9 Validation of Results Using TCP/Dual and TCP/Vegas

In this chapter, we show that the DCA algorithms associated with TCP/Vegas and TCP/Dual are not able to reliably avoid packet loss and consequently do not improve TCP throughput. The analysis extends the simulation analysis performed in Chapter 8 by examining the performance of a TCP/Vegas and TCP/Dual model. The performance metrics of interest are the retransmission rate and the average throughput experienced by the flow.

The results show that the Vegas-CAM algorithm leads to a throughput reduction of 21% over the Emory path and 3% over the ASU path compared to a TCP/Reno connection. TCP/Dual suffers a 14% throughput degradation over the Emory path and 15.8% over the ASU path. Various aspects of each algorithm leads to unique behaviors but because both Vegas and Dual are DCA algorithms, they suffer the problems associated with DCA that we have identified in previous chapters.

We find that the enhanced loss recovery algorithm of Vegas is actually quite effective. We show that a version of Vegas that has the CAM algorithm disabled will improve TCP throughput by 69% over the Emory path and 40% over the ASU path. We refer to this algorithm as TCP/Reno-Vegas. For comparison, we find that the TCP/Newreno algorithm (a widely accepted enhanced loss recovery algorithm) improves throughput by 72% over the Emory path and 12.8% over the ASU path.

A further result (which we state here but do not discuss further in this chapter) is that if we replace a TCP/Reno connection with either a TCP/Vegas-CAM or a TCP/Dual connection, the congestion level at the bottleneck links does not decrease (using the same methodology as described in Section 8.2).

We organize this chapter as follows. We first present the Vegas and Dual simulation models. In the second section, we provide the results of the simulation analysis. To focus the analysis on DCA, in the second section we use a version of TCP/Vegas that has the enhanced loss recovery algorithm disabled. We refer to this algorithm as TCP/Vegas-CAM.

9.1.1 TCP/Dual Model

In an earlier chapter (i.e., section 2.4.1), we described the TCP/Dual algorithm. In this section, we provide details of the TCP/Dual simulation model. In the previous chapter, we described the extensions to the TCP/Reno model that extracts per packet RTT time samples (i.e., the *tcpRTT* samples). The TCP/Dual model, which also uses the *tcpRTT* samples, sets the *Dmin* and *Dmax* to the minimum and maximum *tcpRTT* value observed during the connection. The authors in [WANG92] set *alpha* to ½ which places the threshold in the middle of the maximum and minimum RTT range. In most of our runs (both the simulation and the traced connections) we periodically see very large RTT values which causes the *dual_threshold* to get very high when *alpha* is ½. As the threshold moves above the average *tcpRTT* value, the level of control exerted by DCA decreases. We saw the same effect in our measurement analysis. When we ran the DCA algorithm on the *tcpRTT* time series data derived from the TCP traces, we found that a *threshold* set to twice the standard deviation was too high and prevented the congestion detection algorithm from predicting any of the loss events. We find that a value of ¼ for *alpha* provides a more reasonable threshold level. Therefore, the TCP/Dual algorithm will use the following threshold:

$$dual\_threshold = \tfrac{3}{4} * Dmin + \tfrac{1}{4} * Dmax.$$

Every *tcpRTT* sample, the *Dmin* and *Dmax* are possibly updated. Every other RTT, if the *tcpRTT* sample exceeds the threshold, the *cwnd* is reduced by 12.5% and the *ssthresh* is set to 2 (packets) forcing congestion avoidance.

9.1.2 TCP/Vegas Model

We use the TCP/Vegas model that is provided with the *ns* simulator. As described in Section 2.4.1, TCP/Vegas represents two independent congestion control enhancements: an additional loss recovery algorithm and the *congestion avoidance mechanism* (i.e., CAM). The following shows that it is necessary to isolate each algorithm to get a clear understanding of Vegas. We illustrate this with a simulation experiment that is designed to compare the performance of TCP/Reno, TCP/Vegas and TCP/Newreno.

TCP/Newreno represents an enhanced loss recovery algorithm. We run run three connections (a Reno, Vegas and a Newreno connection) over the Emory and ASU paths (along with the same level of background traffic that was used in the TCP/DCA analysis in Chapter 8). The experiment consists of five runs (each run is 500 seconds). For each run, we find the relative change in TCP throughput between the two enhanced protocols with respect to the TCP/Reno connection. Table 9-1 shows the average of the results for the five runs. As an example, the first row and column indicates that over the Emory path the TCP/Vegas connection (with CAM and the enhanced loss recovery enabled) experiences an increase in throughput of 1.1% compared to the Reno connection and reduces the timeout percentage by 71%. The timeout percentage is the ratio of the number of retransmissions that required a timeout for recovery to the total number of retransmissions. Over the ASU path, TCP/Vegas improves throughput by 24.7% and reduces the timeout ratio by 49%.

Table 9-1. Comparison of TCP/Vegas and TCP/Newreno with TCP/Reno

Model	TCP/Vegas Throughput Change (95% confidence interval) Change in % of TO's (95% confidence interval)	TCP/Newreno Throughput Change (95% confidence interval) Change in % of TO's (95% confidence interval)
Emory	+1.1% (-4.5,6.7) -71% (-78,-62)	+69% (54,84) -85% (-90,-80)
ASU	+24.7% (12.8, 36.5) -49% (-54,45)	+17.3% (4.9,30) -26.8% (-30, -23)

The throughput improvement experienced by TCP/Vegas is due to the enhanced loss recovery algorithm (we prove this shortly). In the experiment, the TCP/Reno connection experiences a timeout rate of 22% over the Emory path and 35% over the ASU path. Timeouts occur for two reasons. Either multiple packets are dropped within a single RTT or the number of packets in flight when loss occurs is too low such that three duplicate acknowledgements do not arrive at the sender. The Vegas enhanced loss recovery algorithm addresses both problems by using a more aggressive retransmission strategy based on a fine grained retransmission timeout. The scheme allows a recovery to occur without having to wait for three duplicate acknowledgements to arrive. By also checking for fine-grained retransmission timeouts after a

retransmitted packet is acknowledged, the algorithm can reduce further the number of coarse-grained TCP timeouts which will occur if more than one segment is lost within a RTT.

Table 9-1 also shows that the TCP/Newreno connection experiences a similar reduction in the frequency of timeouts and a very large improvement in TCP throughput. As described in [FLOY99], Newreno defines a fast recovery procedure that activates when three duplicate ACKs are received and ends either with a timeout or when an ACK arrives that acknowledges all of the data up to and including the data that was outstanding when the fast recovery procedure began. Essentially, if a "partial ACK" arrives (i.e., an ACK that acknowledges the retransmitted segment and possibly additional segments, but not all of the outstanding data), the sender adjusts the congestion window by the amount of data acknowledged by the ACK and then inflates the cwnd by 1 MSS and sends a new segment. The sender stays in this recovery mode until the highest sent segment prior to entering recovery has been acknowledged.

The TCP/Newreno results suggest that significant benefits can be derived from an enhanced loss recovery algorithm. The TCP/Vegas results (i.e., a 1.1% throughput improvement) are impacted by both the enhanced loss recovery as well as CAM. We find that over the ASU path, TCP/Vegas shows better performance than TCP/Newreno. As the congestion level increases, the benefits of Newreno diminish because the number of packets in flight will generally be low reducing the chances for a single duplicate acknowledgement recovery. We deduce that the level of control exerted by the CAM algorithm drops as the loss rates increase which allows the enhanced recovery algorithm to dominate (we show this in the next section).

We create a version of TCP/Vegas, called TCP/Vegas-Reno, that is supports only the enhanced loss recovery of Vegas (i.e., CAM is disabled). We rerun the previous simulation experiment to see how TCP/Vegas-Reno performs. Table 9-2 shows that by removing the congestion avoidance algorithm (i.e., CAM), the Vegas enhanced loss recovery algorithm leads to comparable (possibly better) performance improvements with the Newreno algorithm. However, it is unclear if it is valid for an algorithm to

continue with a retransmission strategy that is more aggressive than TCP/Reno as the congestion levels increase. We identify this study as a future work item.

Table 9-2. Comparison of TCP/Vegas-Reno and TCP/Newreno with TCP/Reno

Model	TCP/Vegas-Reno Throughput Change (95% confidence interval) Change in % of TO's (95% confidence interval)	TCP/Newreno Throughput Change (95% confidence interval) Change in % of TO's (95% confidence interval)
Emory	+69%(48,90) -69% (-73, -66)	+76% (63,89) -86.5%(-89,-83)
ASU	+46% (22.6,69) -64% (-68.4,-59.6)	+12.8% (-2.2,28) -25% (-35,-15)

For the remainder of this Chapter, we use a version of TCP/Vegas that has the enhanced loss recovery algorithm disabled. We refer to this algorithm as TCP/Vegas-CAM.

9.2 Simulation Analysis

Figure 9-1 shows the simulation results for three end-to-end connections (TCP/Reno, TCP/Dual and TCP/Vegas-CAM) over the Emory simulation model. The simulation is identical to that used in the previous section except that the three end-to-end connections under observation are Reno, Dual and Vegas-CAM. The top curve reflects the *tcpRTT* time series of the TCP/Reno connection. We present further analysis of the behavior of the DCA algorithms used by CAM and Dual in the next subsection. The objective of Figure 9-1 is to introduce the behavior associated with the two DCA algorithms. The lower graph plots the throughput of the Reno connection (dashed-dotted curve), the Vegas-CAM connection (dashed curve) and the Dual connection (solid curve). For the congestion dynamics associated with the run, Vegas-CAM appears to be the most reactive while the Dual algorithm falls somewhere in between Vegas and Reno.

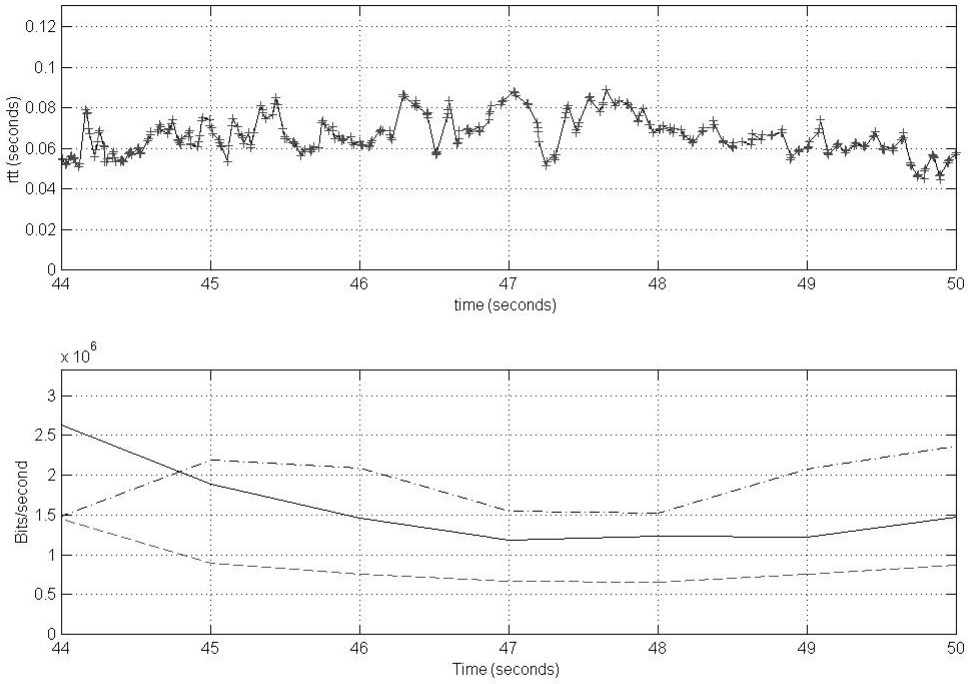

Figure 9-1. Simulation results of Reno, Vegas-CAM and Dual over Emory path

We define the following simulation experiment. Over the two paths, we run three connections simultaneously: TCP/Reno, TCP/Vegas-CAM and TCP/Dual. As we have done in previous experiments, we perform each run (which is 500 seconds) five times. We are interested in observing the end-to-end throughput and packet loss rate experienced by the TCP/Vegas-CAM and TCP/Dual connections compared to the TCP/Reno connection. Table 9-3 summarizes the average results of the experiment. We see that Vegas-CAM degrades TCP throughput by 21.7% and 2.7% over the Emory and ASU paths respectively while TCP/Dual degrades throughput by 13.9% and 19% over the paths. We make the following observations:

- Both algorithms appear to increase the packet loss rate over a more congested path.
- The level of throughput degradation decreases for Vegas-CAM over a more congested path.

We comment on both observations before presenting further analysis of Vegas and Dual. The results indicate that the TCP/Dual and TCP/Vegas connections experience a higher packet loss rate over the ASU path and a lower loss rate over the Emory path. The increase in the loss rate is significantly higher with TCP/Vegas over the ASU path. There might be a minor impact between a DCA reaction to congestion and the base TCP/Reno recovery procedure (we have not studied this). There are minor code differences between the *ns* TCP/Reno model and the TCP/Vegas model in the recovery portions of the algorithm. We confirm that this accounts for at least a part of the difference by simulating a version TCP/Vegas-CAM with the CAM algorithm disabled (i.e., which makes it a TCP/Reno protocol). Comparing the performance of this protocol with that of TCP/Reno, we continue to see that the model exhibits a higher loss rate over the ASU path than the TCP/Reno model by roughly 5-7%. Regardless of the specific cause, the results of TCP/Dual and TCP/Vegas over the ASU path indicate that DCA becomes less effective at avoiding loss as the congestion level increases.

The second observation requires further understanding of the Vegas protocol. In Section 9.2.2 we will show that the explanation involves the use of throughput as opposed to RTT as the metric used in the congestion decision.

Table 9-3. Performance of TCP/Vegas-CAM and TCP/Dual with respect to TCP/Reno

Model	% Reduction of Packet Loss TCP/Vegas-CAM, TCP/Dual (95% confidence interval)	% Reduction of Throughput TCP/Vegas-CAM, TCP/Dual (95% confidence interval)
Emory	-3 (-12.4, 6.2) -6.8 (-22.6, 8.9)	-21 (-27,-15) -14.2 (-22.9,-5.43)
ASU	+7% (-6,21) +1.2% (-9,2)	-7.3 (-20.3,5.8) -18 (-23.3,-12.9)

9.2.1 Analysis of TCP/Dual

In this section we provide insight into the operation and performance of the Dual DCA algorithm. Figure 9-2 provides a visualization of the dynamics experienced by the TCP/Dual connection for a one second

time period during one of the Emory simulation runs associated with Table 9-3. The top graph plots the *tcpRTT* time series that is observed by the TCP/Dual algorithm for a one second period during the simulation. The lower two graphs plot the queue levels at the two bottleneck links. As shown in the network model (i.e., Figure 7-7), link 4-5 is 45mbps and link 7-8 is 155mbps.

TCP/Dual makes a congestion decision every other RTT based on the *tcpRTT*. The diamonds at the top of the *tcpRTT* graph in Figure 9-2 indicate when Dual reacts to an observed increase in RTT. The upside down triangles indicate when the Dual algorithm makes a congestion decision that does not result in a send rate reduction. Three reactions occur at times 80.1, 80.3 and 80.8 seconds. The first two reactions are "unnecessary" as they react to queue fluctuation at link 4-5 that is not associated with packet loss. The third reaction is in response to queue buildup that leads to multiple packet loss at link 4-5. Four packets are lost. The first and second loss event are associated with segments that are sent at time 80.79 seconds. The corresponding queue buildup is not detected until after the segments that eventually are dropped are first transmitted. A congestion decision occurs at time 80.82 seconds that results in a send rate reduction. However the reaction is not able to avoid loss (i.e., in the graph we see that there is a loss tick mark under the diamond which means that Dual reacted but the segment that is sent next is dropped). The fourth loss event is preceded by a significant increase in RTT that is observed in the *tcpRTT* samples, however the increase falls in between congestion decisions (i.e., which can only occur every 2 RTT's) and consequently does not react to the RTT increase.

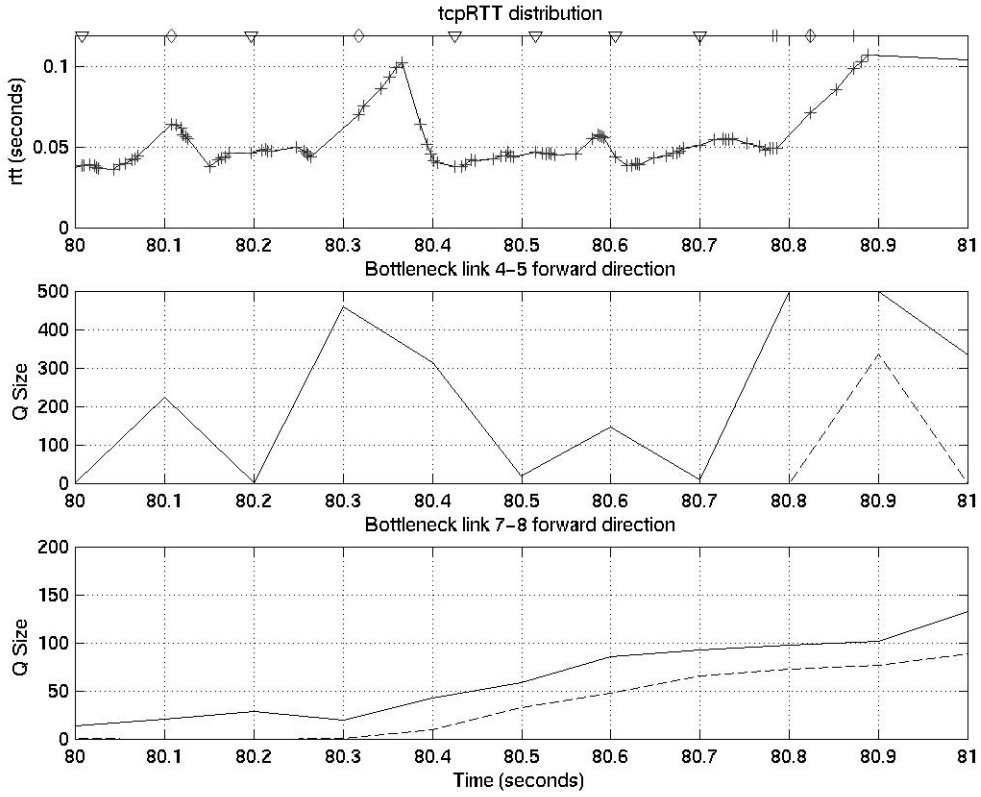

Figure 9-2. TCP/Dual simulation results over Emory path

We summarize the TCP/Dual DCA algorithm as follows:

- We see evidence of the same problems associated with TCP/DCA discussed in Chapter 8:
  - ♦ Congestion indications (i.e., increases in RTT) arrive at the sender too late to avoid loss.
  - ♦ A send rate reduction does not guarantee that loss is avoided.
  - ♦ The algorithm reacts to queue buildup which is not associated with loss
  - ♦ The algorithm is not able to detect queue buildup that occurs at a router with higher link capacities (i.e., the Dual *tcpRTT* samples do not detect the slowly growing queue level at link 7-8 because the queue delays are effectively insignificant compared to the queue delays incurred at link 4-5).
- Dual's infrequent decisions make the congestion probe more granular than the TCP/DCA algorithm which further impedes the algorithms abilities to reliably avoid loss.

- The Dual algorithm is dependent on accurate *Dmin* and *Dmax* values. One large *tcpRTT* sample (perhaps caused by a routing update within the network) can effectively turn off the Dual control algorithm by raising the *dual_threshold* significantly above the average *tcpRTT* level.

9.2.2 Analysis of TCP/Vegas

In this section we provide insight into the operation and performance of the CAM algorithm. Figure 9-3 shows the behavior of TCP/Vegas-CAM for the same one second portion of the Emory simulation discussed in the previous section. The upper curve illustrates the RTT samples that are used by Vegas. The *ns* Vegas implementation computes an RTT sample by averaging all the packet *tcpRTT* samples for the last RTT time period. The two lower curves show the queue levels of the bottleneck links.

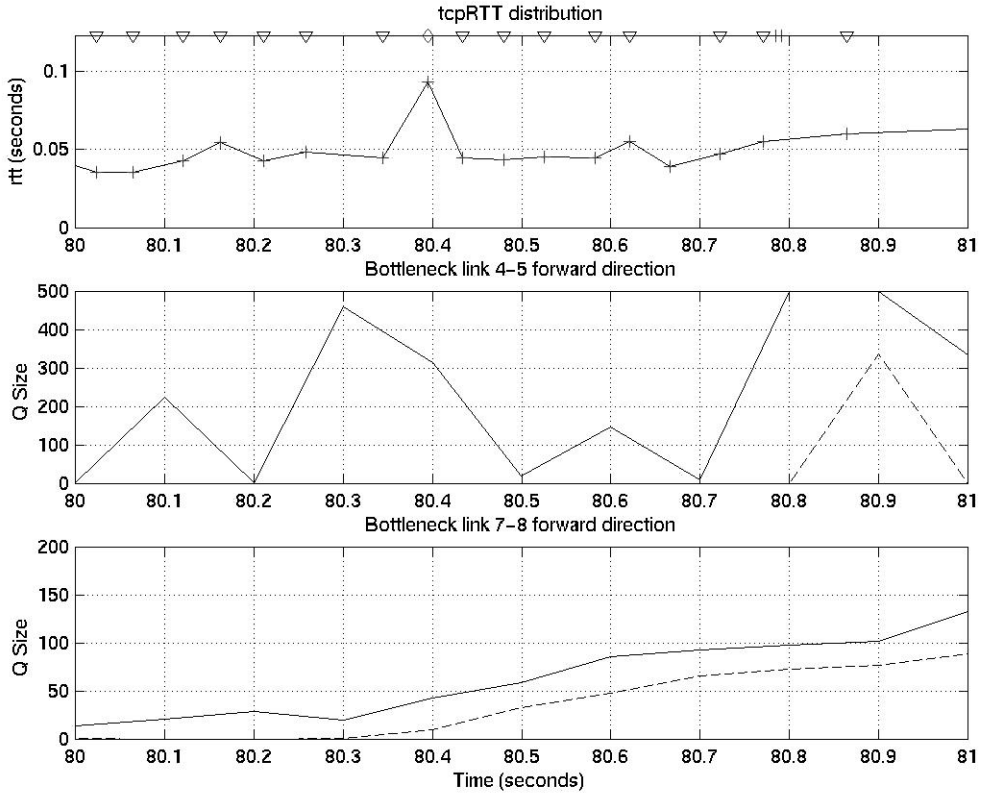

Figure 9-3. TCP/Vegas simulation results over Emory path

The top graph of Figure 9-3 illustrates the congestion reactions by CAM. At the top of the curve, the upside down triangles represent the CAM decision to "stay" at the current sending rate. The diamond indicates a decision to lower the *cwnd* by 1 segment. We observe that CAM reduces the send rate less frequently than TCP/Dual and instead tends to stay at the same rate as a result of the "stay" reactions.

While Dual experienced 4 dropped packets during the 1 second period, TCP/Vegas experiences two losses. As in the Dual case, the first 2 packet drops are not preceded by a significant increase in time for Vegas to be able to react. It is interesting to observe the last RTT sample at time 80.86 seconds does not reflect the very large amount of queue delay that exists (as the TCP/Dual *tcpRTT* samples did). The Vegas RTT sample is on the order of 60 milliseconds and the *tcpRTT* sample taken at the same time is roughly 90 milliseconds. The reason is because the Vegas RTT is the average of some number of previous RTT values (in this case it was 5) and therefore even though the last RTT sample was 90milliseconds the average smooths the RTT. Because of the small send rate reduction and because the algorithm filters short time scale RTT fluctuations (by using the an average of several RTTs), Vegas is really an example of an algorithm that is optimized to respond to long term congestion. This explains why the Emory simulation results reflect only a 3.3% reduction in packet loss (while TCP/DCA was able to lower the rate as much as 14% if allowed to react every RTT).

A difference between Vegas and TCP/Dual (and TCP/DCA) involves the metric that is the basis for the congestion detection mechanism. In TCP/Dual and TCP/DCA, the RTT is used. Vegas however uses RTT indirectly by using variations in throughput as the congestion detection metric. This explains why the throughput degradation experienced by the TCP/Vegas connection decreases over the ASU path compared to the Emory path. The following analysis helps to illustrate this.

Anytime the sampled RTT is larger than the minimum RTT value (i.e., the *baseRTT*) there will be a difference between the actual and the expected throughput. Vegas then converts this difference in throughput to an estimate of the number of buffers consumed by the connection. This *diff* value becomes

the congestion indication which is then compared to static thresholds (i.e., *alpha* and *beta*). The congestion decision is therefore based on the sampled RTT value, the minimum RTT value and the current congestion window. The following shows that the level of control exerted by Vegas is generally driven by the congestion window.

The *Diff* value in bytes can be written as:

$$Diff = (W/BaseRtt - W'/Rtt) * BaseRtt$$

where *W* is the current window, *W'* is the amount of data sent during the measurement period, and *Rtt* is the current RTT sample. As long as the sender has data to send, no packets are lost and the receiver ACKs each packet, *W'* will be the current window, *W*. Therefore:

$$Diff = W - W * BaseRtt / Rtt$$

$$Diff = W(1 - BaseRtt / Rtt)$$

Clearly, *Diff* is 0 when *BaseRtt* = *Rtt*, and is positive when *Rtt* > *BaseRtt*. The upper bound of (*1-BaseRtt/Rtt)* is 1 which means that the largest *Diff* value that can ever be observed is the current window size (which makes sense since the *Diff* estimates the amount of buffers consumed by the flow). As W gets small (in the 1-3 packet range), the *Diff* becomes insensitive to even large RTT variations.

In the Emory simulations discussed earlier in this section, the congestion window is large (in the 7-9 packet range) making the Vegas control algorithm reactive to RTT fluctuations. The congestion window associated with the Vegas connection during the ASU runs is in the 2 to 5 packet range. As explained above, this makes Vegas less responsive to an increase in RTT and explains why we see less reduction in the throughput by Vegas over the ASU path compared to the Emory path. In effect, the CAM algorithm will make higher bandwidth TCP flows more reactive than lower bandwidth flows.