

# The Incremental Deployability of RTT-Based Congestion Avoidance for High Speed TCP Internet Connections

**Jim Martin**

**Networking Software Division, IBM Corporation  
Research Triangle Park, NC, 27709-12195  
jjm2@us.ibm.com**

**Arne Nilsson**

**Department of Electrical and Computer Engineering  
North Carolina State University,  
Raleigh, NC, 27695-7914  
nilsson@eos.ncsu.edu**

**Injong Rhee**

**Department of Computer Science  
North Carolina State University  
Raleigh, NC, 27695-7534  
rhee@eos.ncsu.edu**

## ABSTRACT

Our research focuses on end-to-end congestion avoidance algorithms that use round trip time (RTT) fluctuations as an indicator of the level of congestion. The algorithms are referred to as *delay-based congestion avoidance* or DCA. Due to the economics associated with deploying change within an existing network, we are interested in an incrementally deployable enhancement to the TCP/Reno protocol. TCP/Vegas, which is a DCA algorithm, has been proposed as an incremental enhancement. Requiring relatively minor modifications to a TCP sender, TCP/Vegas has been shown to increase end-to-end TCP throughput primarily by avoiding packet loss. We study DCA in today's best effort Internet where IP switches are subject to thousands of TCP flows resulting in congestion with time scales that span orders of magnitude. Our results suggest that RTT-based congestion avoidance may not be reliably incrementally deployed in this environment. Through extensive measurement and simulation, we find that when TCP/DCA (i.e., a TCP/Reno sender that is extended with DCA) is deployed over a high speed Internet path, the flow generally experience degraded throughput compared to an unmodified TCP/Reno flow. We show that the congestion information contained in RTT samples is not sufficient to reliably predict packet loss and that the congestion reaction by a DCA flow, assuming that the flow consumes a small fraction of the resources at the bottleneck, has minimal impact on the congestion level over the path when the total DCA traffic at the bottleneck consumes less than 10% of the bottleneck bandwidth.

## 1 Introduction and Motivations

A fundamental aspect of protocols that operate in TCP/IP networks is the congestion control mechanisms utilized throughout the network. Various forms of congestion avoidance algorithms designed to prevent congestion have been proposed to improve end-to-end performance. The addition of slow-start and congestion avoidance to TCP in the late 1980's was crucial to ensuring the stability of the Internet [JACO89,ALLM99]. More sophisticated end-to-end congestion avoidance algorithms that monitor round trip time (RTT) and use increases in RTT samples as an indication of congestion have been proposed (e.g., TCP/Vegas, TCP/Dual) [BRAK94, WANG92]. We refer to such algorithms as *delay-based end-to-end congestion avoidance* (DCA). The benefits claimed by DCA algorithms are increased TCP throughput due to reduced numbers of packet losses and timeouts, and reduced congestion level over the path. Another advantage is that they do not require anything from the network. All these properties support the case for the incrementally deployability of such algorithms in today's best effort Internet. However, previous studies of DCA algorithms have concentrated on networks where DCA flows consume a significant percentage of the total traffic. In this paper, we focus on the performance of DCA algorithms in a high speed network where DCA flows constitute only a fraction of the total traffic.

The following describes the attributes of DCA:

- DCA augments the TCP/Reno protocol.
- DCA monitors TCP packet round trip times allowing the algorithm to reside entirely within the TCP sender.
- RTT variations are assumed to be caused by changes in queueing delays experienced by packets being timed.
- Based on RTT variations, DCA makes congestion decisions that reduce the transmission rate by some percentage by adjusting the TCP congestion window (*cwnd*).

The importance of an *incrementally deployable* enhancement cannot be overstated. It can take years for Internet protocols to standardize and even longer for Internet standards to be widely deployed. An incremental enhancement to TCP that meets the following requirements will have the best chance for deployment:

- It must improve the throughput of the TCP connection that employs the enhancement.
- It should not reduce the performance of other competing TCP flows on the same path where the "enhanced" TCP flow travels.
- Ideally it requires changes only to a TCP sender.

Furthermore, the above properties must hold regardless of the number of "enhanced" TCP flows on the same end-to-end path. This implies that the properties must hold even if there is only one "enhanced" flow in the path. To support a wide deployment of TCP enhancements, there must exist sufficient economical incentives for adopting them. When these incentives are weak and further, enhancements may even result in temporary sacrifice of resources for users at least for a while before its wider deployment, the eventual wide deployment of such protocols becomes unlikely. Therefore, our research assesses the benefits associated with incremental deployment where "enhanced" flows constitute only a fraction of the total traffic in the bottleneck link.

We further limit the scope of our study of DCA to Internet environments where the lowest link capacity along the path is 10 Mbps. Measurement studies have shown that Internet switches, most of which support link speeds of at least 45 Mbps, are subject to thousands of low bandwidth, ON/OFF TCP flows [CLAF97,THOM97]. Thus, we study the performance of DCA under realistic today's Internet environments where thousands of TCP flows may come and go over short time periods.

In this paper, contrary to the findings of prior study on DCA algorithms, we find evidence suggesting that RTT-based congestion avoidance may not be reliably incrementally deployed over high speed Internet paths. Through measurement study over 7 typical high speed Internet paths, we find that congestion information contained in RTT samples of TCP cannot be reliably used to predict packet loss. The success of DCA highly depends on a strong correlation between packet loss events and increases in RTT prior to the loss. By tracing TCP connections over multiple high speed Internet paths, we observe that the level of correlation is weak. Depending on the path, we find that only 7-18% (on average) of observed loss events were preceded by a significant increase in RTT. Furthermore, even though we assume that every loss predicted by DCA is avoided, our analysis indicates that noise in RTT samples causes degraded throughput by guiding TCP to reduce at wrong times (when there is no loss). We modify an analytic TCP throughput model given by Padhye et al. [PADH98] to assess the impact of DCA's congestion reactions (both right and wrong ones). Based on measured data the throughput model predicts that DCA degrades TCP throughput in the range of 4% to 70%. Through simulation and Internet measurement, we show that this lack of information in RTT samples of TCP is due to the following two reasons.

- ◆ A TCP constrained RTT congestion probe is too coarse to accurately track the bursty congestion associated with packet loss over high speed paths.
- ◆ A DCA algorithm cannot reliably assess the congestion level at the router. Short term queue fluctuations which are not associated with loss at a router, persistent queueing and congestion at multiple bottlenecks (especially when the congested routers are provisioned differently) make it difficult for a DCA algorithm to make accurate congestion decisions.

Another claimed benefit of DCA is that it reduces the overall queue level so that it contributes to reduced packet loss at the bottlenecks. This could be indeed true when more reactive flows, such as DCA, consume a large percentage of the total bandwidth in the bottleneck link. However, this becomes less obvious when they constitute only a fraction of the total traffic in a congestion link. As DCA gives up more bandwidth, more aggressive flows such as TCP/Reno are likely to steal more bandwidth, leaving the queue level unaltered.

To verify this conjecture, we resort to simulation using the *ns* simulation tool [NSSIM] since the queue levels within the network cannot be observed through the Internet measurement data. We construct *ns* models that exhibit similar end-to-end traffic characteristics as two of the 7 Internet paths being measured. These models closely match the loss behaviors and burstiness of RTT variations of the two paths. Under these models, we first confirm the result we found from the Internet measurement data: a DCA flow suffers from throughput degradation. Further, we

observe that a single DCA flow does not alter the loss and RTT dynamics of the simulated path significantly. It is also interesting to see how the “global” benefits of DCA might scale as the amount of DCA traffic in the network increases. We observe that the loss rate at a congested router only begins to decrease when DCA accounts for at least 10% of the total traffic. At a high speed switch, this can correspond to traffic generated by hundreds of TCP flows which clearly exceed incremental deployment.

This paper is organized as follows. First we overview related work. Then we present the measurement analysis and throughput analysis followed by the simulation analysis. We end the paper with conclusions and a discussion of future work.

## 2 Related Work

[BRAK94,AHN95] show that TCP/Vegas can improve TCP throughput over the Internet by avoiding packet loss. However, these studies were based on Internet paths that existed in the early 1990’s which generally involved at least one T1 speed link and consequently allows any given flow to consume a significant fraction of available bandwidth. The studies also did not isolate the impact of the congestion avoidance algorithm (i.e., CAM) from the enhanced loss recovery enhancement.

A key aspect of our research focuses on assessing the ability of a TCP constrained RTT-based congestion sampling algorithm in predicting future packet loss events. Several previous studies are relevant. Bolot studied end-to-end packet delay and loss behavior over the Internet [BOLO93]. However, he did not focus on the correlation between an increase in RTT with packet loss. Moon et. al., looked at the correlation that exists between a loss conditioned delay (i.e., a packet delay immediately preceding a loss event) and packet loss [MOON99]. Their motivations were similar to ours in that they wanted to see if an endpoint could predict loss. They found a higher level of correlation than we did. Their method utilized a more frequent, one-way delay based UDP-based probe technique. A further difference is that they do not take into account the feedback time associated with TCP’s closed loop control and consequently their results do not correctly portray the level of correlation that is usable by TCP. Paxson also looked at the correlation between one way packet delay variation and loss [PAXS97]. He concludes that loss is weakly correlated to rises in packet delay and conjectures that the linkage between the two is weakened by routers with large buffer space and/or because the end-to-end delay reflects the accruing of a number of smaller variations into a single, considerably larger variation. Our work finds a similar result and we also present a set of conjectures to explain the measured results. Additionally, we also explore what happens if a DCA algorithm attempts to utilize the little correlation that does exist between increases in delay and packet loss.

### 3. Measurement Analysis

The objective of the measurement analysis is to show that an increase in a per packet RTT sample (i.e., the *tcpRTT* samples) is not a reliable indicator of future packet loss events and cannot be used to improve TCP throughput. In overview, we trace many TCP connections over different paths. We then post-process the trace files to extract the *tcpRTT* time series (along with the loss indication) associated with each traced connection. This data is the basis of our analysis.

#### 3.1 Data Collection Methodology

We selected seven high speed Internet paths. Each path consists of many hops (at least 11) with a minimum link capacity of 10 Mbps. The sender of each TCP connection is a host located on the campus of North Carolina State University, each of the 7 receivers is located over the Internet. We run a bulk mode TCP application between the host and each destination. The TCP sender in our experiments as well as the trace point (via *tcpdump*) is a 333Mhz PC running freeBSD. The machine is equipped with a 3COM PCI ethernet adapter and is attached to a campus network via a 10 Mbps ethernet connection.

Table 3-1 describes each of the 7 paths. Of the 7 paths, five are located outside of North America. For three of the paths, we used the *ttcp* application [MILE84]. For the others we used a Unix tool called *echoping* which can send any amount of TCP data to a discard server [BORT99]. The five discard servers were located on Web servers. Over the course of five days we traced TCP connections at regular intervals throughout the day (five runs each day beginning at 9:00AM followed by a run every two hours). Each run transferred between 6 Mbytes and 20Mbytes (depending on the path) and lasted anywhere from 3 minutes to 45 minutes depending on the level of congestion over the path.

Table 3-1 Summary of traced paths

Path #	Destination Host	# Hops	Maximum Window	MSS	service
1	dilbert.emory.mathcs.edu	12	17376	1448	ttcp
2	comeng.ce.kyungpook.ac.kr	19	17376	1460	ttcp
3	cgg.ee.ntust.edu.tw	17	8760	1460	ttcp
4	www.nikhef.nl	17	4096	512	discard
5	www.snafu.de	13	17520	1460	discard
6	icawww1.epfl.ch	18	8760	1460	discard
7	www.eas.asu.edu	14	8760	1460	discard

Table 3-2 summarizes the measured performance of each path. Path 1 exhibits the best performance and path 7 is the worst. Table 3-3 shows the average throughput measured for the runs over each path based on the time of day (Eastern Standard Time) corresponding to the time when the connection was traced. The only consistent trend is that all paths (except for path 2) experienced their worst performance during one of the afternoon runs.

Table 3-2. Summary of measured performance

Path #	Avg RTT (seconds) (standard deviation)	Avg throughput (Kbytes/sec) (standard deviation)	Loss rate(%) mean (standard deviation)	% of loss that ended in Time-out mean (standard deviation)
1	.066(.012)	185.1(77.5)	.8(1.1)	13.1(8.0)
2	.249(.056)	46.4(19.2)	1.7(3.2)	14.5(11.6)
3	.32(.037)	12.8(7.6)	6.5(7.1)	48.3(9.8)
4	.117(.02)	23.8(8.5)	.89(1.1)	56.1(14.9)
5	.174(.017)	71.7(22.4)	.65(.78)	15.2(9.2)
6	.171(.01)	46.2(6.6)	.55(1.1)	11.2(7.5)
7	.179(.06)	10.8(7.3)	10.5(4.1)	47.9(13.4)

Table 3.3. Average throughput in Kbytes/sec with standard deviation based on time of day (EST)

Path #	9:00AM	11:00AM	1:00PM	3:00PM	5:00PM
1	228.4(42.6)	192.6(57.4)	241.9(63.7)	113.8(65.3)	148.6(90.4)
2	48.7(18.9)	41.5(20.5)	42.4(18.7)	52.8(15.4)	46.7(27.5)
3	11.2(7.4)	14.3(4.8)	16.5(7.2)	15.7(8.7)	5.9(7.4)
4	22.2(10.3)	27.4(8.4)	17.1(5.2)	22.4(7.1)	28.9(8.3)
5	84.2(13.7)	73.4(22.4)	77(15.1)	47.8(24.6)	76.1(22.5)
6	45.2(6.2)	48.7(2.2)	47.5(2.7)	40.1(11.7)	49.7(1.1)
7	17.7(8.7)	11.5(5.7)	7.7(5.3)	5.8(4.2)	11.2(8)

We post-processed a *tcpdump* trace to obtain the round trip delay associated with each data packet that has a unique acknowledgement. We refer to these RTT samples as the *tcpRTT* time series. The *tcpRTT* samples provide more congestion information than that provided in TCP's existing RTT algorithm (used for TCP's retransmit timeout calculation) because the *tcpRTT* samples are more frequent and they are based on a more precise time measurement.

The algorithm that is used to generate the *tcpRTT* time series is summarized as follows. (We describe the algorithm as it would be implemented by a TCP/Sender even though we actually run the algorithm on a *tcpdump* trace.) The sender time-stamps the departure time of every packet. To filter out error in RTT samples caused by the TCP delayed acknowledgement, *tcpRTT* samples are generated only for the highest segment acknowledged by an ACK. Furthermore only ACKs that acknowledge more than one segment of data will generate a *tcpRTT* sample. During periods of recovery, the algorithm does not take any *tcpRTT* samples to avoid errors.

The *tcpRTT* time series consists of the following tuple:

$$i : (time_i, tcpRTT_i, l_i) \text{ where } l_i = \begin{cases} 1 \\ 0 \end{cases}$$

When  $l_i = 1$ , this implies that the next segment sent after  $time_i$  is dropped and when  $l_i = 0$ , the segment is not dropped. In the event that multiple loss events are associated with the same  $tcpRTT$  sample, rather than having a duplicate entry, we keep only one. Therefore, our analysis treats a burst of loss as a single loss event. Also, packets that are retransmitted more than one time will be considered as separate loss events as long as they have a unique  $tcpRTT$  sample.

### 3.2 Analyzing the Loss Conditioned Delay

We are interested in learning if the  $tcpRTT$  sample prior to loss (or perhaps the average of some small number of  $tcpRTT$  samples prior to loss) is greater than a smoothed average of preceding  $tcpRTT$  samples. In other words, we want to know how frequently loss events in a connection might have been predicted by monitoring RTT. To help assess this, we have developed three metrics which we apply to  $tcpRTT$  time series data.

#### 3.2.1 Loss Conditioned Delay Indication Metric

The *correlation indication metric* counts the number of times that the  $tcpRTT$  samples just prior to packet loss are greater than the average of some previous  $tcpRTT$  samples. We define the following delay event:

$$\{sampledRTT(x) > (windowAVG(w) + std)\}$$

The  $sampledRTT(x)$  is the average of the  $x$  number of  $tcpRTT$  samples prior to the transmission of a dropped segment. We refer to this as the *loss conditioned delay*. In other words:

$$sampledRTT_i(x) = \frac{\sum_{j=(i-x+1)}^i tcpRTT_j}{x}$$

The parameter,  $x$ , controls the size of the moving window associated with the  $sampledRTT$  and determines the responsiveness of the algorithm. Similarly,  $windowAVG(w)$  is a moving window average of the previous  $w$   $tcpRTT$  values prior to the transmission of a segment that is dropped. The  $std$  is the standard deviation associated with the  $windowAVG(w)$ . This serves to filter  $sampledRTT$  values.  $sampledRTT(x)$  approximates the “instantaneous” RTT values while the  $windowAVG(w)$  is a longer term average. The difference,  $sampledRTT(x) - windowAVG(w)$ , is reflective of an increase or decrease in queue delays (relative to the average of the immediately preceding  $w$  RTT samples) as experienced by a packet from which the latest  $tcpRTT$  sample is obtained.

For a given run, we calculate the  $sampledRTT(x)$  and the  $windowAVG(w)$  associated with each loss event. We count the number of times that the delay event (i.e.,  $(sampledRTT(x) > windowAVG(w) + std)$ ) is true for each loss.

Dividing this count by the total number of packet loss occurrences estimates the probability that the  $tcpRTT$  samples prior to a loss are higher than some average. We refer to this as the *correlation indication metric* and define it as follows:

$$P[sampledRTT(x) > windowAVG(w) + std]$$

Different values of  $x$  and  $w$  can provide useful information. For example, a value of (2,5) for the  $(x,w)$  pair provides an indication that the most recent  $tcpRTT$  samples are increasing while a value of (2,200) is an indicator that the magnitude of the  $tcpRTT$  samples is greater than the average over some larger amount of time. Through experiments, we found that the  $(x,w)$  pair of (2,20) is generally most effective in predicting loss events across a range of path dynamics.

Table 3.4.  $P[sampledRTT(2) > windowAVGRTT(20) + std]$

Path #	9:00AM	11:00AM	1:00PM	3:00PM	5:00PM
1	.186(.1)	.2(.036)	.137(.068)	.162(.034)	.149(.022)
2	.19(.044)	.137(.04)	.135(.028)	.15(.022)	.107(.039)
3	.065(.047)	.142(.064)	.14(.058)	.14(.14)	.08(.1)
4	.34(.06)	.11(.06)	.1(.017)	.132(.165)	.099(.07)
5	.21(.08)	.16(.07)	.21(.16)	.128(.022)	.19(.17)
6	.34(.29)	.1(.06)	.047(.036)	.054(.035)	.097(.077)
7	.081(.011)	.072(.015)	.073(.022)	.063(2.0)	.068(.023)

Table 3.4 illustrates the results of the metric applied to the traced data grouped by paths and by time of day (i.e., the time at which the trace was obtained). The parameters of the analysis were (2,20) with a threshold of one standard deviation. As an example, the path 1 data associated with the 9:00AM traces suggests that 18.6% of all loss events were preceded by a significant increase in RTT. Paths 4 and 6 were able to identify 34% of the loss events. Path 7 was able to detect loss the least frequently (due to the higher congestion levels). Taking the average of the results over each time period for each path, the average loss conditioned delay indication metric ranged between 7 – 18% depending on the path.

This result by itself is not indicative of the performance of DCA. It simply says that in general it would be difficult for DCA to reliably predict loss events over the Internet. The results clearly indicate that some level of correlation exists between loss and increases in RTT. The more significant problem (which we analyze in a subsequent section) is that in its attempts to avoid packet loss, a DCA algorithm will react unnecessarily (i.e., reacting to an increase in RTT that would not have lead to loss) resulting in TCP throughput degradation.

### 3.2.2 Loss Conditioned Delay Correlation Metric (LCDC)

The *loss conditioned delay correlation metric* (LCDC) provides a visual indication of the magnitude and time scale of correlation that exists between an increase in *tcpRTT* samples and loss events. Our algorithm is essentially identical to that used in [MOON99]. Moon et. al., define a *lag* that is used in calculating the average delay conditioned on loss. For each packet loss occurrence in a trace, lag ‘-1’ is the *tcpRTT* sample prior to the transmission of a segment that is lost (and lag ‘-2’ is the second *tcpRTT* sample before the transmission of the lost segment, up to some large number of samples such as lag ‘-300’). Likewise, lag ‘+1’ is the *tcpRTT* sample that is obtained after a dropped segment is initially transmitted (up to lag ‘+300’). The *average packet delay conditioned on a loss at a lag j* is defined to be the average of *tcpRTT* samples of packets whose *j*<sup>th</sup> prior packet is lost. DCA has a much better chance of being successful if this metric shows a distinct peak in the loss conditioned delay in the lags immediately prior to packet loss indicating that there is a tendency for the *tcpRTT* values to increase prior to loss.

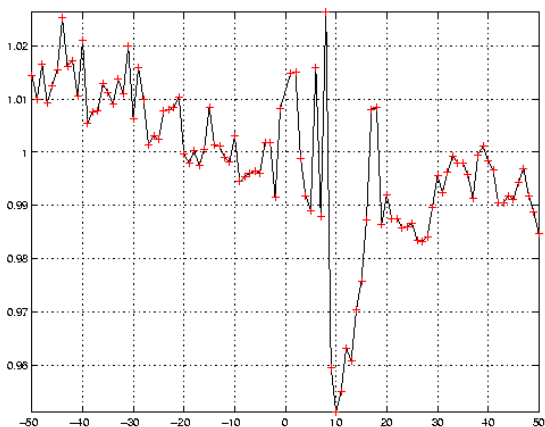


Figure 3-1 LCDC metric for path 1

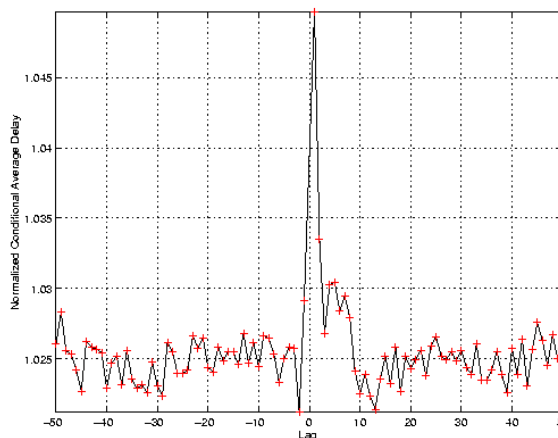


Figure 3-2. LCDC metric for path 7

Figure 3-1 illustrates the metric applied to the data obtained over path 1 (the aggregate data over all 25 runs). The data points plot the values of the average packet delay conditioned on loss for both positive and negative lags with the average delay normalized with the average RTT observed throughout the trace. The figure indicates some correlation exists at lag ‘-1’ and an even larger level at lag ‘-10’. For most of the paths, we see a higher level of correlation associated with the *tcpRTT* samples that are obtained *after* the dropped segment is initially transmitted. The LCDC results suggest that the time scale of the congestion associated with packet loss tends to be on the order of several *tcpRTT* samples. Figure 3-1 shows that the peak in the normalized loss conditioned delay is not unique which implies that the level of correlation between an increase in RTT and loss is fairly weak. Figure 3-2 illustrates the metric applied to path 7. Here we do see a unique peak in the average loss conditioned delay, however the peak is centered around lag ‘+1’. The *tcpRTT* samples obtained from packets that travel over the network closer in time to

the loss events tend to reflect a higher level of correlation suggesting that the queue buildup associated with some percentage of loss at a router is of time scale less than a RTT. The metric results from the five other paths are similar. We find less correlation than that observed in [MOON99]. We believe the reason is because our method measures the correlation observed by a TCP constrained sender-based probe algorithm rather than by a more frequent one-way delay UDP-based probe.

### 3.2.3 Loss Conditioned Delay Cumulative Distribution Function (CDF) Metric

The *loss conditioned delay cumulative distribution function (cdf) metric* simply plots the cdf's of the *tcpRTT* and *sampledRTT* distributions. The latter provides the probability of a packet loss for a given *tcpRTT* value. Plotting the cdf's of the two distributions together illustrates the relationship between the level of *tcpRTT* prior to loss (i.e., the *sampledRTT* values) with all *tcpRTT* samples. If the two distributions appear identical, this suggests that there is nothing statistically unique about the *tcpRTT* samples prior to loss (compared to other *tcpRTT* samples). Figure 3-3 illustrates the metric results applied to path 1. The darker colored CDF is the *tcpRTT* distribution and clearly shows the variability associated with the RTT samples. The light colored CDF illustrates the *sampledRTT* distribution. The metric shows that the RTT that precedes loss is not restricted to the larger RTT samples which clearly would make it difficult for a DCA algorithm to uniquely identify the RTT increase associated with loss. The metric applied to the other 6 paths shows similar results.

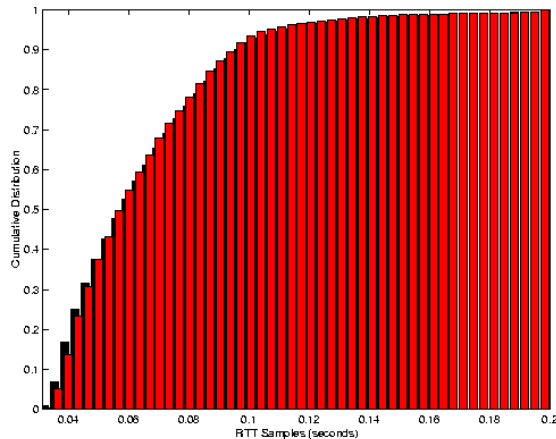


Figure 3-3. Loss conditioned delay CDF for path 1

### 3.3 Throughput Analysis

For each traced TCP connection we run a simple DCA algorithm over the *tcpRTT* time series. In brief, the algorithm assesses the following congestion decision based on the delay event defined earlier:

$$sampled(2) > windowAVG(20) + std$$

When the delay event is true, TCP/DCA reduces the congestion window (*cwnd*) by 50% which is equivalent to the TCP reaction of a loss event that is recovered using the base TCP loss recovery algorithm (i.e., via a triple duplicate acknowledgement which we refer to as a TD event). A 50% send rate reduction is justified for two reasons. First, the RED/ECN algorithm mandates a 50% *cwnd* reduction in response to a congestion indication [FLOY99]. Second, a 50% reduction (rather than a smaller reduction) improves the chances that a loss event will be avoided. We explore the use of smaller *cwnd* reduction values in the simulation analysis in the next section.

Our method assumes that a DCA algorithm makes a congestion decision prior to the transmission of all packets that are dropped. For the *tcpRTT* samples that precede loss events, we perform the congestion decision and increment the variable *lossAvoidedCount* if the decision is true. We further assume that the algorithm will make additional congestion decisions as the *tcpRTT* samples are generated. We limit the frequency of congestion reactions to once per congestion epoch. A congestion epoch begins when the RTT increases and terminates when the RTT returns to some baseline level. When DCA reacts, a count (*congestionReactionCount*) is incremented. If a loss event occurs within one RTT time period, we also increment *lossAvoidedCount* (i.e., we assume the reaction avoids the loss). The base loss rate (*p-loss*) experienced by the connection is the number of retransmissions (*numberRetransmits*) divided by the number of packets transmitted (*packetsSent*).

We adapt a TCP throughput model [PADH98] to help quantify the tradeoff involving DCA reactions that are successful and the DCA reactions which are unnecessary. The model has one component that models the throughput loss caused by TD events and a second (and separate) component that models the loss caused by timeout recoveries. The probability of a TD event and that of a timeout are both derived from a base packet loss probability. We extend the model to use two loss rates instead of one. One probability (*p-TD*) represents the packet loss rate that is used to model recoveries by TD events (we treat DCA event reactions as TD events):

$$p\text{-}TD = (numberRetransmits + congestionReactionCount - lossAvoidedCount) / packetsSent$$

A second probability (*p-timeout*) represents the packet loss probability that is used to generate the probability of a timeout and accounts for the reduction in the number of timeouts due to the DCA delay reactions that succeed in avoiding loss (and the subsequent timeout recovery).

$$p\text{-}timeout = (numberRetransmits - lossAvoidedCount) / packetsSent$$

If DCA is successful, the *p-TD* will be similar in value with *p-loss* and the *p-timeout* will be much lower than *p-loss*. In actuality, the *p-timeout* might be several times lower than *p-loss*, however the *p-TD* will be several times larger than the *p-loss*. We compare the predicted DCA throughput with the predicted throughput of a Reno connection (using the original, unadjusted throughput model) to determine the amount of degradation caused by TCP/DCA.

A crucial assumption that we make in our analysis is that a reaction by DCA will not significantly impact the congestion process of the observed paths. If this were not true, then the DCA reactions might significantly alter the the RTT variations over the path and lower the loss rates which would invalidate our method. We validate this assumption in section 4 of this paper using simulation.

Table 3-5 illustrates the predicted reduction in throughput caused by DCA. Each entry in the table represents the level of throughput reduction (averaged over five runs) along with the standard deviation and the 95% confidence interval. As an example, based on the 9:00 AM data, if we replace a TCP/Reno flow over path 1 with a TCP/DCA flow, the DCA connection would experience degraded throughput in the range of 40 to 70% (with 95% confidence) compared to the original TCP/Reno flow. The amount of degradation decreases as the loss rate increases. This makes sense as the ratio of unnecessary decisions to correct decisions by DCA (i.e., DCA reactions that actually avoided a loss) tends to decrease as the congestion level increases simply because there are fewer RTT increases that are not associated with loss.

Table 3-5. Percentage of throughput degradation: mean (standard deviation) with 95% confidence intervals

Path #	9:00AM	11:00AM	1:00PM	3:00PM	5:00PM
1	-54(13) (-70,-40)	-50(9) (-60,-40)	-53(17) (-73,-34)	-33(15) (-50,-16)	-44(22) (-70,-20)
2	-50(14) (-66,-33)	-46(14) (-62,-29)	-47(17) (-67,-26)	-59(7.6) (-68,-50)	-50(27) (-81,-19)
3	-20(14) (-36,-4)	-20(6) (-28,-13)	-26(11) (-39,-13)	-25(19) (-46,-3)	-7(11) (-20, 5.5)
4	-39(15) (-56,-21)	-50(2) (-64,-36)	-41(12) (-55,-28)	-49(9.4) (-60,-38)	-54(11) (-66,-41)
5	-43(20) (-66,-20)	-57(6) (-63,-50)	-59(6.1) (-66,-52)	-56(10) (-67,-45)	-62(2) (-65,-59)
6	-41(12) (-55,-27)	-45(8) (-54,-36)	-33(12) (-47,-20)	-20(17) (-39,0)	-31(16) (-49,-13)
7	-19(9) (-29,-9)	-13(7) (-21,-4)	-5.4(6.6) (-13,2.2)	-4.2(2.1) (-6,-2)	-12(10) (-24,-1)

## 4 Simulation Analysis

The objectives of the simulation analysis are to validate our measurement analysis and also to extend it in a manner that was not possible with measurement. In particular, the objectives are:

- Obtain additional insight into why a DCA congestion probe can predict at the most (on average) 7-18% of loss events. We can only do this by looking at the actual bottleneck link queue levels along with the *tcpRTT* time series.
- Validate our measurement results that a TCP/DCA algorithm will indeed degrade TCP throughput as compared to TCP/Reno.
- Provide additional validation of our result by showing that TCP/Vegas and TCP/Dual also degrade throughput.



bs	200	200	200	900	200	200	200	600	200	900	200	200	200	200		
pd	.1	.5ms	.1m	.25ms	3.5ms	.07ms	.5ms	.5ms	.9ms	25ms	.5ms	.4ms	.25ms	.5ms	.1ms	
lc	10m	155m	155m	155m	45m	155m	155m	155m	100m	155m	45m	155m	100m	100m	10m	
h1	----- r1	---- r2	----- r3	---- r4	----- r5	----- r6	----- r7	----- r8	----- r9	----- r10	----- r11	--- r12	--- r13	--- r14	---- dh1	
h2	--															-- dh2
h3	--															-- dh3
	ncsu.net		isp # 1			isp # 2				isp # 3			asu.edu			

Figure 4-1-2. Network model of ASU path

We develop a simulation model for the ASU path in a similar manner (illustrated in Figure 4-1-2). Our measurements indicates that the path is highly congested. The majority of loss and congestion occurs at link 8-9 which is located at the MAE-EAST exchange. However, as in the Emory path, we saw evidence of multiple points of congestion. Therefore, we constructed the background traffic such that there was high loss and sustained congestion at link 8-9 and bursty congestion (with low loss) at links 4-5, 10-11 and 7-8.

We design a set of TCP and UDP flows to create bi-directional background traffic. We found that even when using thousands of low bandwidth, ON/OFF TCP flows (using pareto traffic generator configured to emulate the traffic generated by a “web” user as described in [BARF98]) we could not duplicate the burstiness associated with RTT variations observed in the traces over the paths. Therefore we used a combination of TCP flows (several hundred) along with several high bandwidth, ON/OFF UDP flows. In the Emory path, link 4-5 consists of roughly 80% TCP traffic and 20% UDP. Link 7-8 consists of 12% UDP (and 88% TCP) traffic. In the ASU path, a larger percentage of traffic is TCP (92% at all links except for link 7-8 where 50% of the traffic is UDP). For the ASU path, we tuned the background traffic so that the majority of loss occurs at link 8-9. We used several hundred low bandwidth TCP sources to create the heavy load at link 8-9.

Figures 4-2 and 4-3 illustrate a portion of observed behavior over the Emory paths (the measured behavior and the simulated behavior respectively). The top curve in both figures plots the *tcpRTT* time series (the “+” marks forming the curve are the samples) of an end-to-end TCP/Reno connection between h1 and hd1. The “+” marks along the top border of the graph identify the *tcpRTT* samples that occur just prior to the transmission of a segment that is dropped somewhere along the path. The lower curve plots the TCP “goodput” (i.e., the rate that data is acknowledged) in intervals of .5 seconds (the dark line) and 2 seconds (the dashed line). The end-to-end TCP/Reno connection under observation is configured similarly to the TCP stack used in the measurements. The maximum window size is limited to 12 packets with a maximum segment size of 1448 bytes. An ftp application sources the TCP connection. The sink is configured to perform delayed acknowledgements. The loss rate at link 4-5 is 1.5%, .86% at link7-8 and .6% at link 10-11. The loss rate experienced by the traced TCP/Reno connection is about 1%. The loss rate experienced by the simulated TCP/Reno connection is .55% which explains why the throughput exhibited in Figure 4-3 is higher than the measured results shown in Figure 4-2. We ran the three metrics that we defined in the previous section on the *tcpRTT* extracted from traces of simulation runs and found that the model

reasonably captures the level of correlation between packet loss and increases in RTT that we observed over the actual path.

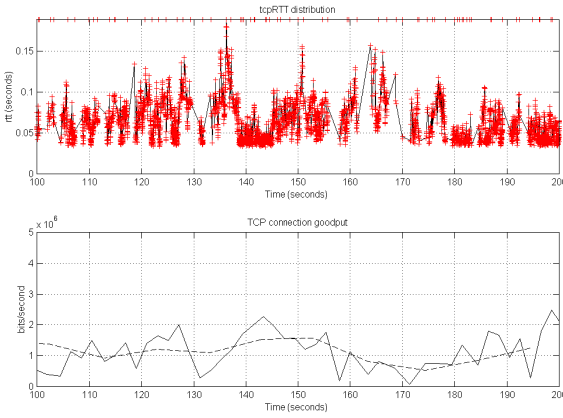


Figure 4-2. Measured results over Emory path

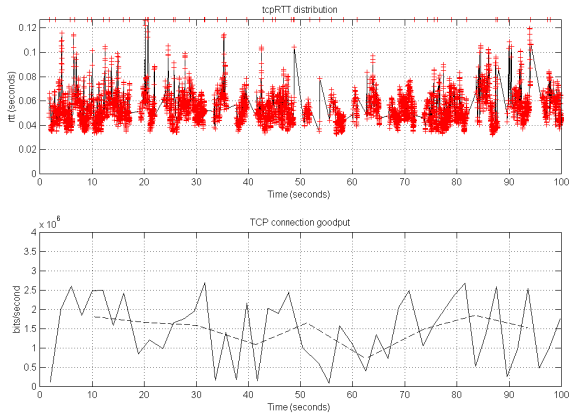


Figure 4-3. Simulated results over Emory path

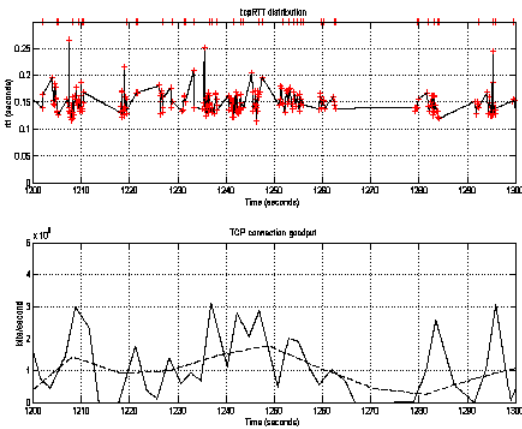


Figure 4-4. Measured results over ASU path

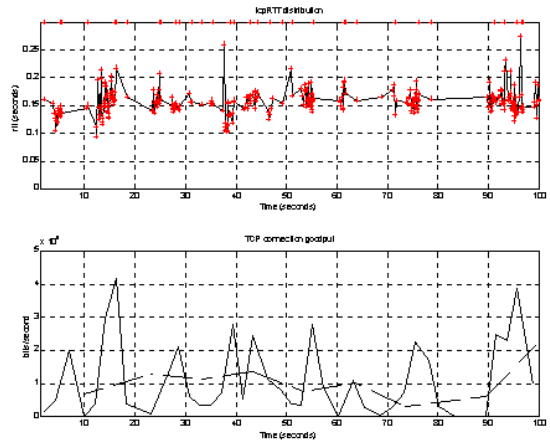


Figure 4-5. Simulated results over ASU path

Figures 4-4 and 4-5 illustrate the measured and simulated results of the ASU path. The loss rate experienced by the end-to-end TCP/Reno connection in both cases is roughly 7%. The majority of loss occurs at link 8-9 which experiences sustained congestion.

## 4.2 Analysis of DCA

A key part of our simulation analysis is to extend the measurement results in a manner that is only possible with simulation. We want to confirm our intuitive understanding of why a DCA congestion probe can only detect on average 7-18% of loss events. The fundamental problem is that a TCP constrained RTT sampling process prevents an endpoint from accurately tracking the bursty congestion associated with packet loss. Factors such as the bursty traffic arrival processes at high speed switches along with the dynamics of TCP's congestion control algorithms make it difficult for a TCP endpoint to reliably avoid packet loss. For example, TCP tends to send packets "clumped" together which makes the probe more granular especially over long paths. We verify this through simulation.

As an example, Figure 4-6 illustrates a portion of a run over the ASU model. The top curve plots the *tcpRTT* time series of the end-to-end TCP/Reno connection that was under observation. The second curve plots the queue level at link 7-8 and the lower curve plots the queue level at link 8-9. In the queue plots, the solid line plots the maximum queue level observed every .1 second and the dashed curve plots the minimum queue level observed. A single loss event occurs at time 47.15 seconds (at link 7-8). Due to the congestion level (the TCP connection experiences a loss rate of 7%), the granularity of the RTT samples is extremely coarse. The *tcpRTT* sample at 46.95 seconds is the RTT sample preceding the transmission of the packet that is dropped and completely misses the queue buildup that is associated with loss. Clearly, in environments where the queue buildup is less than one RTT time, it is not possible for DCA to avoid loss. A further problem is that the magnitude of the queue delay associated with loss at link 7-8 is relatively insignificant compared to the delays associated with link 8-9 (due to differences in link capacities and maximum buffer sizes). An end-to-end congestion probe (a TCP constrained probe or even a more fine-grained probe) is simply not able to accurately assess the congestion level of a router.

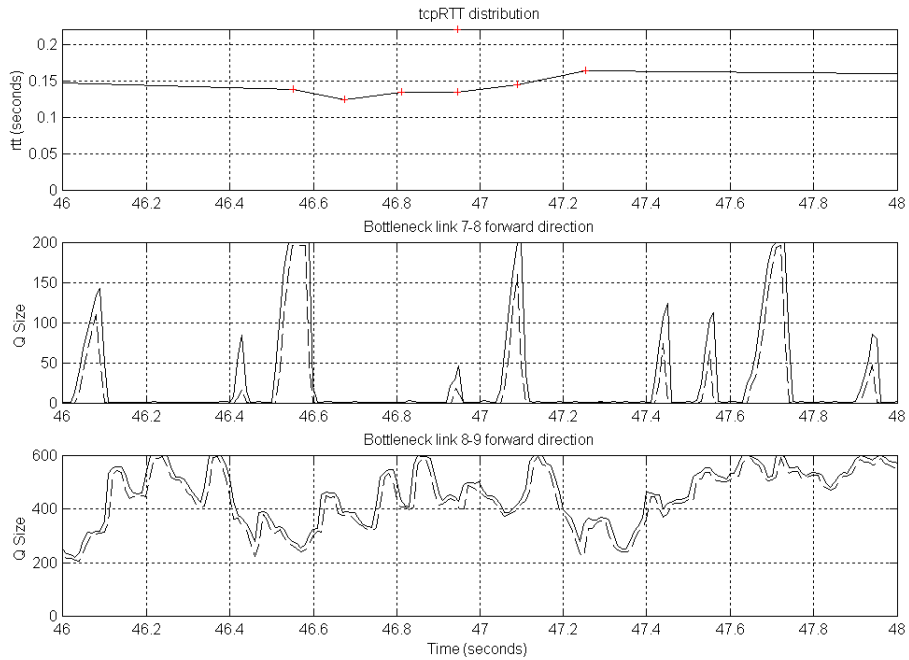


Figure 4-6. ASU simulation results

### 4.3 Simulating the TCP/DCA Protocol

The TCP/DCA protocol augments a TCP/Reno sender with additional DCA congestion decisions. One of the parameters associated with the TCP/DCA algorithm is the level of the send rate reduction when DCA reacts to an increase in RTT (i.e., the *congestionReductionLevel*). The algorithm engages each time a *tcpRTT* sample is generated, however it will react no more frequently than once per congestion epoch. After the normal TCP *cwnd* adjustment (either slow start or congestion avoidance), the following additional logic is run:

```

If (tcpRTT is updated) {
    If (sampledRTT(2) > windowAVG(20) + std)
        cwnd = cwnd - (cwnd * congestionReductionLevel)
}

```

The simulation experiments compare the throughput achieved by two TCP connections (one Reno and one DCA) that run concurrently over the path (either the Emory or the ASU models). Table 4-1 shows the impact that the *congestionReductionLevel* has on TCP throughput (based on 10 500 second simulation runs). For example, over the Emory path, a 50% send rate reduction leads to an average throughput degradation of 37% while a 12.5%

reduction leads to a smaller level of degradation (6.8%). If we assume that the algorithm reacts the same number of times regardless of the level of send rate reduction, it is expected that a smaller send rate reduction will cause less throughput degradation and that a send rate reduction of 50% will be more successful at avoiding loss than a 12.5% reaction. Table 4-2 shows this trend for the Emory path. Over the Emory path, DCA is able to lower the loss rate experienced by the connection by roughly 7%. However, a reduction level of 12.5% does not appear to lower the loss rate. Over the ASU path we see higher variation in loss rate reductions for the lower *congestionReductionLevel* values. We conclude that DCA is less effective at avoiding loss over the ASU path.

Table 4-1. Throughput degradation by varying the *congestionReduction* level

<i>CongestionReductionLevel</i>	% reduction of throughput over Emory mean(std) (95% confidence interval)	% reduction of throughput over ASU mean(std) (95% confidence interval)
50%	-37(8.1) (-42,-31)	-13.2(9.4) (-19.8,-6.6)
25%	-12(12) (-21,-4)	-9.8(20) (-24,4)
12.5%	-6.8(8.6) (-13, -.8)	-7.4(15) (-18.1,3.1)

Table 4-2. Loss rate reduction by varying the *congestionReduction* level

<i>CongestionReductionLevel</i>	% reduction of packet loss over Emory: mean(std) (95% confidence interval)	% reduction of packet loss over ASU: mean(std) (95% confidence interval)
50%	-7.5(12.8) (-16.5, 1.5)	-2.4(15.6) (-13.4,8.6)
25%	-8(17.5) (-20,4.2)	7.8(25.7) (-10.2,26)
12.5%	3.4(15) (-7.4,14.3)	7.8(21) (-6.7,22.6)

In order to match the measured end-to-end dynamics, the Emory model relied on a link level traffic models with high bandwidth UDP flows to define the loss dynamics over the path. Although the ASU model relied on a larger percentage of TCP traffic, it still consists of a fairly significant level of UDP traffic (8%). To be complete, we also evaluate DCA when the loss was driven entirely by TCP flows (even though this leads to dynamics that differ from the dynamics of the traced connections). We modify the congestion dynamics associated with the Emory model such that loss is isolated to link 7-8 and is driven many by low bandwidth, ON/OFF TCP flows (rather than by a combination of TCP and UDP flows). We design two cases: a heavily congested scenario and a lightly congested scenario. In both cases, the background traffic consists of 2200 TCP flows along with a small amount of low bandwidth UDP traffic (less than 5% of the traffic is UDP). The idle time associated with the pareto traffic generator attached to the TCP flows is 1-4 seconds in the highly congested case and 3-5 seconds in the lightly congested case. The loss rate in the heavily congested case is in the range of .8 to 2% and the link experiences

sustained congestion. In the lightly congested case, the loss rate is low (less than .5%) and the link experiences periodic epochs of congestion.

Table 4-3 illustrates the results. In both cases, the throughput degrades significantly. In the less congested case, it turns out that DCA is somewhat successful at avoiding loss (up to 30% of loss events were avoided). Running the correlation indication metric on the tcpRTT time series generated by the TCP/DCA connection confirms that the path exhibits a significantly higher level of correlation between loss and increased RTT (as compared to the measured data). With our models, we found that when the background traffic consists of primarily TCP flows, we could not duplicate the burstiness that we observed over the measured path. The result is that the queue levels increase at a slower rate and this gives DCA a better chance at avoiding loss. However, even in this “best case” environment, the “unnecessary” reactions by DCA to RTT increases not associated with loss drives the throughput down (by 42% in the moderately congested case and by 53% in the more congested case).

Table 4-3. Altered Emory model : DCA results

Average queue level at link 7-8 (capacity is 200 packets)	% reduction of throughput (mean,std) (95% confidence interval)
143.7	-42.4 (6.9) (-47.3, -37.5)
38.3	-50(4.5) (-53.5, -4.7)

#### 4.4 Simulating the TCP/Vegas and TCP/Dual Protocols

To further confirm our claim that DCA will generally result in throughput degradation, we measure the performance of TCP/Vegas and TCP/Dual over the three simulation models that we have developed. The model of TCP/Dual is a straightforward extension to TCP/Reno and is described in [WANG92]. The *ns* simulator provides a model of TCP/Vegas which we use. To evaluate the DCA algorithm of Vegas, we had to isolate the Vegas enhanced loss recovery improvement from the *congestion avoidance mechanism* (i.e., CAM) algorithm.

We compared a version of Vegas with only the enhanced loss recovery algorithm active (i.e., which we refer to at TCP/Reno-Vegas) to TCP/Reno and to TCP/Newreno. We ran three connections (a Reno, 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 the previous section). 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. We find that the TCP/Reno-Vegas flow is able to improve TCP throughput by about 70% over the Emory path. The Newreno algorithm is able to improve throughput by 76% over the path. Both enhanced loss recovery algorithms improve throughput by significantly reducing the frequency of timeouts (on the order of 65% in our example). Based on these limited experiments we confirm that enhanced loss recovery algorithms can be highly effective although we wonder if they might be too effective during periods of heavy congestion.

The rest of this section focuses on a version of TCP/Vegas that disables the enhanced loss recover (we refer to this protocol as TCP/Vegas-CAM) and TCP/Dual. Table 4-4 shows the level of performance degradation experienced by a Vegas-CAM flow (i.e., a Vegas protocol without the enhanced loss recovery algorithm) and TCP/Dual. Both algorithms result in throughput degradation although the decrease is not as large as with DCA. This is primarily because both protocols react less aggressively to congestion. The throughput reduction caused by Vegas is less over the congested ASU path. Because Vegas reacts indirectly to RTT (i.e., changes in throughput), the level of control by Vegas decreases as its sending rate decreases. Another factor is that the path suffers from sustained congestion. Vegas is more likely to underestimate the state of congestion over a path that exhibits sustained congestion than DCA or Dual. The latter two algorithms will still react to RTT fluctuations that exceed the base level of queueing that exists while Vegas tends to be less responsive to short term RTT fluctuation.

We repeated the experiment using the modified Emory model discussed previously. The goal is to see similar results when traffic is less bursty and dominated by TCP. Table 4-5 shows that the throughput reduction is much lower (-1%). As with the ASU model, the congestion level at link 7-8 in the modified Emory model suffers from sustained congestion which causes CAM to significantly reduce its level of control. When traffic is less bursty and highly congested, Vegas-CAM is less reactive. Its benefits seem negligible. Our results differ from prior studies of TCP/Vegas because we have isolated our study to CAM and we have limited our study to high speed paths where a given flow consumes only a fraction of available bandwidth over the path (which we believe was not true in previous studies).

Table 4-4. Performance of TCP/Vegas-CAM and TCP/Dual with respect to TCP/Reno

Model	% Reduction of Throughput TCP/Vegas-CAM, TCP/Dual (95% confidence interval)
Emory	-21 (-27,-15) -14.2 (-22.9,-5.43)
ASU	-7.3 (-20.3,5.8) -18 (-23.3,-12.9)

Table 4-5. Altered Emory model : Vegas results in heavily congested scenario

Average queue level at link 7-8 (capacity is 200 packets)	% reduction of throughput (mean,std) (95% confidence interval)
143.7	-1 (16.6) (-12.3, 11)

## 4.5 Assessing the Impact of DCA on Network Congestion

To validate our claim that DCA is not incrementally deployable (and also to validate our assumptions made in the throughput analysis), we need to confirm that the congestion level of the path is not significantly impacted by the

reactions of a DCA flow. We observe an end-to-end TCP/Reno connection over both of the paths. We perform a second run, replacing the TCP/Reno connection with TCP/DCA. The background traffic sources use their own random number generator so the sample of the application data presented to the network in both runs is the same. We are interested in observing the queue level during each run. Repeating this experiment 10 times for 500 second runs, we find that on average the loss rates, average queue levels and number of queue oscillations are essentially unchanged as a result of the less aggressive behavior by the DCA flow.

An interesting question is how much DCA traffic must be in the network before we see reduced congestion over the path. For an analysis of a larger deployment of DCA, it becomes important to have a realistic traffic patterns which implies that unresponsive link level traffic models might not be the right approach. [CLAF97] has observed that UDP traffic represents about 5% of all bytes that flow over an Internet backbone hop. It seems reasonable to keep this amount of UDP traffic in the model. We utilize the modified Emory model from the previous section where congestion at the bottleneck is generated by 2200 low bandwidth, ON/OFF TCP flows. 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 4 seconds. Roughly 95% of the traffic is TCP and 5% is UDP. We first run an end-to-end DCA connection over the path when all 2200 background traffic flows are TCP/Reno. We repeat this multiple times and obtain the average loss rate and average queue level at link 7-8. Then we repeat the experiment but we replace 50 of the TCP/Reno background flows with DCA flows. We continue doing this, increasing the amount of DCA traffic until the final experiment where 95% of the background traffic is DCA (and 5% UDP).

We actually do 3 cases. The first case is as described above. The dark curve in Figure 4-7 illustrates the results. The loss rate at the bottleneck does not begin to drop until at least 10% of the traffic that arrives at the link is DCA. In second case, we repeat the analysis using TCP/Vegas instead of TCP/DCA. The dashed-dotted line in Figure 4-7 shows that TCP/Vegas is essentially inactive in this environment (due to the sustained congestion). For the third case, we use DCA flows but we reduce the level of congestion at the link (by increasing the idle times associated with the pareto traffic generators). The dashed curve in Figure 4-7 suggests that in a less congested environment, an even larger percentage of DCA traffic is necessary to reduce the congestion level at a bottleneck link.

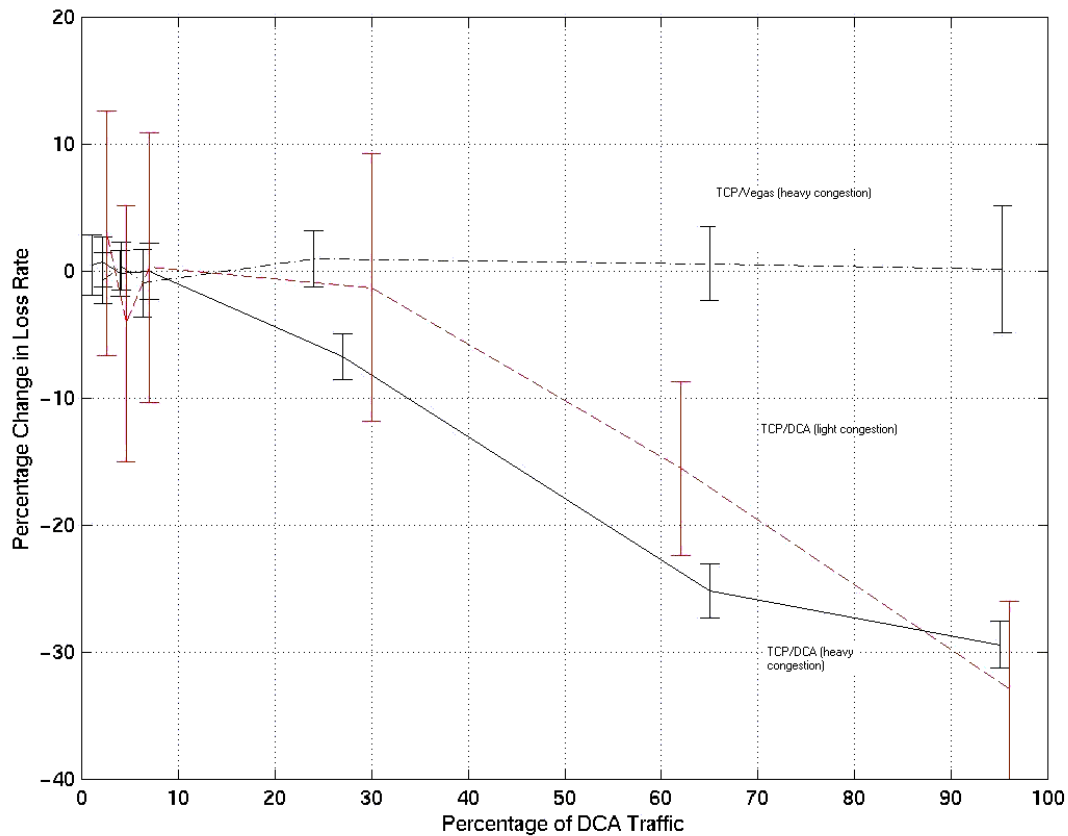


Figure 4-7. Reduction of loss rates as the amount of DCA traffic increases over the Emory path

## 5 Conclusions

We have studied the performance of a class of TCP end-to-end congestion avoidance algorithms (i.e., DCA algorithms) which use an increase in a packet RTT as an indicator of congestion and future packet loss. We have provided significant evidence that supports our claim that DCA cannot be incrementally deployed over high speed Internet paths. We have also presented evidence suggesting that DCA can provide “global” improvements to a network provided there is a sufficient amount of DCA traffic at the bottleneck. However, we have seen that the number of flows required might be large and likely exceeds an incremental deployment. If a proposed enhancement is not able to show any benefit for a single deployment, we believe the chances are very slim that the enhancement would ever be fully deployed.

Our work does have several limitations. First, the measurements represent a small sample of Internet dynamics. Second, the throughput analysis assumes that the analytic throughput model that we use is accurate (at least to some degree). Finally, the simulation models, especially the design of the background traffic, relies in part on conjecture. The simulation analysis of TCP/DCA and TCP/Vegas is more dependent on the end-to-end

characteristics of the path models which we were able to validate to some degree. Several of our other results, however, that we derive by looking within the simulated network (e.g., a queue levels) are more dependent on the accuracy of the models.

## 6 Future Work

We plan to evaluate DCA in a low speed environment. Of particular interest is a web server farm behind a firewall connected to the Internet via a T1 speed access link. Our work has suggested that the congestion levels within a network will improve as the amount of DCA traffic increases. We would like to show that the benefits of an Internet that is dominated by DCA traffic are not as great as an Internet that is dominated by RED/ECN flows.

A problem that we struggled with was how to accurately model the dynamics of the Internet. We obtained satisfactory results using a combination of many TCP flows with along with several high bandwidth UDP flows. To accurately assess the impact of a large deployment of DCA, the design of the background traffic becomes vital. If we use mostly TCP flows, we lose to some degree the characteristics of the measured paths. If we use too much UDP traffic, this might interfere with the evaluation of a large scale deployment of DCA. This problem exists for any study involving large scale simulation of Internet transport protocols (e.g., RED/ECN and differentiated services). We plan on exploring this in the future.

## 7 References

- [AHN95]: J. Ahn, P. Danzig, Z. Liu, L. Yan, "Evaluation of TCP Vegas: Emulation and Experiment", ACM SIGCOMM95.
- [ALLM99]: M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.
- [BARF98]: P. Barford, M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation", ACM SIGMETRICS '98, July, 1998.
- [BORT99]: S. Bortzmeyer, the *echoping* measurement tool available at [http://www.ensta.fr/internet/unix/sys\\_admin/echoping.html](http://www.ensta.fr/internet/unix/sys_admin/echoping.html).
- [BRAK94]: Brakmo, S. O'Malley, L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance", ACM SIGCOMM94, 1994.
- [BOLO93]: J. Bolot, "End-to-end Packet Delay and Loss Behavior in the Internet", ACM SIGCOMM93.
- [CLAF97]: K. Claffy, G. Miller, K. Thompson, "The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone", [http://info.isoc.org/inet98/proceedings/6g/6g\\_3.htm](http://info.isoc.org/inet98/proceedings/6g/6g_3.htm).
- [FLOY99]: S. Floyd, K. Ramakrishnan, "A Proposal to add Explicit Congestion Notification to IP", Experimental RFC 2481, Jan 1999, <ftp://ftp.isi.edu/in-notes/rfc2481.txt>.
- [JACO88]: V. Jacobson, "Congestion Avoidance and Control", ACM SIGCOMM88, 1988.
- [JACO89]: V. Jacobson, C. Leres, S. McCanne, *tcpdump* available at <ftp://ftp.ee.lbl.gov>.
- [MILE84]: R. Miles, *ttcp* measurement tool, available at <http://www.freebsd.org/ports>.
- [MOON99]: S. Moon, et. Al., "Correlation of Packet Delay and Loss in the Internet", INFOCOM 1999.
- [NSSIM]: The Network Simulator. Available at : <http://www-mash.cs.Berkeley.EDU/ns/>.
- [PADH98]: J. Padhye, et. Al., "Modeling TCP Throughput: A Simple Model and its Empirical Validation", ACM SIGCOMM98, 1998.
- [PAXS97]: V. Paxson, PhD. Thesis, University of California Berkeley, 1997.
- [THOM97]: K. Thompson, G. Miller and R. Wilder, "Wide Area Internet Traffic Patterns and Characteristics", IEEE Network, Nov 1997. <http://www.vbns.net/presentations/papers/MCItraffic.ps>
- [WANG92]: Z. Wang, J. Crowcroft, "Eliminating Periodic Packet Losses in the 4.3-Tahoe BSD TCP Congestion Control Algorithm", ACM Computer Communication Review, April 1992.