Part 1

Measurement Analysis

The time scale associated with congestion at a bottleneck router within the Internet can range from milliseconds to many minutes [PAXS97]. Even during periods of sustained congestion, the bursty traffic arrival processes associated with aggregated TCP/UDP flows will tend to make the instantaneous queue level fluctuate rapidly. Leland et. al., describes this behavior as "traffic spikes ride on longer-term ripples, that in turn ride on still longer term swells" [LELA94]. It is reasonable to assume that the self-similarity associated with aggregate TCP/UDP flows accounts for a significant percentage of the packet loss that occurs throughout the Internet. An RTT-based DCA algorithm attempts to track congestion along the path by assuming that an increase in a packet RTT sample reflects the waiting time experienced by the "probe" packet in a queue at a congested router. While a TCP constrained probe process can only provide very coarse assessment of the actual congestion level, the probing mechanism does provide some congestion information. We study the effectiveness of DCA in utilizing the congestion feedback provided by packet RTT measurements to avoid packet loss and consequently improve TCP throughput.

There are two approaches, not necessarily mutually exclusive, available to a DCA algorithm that can be used to avoid packet loss:

- If a connection experiences large time scale congestion (i.e., a long delay epoch where the RTT is larger than some average for a time period greater than at least several RTT's), a DCA algorithm can reduce the chances of packet loss during a long delay epoch by limiting the TCP send rate. Once the congestion backs off, the DCA algorithm stops restraining the sender. Assuming that the packet loss rate over the path is higher during the periods of congestion, it is possible for the DCA algorithm to reduce the effective packet loss rate experienced by the connection by not sending as many packets during higher loss periods.

- Even if loss occurs during a period of long term congestion, loss events are generally accompanied by a short term increase in queueing (extends beyond a longer-term level of queueing). DCA can detect and react to the queue buildup that precedes loss as long as the time scale associated with the buildup is at least one RTT. The objective here is to avoid a loss event that the DCA algorithm predicts will occur within the next RTT.

Assuming the objective is to avoid packet loss, DCA will be most effective if it is successfully able to implement the latter approach. If the DCA algorithm reacts only to the congestion that is associated with packet loss, it will only reduce the send rate when it has to. For either approach, the following requirement must hold. The DCA algorithm must be able to uniquely identify the congestion that is associated with packet loss (regardless of the time scale). For example, if a connection experiences many congestion epochs, most of which are not associated with loss, an algorithm that reduces the send rate during all periods of long term congestion will effectively reduce the connection's throughput unnecessarily. For an algorithm that attempts to avoid loss events by detecting the short term queue buildup that precedes the loss, if the congestion dynamics are bursty such that there are many increases in RTT, it will be difficult for the DCA algorithm to reliably predict future packet loss events. To uniquely identify the congestion associated with packet loss, at least one of the following two conditions must hold:

- The increase (i.e., the magnitude) in the RTT prior to a loss needs to be higher than the average of some number of previous RTT samples. The higher the magnitude of the increase in RTT prior to loss (with respect to previous RTT samples) is, the easier it will be for a DCA algorithm to accurately predict future loss.
- The number of congestion increases that are not associated with packet loss needs to be low. If so then it will be easier for a DCA algorithm to predict packet loss.

The objective of the next several chapters is to show that DCA is not incrementally deployable for use over high speed Internet paths. We do this by presenting evidence, based on measurements taken over actual TCP connections, that indicates that the dynamics associated with congestion over high speed Internet paths conflict with those required by either of the two approaches described above that a TCP/DCA

algorithm might use to avoid packet loss. We select 7 paths that we feel are typical of high speed Internet paths, using *tcpdump*, we trace TCP transfers over each path periodically over 5 days (5 runs over each path each day). We post-process the *tcpdump* traces and extract a per-packet RTT time series (referred to as the *tcpRTT time series*).

The focus of our analysis involves the relationship between loss events and the level of RTT that precedes the loss. By examining the values of the *tcpRTT* samples that immediately precede the transmission of a dropped segment (i.e., a "*loss conditioned*" delay), we can assess the level of correlation that exists between packet loss and increases in RTT samples. We present a set of metrics that helps to assess and quantify the level of correlation between increases in *tcpRTT* samples and packet loss events that exists for a given connection. Our results are based on data obtained by tracing TCP connections over 7 high speed Internet paths taken over the course of 5 days. The statistics obtained from the metrics based on the measured data suggest that:

- While a TCP constrained per-packet probe generally can detect long term congestion, it is able to detect the queue buildup that precedes packet loss only 30-50% of the time. More significantly, if the congestion detection algorithm attempts to filter out minor RTT increases from larger increases, only 7-18% of packet loss events would have had a chance of being detected and avoided.
- While there generally is a certain level of correlation between packet loss and increased RTT, the magnitude of the RTT increase associated with loss is not easily distinguishable from other more frequent increases not associated with packet loss.

We conclude that increases in *tcpRTT* samples are a weak indicator of future packet loss events. Based on the measurement data, we conjecture that the basic cause of this problem is twofold. First, the sampling abilities of a TCP constrained flow (which are impacted by RTT, TCP's congestion control algorithms and the dynamics of congestion over the path) are insufficient to track bursty congestion associated with packet loss over high speed Internet paths to predict and avoid loss. Second, it very difficult (if not impossible) for an endpoint to make consistent, correct congestion decisions based on an end-to-end RTT-based probing mechanism. For example, even if the end-to-end sampling process accurately reflects the queueing

delay that exists over the path, an endpoint can only guess when packet loss will occur. We validate these conjectures in more detail in Part 2 of this dissertation.

To demonstrate the difficulties a DCA algorithm might encounter over high speed Internet paths, we run a hypothetical DCA algorithm on the *tcpRTT* time series extracted from the TCP traces to see how reliably the algorithm can predict future packet loss events. We modify an analytic TCP throughput model to measure the throughput of the DCA algorithm under the same transmission environment. The net result is that DCA will tend to degrade throughput as it will be reacting frequently to increases in RTT samples that are not associated with packet loss. We show that this is true even if the algorithm is able to avoid a high percentage of actual packet loss. This result, along with our conjecture that the reaction from a single DCA flow will not impact the congestion processes that are active over the path, lead us to conclude that DCA is not incrementally deployable for use over high speed Internet paths.

We organize the analysis as follows. First, we define the methodology that we use to obtain the raw data (i.e., the *tcpRTT* time series along with loss indications). We demonstrate the data collection methodology and present a set of metrics that we have developed that quantify the level of correlation that exists between increases in the packet round trip time samples and packet loss events In the next chapter, we present the measured data and the respective statistical analysis and results. In the final chapter of Part 1 of this thesis, we summarize our findings.

4 Methodology

The objective of the measurement analysis is to show that an increase in per packet RTT samples (i.e., the *tcpRTT* samples) is not a reliable indicator of future packet loss events. In this section, we describe the methodology we use to obtain the data necessary to prove this result. 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. In this section, we describe the method used to obtain the raw TCP trace data and how we extract the *tcpRTT* data. Then we present and validate a set of metrics used in the data analysis (the next section) to quantify the level of correlation between increases in *tcpRTT* samples and packet loss events. In the next section we show the aggregate data and analysis results.

4.1 Data Collection Methodology

We have selected 7 typical high speed Internet connections. Each path consists of many hops (at least 11) with the minimum link capacity 10 mbps (we validate this in section 4.1.1). The sender (and also the trace point) of each TCP connection is a host located at North Carolina State University, each of the 7 receivers is located over the Internet. We run a bulk mode TCP application between the NCSU host and the respective destination. The NCSU host is a 350Mhz PC running freeBSD and is capable of tracing (via *tcpdump*) a TCP flow quite accurately as it rarely drops packets and can trace packets with microsecond resolution. The machine is equipped with a 3COM PCI ethernet adapter and is attached to the NCSU campus network via a 10 Mbps ethernet connection.

Table 4-1 describes each of the 7 paths, 5 of which are located outside of North America. For 3 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 5 servers that we selected are public Web servers that provide the standard TCP discard service.

Table 4-1 Summary of traced paths

Path #	Destination Host	# Hops	MAX Window	MSS	service
1	dilbert.emory.mathcs.edu	12	17376	1448	ttcp
2	comeng.ce.kyungpook.ac.kr	19	17376	1460	ttcp
3	ccg.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

4.1.1 Validating the Minimum Link Capacity of the Path

We used two methods to validate that the minimum link capacity along each path is at least 10mbps. First we use the *pathchar* program. Based on *traceroute*, *pathchar* sends UDP packets along the path with specific TTL values. The program works hop-by-hop trying to estimate the link capacity associated with each link. The algorithm is rather complex, however the basic principle is based on the 'packet pair' concept [KESH94] where if a sender transmits two packets back to back, the packets arrive at the receiver with an interarrival time roughly equivalent to the transmission delay over the lowest capacity link.

```
pathchar to electron.mathcs.emory.edu (170.140.150.48)
 mtu limitted to 1500 bytes at local host
 doing 32 probes at each of 64 to 1500 by 44
 0 pc02-412egrc (152.1.194.58)
 |   6.9 Mb/s,   298 us (2.32 ms),  1% dropped
 1 cmdfhub-5513rsm-1.egrc.ncsu.edu (152.1.191.1)
 |    56 Mb/s,   92 us (2.72 ms)
 2 ccgw3.ncsu.edu (152.1.1.7)
 |    55 Mb/s,   52 us (3.05 ms),  1% dropped
 3 ncsu-gw.ncren.net (198.86.71.1)
 |    31 Mb/s,   264 us (3.96 ms),  1% dropped
 4 rtp7-gw.ncren.net (128.109.243.1)
 |    56 Mb/s,   465 us (5.10 ms),  1% dropped
 5 Serial0-1-0.GW2.RDU1.ALTER.NET (157.130.36.157)
 |    37 Mb/s,   2.44 ms (10.3 ms),  1% dropped
 6 142.ATM2-0.XR2.TCO1.ALTER.NET (146.188.163.166)
 |   109 Mb/s,   355 us (11.1 ms),  1% dropped
 7 292.ATM2-0.TR2.DCA1.ALTER.NET (146.188.161.186)
 |   ?? b/s,   6.99 ms (25.1 ms),  1% dropped
 8 101.ATM6-0.TR2.ATL1.ALTER.NET (146.188.136.17)
 |   ?? b/s,   75 us (25.2 ms),  1% dropped
 9 198.ATM6-0.XR2.ATL1.ALTER.NET (146.188.232.97)
 |    68 Mb/s,   221 us (25.8 ms),  +q 1.90 ms (16.3 KB),  1% dropped
10 194.ATM8-0-0.GW2.ATL1.ALTER.NET (146.188.232.77)
 |    13 Mb/s,   434 us (27.6 ms),  +q 1.32 ms (2.19 KB) *2,  1% dropped
11 emory-gw.customer.alter.net (157.130.65.46)
 |    36 Mb/s,   451 us (28.8 ms),  +q 1.68 ms (7.57 KB) *2,  1% dropped
12 ndg-bbatm.netops.emory.edu (170.140.2.190)
13  *
14  *
15  *
 |    21 Mb/s,   700 us (30.8 ms),  +q 17.6 ms (46.2 KB) *2,  49% dropped
16 electron.mathcs.emory.edu (170.140.150.48)
16 hops, rtt 25.7 ms (30.8 ms), bottleneck 6.9 Mb/s, pipe 26780 bytes
```

Figure 4-1 *pathchar* output between NCSU and emory.edu

Figure 4-1 illustrates the *pathchar* output corresponding to the first path between NCSU and emory.edu. The pathchar output for all 7 paths consistently shows that the minimum link capacity is the 10 mbps link at the endpoints. In our measurement work, we encountered routers that perform ICMP handling in a "low priority" mode, giving processing priority to forwarding (we show this in Part 2 of this thesis). This makes any analysis based on timing data derived from responses from the router prone to error. Therefore, to further assure that the paths we have chosen are high speed, we performed our own packet pair analysis. However, unlike *pathchar*, our method is based on end-to-end timing that will estimate the minimum link capacity along the path. Using *tcpdump*, we trace a *ttcp* flow between the emory.edu host and our NCSU host. Whenever the sender (the emory.edu host) sends consecutive packets, we record the arrival time at the receiver (NCSU is the receiver and the trace point). We accumulate each sample for the duration of the test and plot the probability distribution (the top curve of Figure 4-2). A mode in the 9-10mbps range is a

clear indication of the bottleneck link capacity. The lower curve plots the time series of the estimated minimum link capacity date points. We do this for each path and confirm the *pathchar* results that the lowest capacity link in each path is 10 mbps.

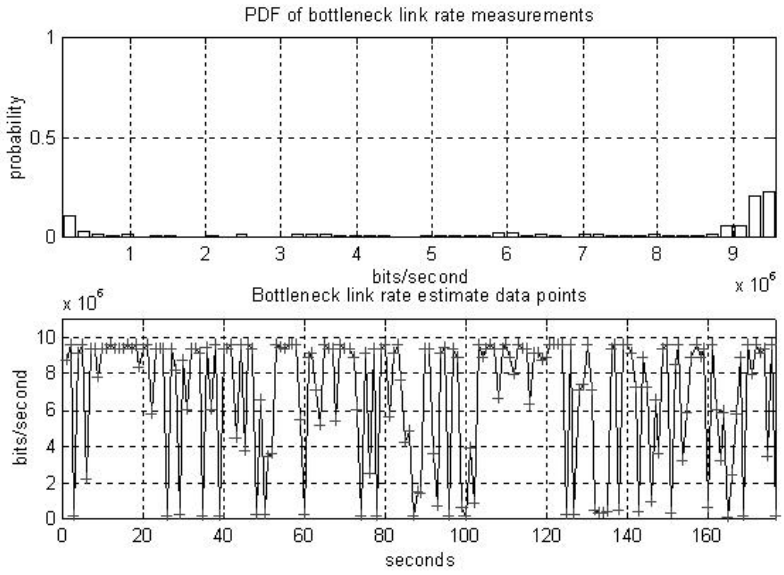

Figure 4-2 Packet pair analysis between emory.edu and ncsu.edu

4.1.2 Extracting packet round trip time samples

Our goal is to extract per-packet round trip time (RTT) samples from a TCP trace. By post-processing a *tcpdump* trace, we gather 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) for two reasons: the *tcpRTT* samples are more frequent and are based on a more precise time measurement.

Deriving TCP per-packet RTT samples from a trace is easier to do when the trace point is located at the sender. At the receiver, *tcpdump* will record the arrival of the data packets and the corresponding acknowledgements from the receiver. It is possible to obtain some RTT samples by timing the interval

between an ACK departure and the arrival of the next sequence number. However, since the sender is not obligated to send additional data once an ACK arrives, this method of extracting the per-packet RTT samples can be inaccurate. In fact, when accessing a busy web server, it is quite likely to observe gaps in the data stream because of application delays. For example, Figure 4-3 illustrates a portion of the sequence/acknowledgement plot associated with an HTTP GET operation of a web object from a web server located at www.freebsd.org. The trace was performed at the receiver. The "+" marks indicate the sequence number of a packet that arrives and the "o" indicates when the receiver sends an acknowledgement. The 12 second delay after time 1 second is an application delay, most likely due to congestion at the server. Thus, it is difficult to extract RTT samples from a receiver based trace.

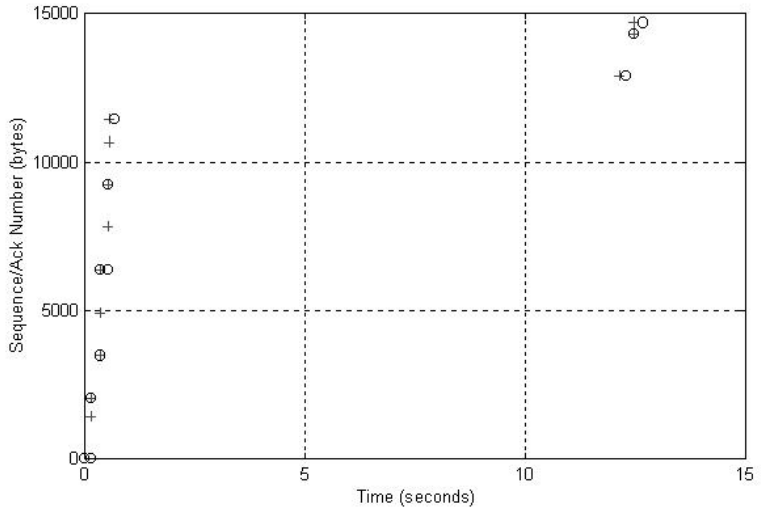

Figure 4-3 Application delay during an HTTP GET web request at http://www.freebsd.org.

To get around this difficulty we perform the at the sender. The goal is to obtain as many samples as possible. Unlike the base TCP RTT estimate which times one segment every RTT, the *tcpRTT* algorithm attempts to obtain a sample for every packet. Difficulties arise due to the TCP delayed acknowledgement and also when packet loss occurs, both of which require special handling by the algorithm.

The algorithm is summarized as follows. The sender time-stamps the departure time of every packet. This is the state maintained by the TCP connection. Once an ACK arrives, the highest segment acknowledged by the ACK generates the *tcpRTT* sample. If the receiver supports the delayed ACK, then typically only

every other packet sent by the sender will result in a *tcpRTT* sample. In other words, if an ACK acknowledges more than one segment, only one *tcpRTT* sample is generated. To filter out errors due to delayed ACKs, ACKs that acknowledge a single segment (or less) of data will not generate a *tcpRTT* sample. During periods of recovery, the algorithm does not take any *tcpRTT* samples to avoid errors. This is described below in more detail.

Figure 4-4 illustrates the algorithm more clearly. Assume that a TCP connection begins in slow start. The first ACK could be used to generate a *tcpRTT* sample, however it would be inaccurate because it includes the receiver's delayed ACK timeout amount. Thus the algorithm throws this sample out. The *tcpRTT* sample corresponding to snum3 will be correct. The sequence of snums 4-6 illustrates how the algorithm deals with packet loss. Since segment 4 is lost, a timeout recovery is required. When a duplicate ACK arrives (e.g., the ACK 3), the *tcpRTT* algorithm stops taking samples until the highest sent SNUM+1 sample. When the ACK for segment 3 arrives, the highest sent SNUM is 6. Therefore, the algorithm will discard all RTT samples until *rtt7*. If this is not done, then once the ACK for segment 6 arrives, the *rtt6* sample will be in error.

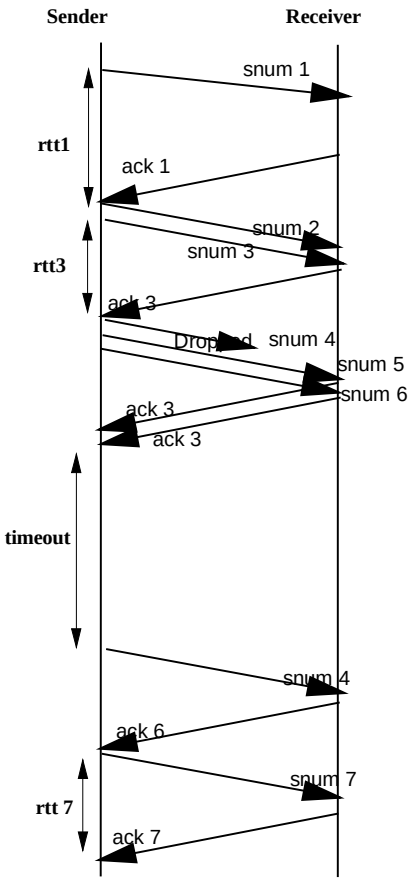

Figure 4-4 Illustration of *tcpRTT* samples

We use an example to demonstrate the algorithm and methodology we use to obtain the *tcpRTT* samples (and packet loss events) from the *tcpdump* trace . We traced a TCP connection exchange between our NCSU host and a web server located 18 hops away in Amsterdam (www.nikhef.nl). The PC at NCSU is the sender (and the trace location) and we are therefore able to obtain the *tcpRTT* time series. We use the *echoping* program to send 655350 bytes to the discard server at www.nikhef.nl.

The top curve of Figure 4-5 plots the *tcpRTT* time series that is extracted from the trace. The lower curve plots the throughput time series. Figure 4-6 expands the results between times 562 and 568 seconds showing the *tcpRTT* samples during a recovery period. The "+" marks represent *tcpRTT* samples. The hash marks at the top of the curve indicate occurrences of packet loss. The lower curve plots the throughput sampled at every 3 seconds (the solid curve) and every .5 seconds (the dashed curve). The two

graphs provide a very good visualization of the dynamics associated with the connection as well as the relationship between the *tcpRTT* samples and packet loss. To help illustrate the TCP recovery, Figure 4-7 plots the acknowledgement arrivals (i.e., the "o" marks) and the corresponding segment transmissions (i.e., the "+" marks). The vertical axis plots the sequence number associated with the ACK arrivals and the segment transmissions. This type of plot, referred to as the sequence/acknowledgement plot, is useful to visualize the dynamics of the connection. For example, the "o" at time 562.1 seconds represents the arrival of an ACK that acknowledges up to packet 345000 (i.e., it acknowledges that the receiver has successfully received up to byte 345000 sent by the sender. We see that the sender responds by sending two additional packets as the acknowledgement opens the send window by 2 packets.

The packet with sequence number 369152 is originally sent at time 563.60253 seconds is dropped. The last two *tcpRTT* samples are at times 563.60251 and 563.652. The first duplicate ACK arrives at time 563.81 and the algorithm will not take another sample until one RTT after the recovery has completed (566.816 seconds). In this example, the last *tcpRTT* sample prior to the retransmission (which occurs at 566.435) was generated after the segment was sent that was dropped (i.e., segment 369152 is originally sent at time 563.60253).

In the next section, we present a set of metrics that will show the level of correlation between increases in the *tcpRTT* samples and packet loss events. The metrics require a time series where the *i'th* sample looks like the following tuple:

$i : (time_i, tcpRTT_i, l_i)$ 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 1 time will be considered as separate loss events as long as they have a unique *tcpRTT* sample

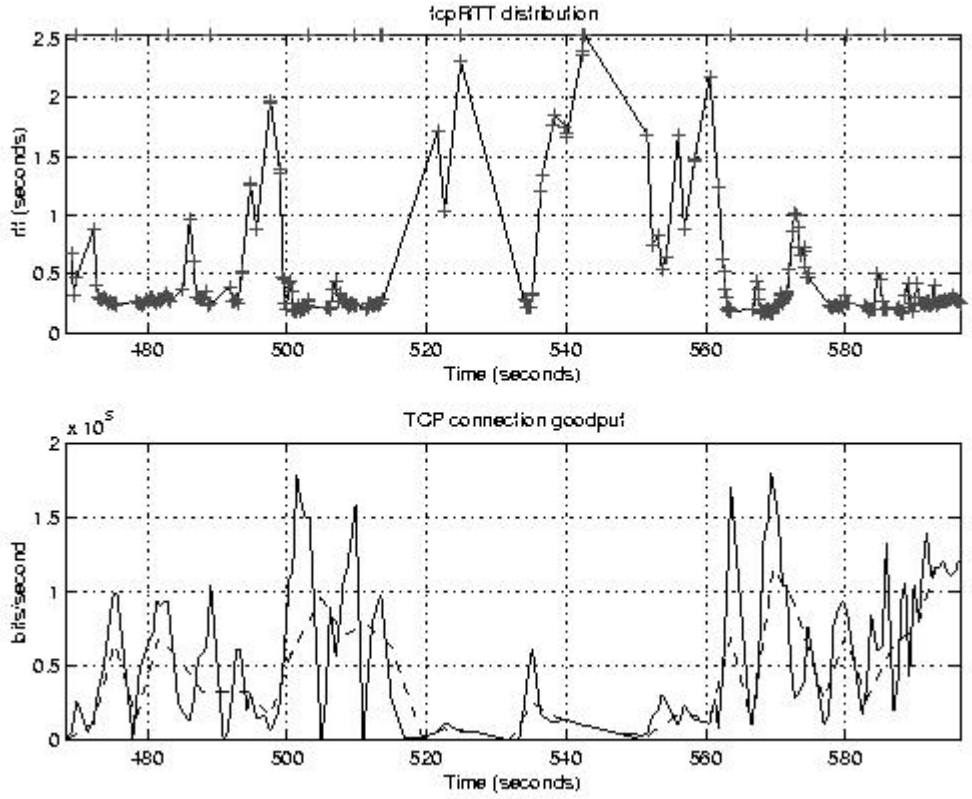

Figure 4-5 Transfer between NCSU and www.nikhef.nl

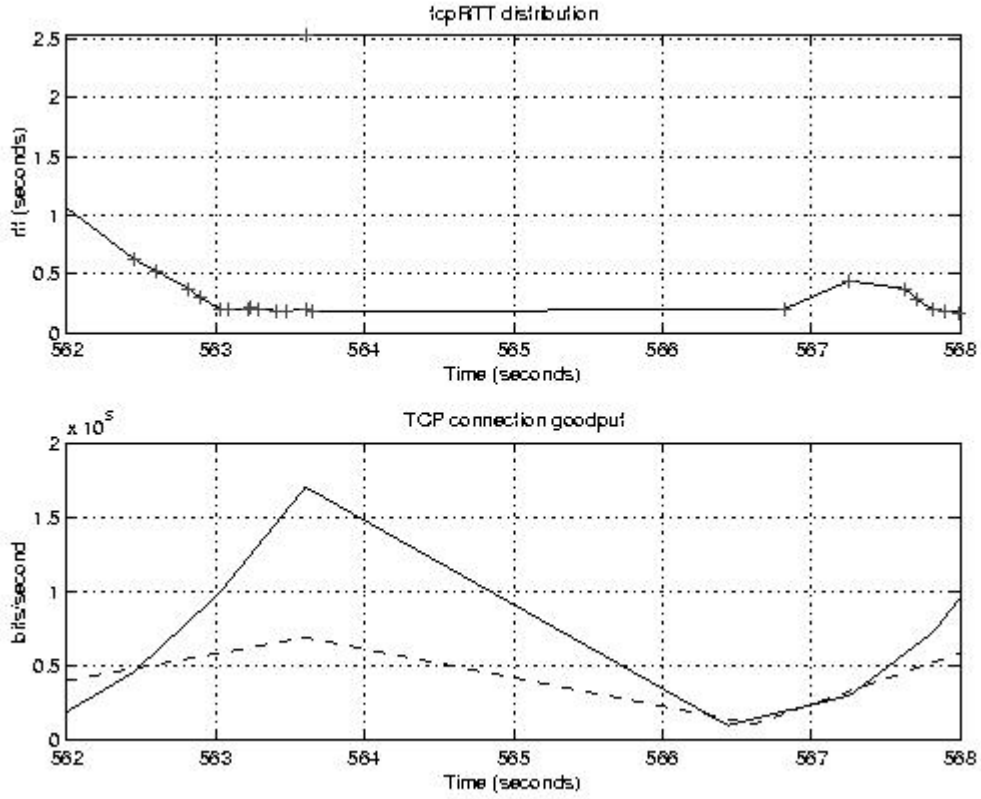

Figure 4-6 Expanded view of transfer between NCSU and www.nikhef.nl

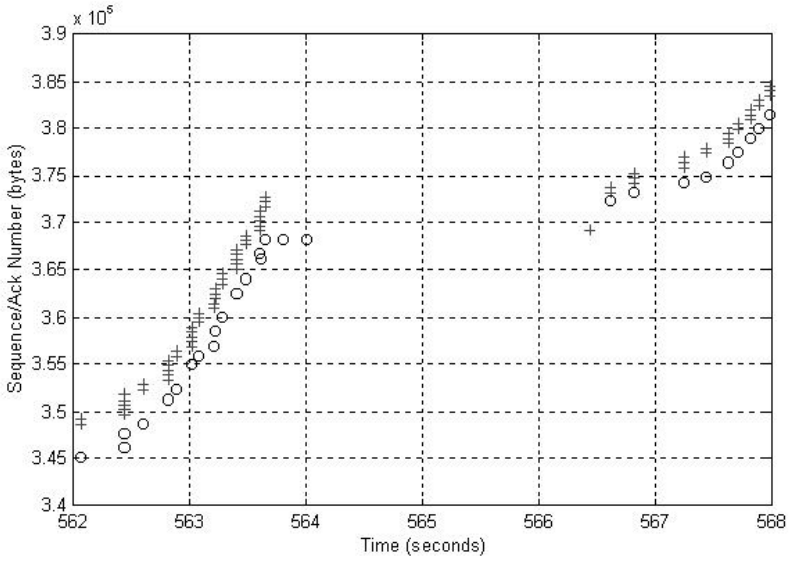

Figure 4-7 Sequence/Acknowledge plot of transfer between NCSU and www.nikhef.nl

4.2 Defining the Metrics

Assuming that the objective of DCA is to avoid packet loss, the algorithm must be able to identify the congestion that is associated with packet loss regardless of the time scale associated with the congestion. Therefore, DCA requires that packet loss events and increases in RTT be tightly coupled (i.e., correlated). We define three metrics that allows us to examine the following aspects of the correlation between RTT and packet loss:

- How frequently loss is preceded by an increase in packet RTT samples.
- A rough estimate of how unique the increase in RTT that precedes loss with respect to all RTT variations.

Specifically, we present the following set of metrics:

- Correlation indication metric: This metric provides an indication of the level of correlation that exists between packet loss and increases in RTT prior to the transmission of a packet that is eventually dropped in the network.
- Loss conditioned delay correlation metric (referred to as LCDC) : This metric reflects the level and time scale of correlation that exists between increases in RTT and packet loss events.
- Loss conditioned delay CDF: This is the cumulative distribution function (i.e., the CDF) of the set of *tcpRTT* values just prior to a loss event (i.e., the loss conditioned delay values). Plotting the loss conditioned delay CDF along with the CDF of the entire *tcpRTT* time series provides insight to the relationship that exists between the two distributions.

4.2.1 Correlation 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 number of previous *tcpRTT* samples. More formally, we define the following event:

$$E_1 : \{sampledRTT_i\ (x) > windowAVG_i(w)\}$$

Define the set *j* as: $\{j : l_j = 1\}$

We define the *sampledRTT(x)* to be 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 number, *x*, is a parameter that controls the size of the moving window associated with the *sampledRTT*. The value of *x* sets the responsiveness of the congestion detection. *x* is a set to 2 and up to 8. The *windowAVG(w)* is also a moving window average of the previous *w* *tcpRTT* values prior to the transmission of the segment that are dropped. In other words:

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

The *sampledRTT*(x) approximates the "instantaneous" RTT value while the *windowAVG*(w) is a longer term average. The difference, *sampledRTT(x) – windowAVG(w)*, is reflective of an increase or decrease in queue delay as experienced by the last "probe" sample.

For a given run, we calculate the *sampledRTT(x)* and the *windowAVG(w)* for each occurrence of packet loss. We count the number of times that the event E_1 is true. Dividing this count by the total number of packet loss occurrences gives the probability of event E_1 conditioned on a loss indication:

$$P[sampledRTT_i(x) > windowAVG_i(w) \mid l_i = 1]$$

We refer to this as the correlation indication metric. We assume that the *sampledRTT(x)* implies the *tcpRTT* samples prior to a loss. Therefore, for the remainder of the thesis, we write the correlation indication metric as (note that we also will drop the subscripts when referring to any of the time series data):

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

The metric provides an indication that the *tcpRTT* samples prior to loss are higher than some average. 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. Similarly we can find the probability that the *sampledRTT* is greater than the average of all *tcpRTT* samples:

$$P[sampledRTT(x) > rttAVG]$$

We primarily use the correlation indication metric with either a (2,5) or (2,20) window size as we are interested in how frequently the *tcpRTT* samples prior to loss show an increase. While the metric is not a true correlation, it is an indication of the level of correlation that exists between increases in *tcpRTT* and packet loss events. The effectiveness of a DCA algorithm will increase as these correlation indicators approach a value of 1. We will show in the next section that a correlation indication level larger than .7 typically indicates a "strong" level of correlation while a value in the range of .4-.6 is either weak to moderate.

4.2.2 Loss Conditioned Delay Correlation Metric (LCDC)

The LCDC indicates the magnitude of the correlation that exists between an increase in *tcpRTT* samples and loss events. The metric is also of benefit as it can provide a visual indication of the time scale of correlation (if any exists). While the correlation indication metric defined previously does not consider the *tcpRTT* samples that are generated after loss occurs, the loss conditioned delay correlation metric provides a view of the level of correlation both before and after loss.

As explained in the background section of the dissertation, the LCDC is based on the concept of a loss-conditioned average delay as defined 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 delay sample prior to the probe that was lost (and lag '–2' is the second delay sample before the lost probe, up to some large number of samples such as lag '–300'). Likewise, lag '+1' is the delay sample associated with the probe packet that arrived after the packet that was dropped (up to lag '+300'). The average packet delay conditioned on a loss at a time lag *j* packets is the average delay of all packets in the trace that have a loss *j* packets before them in the trace. That is:

$$E[d_i \mid l_{i-j} = 1] = \frac{\sum_{k \in P} d_k}{|P|} \quad \text{where} \quad P = \{k : l_{k-j} = 1 \text{ and } l_k = 0\}$$

Where:

d_i is the delay of the *i*-the packet. If l_i =1, the *i*-th packet is lost and d_i is set to 0.

The algorithm that we use is essentially identical to that used in [MOON99]. Lag '0' corresponds to the time that the segment that is dropped is originally transmitted. Lag '-1' will be the *tcpRTT* sample immediately before the time that the dropped packet was originally sent. As we will show in the next section, the effectiveness of DCA will increase if the metric shows a distinct peak in the loss conditioned delay in the lags immediately prior to packet loss. If true, this indicates that there is a tendency for the *tcpRTT* values to increase prior to loss.

4.2.3 Loss Conditioned Delay CDF

The CDFs of the *tcpRTT* and *sampledRTT* distributions are plotted. The latter provides the probability of a packet loss for a given *tcpRTT* value. Plotting 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). As we will see in the next chapter, the more likely result is that loss does occur when the *sampledRTT* is above some minimum *tcpRTT* level (which we define as the *thresholdRTT*). Above this threshold however, the *sampledRTT* values tend to be evenly distributed across the *tcpRTT* distribution. In other words, loss is generally accompanied by an increase in RTT. However, above the *thresholdRTT* level, the *sampledRTT* can be any *tcpRTT* value with almost equal probability. In this environment, it would be difficult for a DCA algorithm to uniquely identify the increases in *sampledRTT* that correspond to packet loss. Because of this, the algorithm will react to many of the *tcpRTT* variations, most of which are not associated with packet loss.

We define the *sampledRTTAVG* to be the average of the *sampledRTT* distribution. In order to increase the chances of a DCA algorithm correctly predicting loss events, the following probability (which can be estimated by from the loss conditioned delay CDF curve) that describes how frequently the *tcpRTT* samples remain low except when loss occurs should be larger than .5:

$$P[tcpRTT < sampledRTTAVG]$$

4.2.4 Validating the Metrics

To validate the correlation metrics defined above, we use simple simulation experiments. We simulate the network illustrated in Figure 4-8. There are 3 TCP/Reno sources (all ftp applications) competing for bandwidth over a T1 WAN (configured with a 20ms propagation delay). The ftp data flows between sources S1, S2 and S3 to destinations D1, D2 and D3. We incorporate the method that we used to post-process the TCP traces to generate the *tcpRTT* time series associated with one of the connections. The top

graph of Figure 4-9 plots the bottleneck link queue levels. We extend the *ns* simulator with a queue monitor capability such that every .1 second, we obtain a data point consisting of the minimum and the maximum queue levels during the interval. Looking at Figure 4-10 (which shows a more detailed view between seconds 10 and 18), the solid line in the upper curve plots the maximum queue level during the sample period and the dashed curve plots the minimum level during the period. The middle curve of the two figures plots the *tcpRTT* time series. The *tcpRTT* time series is able to track the queueing at the bottleneck link quite well. The lower graph plots the goodput (the dark line) and the send rate (the light line). The router is configured with 10 buffers. The packet loss rate associated with the link is about 1%.

The figures clearly show that the *tcpRTT* samples tend to be at their maximum when loss occurs. Also, the time scale of congestion associated with packet loss is large. Essentially, this simulation presents all the right conditions that would contribute to the effectiveness of a DCA algorithm. The correlation indication metrics are:

P[sampledRTT > windowAVG(5)] = .72
P[sampledRTT > windowAVG(20)] = .81
P[sampledRTT > rttAVG] = .91

The metrics indicate that 72% of the time, the average of the two *tcpRTT* values prior to the transmission of a lost segment was higher than the average of the previous 5 *tcpRTT* samples. The fact that the correlation level increases as the length of the window associated with the longer term moving average of previous *tcpRTT* samples indicates that the time scale of the queue buildup associated with packet loss is large (i.e., many RTT's).

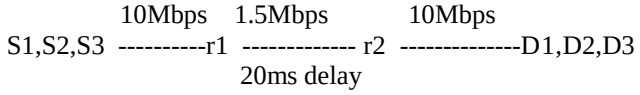

Figure 4-8 Simple T1 Simulation Network

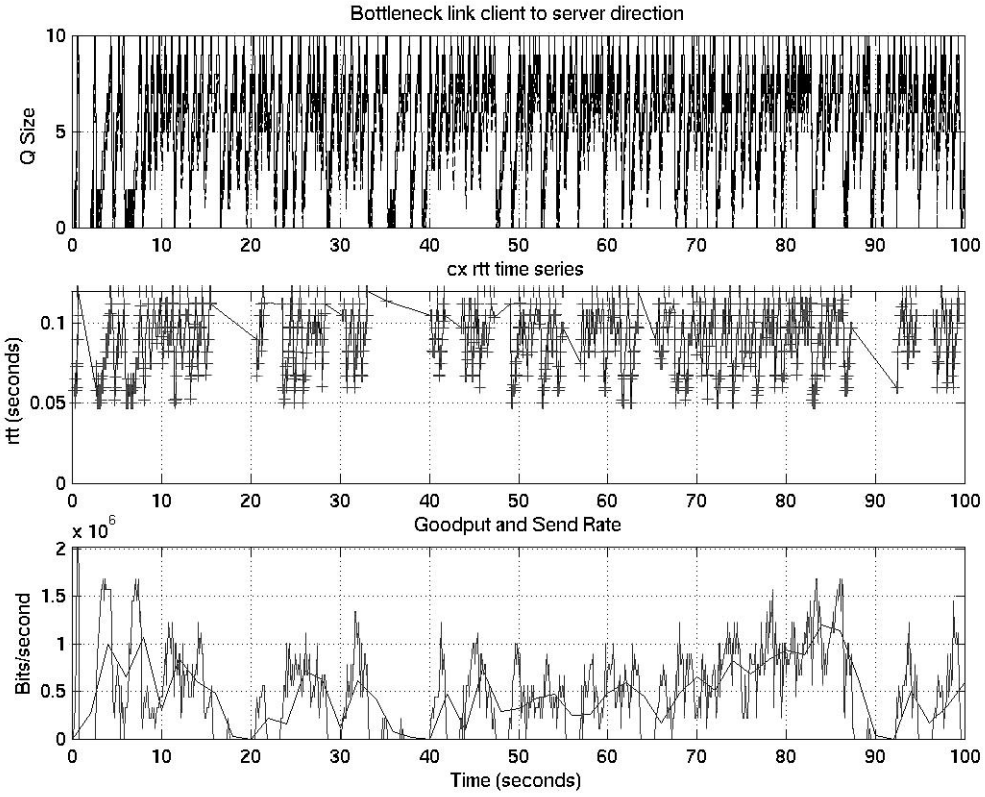

Figure 4-9. 3 ftp TCP/Reno flows competing

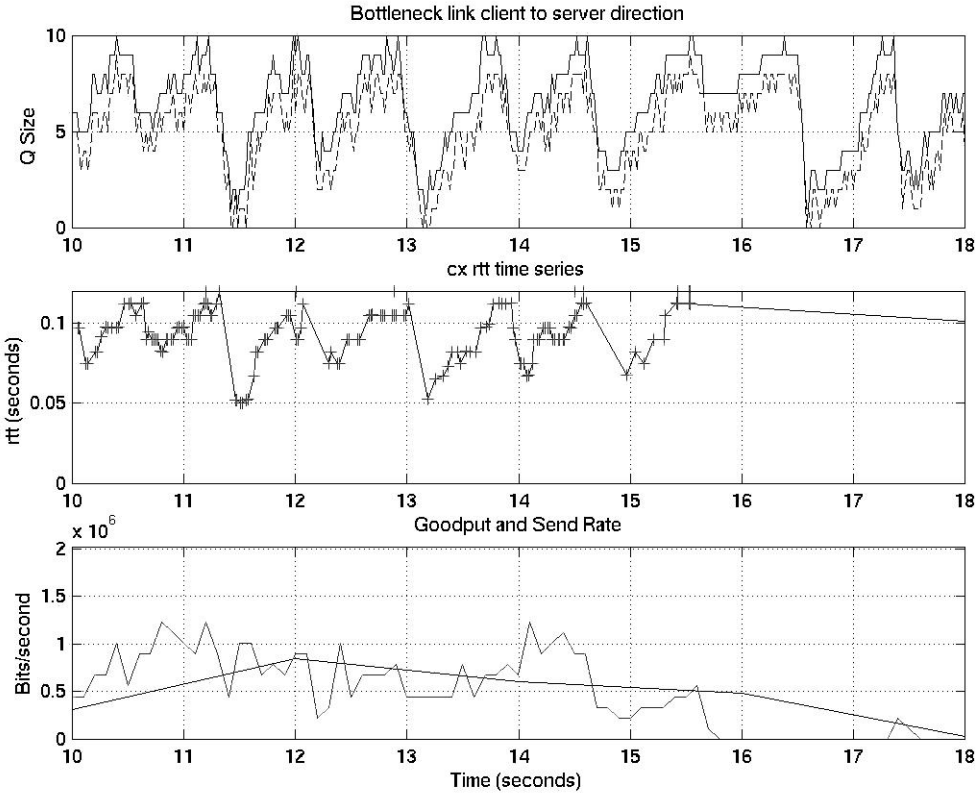

Figure 4-10. Detailed view of 3 ftp TCP/Reno flows competing

Figure 4-11 illustrates the loss conditioned correlation metric for the same simulation run. The curve shows a distinct increase in the loss conditioned delay beginning roughly 7 lags prior to the loss. It is interesting to observe the decrease in *tcpRTT* samples immediately following loss. The 3 connections are essentially synchronized, each connection experiences and reacts to loss at roughly the same time. Therefore, once loss occurs, the queue is able to clear and the subsequent *tcpRTT* samples after loss detects the decrease in RTT.

.

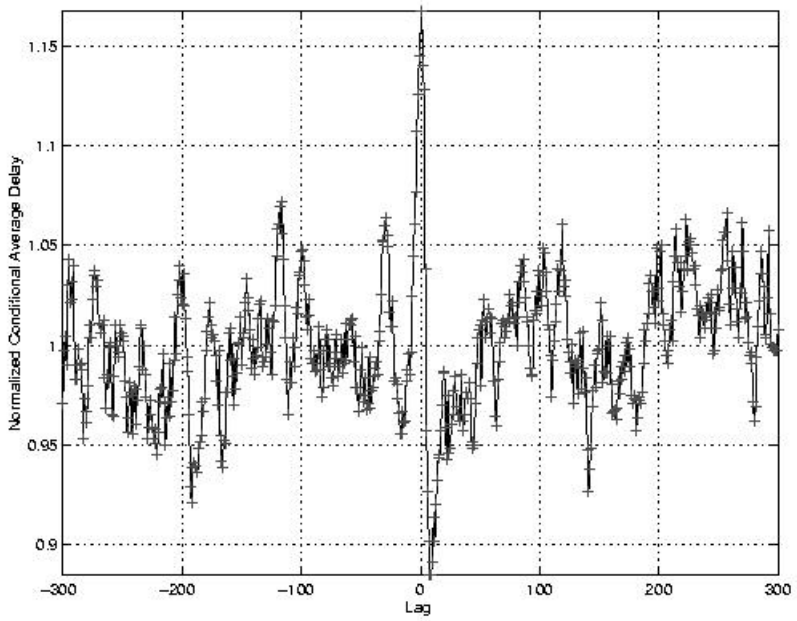

Figure 4-11. *ns* simulation run: loss conditioned delay correlation metric

Figure 4-12 illustrates the CDF of the *tcpRTT* time series (the dark bar plot) and the *sampledRTT* time series (the lighter bar plot). The *sampledRTT* distribution clearly shows that the increase in the *tcpRTT* samples prior to loss are confined to the larger *tcpRTT* values. In other words, when loss occurs, the *sampledRTT* tends to be very large (i.e., larger than the average *tcpRTT* value). This is reflected in the following probability which we estimate from Figure 4-12:

$P[tcpRTT < sampledRTTAVG] = .72$

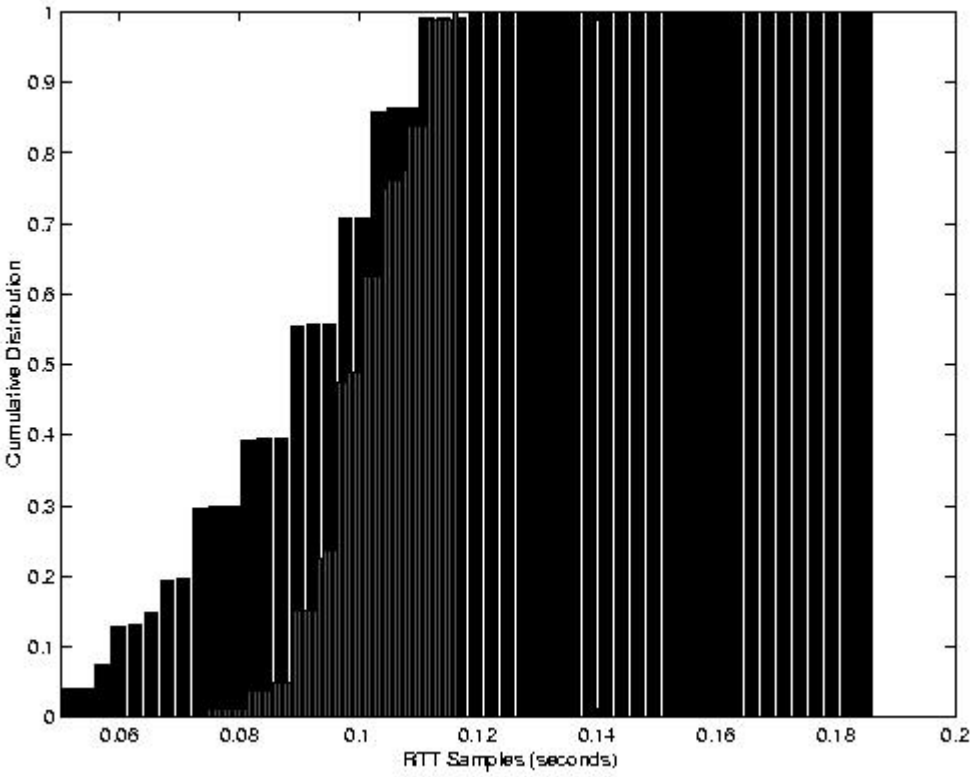

Figure 4-12. *ns* correlation simulation run: CDF metric

If we increase the bottleneck router buffer size to 50 packets and add a 3% random drop process (i.e., based on a uniform distribution) , the correlation indication metric data becomes:

P[sampledRTT > rttAVG] = .37
P[sampledRTT > windowAVG(5)] = .45
P[sampledRTT > windowAVG(20)] = .45

The data indicates that the *tcpRTT* samples prior to loss do not tend to be increasing prior to loss. Furthermore, the magnitude of the *tcpRTT* sample before loss is relatively low. Figures 4-13 and 4-14 show the dynamics associated with the run. Again we see that the *tcpRTT* time series is able to track the majority of the long term congestion.

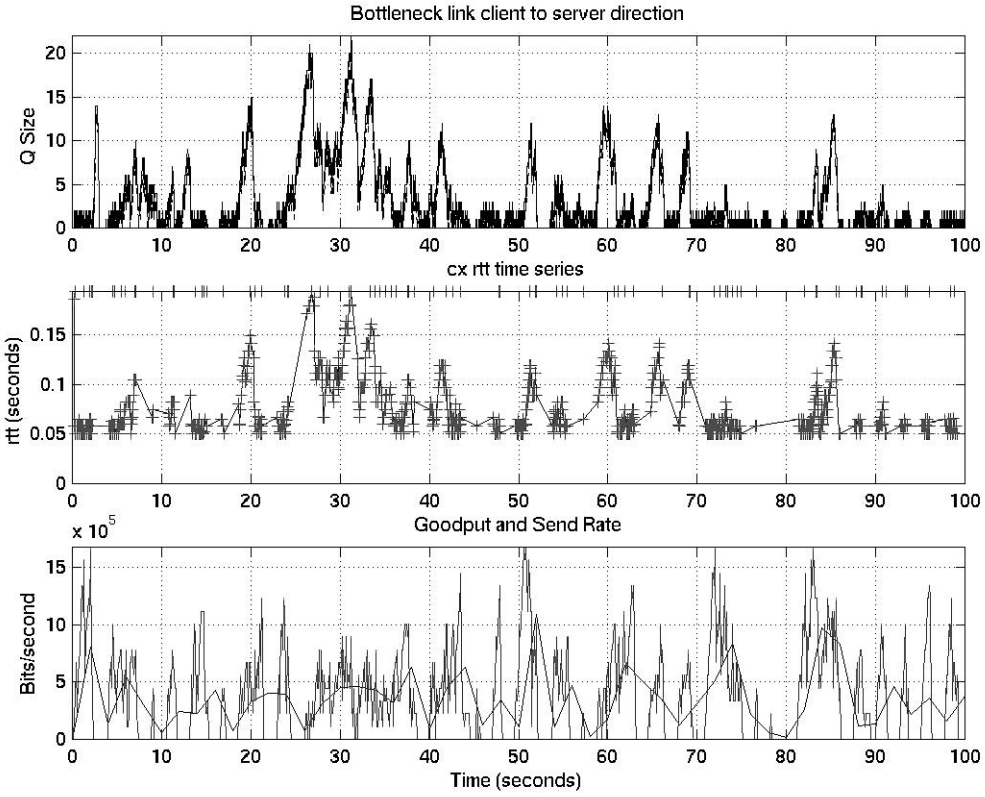

Figure 4-13: *ns* correlation run with 3% random loss

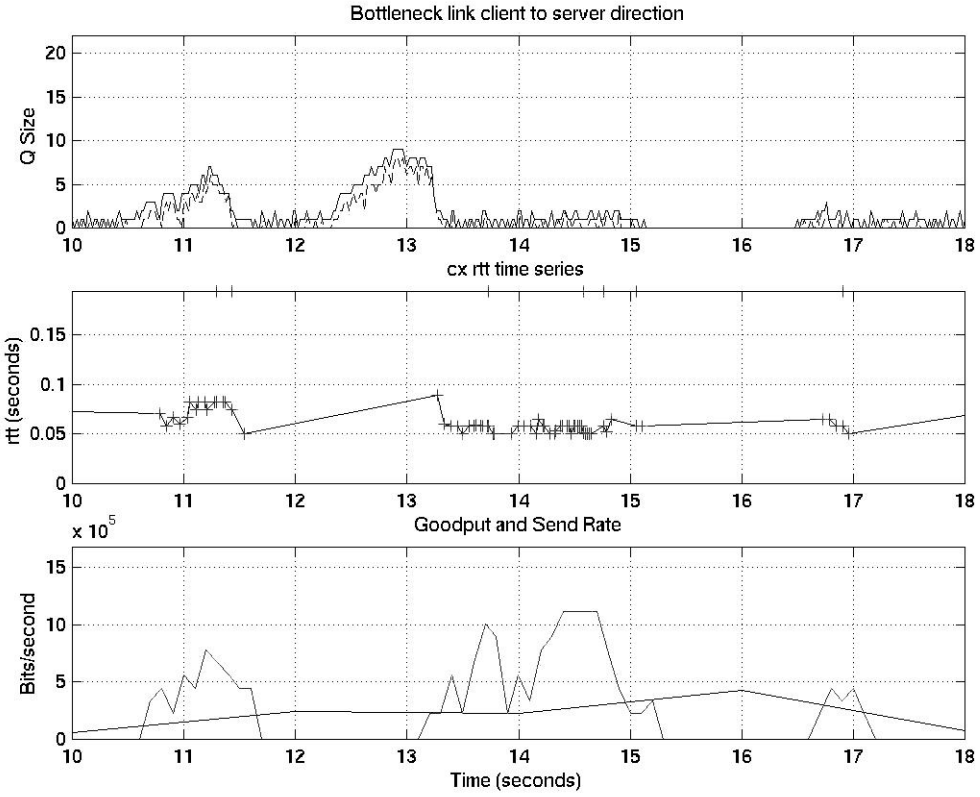

Figure 4-14. *ns* correlation run with 3% random loss details

Figure 4-15 shows the level of loss conditioned delay correlation associated with the run. In this case, there is no longer a clear correlation between an increase in *tcpRTT* with a future packet loss event. The graph does show that there is a tendency for the *tcpRTT* samples after a packet loss to decrease. Because the TCP connection consumes a significant amount of bandwidth, once it reacts (via packet loss), this causes the queue to clear which is observed by the subsequent *tcpRTT* samples. Figure 4-16 shows that the *sampledRTT* distribution follows the *tcpRTT* distribution closely. All loss is caused by the random drop process rather than from buffer overflow. Therefore we would expect the *sampledRTT* values to essentially be random with respect to the *tcpRTT* distribution. This is supported by the following probability:

$P[tcpRTT < sampledRTTAVG] = .5$

Figure 4-16 reflects this although there is some bias away from the lowest *tcpRTT* samples. This is most likely because there are not enough sample points. Figure 4-17 illustrates the same result only using the probability density distributions rather than the CDFs.

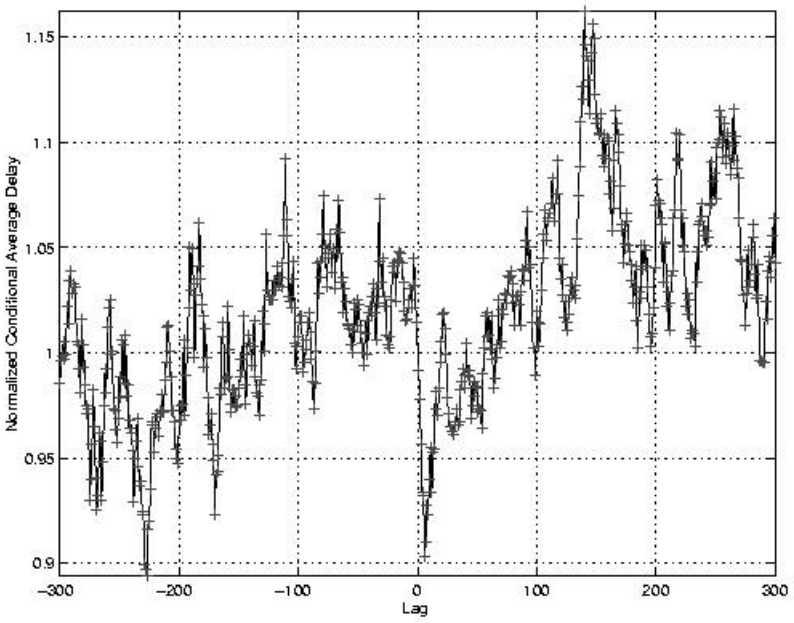

Figure 4-15. Loss conditioned delay correlation metric for *ns* run with random loss

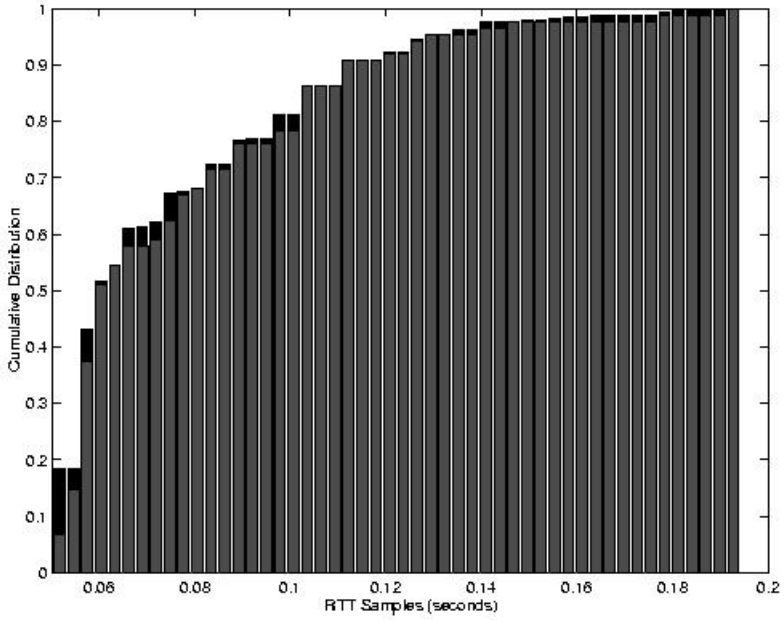

Figure 4-16. CDF metric for *ns* run with random loss

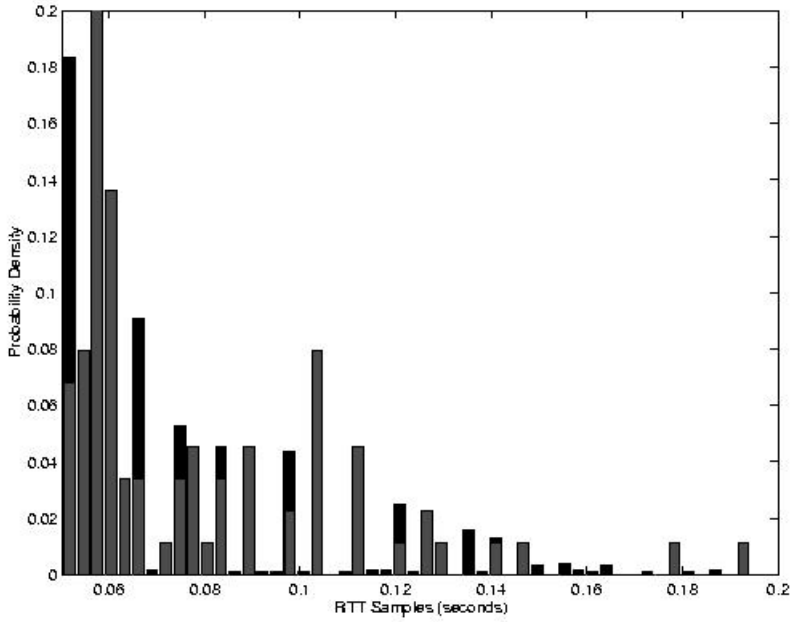

Figure 4-17. PDF metric for *ns* run with random loss