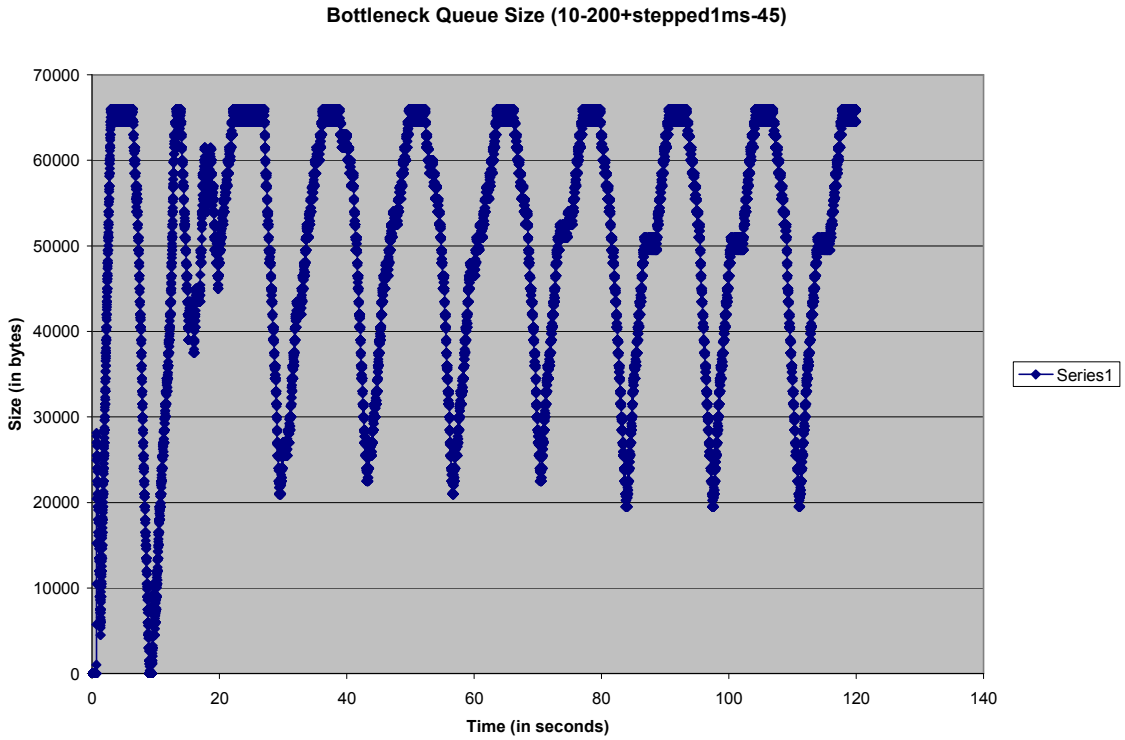

Figure 7: qm stepped1ms

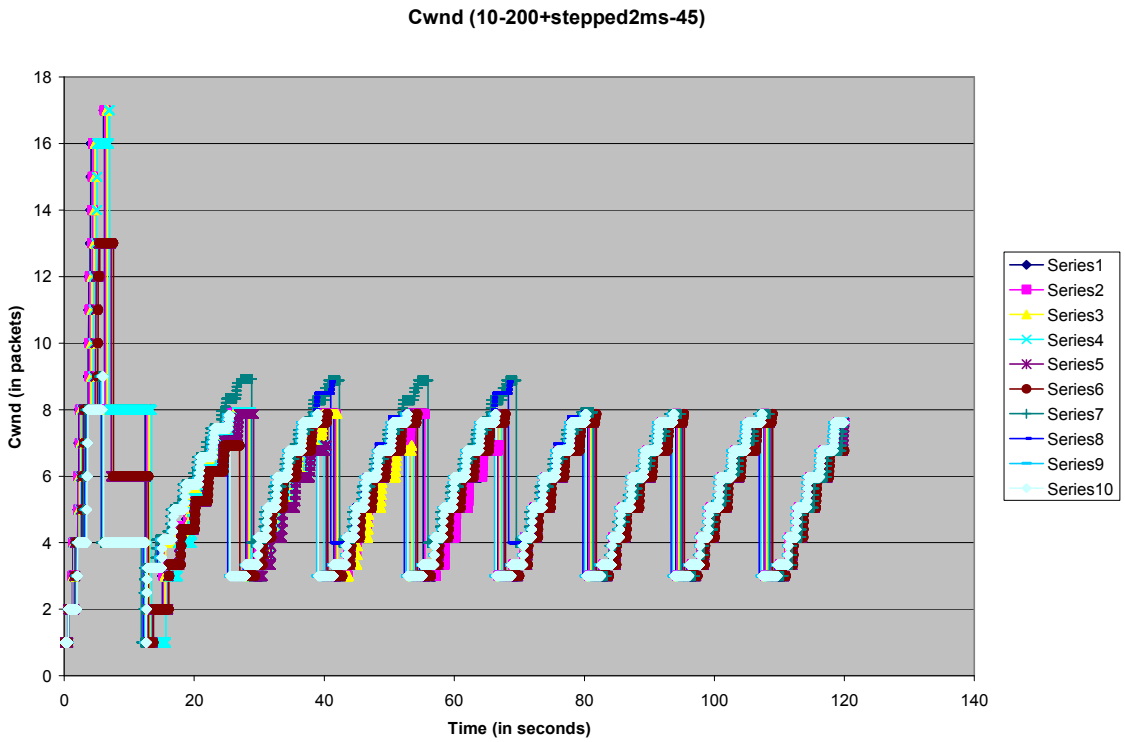

Figure 8: cwnd stepped1ms