8 Simulation Analysis of TCP/DCA

On the simulated paths developed in Chapter 7, we run the hypothetical DCA algorithm we developed in Chapter 5 (i.e., the TCP/DCA algorithm). Through these experiments, we obtain the following results:

- Compared to a TCP/Reno connection, a TCP/DCA flow is able to avoid packet loss (by up to 8%), however the flow will experience throughput degradation on the order of 30%. These results are similar to the measurement results.
- The reactions of a DCA algorithm in response to increases in RTT do not significantly impact the congestion process over the path. We show this by first running a TCP/Reno connection over the simulated path and then in a different simulation run we replace the TCP/Reno connection with a TCP/DCA connection. By using identical background traffic sample paths in both runs, we find that the packet loss rates are roughly the same. We also analyze the queue fluctuation at the bottleneck links and find that the average queue length and the number of queue oscillations are not significantly impacted by the DCA congestion decisions. We do find that a small change in the behavior of a flow can affect the sample path of the loss process that operates at the congested link. The level of the change depends on the amount of resources consumed by the DCA flow and by the competing flows whose samples paths are perturbed. However, assuming that the flows involved consume only a fraction of total resources, we show that the impact of the perturbation on the queue dynamics does not affect the level of congestion at the bottleneck nor does it affect the relationship between packet loss and increases in RTT that an end-to-end DCA algorithm would observe.
- A TCP/DCA connection will experience throughput degradation over a range of TCP/DCA algorithm variations. Of particular interest is to see the impact that different levels of send rate reductions has on the end-to-end behavior. We show that as DCA reacts with a smaller send rate reduction the amount of throughput degradation decreases. However the algorithm becomes less effective in avoiding loss. Assuming that the number of reactions is the same, an algorithm that reacts to an increase in RTT by reducing the *cwnd* by 50% will have a better chance at reducing loss compared to a send rate reduction of 12.5% because the connection will consume fewer buffers over the next RTT.

This chapter is organized as follows. In the first section, we define and illustrate the TCP/DCA protocol. In the next section, we show that the congestion decisions of a TCP/DCA flow has a minimal impact on the congestion process over the paths. In the final section, we show that our thesis holds for different variations of the TCP/DCA algorithm.

8.1 The TCP/DCA Protocol

The TCP/DCA protocol extends a TCP/Reno sender using the DCA algorithm that was used in the throughput analysis (described in section 5.3.1). In order to generate the *tcpRTT* samples, a TCP/DCA sender measures the transmission time of all segments that are in flight. When an acknowledgement arrives that acknowledges more than one segment, a single *tcpRTT* sample is generated. Samples are not generated during periods of recovery.

The TCP/DCA congestion decision is:

$$sampledRTT(x) > windowAVG(w) + threshold$$

We will use a fixed *threshold* equal to the standard deviation associated with the *windowAVG(w).* As described below, the size of the windows associated with the moving averages (i.e., *x* and *w*) is a parameter of the simulation. The following parameters are defined:

- *congestionReduction*: this specifies the level of the send rate reduction when the congestion decision requires a reaction to an increase in RTT. The analysis in this section will use a value of 50%. The *cwnd* is adjusted as follows:

  $$cwnd = cwnd - (cwnd * congestionReduction)$$

- *congestionDecisionFrequency*: this specifies how frequently the congestion decision is performed. Typical values are once per congestion epoch (where an epoch is defined as a time period that begins with an increase in RTT and that terminates when the RTT subsides to it original value) or once every *X* RTT periods.

- *windowAVGWindowSize*: this specifies the size (*w*) of the window associated with the *windowAVG(w)* values.
- *sampledRTTWindowSize*: this specifies the size (*x*) of the window associated with the *sampledRTT(x)* values.

Previously defined DCA algorithms react at discrete time intervals. For example, the congestion decision of the TCP/Vegas algorithm engages once each RTT upon the arrival of a selected acknowledgement. The rational is to minimize the processing overhead required by the algorithm. Our goal is to assess DCA at its best therefore we do not consider the impact of overhead. With each *tcpRTT* sample, TCP/DCA will search for the beginning or the end of a congestion epoch and possibly run the congestion decision algorithm. This allows the algorithm to react to an increase in RTT immediately rather than to wait for the next decision time. The objective is to detect and avoid the 7-18% of the loss events that our measurements indicate are preceded by a significant increase in RTT.

Figure 8-1 shows simulation results of the TCP/DCA protocol. The top curve plots the *tcpRTT* time series for a 10 second portion of simulation over the Emory simulation model. The middle and lower curves show the queue levels of the two bottleneck links in the path. In the *tcpRTT* curve, the hash marks at the top of the curve indicate occurrences of packet loss. The diamonds at the top of the curve indicate when the TCP/DCA algorithm reacts to congestion. The figure indicates that there are three loss events that occur at times 91.2, 91.3, and 94 seconds. The *tcpRTT* curve illustrates that DCA reactions are limited to once per congestion epoch. As an example, the first three DCA reactions that occur between time 90.1 seconds and 91.5 seconds occur for each of the three largest RTT increases. Clearly it is not possible to tell if a particular reaction actually avoids loss. As we explain further, simulation experiments based on the Emory model show that TCP/DCA is able to reduce the packet loss level only by 8%. Therefore, the majority of the reactions indicated in Figure 8-1 are likely to be "unnecessary".

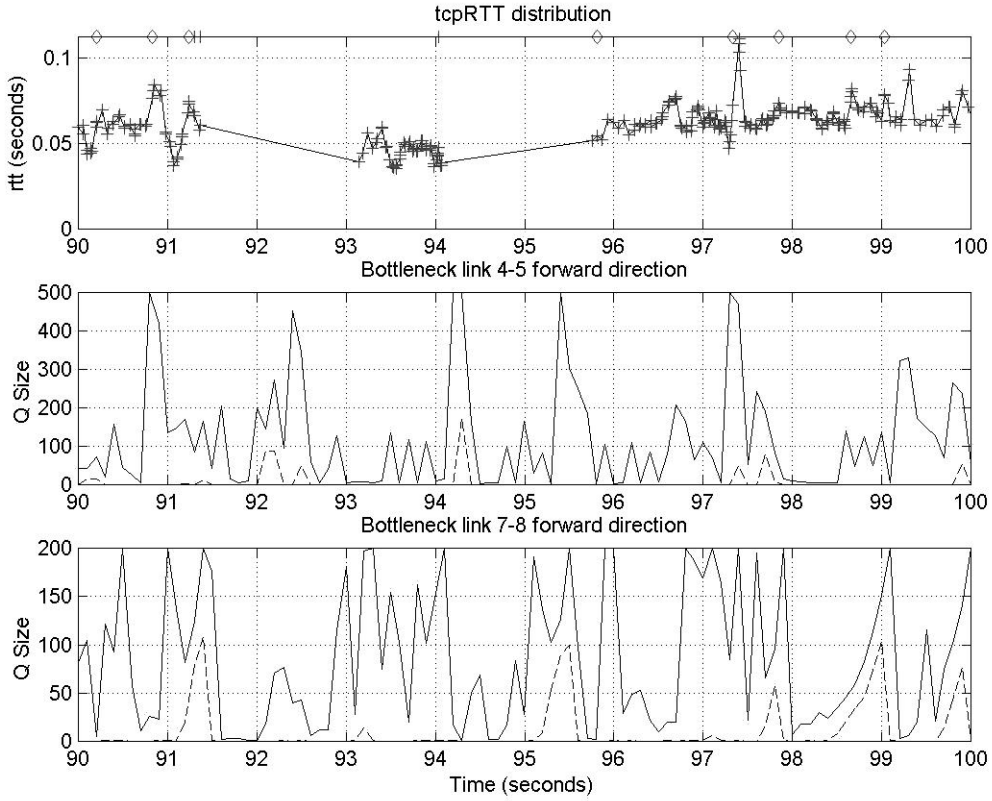

Figure 8-1. TCP/DCA simulation result over the Emory path

We are interested in comparing the performance of a TCP/DCA and a TCP/Reno connection. Over each of the two path models presented in the previous chapter, we run two end-to-end connections (a TCP/Reno and a TCP/DCA protocol) over the path. The TCP parameters associated with each of the two connections under observation are identical (i.e., TCP window sizes, segment sizes). We perform 10 simulation runs (500 seconds long). At the end of each run we compute the effective loss rate and throughput experienced by the two connections. Table 8-1 shows the average results from all the runs for each path (along with the corresponding 95% confidence intervals).

For the Emory path, the results indicate that TCP/DCA is able to reduce the level of packet loss by 7.5%. However, in doing so, it reduces the throughput (as compared to the TCP/Reno run) by 37%. The ASU results indicate that in a highly congested environment, DCA will not reduce the packet loss rate and but it will decrease throughput on the order of 13%. The measurement analysis predicts different levels of

throughput degradation (50% for the Emory and 9% for the ASU paths). The difference between the results from the simulation and the measurement analysis can be attributed to the following.

- Differences between the simulation model and the Internet path.
- Error in the analytic TCP throughput model.

Table 8-1. Assessment of the impact of TCP/DCA on end-to-end TCP performance

Model	% reduction of packet loss (mean,std) (95% confidence interval)	% reduction of throughput (mean,std) (95% confidence interval)
Emory	-7.5(12.8) (-16.5, 1.5)	-37(8.1) (-42,-31)
ASU	-2.4(15.6) (-13.4,8.6)	-13.2(9.4) (-19.8,-6.6)

8.2 Validation that DCA Reactions Have Minimal Impact on the Network

The objective is to show that the queue dynamics at a bottleneck link are not significantly impacted when a Reno flow is replaced with a DCA flow. Using simulation we will show that:

- The congestion level (i.e., the loss rate and the average queue level) at the bottleneck does not decrease.
- While the sample path associated with the queue oscillation might be slightly altered, the relationship between packet loss and RTT is not affected.

We introduce our method by performing one 200 second simulation run using an end-to-end TCP/Reno connection over the Emory and ASU path models. We obtain the loss rate and throughput experienced by the flow. Then we perform another simulation run where a TCP/DCA connection is used in place of the Reno connection. By comparing the throughput and packet loss rate of the DCA run with that of the Reno run we are able to assess the impact of DCA on the network traffic (or congestion levels) relative to Reno.

All of the traffic generators that are used in the model to create background traffic use their own random stream. Consequently, when we replace the TCP/Reno connection with a TCP/DCA connection, the sample paths of the background traffic are identical in both simulation runs. In other words, the amount of application data and the rate at which it is offered to the transport protocol will be identical in both simulation runs.

We configure the TCP/DCA algorithm to be as reactive as possible. We want to assess the ability of one flow to reduce the congestion level over the path. It seems reasonable to assume that if we show that a flow that reacts aggressively to congestion does not reduce the level of congestion, then a flow that reacts less aggressively to congestion will also not impact the congestion level over the path. The parameters of the TCP/DCA algorithm are the same as in the previous section (i.e., the *congestionReaction* level is 50%) except the algorithm can react more than once per congestion epoch. We set the *congestionDecisionFrequency* to one RTT allowing the algorithm to react as frequently as every round trip time.

We start by visually showing that the congestion dynamics at the bottleneck links are not significantly impacted by the TCP/DCA reactions. Figure 8-2 illustrates the traffic dynamics during a 10 second period of TCP/Reno simulation over the Emory model. The figure plots the same *tcpRTT* time series observed by the Reno connection and the queue levels of the two bottleneck links (link 4-5 and link 7-8 as illustrated in Figure 7-7). Figure 8-3 shows results from another run with the TCP/Reno connection replaced with a TCP/DCA connection. Comparing the top curves of Figures 8-2 and 8-3 shows that each algorithm gets a slightly different view of the congestion depending on when loss occurs. The middle and lower queue curves shown in each figure suggest that the queue dynamics at link 4-5 are unaffected by the TCP/DCA algorithm while the dynamics associated with link 7-8 are slightly impacted.

The background traffic at both links in the Emory path consists of a combination of both TCP and UDP traffic. Link 4-5 has a significant amount of UDP traffic (by monitoring the traffic at the link, we find that 20% of the traffic is UPD and 80% is TCP). The behavior of a UDP flow will not be altered by changes in

the behavior of competing traffic which explains why the queue oscillations at link 4-5 are essentially unchanged between the two runs. The traffic mix over link 7-8 is roughly 88% TCP and 12% UDP traffic. We wanted to create a congestion process that contains a mixture of large time scale congestion (e.g., see time 98-99 seconds in Figure 8-2) along with short time scale traffic spikes. We found it difficult to create exactly what we wanted. We settled on a traffic mix containing a small set of high bandwidth ON/OFF TCP flows (on the order of 30-40 flow each with a maximum window size equivalent to 64Kbytes) along with a small amount of UDP traffic (12%). Comparing the lower plots of Figure 8-2 and 8-3, we observe that the level of congestion in the two simulation runs appears the same (we confirm this shortly). The difference is that the sample path of the queue oscillations is slightly altered.

If the contribution of traffic by the original TCP/Reno flow was large with respect to the level of all the traffic that flows over the link then the reactions of DCA could significantly alter the level of congestion at a bottleneck link. However, the relative contribution of the end-to-end TCP/Reno flow is small (at most an entire window of data or 12 packets will be queued which represents 6% of a total of 200 buffers available at the congested router). By monitoring arriving traffic at the bottlenecks, we find that the single DCA flow under observation generally contributed less than .5% of the total traffic. We will show shortly that the loss rate, the average queue level and the average number of queue oscillations does not change because of the different behavior of a single DCA connection.

There is another effect that explains the altered queue dynamics. If the behavior of a single connection changes, it is likely that the sample path associated with the packet loss process will change. One perturbation leads to other perturbations which lead to others. The impact of the perturbations is proportional to the resources consumed by flows involved. A change in the sample path of one high bandwidth TCP connection can be significant and this explains the difference in the queue behaviors.

We redesigned the background traffic at link 7-8 to consist of 1800 ON/OFF TCP flows with idle times in the range of 2-6 seconds. We find that the level of perturbation that occurs when we replace a Reno connection with a TCP/DCA connection is still noticeable but it is less than the level of perturbation

illustrated in Figures 8-2 and 8-3. If we add even more TCP flows (2200 in all) such that the link becomes saturated (i.e., sustained queueing exists throughput the simulation), the level of change in the queue oscillation becomes very minor. As the congestion level increases, each flow consumes fewer resources thereby making the actions of a single flow less significant. We also see this behavior in the ASU simulation.

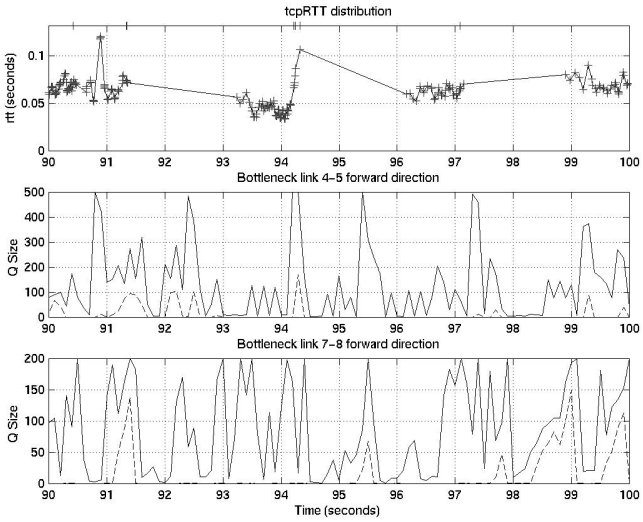

Figure 8-2. TCP/Reno run over Emory path

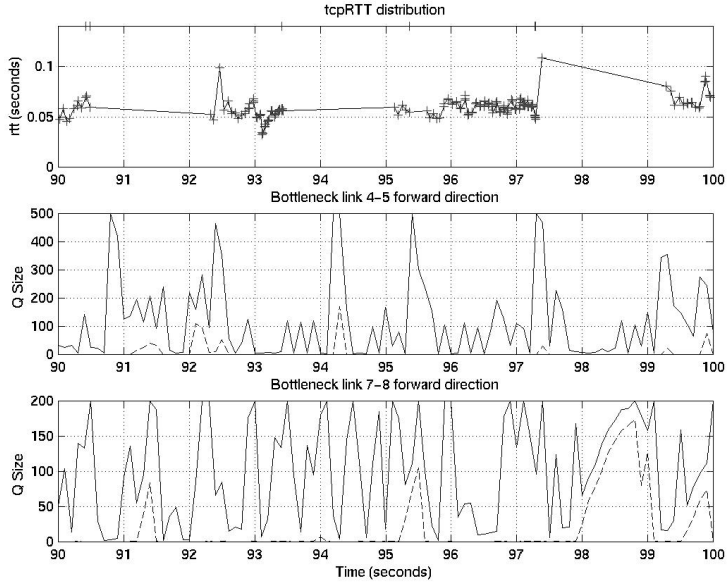

Figure 8-3. TCP/DCA run over Emory path

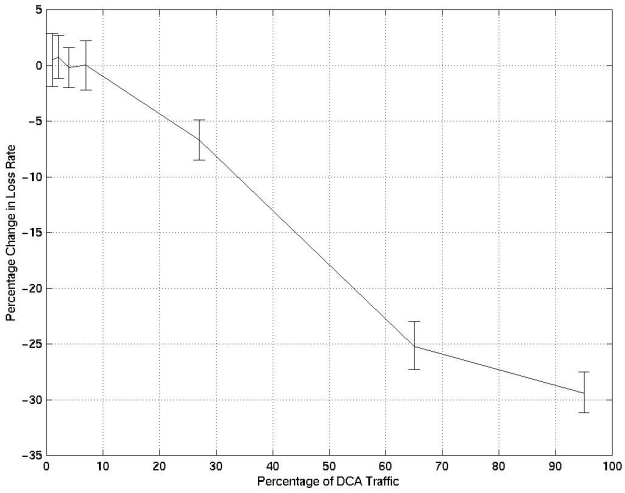

Figure 8-4. Reduction of loss rate as the amount of DCA traffic increases

Continuing with the modified Emory simulation model, we want to find the percentage of DCA traffic that is required to lower the level of congestion at a bottleneck. Focusing on link 7-8, we set the background traffic such that there are 2200 low bandwidth, ON/OFF TCP flows. Roughly 95% of the traffic is TCP and 5% is UDP. Each flow emulates a web user by setting a pareto application traffic generator with burst parameters set to burst a realistic number of packets (in the range of 6 packets to something much larger) using an idle time in the range of 1 to 10 seconds. Figure 8-4 shows that the loss rate at the router begins to drop once the amount of DCA traffic at link 7-8 exceeds 10%. It is interesting to observe that even when DCA traffic dominates the link, the loss rate is only reduced by 30%. We also find that the average queue level of the bottleneck link is unchanged until the DCA traffic exceeds 60%. Even when 95% of the traffic is DCA, the average queue level is reduced by only 5%. We repeated this experiment using different DCA

algorithm parameters and found similar results. We also replaced all DCA connections with TCP/Vegas connections (we present a Vegas simulation analysis in the next chapter) and find similar results.[1]

The result described above is based on an experiment over a highly congested network. It is possible that DCA is more effective at reducing the congestion level during less extreme conditions. Our point here was simply to get one data point that estimates how much DCA traffic is required to see an improvement in the congestion level. A value of 10% seems reasonable.

Figure 8-5 and 8-6 shows the results of a TCP/Reno and TCP/DCA run over the ASU path. The *tcpRTT* samples indicate that each connection observes a different view of the congestion. Unlike the Emory case, the queue dynamics at both of the bottleneck links are essentially identical. The amount of UDP traffic at link 7-8 is large (50%) which explains why the queue level at link 7-8 is identical in the two simulation runs. The traffic at link 8-9 is dominated by low bandwidth TCP flows (roughly 200 connections with idle times in the .5-1 second range). The traffic seen at the router is 92% TCP and 8% UDP. In the conditions associated with link 8-9, the impact of a single flow is minimal. Because of the heavy load, the queue experiences sustained queueing. As we discussed in the Emory simulation the queue dynamics at congested links that are dominated by low bandwidth TCP flows are determined by the increase and decrease in the number of flows rather than by the behavior of individual sessions.

[1] We also replaced the DCA connections with RED/ECN (turning on RED at all the bottlenecks in the model using a *thresh_* value of 50 and a *maxthresh_* of 150 packets) and found that the average queue level at link 7-8 was reduced by 50%.

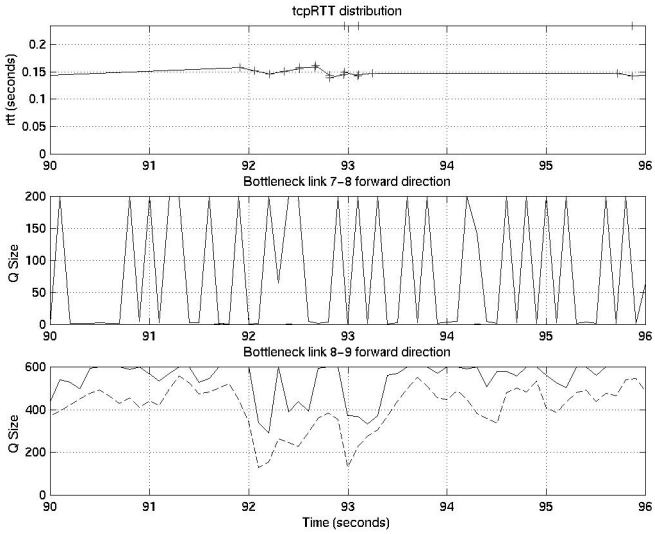

Figure 8-5. TCP/Reno run over ASU Path

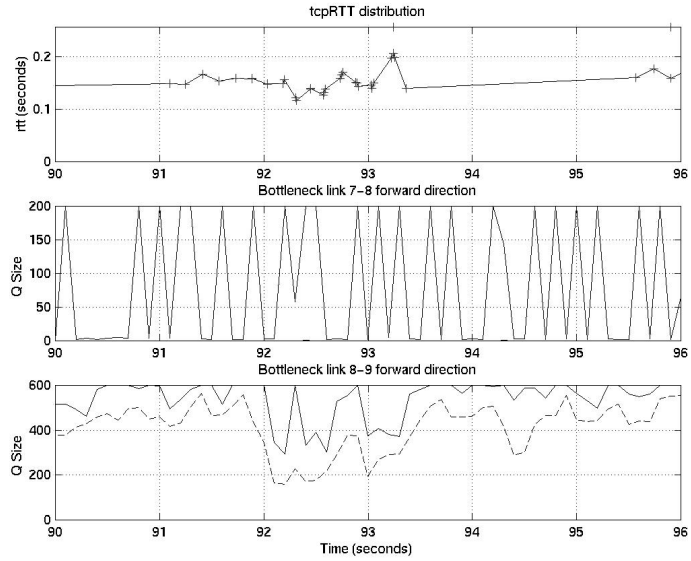

Figure 8-6. TCP/DCA run over ASU Path

We define an experiment to consist of two simulation runs. In the first run, we simulate two TCP/Reno connections competing over either the Emory or the ASU paths. In the second we simulate one TCP/Reno and one TCP/DCA connection competing. We monitor the throughput and loss rate of the TCP/Reno connection in the first run and those of the TCP/DCA connection in the second run. To obtain an accurate

assessment of RTT, we monitor the RTT from the TCP/Reno connection in both runs. If we were to monitor the RTT of the TCP/DCA connection, the DCA algorithm would skew the average RTT as there would be fewer samples taken when the RTT was large. We perform five sets of the experiment using different random number generator seeds. Table 8-2 shows the average results. We find that the level of throughput degradation and the reduction of the packet loss rate experienced by the connection is similar to the results from the previous section (Table 8-1). The throughput degradation is larger because we have configured the DCA algorithm to react more frequently (up to every RTT period). The third column of Table 8-2 shows that the average RTT observed by the TCP/Reno connection in both runs is the same. This result suggests that the average queue levels at the bottlenecks are not affected by the DCA reactions.

Table 8-3 shows the impact that DCA has on the bottleneck links. The first three columns describe the difference in the queue dynamics at link 4-5 caused by DCA and the last three columns describes the dynamics at link 7-8. The first column indicates the change in the loss rate at link 4-5 when the TCP/Reno connection is replaced by DCA. The second column indicates the average queue length (along with the 95% confidence interval associated with the data) and the third column indicates the relative change in the number of queue oscillations (i.e., congestion epochs).

Based on the data in Table 8-3, we see that the loss rates at both links are unchanged. The average queue level at link 4-5 does reflect a small decrease that corresponds to roughly 3 packets. In other words, the queue level average is about 3 packets lower when the TCP/DCA connection replaces the TCP/Reno connection. This is because

- The TCP/DCA connection will use fewer buffers during times of congestion.
- The load offered to the network in both runs (i.e., in the Reno case and the DCA case) is identical.
- The loss rates are low (1%) which means that other connections generally do not need the buffers that are not consumed by the DCA flow.

The third and sixth columns indicate that the queue level at link 4-5 experienced a slightly larger number of queue oscillations while the link 7-8 reflects a slightly smaller number. We believe that this is only statistical variation and does not indicate that some aspect of the queue dynamics has changed.

Table 8-2. Impact of DCA on a connection over the Emory path

Loss Rate Change	Throughput Change	Avg RTT Change
-8%	-52%	0%

Table 8-3. Impact of DCA on the network over the Emory path

Link 4-5 Loss Rate Change	Link 4-5 Queue Avg Change (95% confidence interval)	Link 4-5 Change in the Number of Queue Oscillations	Link 7-8 Loss Rate Change	Link 7-8 Queue Average Change (95% confidence interval)	Link 7-8 Change in the Number of Queue Oscillations
0	-2.8% (-4.9%,-.7%)	+2.4%	0%	-.014 % (-1.9%,1.9%)	-1.9%

Our simulation results confirm that the congestion reactions of DCA will not reduce the congestion level at the bottleneck links. However, we have seen that the queue dynamics at link 7-8 in the Emory simulation are slightly impacted by DCA (by comparing the lower curves in Figures 8-2 and 8-3). We contend that the impact is not significant enough to invalidate the measurement throughput analysis from Chapter 5 for several reasons (an assumption of the throughput analysis was that the DCA reactions does not significantly alter the dynamics of the congestion process over the path). First the perturbations to the network are small. Second, we have shown that the loss rates, average queue level and number of epochs does not change significantly. Finally, we will show that the relationship between RTT and loss events is does not change. We use the correlation metrics defined in Chapter 4 on the aggregate data from the Reno simulation runs and compare the results with those from the DCA simulation. Table 8-4 shows that the correlation indication metric results indicate that DCA has no impact. Figures 8-7 and 8-8 illustrate that

the loss conditioned delay correlation metric applied to the aggregate *tcpRTT* time series data from the Reno runs over the Emory model is almost identical to the metric applied to the aggregate DCA *tcpRTT* data.

Table 8-4. Correlation indication metric results for the Emory simulation

Metric	Reno Data	DCA Data
P[sampledRTT(2)>windowAVG(5)]	.33	.30
P[sampledRTT(2)>windowAVG(20)]	.32	.30
P[sampledRTT(2)>windowAVG(20)+std]	.1	.1
P[sampledRTT(5)>rttAVG	.4	.35

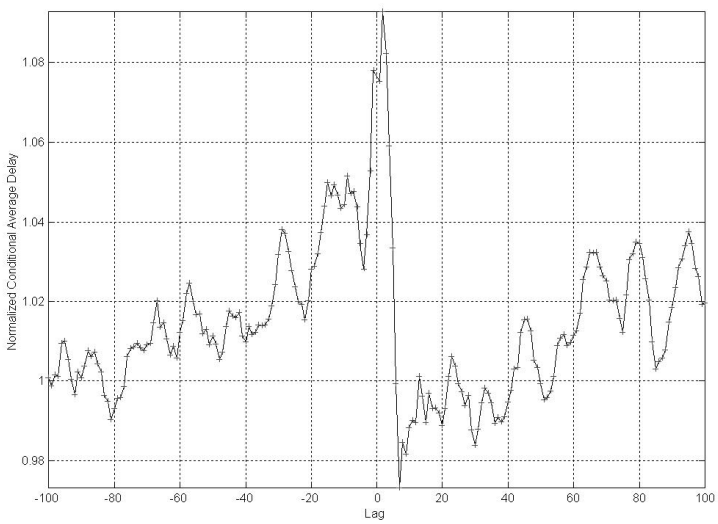

Figure 8-7. Loss conditioned correlation delay metric on aggregate Reno data using Emory model

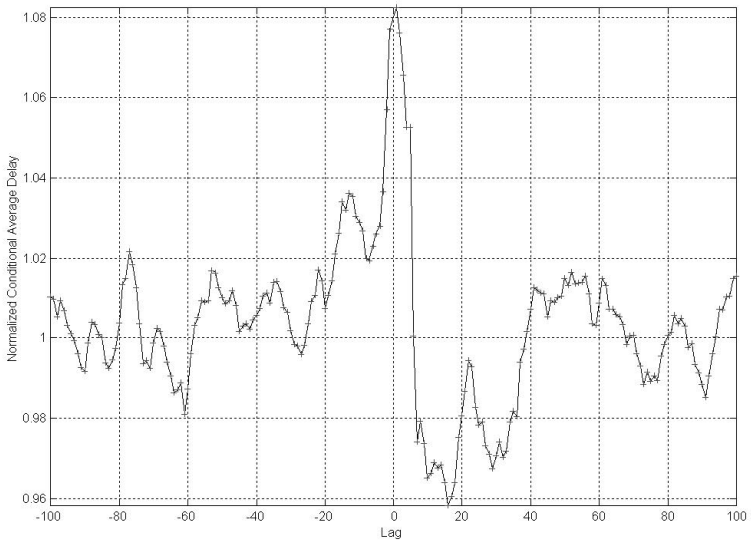

Figure 8-8. Loss conditioned correlation delay metric on aggregate DCA data using Emory model

Tables 8-5 and 8-6 illustrate the results over the ASU path. For this experiment, TCP/DCA is more effective at avoiding packet loss than the earlier experiment over the ASU path (i.e., Table 8-1). The difference is that we have configured TCP/DCA to react more frequently (once per RTT rather than once per epoch) to maximize its impact on the traffic dynamics. Table 8-6 indicates that the queue dynamics at both bottleneck links are not at all changed (or altered) by the congestion reactions of DCA. Note that we can not count the number of queue oscillations at link 8-9 because of the sustained congestion.

Table 8-5. Impact of DCA on a connection over the ASU path

Loss Rate Change	Throughput Change	Avg RTT Change
-7%	-15%	0%

Table 8-6. Impact of DCA on the network over the ASU path

Link 7-8 Loss Rate Change	Link 7-8 Queue Avg Change (95% confidence interval)	Link 7-8 Change in the Number of Queue Oscillations	Link 8-9 Loss Rate Change	Link 8-9 Queue Avg Change (95% confidence interval)	Link 8-9 Change in the Number of Queue Oscillations
0%	-.02% (-.07,.03)	0%	0%	-.02% (-.2, .18)	*

8.3 Confirming the General Result

In this section we confirm that DCA will not improve TCP throughput for different variations of the algorithm. In particular we show:

- As we reduce the *congestionReduction* level, the amount of throughput degradation decreases although we never see the throughput improve. As the *congestionReduction* decreases, the ability to avoid loss also decreases. This is because a smaller send rate reduction in response to an increase in RTT has a lower chance of avoiding packet loss than a larger reaction.
- As we increase the number of times DCA is allowed to react, we see an increase in the amount of loss the algorithm can avoid but the level of the throughput degradation increases as well.
- As we increase the window sizes associated with the *sampledRTT* and *windowAVG*, we find that the algorithm becomes less responsive to congestion. Increasing the *x* associated with the *sampledRTT* tends to filter out RTT increases associated with short term queue delays. This reduces the ability of the algorithm to avoid loss. Increasing the *w* associated with the *windowAVG* significantly increases the threshold level as the standard deviation associated with a longer window will be larger than the deviation associated with a shorter window. This makes the algorithm react only to the larger

increases in RTT. Because it reacts less frequently, the amount of throughput degradation is low however the algorithm is not able to reduce the packet loss rate.

Our method is slightly different from that used in the previous experiments. We define an experiment to consist of a single simulation run. We compare the throughput and loss experienced by an end-to-end TCP/Reno and a TCP/DCA connection. By performing multiple runs, we derive the results stated above.

We first calibrate our method by observing the base variation associated with the throughput and loss of TCP/Reno. Table 8-7 illustrates the statistical results of 10 simulation runs designed to compare the behavior of two competing TCP/Reno connections. We see roughly what we would expect over the Emory path. The difference in the packet loss rates experienced by the two Reno connections should be 0. Due to the small number of runs (set to 10), we do not see exactly 0. We see even more bias in the throughput. Clearly additional runs are required in order for the statistics to converge. However, because of the processing requirements of large scale simulation, we had to limit the number of simulation runs to 10. The 95% confidence intervals associated with both the loss rate and the throughput indicates the level of variation associated with loss rates and throughput that can be expected with Reno connections. In the DCA that analysis we present in this section, we search for trends in the data that rise above this statistical "noise" level.

Tables 8-8 and 8-9 show the impact that the *congestionReduction* level has on the effectiveness of the DCA algorithm. The Emory results (i.e., Table 8-8) confirm our hypothesis that as the amount of the send rate reduction decreases, the level of throughput degradation decreases as well as the ability to avoid packet loss. When the send rate reduction is 12.5%, the algorithm is not able to avoid packet loss. For the ASU results (Table 8-9), the algorithm is able to reduce the loss rate only by 2% when the level of send rate reduction is 50%. For smaller send rate reductions, we actually see an increase in the loss rate. We believe that there is some interaction in the Vegas code between CAM and the base congestion control algorithms which reduces the effectiveness of the fast retransmission/recovery algorithms. Tables 8-8 and 8-9 indicate that as the level of the send rate reduction decreases, the amount of throughput degradation

experienced by the flow decreases. Because the DCA reactions do not decrease the congestion level over the path, the algorithm will react roughly the same number of times regardless of the amount of the send rate reduction. This explains why the amount of throughput degradation decreases.

Table 8.7 Reno-Reno baseline

Path	% reduction of packet loss (mean,std) (95% confidence interval)	Reduction of throughput (mean,std) (95% confidence interval)
Emory	+.6% (18.9) (-12.7, +13.9)	+2.1% (12.4) (-6.6,+10.9)
Asu	4.7 %(15) (-5.8, 15.2)	.7% (11.2) (-7.1, 8)

Table 8-8. Varying the *congestionReduction* level for the Emory model

CongestionReduction level	% reduction of packet loss (mean,std) (95% confidence interval)	% reduction of throughput (mean,std) (95% confidence interval)
50%	-7.5(12.8) (-16.5, 1.5)	-37(8.1) (-42,-31)
25%	-8(17.5) (-20,4.2)	-12(12) (-21,-4)
12.5%	3.4(15) (-7.4,14.3)	-6.8(8.6) (-13, -.8)

Table 8-9. Varying the *congestionReduction* level for the ASU model

CongestionReduction Level	% reduction of Packet Loss (mean, std) (95% confidence interval)	% reduction of Throughput (mean,std) (95% confidence interval)
50%	-2.4(15.6) (-13.4,8.6)	-13.2(9.4) (-19.8,-6.6)
25%	7.8(25.7) (-10.2,26)	-9.8(20) (-24,4)
12.5%	7.8(21) (-6.7,22.6)	-7.4(15) (-18.1,3.1)

Tables 8-10 and 8-11 show the impact that varying the *congestionDecisionFrequency* has on the effectiveness of the DCA algorithm. Again, the results over the Emory path are rather expected. Table 8-10 shows that DCA is able to avoid loss more effectively when the algorithm is allowed to react more frequently. However this causes an increase in the throughput degradation. Over the ASU path, there

appears to be a similar trend although the statistics reflect wider variations in the data as compared to the Emory data. When the *congestionDecisionFrequency* is set to once an epoch, there will be fewer reactions as an epoch might last for many RTTs. The algorithm is most effective at avoiding loss when it is allowed to react every RTT. By reducing the loss rate by 6%, a significant number of timeouts are avoided which explains why the amount of throughput reduction is less compared to when the algorithm is allowed to react every other RTT.

Table 8-10. Varying the *congestionDecisionFrequency* parameter for the Emory model

CongestionDecisionFrequency	% reduction of packet loss (mean,std) (95% confidence interval)	% reduction of throughput (mean,std) (95% confidence interval)
every epoch	-7.5(12.8) (-16.5, 1.5)	-37(8.1) (-42,-31)
every 2 RTT's	-16.2 (18.6) (-29, -3)	-46.3(6) (-51,-42)
every RTT	-14.2(18) (-27,-1.5)	-49(8.4) (-54.5,-43)

Table 8-11. Varying the *congestionDecisionFrequency* parameter for the ASU model

CongestionDecisionFrequency	% reduction of packet loss (mean,std) (95% confidence interval)	% reduction of throughput (mean,std) (95% confidence interval)
every epoch	-2.4(15.6) (-13.4,8.6)	-13.2(9.4) (-19.8,-6.6)
every 2 RTT's	-3.3(17.2) (-15.4,8.8)	-17(12.6) (-26,-8.1)
every RTT	-2.8(17.9) (-15.4,9.8)	-17.6(10.1) (-25,-10.5)

Tables 8-12 and 8-13 show the impact of varying the moving window parameters on the effectiveness of DCA. Both the Emory and the ASU results show that as either of the window size parameters increases, the degradation in throughput decreases and the algorithm becomes less effective at avoiding loss. This reinforces our measurement analysis conclusion where we found that the most effective (*x,w*) combination is (2,20). Increasing the x value to 6 implies that the instantaneous RTT estimate is based on the last 6

tcpRTT samples. This reduces the responsiveness of the algorithm by filtering short term variations in RTT. Effectively, the algorithm reacts less frequently which explains the lower level of throughput degradation and the lower level of packet loss reduction. Increasing the *w* to 200 has the same effect but for different reasons. As the *windowAVG* window size increases, the variance associated with the moving window increases as well. This causes the *threshold* level to increase which also reduces the responsiveness of the algorithm.

Table 8-12. Varying the *(x,w)* parameters for the Emory model

(x,w) values	% reduction of packet loss (mean,std) (95% confidence interval)	% reduction of throughput (mean,std) (95% confidence interval)
2,20	-7.5(12.8) (-16.5, 1.5)	-37(8.1) (-42,-31)
6,20	-6.3%(13.8) (-22,9.7)	-25.4%(7) (-33,-17)
2,200	-5.6%(17) (-25,13)	-20.4%(8.8) (-30,-10)

Table 8-13. Varying the *(x,w)* parameters for the ASU model

(x,w) values	% reduction of packet loss (mean,std) (95% confidence interval)	% reduction of throughput (mean,std) (95% confidence interval)
2,20	-2.4(15.6) (-13.4,8.6)	-13.2(9.4) (-19.8,-6.6)
6,20	-.5%(3.9) (-5,4)	-5.6(13) (-20,9)
2,200	+.12%(18) (-21,21)	-4.2%(8.4) (-14,5.4)