3 Related Work and Motivations

Our research explores the effectiveness of a RTT-based congestion avoidance (i.e., DCA) algorithm in providing a performance improvement to TCP/Reno over high speed Internet paths. Related research includes a significant amount of TCP performance improvement studies and also Internet measurement studies that have explored the dynamics associated with packet loss and packet delays over the Internet.

Relevant TCP research includes:

- Enhanced gateway congestion control (e.g., RED [FLOY93] and RED/ECN [FLOY99]).
- Enhanced loss recovery algorithms (e.g., NewReno [HOE96]).
- Packet delay based congestion avoidance (e.g., TCP/Vegas [BRAK94] and TCP/Dual [WANG91]).

In general, a TCP performance improvement should not allow a flow to consume more bandwidth than a similar TCP/Reno flow. However, a flow is allowed to utilize bandwidth more efficiently. TCP/Reno is a remarkably robust transport protocol although it does have a fundamental limitation: it cannot recover from multiple packet losses within a single RTT period without a timeout. A significant portion of previous TCP research focuses on avoiding this problem (e.g., RED, TCP/Sack, NewReno and TCP/Vegas).

A significant benefit of RED's is its ability to "spread" out packet loss and reduce the chances of a single flow experiencing multiple packets dropped within a RTT. Enhanced loss recovery (NewReno and TCP/Sack) is also useful to help recover more efficiently once a burst of loss occurs. The work of [BRAK94] and [AHN95] shows that TCP/Vegas is able to increase TCP throughput anywhere from 40-70% (5-20% in the latter study) primarily by avoiding time-outs. However both of the TCP/Vegas studies were conducted primarily over lower speed lines. As mentioned in the previous section, Vegas consists of three separate ideas, two of which are associated with congestion avoidance (i.e., the CAM algorithm). A study of the results in [BRAK94] and [AHN95] shows that Vegas reduces the number of retransmissions (as compared to TCP/Reno) and timeouts. This indicates that the CAM algorithm was engaging more

frequently than the enhanced loss recovery algorithm because enhanced loss recovery, by design, results in a larger number of retransmissions with fewer timeouts.

Our work differs from that of [BRAK94] and [AHN95] in that we are specifically interested in high speed Internet paths. The initial Vegas analysis was done when a T1 Internet connection was considered high speed. Today, most large companies connect to the Internet with at least a 45mbps connection. Most of the problems that an end-to-end congestion avoidance algorithm must deal with are caused by the congestion dynamics associated with high speed environments. A fundamental assumption that we make in our analysis is that a congestion reaction by a single DCA flow will have no effect on the congestion processes that are active over the path. Unlike the specific protocol analysis done by [BRAK94], our analysis is meant to be broad enough such that the results apply to any DCA algorithm that is run over a high speed path.

Our goals also differ from that of [BIAZ98] who attempts to quantify the effectiveness of three TCP DCA congestion detection algorithms (TCP/Vegas, TCP/Dual and Tri-S) by gathering TCP internal states during data exchanges and, once a packet loss occurs, determine if the algorithm indicates congestion [BIAZ98]. Once packet loss is detected, the goal is to differentiate between loss due to congestion and loss due to data corruption over wireless paths. Biaz finds that only under certain network conditions is TCP/Vegas able to differentiate between network congestion and data corruption. Furthermore Biaz is interested primarily in low speed, wireless paths. Our work differs in that we want to see if any DCA algorithm can predict future occurrences of packet loss over high speed Internet paths.

Additional related work includes several measurement studies involving end-to-end delays and packet loss. Bolot's study of end-to-end packet delay and loss behavior is somewhat relevant in that he measured the round trip delays of small UDP probe packets sent at fixed time intervals and examined the end-to-end packet delays and loss behavior [BOLO93]. He varied the interval between probe packets to study the behavior under different time scales. He performed 10 minute runs using different intervals ranging from 8,20,50.100, 200, 500 milliseconds. The key results from this work are that:

- Probe packets tend to become compressed.
- Queue delays fluctuate rapidly over small intervals.
- Loss is random as long as the probes don't consume more than 10% of the bottleneck link capacity.
- Probe losses are correlated to the "burstiness" of the probe packets

However, Bolot did not specifically look for correlation between delay and loss. The work by Moon et al. [MOON99] in fact specifically attempts to quantify the level of correlation that exists between one-way packet delays and packet loss events. By emitting a small UDP probe packet every 20 milliseconds, they track one-way packet transit times and packet loss. Their results show that an increase in packet delay corresponds to an increase in the probability of packet loss. Their results also suggest that patterns of packet delay can be periodic over some links with a time scale in the seconds range. They conjecture this might happen when TCP traffic at a bottleneck link synchronizes.

We are interested in the method used by Moon et., Al. to quantify the level of correlation between packet delay and loss observed over Internet paths based on the concept of a loss-conditioned average delay. They define a lag which 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:

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

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.

As an example, consider the following delay time series (where a 0 value corresponds to a loss):

delay(i): .21, .2, .3, .33, .4, 0 {loss}, .38, .33, .28, .2, .21, .23, .28, .33, 0 {loss}, .40, .38, .21, .19

There are 2 losses and 17 valid delay measurements. For lag '1', the loss conditioned delay would be (.38 + .4)/2 = .39. For lag '–1', the average delay would be (.4 + .33)/2 = .37. Likewise, for lag '2', the delay is (.33 + .38)/2 = .35 and for lag '–2' the delay is (.33 + .28)/2 = .31. Moon gathered data for 6 Internet paths. For each data set (each of which contains 40000 to 100000 data points), she plots the normalized loss-conditioned average delay (normalized to the average delay of all samples in the run) versus a lag. She limits the range of lag to +/- 500 samples which corresponds to about 10 seconds (5 seconds in the past, 5 seconds in the future).

Her results show an increase in the normalized loss-conditioned delay in the range of 10 to –50 lags. The highest level of correlation exists around lag '0' which makes sense assuming drop tail routers where there will be queue buildup prior to packet loss. Assuming the probes arrive at the receiver every 20 milliseconds, this indicates the time scale associated with a packet loss can extend up to 1 second (50 probes * 20 milliseconds) prior to the loss. We believe that a probe compression phenomenon is most likely happening (i.e., similar to the TCP ACK compression where probes lose their regular spacing as they are queued at bottleneck links). Therefore, the time scale of observed queueing prior to loss might actually be in the 20-40 millisecond range.

The goal of Moon's work is similar to at least one aspect of our work: to study the extent that an observed increase in delay is a good predictor of future packet loss. The key difference between our work and theirs is that our "queue delay" sampling process is based on TCP constrained RTT-based probing while their sampling process is not impacted by congestion control and tends to produce more smooth and frequent samples (every 20 milliseconds) of delays. A TCP DCA algorithm's sampling process is more bursty and less frequent (because it is constrained by congestion control) than a constant rate probing algorithm. The sampling frequency available to a TCP based DCA algorithm will actually decrease as congestion increases (either due to TCP's congestion avoidance algorithms or due to low advertised windows) which

greatly hinders the accuracy of the congestion detection. We compare our results to Moons in a later section.

We conclude this section with a summary of our motivations. With the exception of [MOON99], there has been no previous work that has provided a generic study of DCA. All prior work has analyzed particular DCA algorithms. Furthermore, previous DCA algorithm studies have concentrated on low speed networks. For example, [AHN95] assumes that their results, which are based on an analysis that primarily involves low speed networks, extend to higher speed networks.

A key question this research addresses is if DCA is effective at avoiding packet loss. This is an important research issue given that recent measurements have shown that the growing demand for network bandwidth has increased the level of packet loss across various links in the Internet [PAXS97]. A further motivation for our work was to gain insight into the benefits of DCA to transport protocols other than TCP (e.g., TCP-friendly transport protocols for adaptive continuous media applications or multicast transport).