

EECS 428 Final Project

Background

In prior investigations of large TCP transfers across a shared bottleneck link, we discovered that TCP's bandwidth-probing scheme leads to unfair sharing of the link among nodes with different round-trip times. To rectify this, a fair-queueing scheme, class-based queueing, was introduced at the bottleneck router.

CBQ largely ironed out the fairness issues, allowing the sender/receiver pairs with longer RTTs to obtain more of the bottleneck bandwidth by enforcing even sharing of the link, through fair forwarding of packets from the different arenas. However, persistent queues formed at the bottleneck router with CBQ – undesirable behavior which leads to longer queueing delays and heavier congestion.

Solution

Proportional Integration is an active queue management technique based on control feedback loops. It attempts to force the queue length to some predetermined value q_{ref} by dropping or marking (in the case of ECN) a packet with probability p :

$$p = p + a(q - q_{ref}) - b(q_{old} - q_{ref})$$

The difficulty in successfully using PI for queue management lies in selecting “good” values for a , b , and q_{ref} . These values are dependent on the number of flows entering the queue, the available bandwidth, and the RTT of the flows.

At first we tried using a C program which provided approximate values for the parameters through some simple calculations based on link capacity, RTT, sampling frequency, and number of flows, but these values turned out to be largely useless – ns scripts using the values would run for extraordinarily long times without finishing.

Thankfully (at least at first glance), provided with ns2 is a file, “picoeff.m”, which contains a similar Matlab function which will calculate a , b , and a sampling frequency based on number of flows, RTT, and link capacity. Using this function, we were able to determine values of $a = 3.04 * 10^{-6}$, $b = 3.01 * 10^{-6}$ for Arena 1; $a = 4.91 * 10^{-6}$, $b = 4.86 * 10^{-6}$ for Arena 2; and $a = 7.61 * 10^{-7}$, $b = 7.54 * 10^{-7}$ for Arena 3.

(In actual fact, these parameters were determined in part by setting the sampling frequency, normally calculated by the function, to be 400Hz. This is far higher than normal for a PI controller, but setting it lower produced undersampling behavior, indicated by large oscillations about q_{ref} .)

Later, after clearing up a misunderstanding of the values used in the C program, we were able to obtain working values from it using a frequency of 6Hz; this will be revisited later

Experiment/Results

In our experimentation, we used the Arena topology from earlier work in the D&D project. The backbone router at the West end of the link (with the Elephant sources) was provided with PI-controlled CBQ queues whose parameters were adjusted to the values obtained through our research.

A number of different q_{ref} values (15-60) were tested with the computed parameters obtained from the Matlab script. Figs. 1-3 show some statistics for each arena obtained across these values.

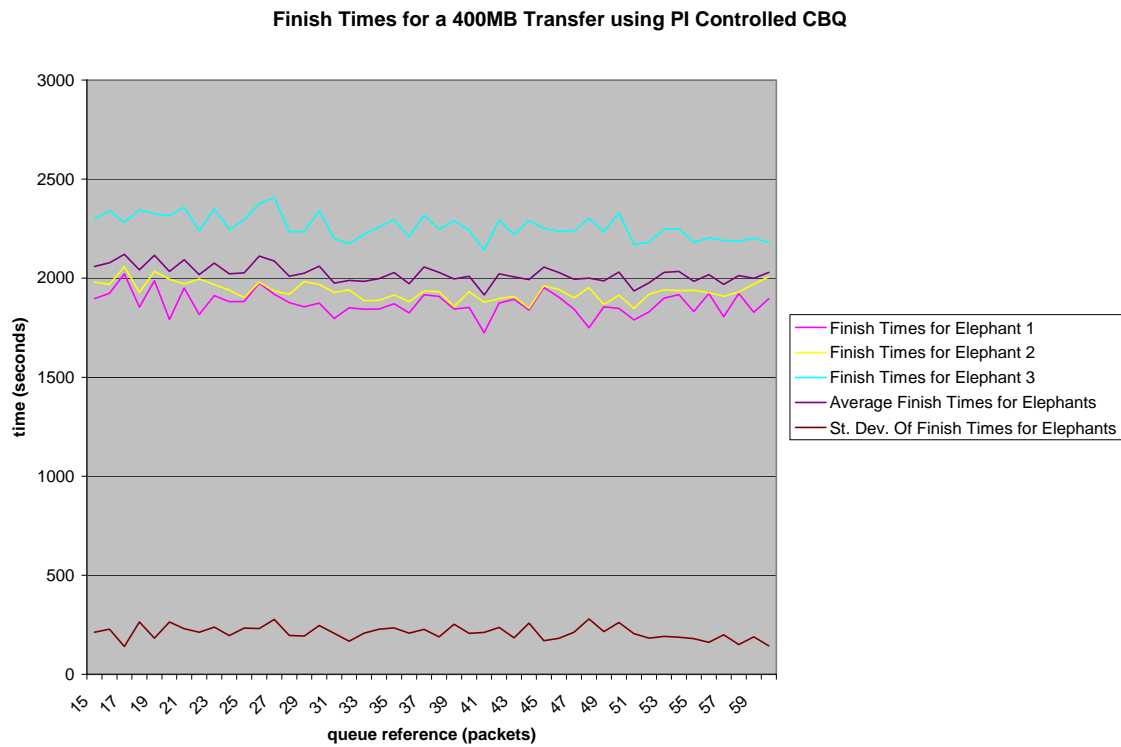


Fig. 1 (Finish Times)

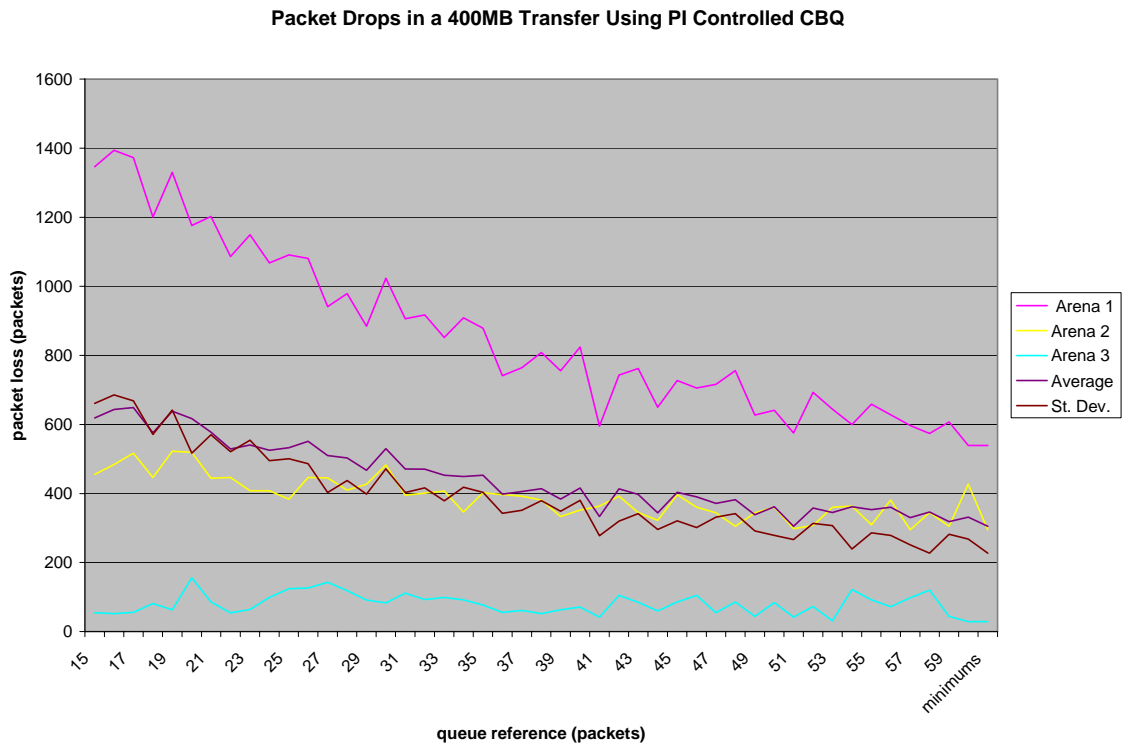


Fig. 2 (Packets dropped)

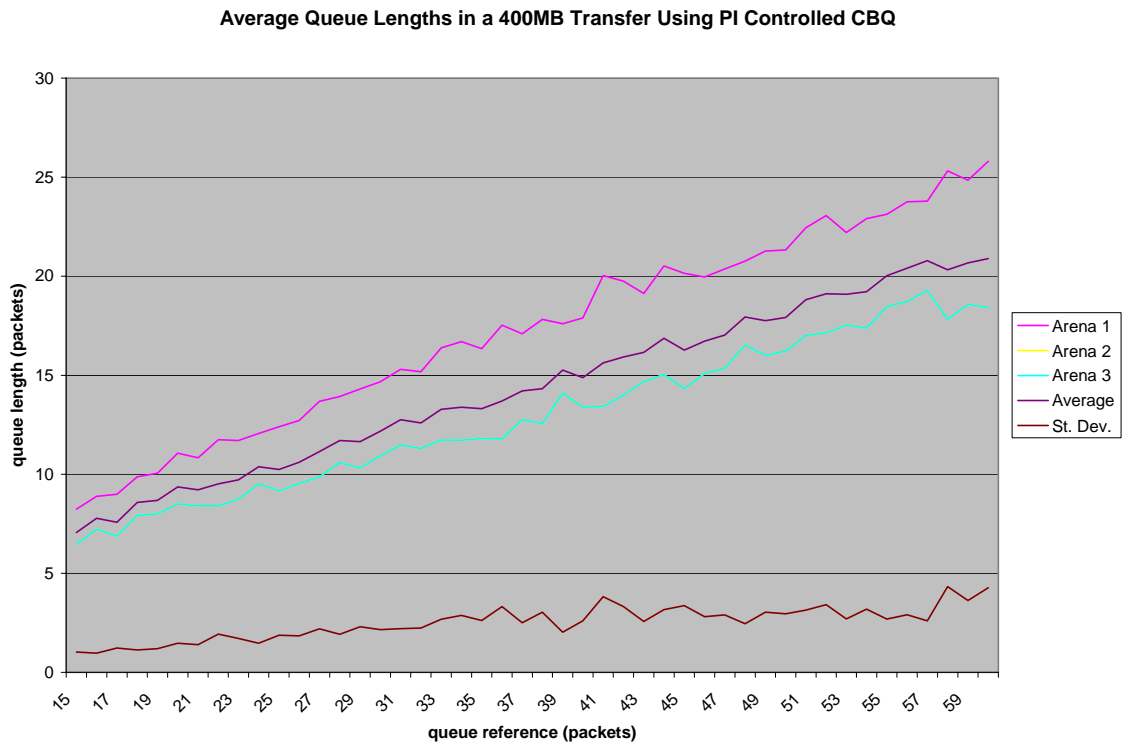


Fig. 3 (Avg Queue Length)

Figures 2 and 3 are particularly revealing, displaying the expected tendencies for PI to drop more packets at lower q_{ref} values (to enforce queue length) and for the average queue length to grow with greater q_{ref} values.

One may notice a sharp dip in transmission time and packet losses at $q_{ref} = 41$. Based on this observation, we elected to run further tests on PI's performance at this value. Fig. 4 shows the instantaneous queue lengths across the duration of the transfers.

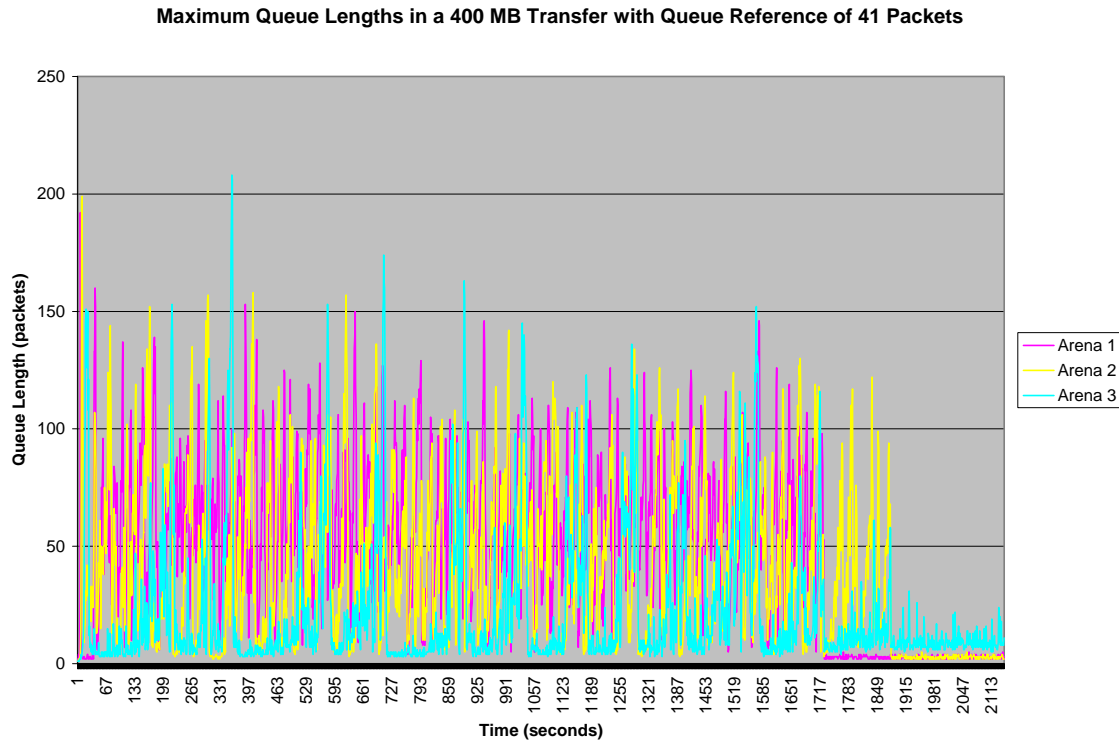


Fig. 4 (Queue Lengths)

Given that we chose q_{ref} to be 41, these data are not promising. Spikes to values above 200 occur, and overall the queue lengths show little sign of adhering to the q_{ref} value.

The possibility of varying the q_{ref} value across the arenas occurred to us next; we chose three test cases, at the points of minimum transmission time (41, 51, and 41 respectively for the arenas), minimum number of packets dropped (60, 57, and 60 respectively), and minimum number of Elephant packets dropped (51, 58, and 60).

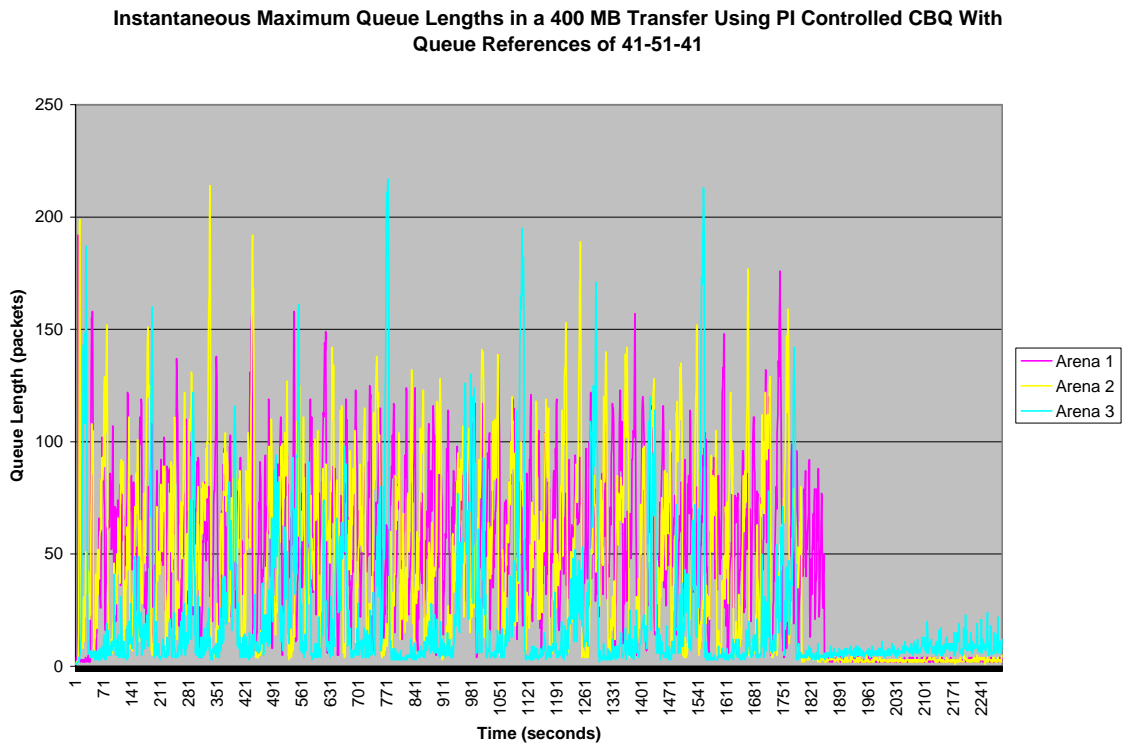


Fig. 4 (Queue Lengths with varying q_{ref})

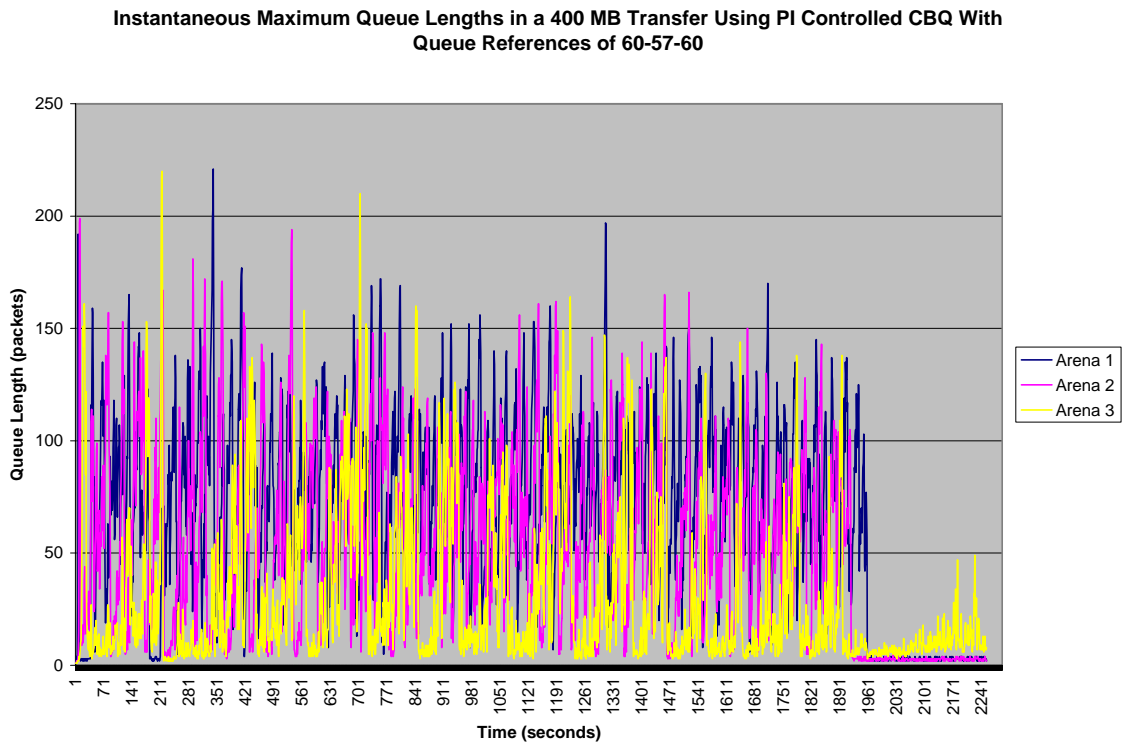


Fig. 5 (Queue Lengths with varying q_{ref})

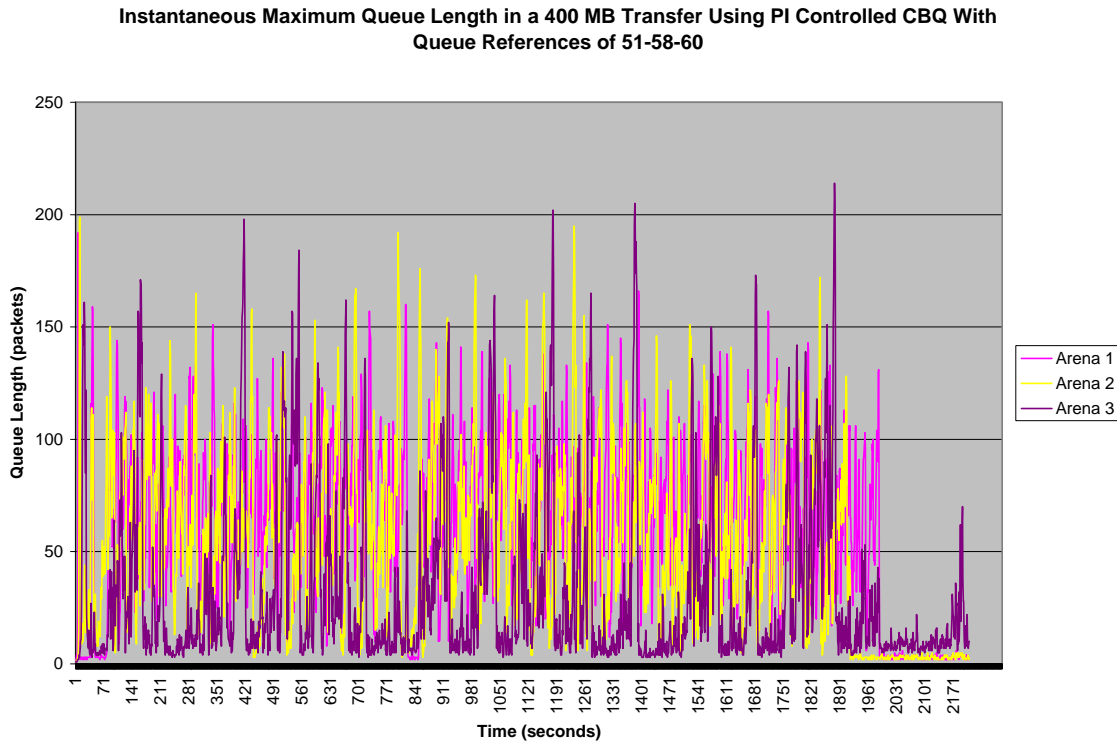


Fig. 6 (Queue Lengths with varying q_{ref})

In terms of overall shape, these results offer no substantial improvement over the results obtained with a uniform q_{ref} value, though they do include some really neat colors.

At the eleventh hour, as previously mentioned, an issue with the C program used to provide values of a and b was resolved, and new values were obtained which allowed the simulation to complete in a reasonable amount of time. The new values were a sampling frequency of 6Hz for all arenas; $a = 0.0959$ and $b = 0.00437$ for Arena 1; $a = 0.00179$ and $b = 0.00143$ for Arena 2; and $a = 3.79 * 10^{-5}$ and $b = 3.72 * 10^{-5}$ for Arena 3. The new simulation used a q_{ref} value of 50, arbitrarily chosen as a value in a “reasonable” range. Arena 1 finished transmitting in 1918.6 s; Arenas 2 and 3 in 2110.3 and 2483.2 s, respectively (cf. with transmission times using Droptail/CBQ, which clocked in at 1974.8, 2012.9, and 2183.8 s, respectively).

The results for Arena 1, with the lowest RTT, are striking and illustrated in Fig. 7. Fig. 8 illustrates the same instantaneous queue length for a previous simulation in which PI was not used.

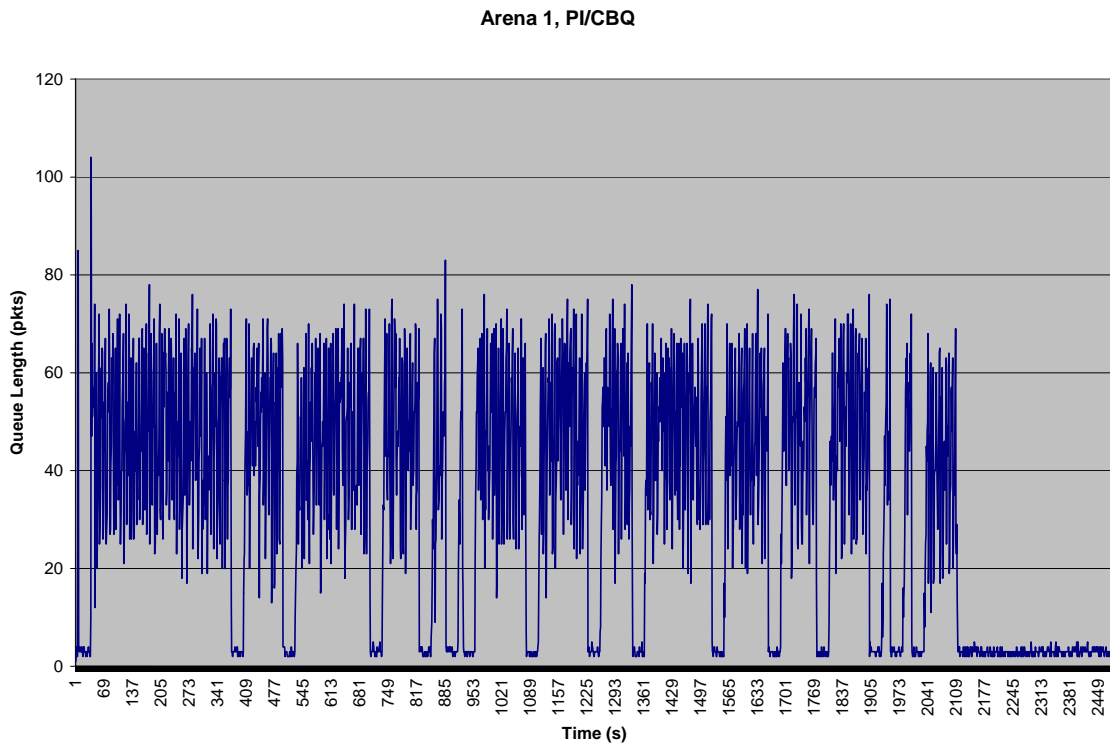


Fig. 7 (Instantaneous queue lengths, Arena 1, PI/CBQ)

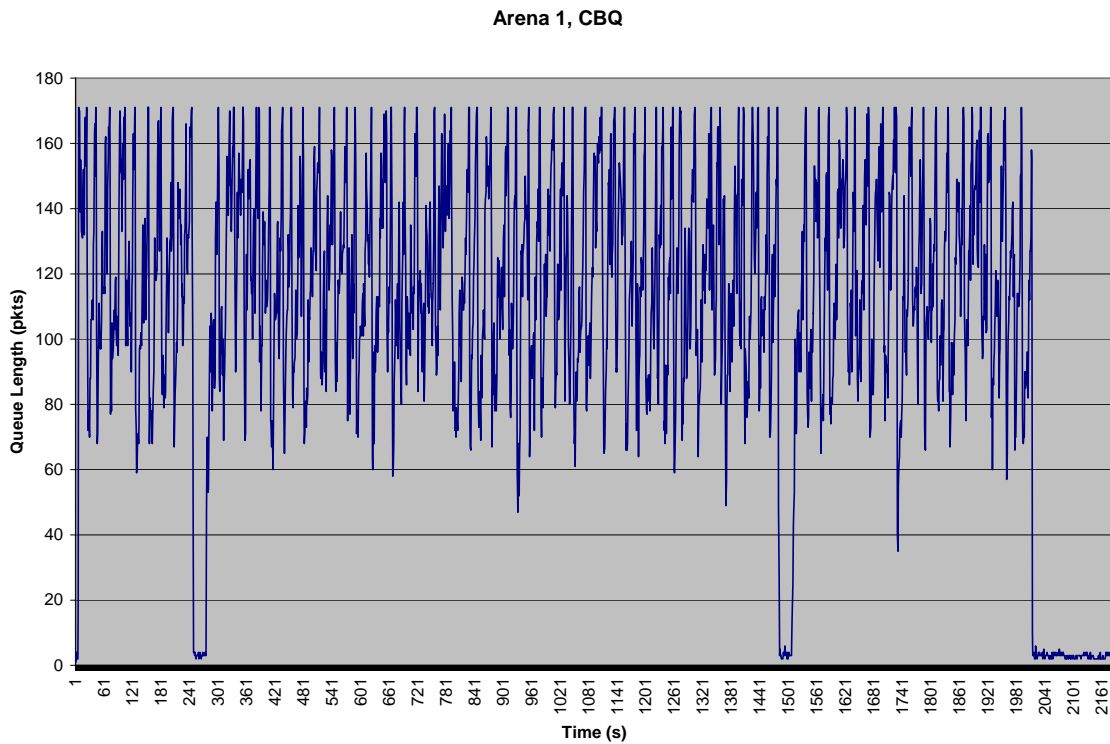


Fig. 8 (Instantaneous queue lengths, Arena 1, CBQ)

As clearly depicted, the queue length tended to cluster much more strongly around a lower value – the approximate mean appears to be around 50, our chosen q_{ref} value.

While PI also did a good job reining Arena 2's queue length in, it didn't achieve quite the clustering about the magical value of 50 that we saw in Arena 1 (see Fig. 9). Part of the reason for this would appear to be a tendency for large oscillations in queue length under Droptail/CBQ (Fig. 10); presumably the longer RTT forces the Elephant sender to back off more often than in Arena 1.

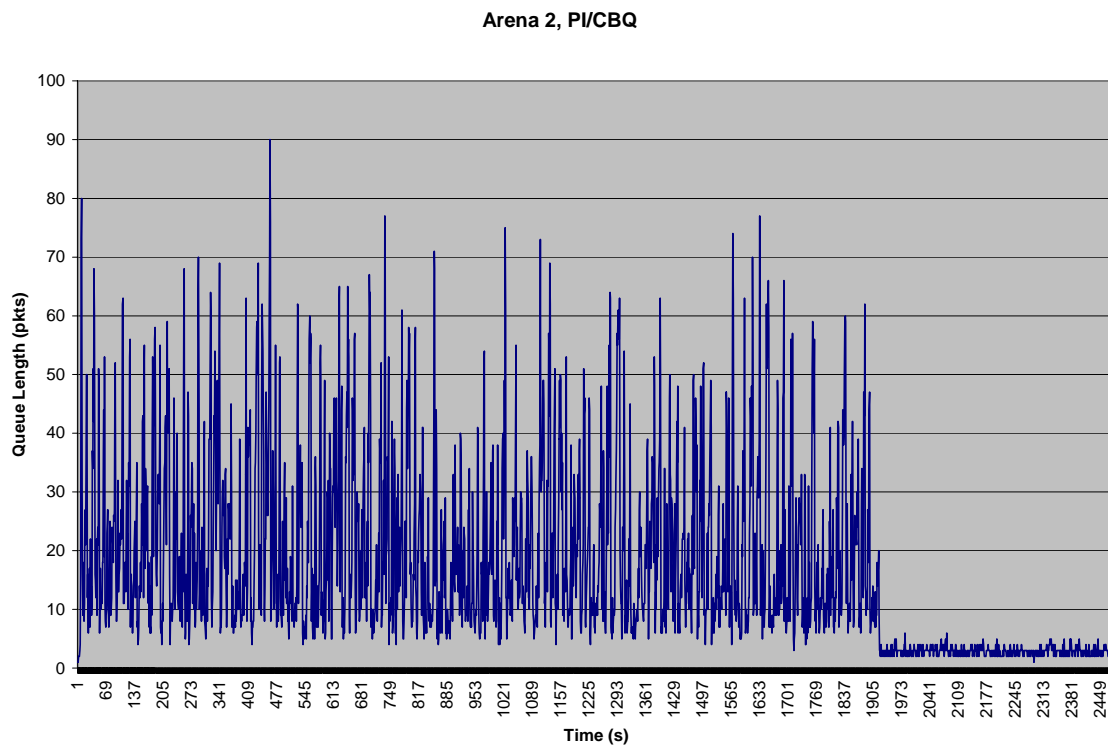


Fig. 9 (Instantaneous queue lengths, Arena 2, PI/CBQ)

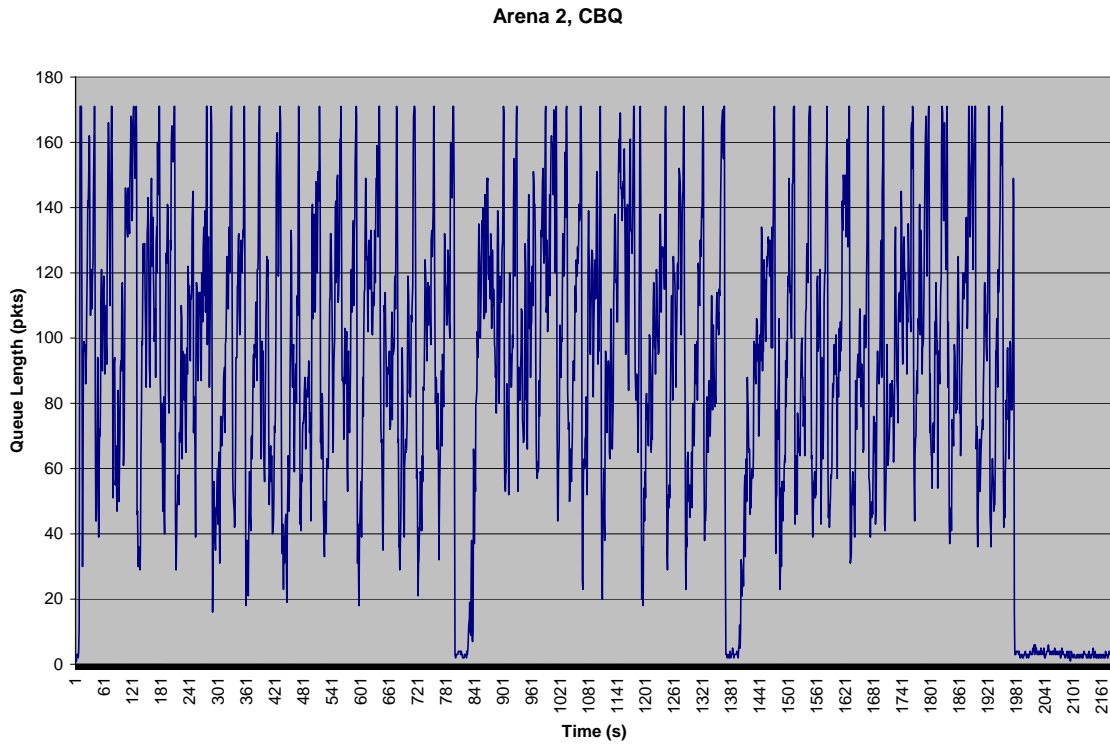


Fig. 10 (Instantaneous queue lengths, Arena 2, CBQ)

In Arena 3, the trend is continued to an even larger degree; the original CBQ queue (Fig. 12) displayed extremely bursty behavior, hovering at low values only to spike momentarily. While PI was able to control this behavior to some degree, the queue length still displays spikes, some well over the q_{ref} value of 50; it's fairly clear that PI was only partially able to moderate the sudden spikes in queue length as the Elephant sends bursts of packets (Fig. 11). PI is clearly better-adapted to controlling a steady-state sender over long periods of time.

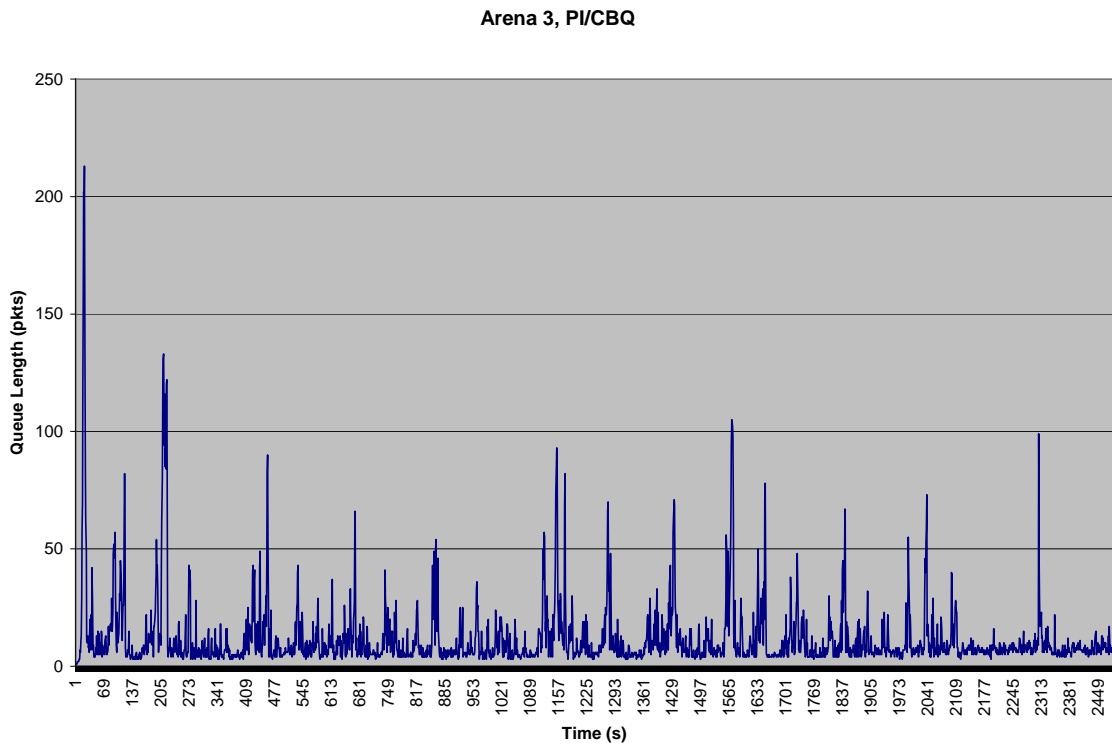


Fig. 11 (Instantaneous queue lengths, Arena 3, PI/CBQ)

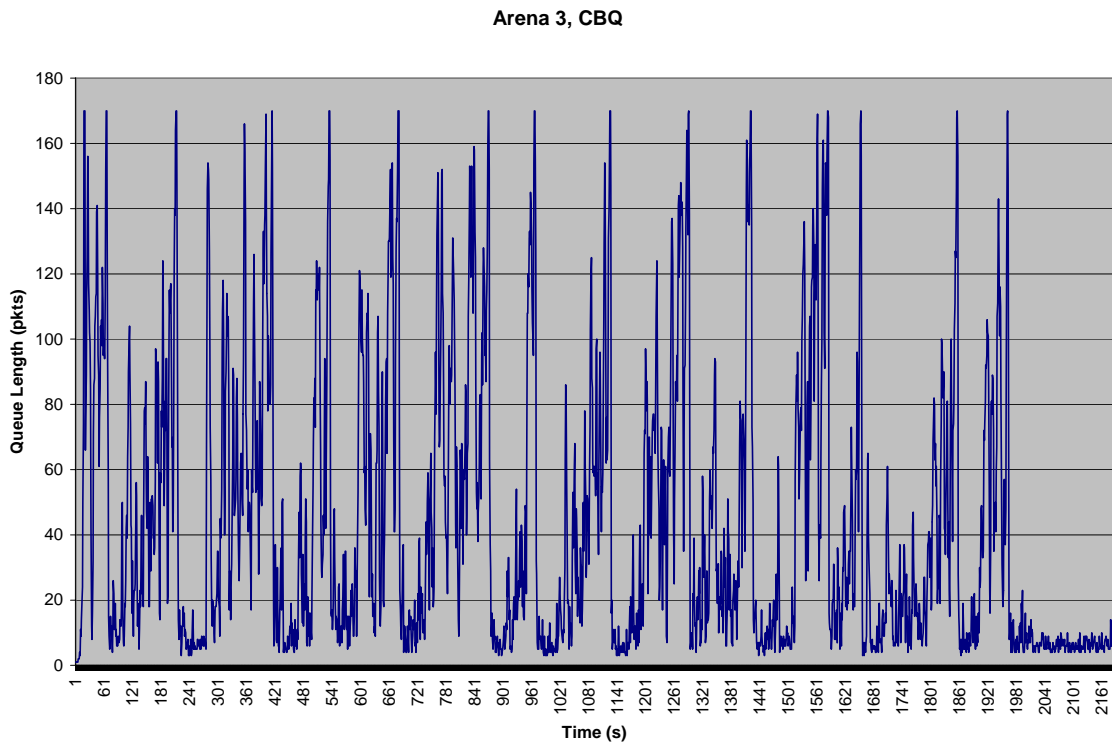


Fig. 12 (Instantaneous queue lengths, Arena 3, CBQ)