

TCP-BFA: Buffer Fill Avoidance

A. A. Awadallah, C. Rai

Computer Systems Laboratory, Stanford University

Gates 3A, Stanford University, Stanford CA 94305, USA, Tel: +1-650-723-1414, Fax: +1-650-725-6949, {aaa,crai}@cs.stanford.edu

Abstract

The main goal of a congestion avoidance algorithm is to maximize throughput and minimize delay (Jain & Ramakrishnan 1988). While TCP Reno achieves high throughput, it tends to consume all of the buffer space at the bottleneck router, causing large delays. In this paper we propose a simple scheme that modifies TCP Reno's congestion avoidance algorithm by throttling back the opening of the congestion window once an increase in round-trip time is perceived. We call the scheme TCP-BFA and have implemented it in the *ns* network simulator and in BSD 4.4. We show through simulations and measurements of real traffic on the Internet that TCP-BFA results in lower router buffer occupancies and lower delays while maintaining a throughput similar to that of TCP Reno. The advantages of TCP-BFA are (1) smaller router buffer size requirements, (2) an order of magnitude improvement in network power (the ratio of throughput to delay), (3) fewer packet losses, (4) faster detection of multiple losses due to lower retransmission timeout estimates, and (5) smoother traffic patterns.

Keywords

TCP, congestion control, congestion avoidance, Internet, transport protocols, optimal window

1 INTRODUCTION

TCP is the most widely used transport protocol for today's Internet data applications. However, the performance and operation of TCP's adaptive retransmission and congestion control mechanism is one of the most widely debated issues in the research community. Revisions for TCP have been proposed over the years (e.g. Jacobson 1988, Jacobson 1990, Jacobson, Braden & Borman 1992, Brakmo, O'Malley & Peterson 1994, Hoe 1996, Mathis, Mahdavi, Floyd & Romanow 1996, Mathis & Mahdavi 1996, Wang & Crowcroft 1991, Wang & Crowcroft 1992, Floyd 1995), with Jacobson's paper (Jacobson 1988) representing a major milestone. Some of these proposed changes have been widely adopted and are part of TCP implementations today.

The source has the primary responsibility for TCP's congestion avoidance

and control. The sink simply sends ACKs back to the source for the data it receives; no explicit congestion control information is sent back. The network also provides no explicit notification about its state to the source (Source Quench messages are rarely used, and Floyd's (1995) Explicit Congestion Notification is yet to be deployed), further complicating the task of the source. The source needs to make control decisions about the network by guessing the state of the network from the information it has: the ACKs it receives from the sink, and the timing information it obtains by estimating the round trip time (RTT) of the transmitted packets.

In the absence of explicit feedback from the network to the source, schemes in two flavors have been proposed to provide high throughput with low delay. The first kind, such as Tri-S (Wang & Crowcroft 1991), RED gateways (Floyd & Jacobson 1993) and SACK (Mathis et al. 1996) (with associated algorithms such as FACK (Mathis & Mahdavi 1996)), extend the functionality of routers or TCP sinks to provide this missing feedback. Others, such as CARD (Congestion Avoidance using Round-trip Delay) (Jain 1989), Keshav (1991), the DUAL algorithm (Wang & Crowcroft 1992), TCP Vegas (Brakmo et al. 1994) and Hoe (1996), focus on the transport-layer protocol at the source. For example, Vegas allows TCP to stop below the point where it would lose packets, hence providing a smoother stream. Ahn, Danzig, Liu & Yan (1995) show that Vegas achieves from 3 to 8% better throughput, with only 1/5 to 1/2 of the losses, as compared to the BSD-Reno distribution, though the stability of Vegas has been questioned by Jacobson (1994).

Reno sources exhibit a start-stop behavior because they increase their window sizes, filling up buffers in the network and consequently suffering a packet loss or a timeout and falling back to a smaller window size. No attempt is made to predict the optimal window size: the size at which the (bandwidth-delay product) pipe is full and the buffers are empty.

In this paper we propose a method to improve congestion control in TCP without requiring any additional explicit information from the sink or the network. Our goals are to

- make TCP seek the optimal window size and operate close to it. This leads to
 - lower delay, which results in higher network power* and faster recovery from multiple losses.
 - lower buffer occupancies, which allows routers to support a higher load for a given buffer size; conversely, a smaller buffer can be used to support the same load (without a corresponding increase in the number of packet losses).
 - a reduction in fluctuations in the round-trip delay and the number of

*Network power is defined as the ratio of throughput to delay (Jain 1989, Kleinrock 1979).

packets in the network, allowing better interaction with non-TCP traffic such as real-time streams.

- only make modifications that are simple enough to incorporate into current TCP implementations.

TCP-BFA is similar to Reno. It realizes the above goals primarily by freezing the congestion window on a sustained increase in RTT. Upon detecting a sustained decrease in RTT, TCP-BFA reverts to normal Reno behavior. Hysteresis is used to avoid rapid flipping between the two states. To detect sustained changes in RTT, TCP-BFA uses an extra state variable that maintains the *signed* RTT variance. Since TCP-BFA relies heavily on RTT estimates, a timer granularity finer than the usual 500ms is required. It should be stressed, however, that this finer granularity is used only for RTT estimates; *the retransmission timeout (RTO) is still calculated with the coarse 500ms granularity for stability reasons* (Jacobson 1994). Since TCP-BFA has the additive increase and multiplicative decrease properties of Reno, with the change that additive increase is sometimes stopped, it is stable* if Reno is stable.

Our simulations show that TCP-BFA requires considerably less buffer space in routers to attain the same throughput as Reno. Both the simulations and the Internet measurements show that TCP-BFA achieves smaller RTT averages – thus up to an order of magnitude higher network power – than Reno. When TCP-BFA shares the bottleneck link with one or more Reno sources, the gains in network power are modest to nonexistent since the Reno sources force higher buffer occupancies. In this situation, simulations indicate that TCP-BFA achieves a higher throughput than Reno, while the measurements show comparable values. It might appear that TCP-BFA will *always* be in competition with Reno, but we can expect a single form of TCP on a large number of paths with the bottleneck at the tail link (e.g., home PCs behind low speed modems).

The proposed algorithm and the concepts behind it are discussed in Section 2. In Section 3 we provide a simple implementation which is a heuristic approximation to the discussed algorithm. Sections 4 and 5 report simulation results and Internet measurements, respectively. Section 6 concludes the paper.

2 OPTIMAL WINDOW SIZE

Consider a source that uses window-based flow control connected to a sink through a number of links and buffers with out-of-band ACKs. Figure 1 (from (Jain 1989)) shows typical throughput, delay and power curves plotted against

*By stable, we mean that it does not fall off the cliff of Figure 1 (see Section 2).

window size for this scenario. An efficient window flow control protocol should operate at the point N^* where the throughput reaches a plateau and the delay starts increasing. Increasing the window size N beyond this point will not significantly improve the throughput but will increase the delay. Note that N^* lies on the knee of the delay curve. It can be shown (Fendick, Mitra, Mitrani, Rodriguez, Seery & Weiss 1991) that the value of N^* is very close, but slightly higher, than the value N_f , where N_f is the number of packets that can be stored in the bandwidth-delay product pipe between the source and the sink (with zero occupancy of the buffers). Basically, we need $N^* > N_f$ to account for the stochastic nature of the traffic arrival pattern: this ensures that the bottleneck link remains fully utilized.

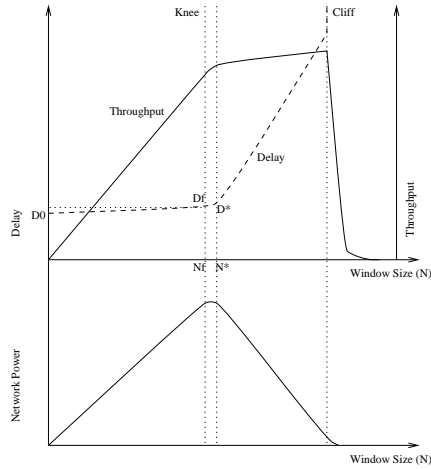


Figure 1 Window flow control: the optimal window size. The plots are for a single source running on an unloaded network.

If the source starts with a window size of one it will experience the smallest RTT (D_0 in the figure). As the window size increases, the RTT will stay constant at D_0 ($\approx D_f \approx D^*$) until the source reaches the point where it starts filling buffers and increasing the RTT (this happens when it reaches N^*). Now,

$$\begin{aligned} \text{for } N < N^*, \quad \text{RTT} &\approx D^* = N^*/\rho, \\ \text{for } N > N^*, \quad \text{RTT} &= N/\rho, \end{aligned}$$

where ρ is the rate at which the ACKs arrive at the source (i.e., our share of the bottleneck link bandwidth). Note that for $N < N^*$, the window is not big enough to cover the RTT and silence periods are introduced, leading to

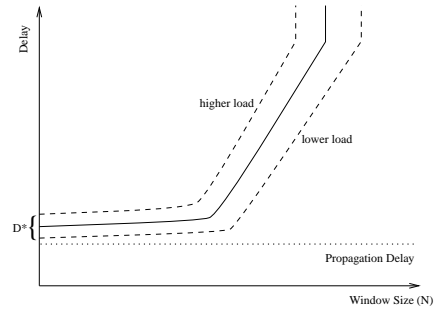


Figure 2 Window flow control: dynamic network behavior. The plots are for a source running on a loaded network.

Implementation

underutilization of the network. On the other hand, for $N > N^*$, packets accumulate at the buffers and $RTT > D^*$.

Figure 2 illustrates the effect on delay caused by varying network load. We see that the points D^* , N^* , and the slope vary with the load. The variation of D^* may not be obvious: it can occur due to routing changes, and/or constant background load generated by open-loop sources (such as multicast or streamed traffic).

The source will have a set of curves depending on the network load and network events (such as failing links, routing changes, etc.), but in all cases the following algorithm* will track the optimal window size N^* :

```
if (  $\partial RTT / \partial N > \epsilon$  ) decrease  $N$ 
else increase  $N$ 
```

where $\partial RTT / \partial N$ is the partial derivative of delay with respect to window size, and ϵ is a small relative threshold value close to zero.

In practice, estimating $\partial RTT / \partial N$ is difficult, given the dynamic nature of the network and its traffic. However, what we need to estimate is the sign of this quantity rather than the magnitude. Observing the fact that, for TCP, N is always increasing (except during packet losses, which can be handled separately), we can simplify the problem further to the estimation of the sign of ∂RTT . This simplification is the essence of our implementation, as described in the following section.

Previous work on tracking the optimal window size has used a variety of metrics: Jain (1989) proposes an algorithm (CARD) based on maximizing network power (see Figure 1); Wang & Crowcroft (1991) use the derivative of sending rate with respect to window size $\partial \rho / \partial N$ (Tri-S); and Brakmo et al. (1994) use the difference between the actual throughput and an expected value (Vegas).

3 IMPLEMENTATION

We now introduce a simple heuristic approach that constrains TCP to operate close to the (continuously varying) N^* . Our changes are easy to integrate with current TCP implementations.

3.1 Timer Granularity

Current TCP implementations use RTT estimates only for the calculation of RTO. Our modified implementation uses RTT estimates for congestion control, and the $500ms$ timer granularity is too coarse for this purpose. We

*This follows from (Jain 1989).

increased the granularity with which RTT estimates are measured to $10ms$, while maintaining the same coarse granularity for RTO estimates. Achieving this finer granularity is straightforward and does not involve any changes in the transmit timer function of TCP; we simply redefined the meaning of a timer tick.

3.2 Signed RTT Variance

We maintain a signed RTT variance* as an extra TCP state variable; this is our approximation to ∂RTT . We cannot use the (unsigned) RTT variance already maintained by TCP because it measures absolute differences. The signed RTT variance uses non-absolute differences and a filter gain constant (α_{srv}) of $\frac{1}{2}$ instead of the $\frac{3}{4}$ used by the unsigned RTT variance of (Jacobson 1988); its computation is otherwise identical to that of the RTT variance already present in Reno implementations. The gain constant of $\frac{1}{2}$ was determined by experiment to provide a good balance between minimizing the error due to randomness and maximizing reactivity to network changes. Using the signed RTT variance instead of variables involving the (current) measured RTT has the effect of filtering out transient network noise. Note that use of the timestamp TCP option improves the accuracy of the signed RTT variance because RTT samples are more frequent.

3.3 Buffer Fill Avoidance

We have introduced an additional TCP variable (BFA Flag) to define a new state, Buffer Fill Avoidance (BFA), in which the congestion window (`cwnd`) is held constant.

This state is entered, either from Slow Start or from Congestion Avoidance, whenever the signed RTT variance indicates a positive drift in RTT. The BFA state is left when there is a negative drift. To prevent rapid oscillation between states, we introduce hysteresis using separate thresholds for setting and clearing the flag. The BFA Flag is set as follows:

```

if ( BFA Flag is on )
    if (Signed RTT Variance <=  $\sigma_{\text{off}}$ )
        switch off BFA Flag
else
    if (Signed RTT Variance >  $\sigma_{\text{on}}$ )
        switch on BFA Flag

```

*Let srv = signed RTT variance, α_{srv} = filter gain constant, and δ = measured RTT - smoothed RTT. Then, $srv_{t+1} = \alpha_{srv} \cdot srv_t + (1 - \alpha_{srv}) \cdot \delta$.

Simulation Results

where σ_{off} and σ_{on} are the switch-off and switch-on thresholds, respectively ($\sigma_{\text{off}} < \sigma_{\text{on}}$ for hysteresis). In addition, the **BFA Flag** is cleared whenever the congestion window drops due to a timeout or a fast retransmit.

If the threshold values are not properly tuned, the source can make false inferences about the state of the network. For instance, if the **BFA Flag** is set and the network load is constant, the signed variance tends to zero. If σ_{off} is non-negative, this will cause the **BFA Flag** to be switched off in spite of the constant load, which is clearly incorrect. Similarly, σ_{on} needs to be constrained to positive values. By adjusting the values of σ_{off} and σ_{on} the source can be tuned from ‘very aggressive’ to ‘extremely well-behaved’. We have used $\sigma_{\text{off}} = -10ms$ and $\sigma_{\text{on}} = 10ms$.

As described in Section 2, a more proactive scheme would decrease the congestion window in an attempt to track the optimal value. We have chosen the simpler scheme of fixing the window and relying on normal TCP mechanisms for the decrease. The reason for this choice is that decreasing the window size makes the source much less aggressive, which causes it to yield bandwidth to Reno sources.

4 SIMULATION RESULTS

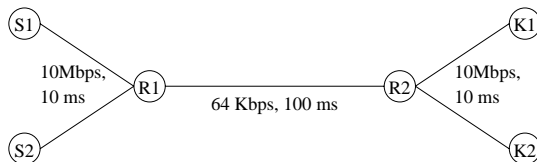


Figure 3 Simulation Scenario

The results presented here have been obtained from the network simulator *ns 2.1* (McCanne & Floyd 1997) developed at LBNL.* The scenario used for most of the simulations is shown in Figure 3. It models network paths with a single bottleneck link. This includes paths with a tail bottleneck (in the case of a machine behind a slow modem link) since the links between S^* and $R1$, and those between $R2$ and K^* , never buffer packets. In fact, simulations with the bottleneck between $R2$ and any of the K^* yield the same behavior.

All routers use tail-drop buffers. The bottleneck link bandwidth of 64 Kbps represents link speeds of today’s modems, and has the excellent side-effect of producing trace files of reasonable size! Simulations with higher bandwidths (such as 1.5 Mbps) gave similar results. The receiver window is set to a large value so that it does not limit the increase in the size of the congestion window.

*The *ns 2.1* implementation of TCP-BFA can be downloaded from <http://klamath.stanford.edu/~aaa/tcp-bfa>.

The TCP agent uses a packet size of 1000 bytes (64 Kbps translates to 8.0 packets/sec). All transfers are FTP bulk data transfers. Buffer size is measured in packets, and throughput in packets/sec.

4.1 Single Sources

A typical queue behavior for Reno and TCP-BFA running separately on completely unloaded networks is shown in Figure 4. The graph clearly indicates that TCP-BFA uses less buffer space and generates smoother traffic. We also observe that Reno suffers periodic losses while there are no drops for TCP-BFA.

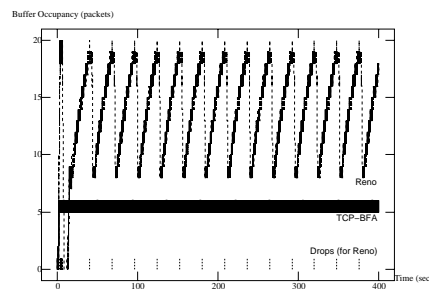


Figure 4 Typical buffer occupancies at the bottleneck *R1* for separate simulation runs of Reno and TCP-BFA (buffer size = 20 packets).

Figures 5 and 6 show throughput and power after 400 seconds of simulation time, plotted against the bottleneck buffer size.* Figure 5 shows that Reno's throughput falls when the buffer size increases. This happens because the maximum delay possible (the delay when the buffer is full) increases with the buffer size. A large delay causes an increase in retransmission timeout (RTO) estimates, leading to a larger recovery time when there are multiple losses. In this scenario, multiple losses occur only during the initial slow-start. These simulations were run for a constant time of 400 seconds; the decrease in throughput will be greater for connections that are more short-lived.

The drop in Reno's network power seen in Figure 6 is due to the compounded effects of a lower throughput and a higher delay. In contrast, for TCP-BFA, the throughput and power remain constant regardless of the buffer size. This is because TCP-BFA stops increasing its window size when the buffer starts to fill.

*Note that each point on these plots represents a complete simulation run.

Simulation Results

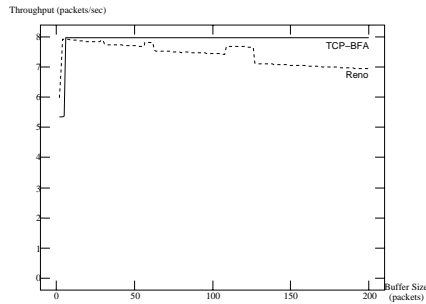


Figure 5 Throughput against bottleneck buffer size for Reno and TCP-BFA sources running separately.

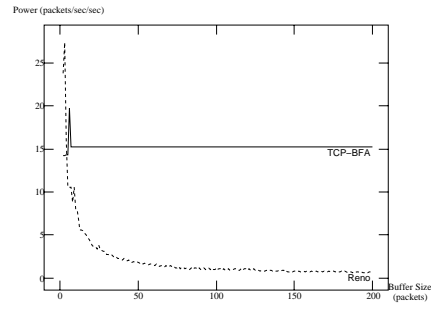


Figure 6 Network power against bottleneck buffer size for Reno and TCP-BFA sources running separately.

4.2 Competing Sources

TCP-BFA sources running in competition with each other retain the benefits mentioned for a single source in Section 4.1. They avoid overuse of network buffers while maintaining high throughput and generate smooth traffic patterns. Figure 7 shows that two BFA sources (starting together) do not take any more buffer space than one source (compare with Figure 4). For Reno sources also, the average buffer occupancy stays the same but there is greater fluctuation and more frequent drops.

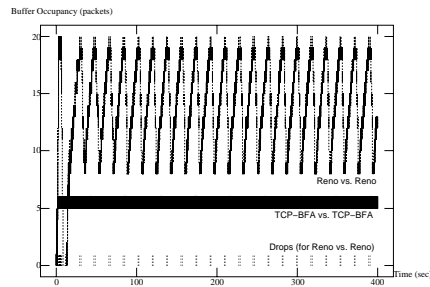


Figure 7 Bottleneck buffer occupancy for: Reno vs. Reno and TCP-BFA vs. TCP-BFA (buffer size = 20 packets).

Figures 8 and 9 are plots of throughput and network power versus bottleneck buffer size for three different scenarios: TCP-BFA vs. TCP-BFA, TCP-BFA vs. Reno and Reno vs. Reno, with both sources starting at the same time in each case. The simulation time for each run is 1000 seconds; when the buffer size is greater than 50 packets, it takes longer to reach steady state (see Figure 10).

First we note that after a certain buffer size all scenarios have similar ag-

TCP-BFA: Buffer Fill Avoidance

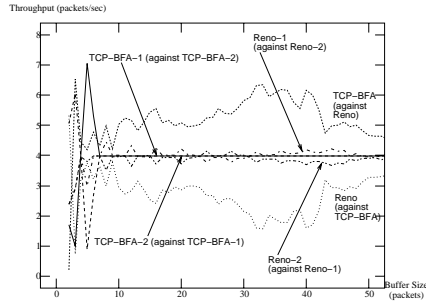


Figure 8 Throughput against bottleneck buffer size for TCP-BFA vs. TCP-BFA, TCP-BFA vs. Reno and Reno vs. Reno

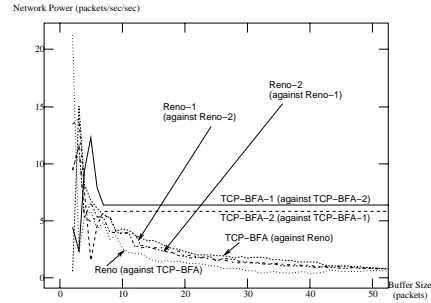


Figure 9 Network power against bottleneck buffer size for TCP-BFA vs. TCP-BFA, TCP-BFA vs. Reno and Reno vs. Reno.

gregate throughput. This is because the buffers are never empty, hence the bottleneck link is always fully utilized. The plots show that when TCP-BFA sources compete with each other, network power is much higher than for any of the other scenarios. When TCP-BFA competes with Reno, TCP-BFA achieves higher throughput and network power. However, the network power is far less than that achieved while competing with another TCP-BFA source. This is because the Reno source will fill up the network buffers anyway, thus increasing delay. TCP-BFA will observe this same high network delay.

Figure 10 shows that TCP-BFA on average maintains a higher congestion window than Reno. This explains why TCP-BFA achieves higher throughput. The reason why Reno has smaller window sizes is that it suffers more losses. This is because, for any drop-tail buffer, packets are more likely to be dropped if they arrive in bursts. A TCP source generates back-to-back packets when its window size is increasing since multiple packets are sent for each ACK received. When the window size is constant, as in TCP-BFA's Buffer Fill Avoidance state, no back-to-back packets will be generated. Note that Reno sources will not have a disadvantage in RED buffers since they do not punish bursty sources. This conclusion has been verified by initial simulations with RED routers.

4.3 Fairness

Figure 8 shows that TCP-BFA sources starting together are perfectly fair,* with exactly the same throughput. Figure 11 shows the throughput plotted against buffer size for six TCP-BFA sources starting at different times. The first source initially stops its window size increase at a higher value than the others because it was the only source in the network when it started. This

*By fairness we mean the equal division of bandwidth between competing sources.

Simulation Results

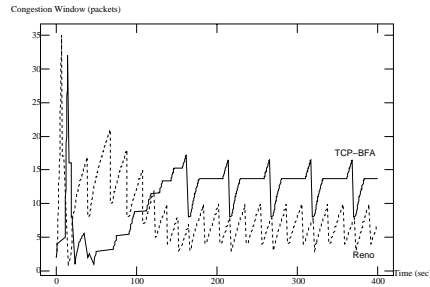


Figure 10 Typical congestion window variation for Reno and TCP-BFA sources in competition (buffer size = 20 packets).

unfair situation will persist unless there are losses or changes in delay. For our simulation scenario, Figure 11 shows that there are no losses (and therefore persistent unfairness) for large buffer sizes.

Figure 12 shows plots of Jain's (1991) Fairness Index* (ϕ) against buffer size for six TCP-BFA or Reno sources starting either together or at different times.

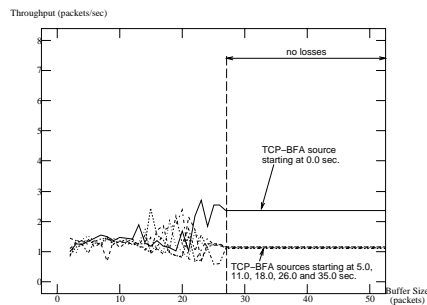


Figure 11 Throughput against buffer size for six TCP-BFA sources starting at different times.

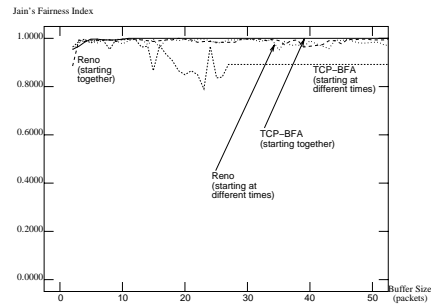


Figure 12 Jain's Fairness Index plotted against bottleneck buffer size for six Reno or TCP-BFA sources, starting together or at different times.

We conducted simulations for asymmetric delay paths with two sources. For a scenario with 25% difference in propagation delays, TCP-BFA ($\phi = 1.0000$ when starting together, $\phi = 0.8989$ when starting at different times) was fairer than Reno ($\phi = 0.6164$ when starting together, $\phi = 0.6467$ when

*Jain's Fairness Index $= \phi = \frac{[\sum_{i=1}^n x_i]^2}{n \cdot \sum_{i=1}^n x_i^2}$ where n is the number of sources, and x_i is the throughput for the i^{th} source. The index is bounded between 0 and 1, with 1 indicating that all sources had the same throughput. Note that Jain's Fairness Index is a generic metric that can be applied to any resource.

starting at different times). The drop in ϕ for Reno (compared to the symmetric case) is due to the fact that sources increase their congestion windows at different rates. TCP-BFA, however, maintains ϕ similar to the symmetric case because the window size is mostly constant, and the signed RTT variance is independent of the path delay.

Losses help improve fairness because they reset state and prevent persistence of an unfair situation, such as the one in Figure 11. One way to improve the fairness of TCP-BFA would be to prevent it from remaining in the Buffer Fill Avoidance state for a long time. This could be achieved by adding a timer that forces the source to switch off the BFA Flag, leading to a window size increase as in Reno, ultimately causing losses. We have chosen not to add this extra mechanism to TCP-BFA since situations without any losses or changes in delay do not exist in the real world. Internet measurements with competing sources show that TCP-BFA is fairer in real-world situations than our simulations would lead us to believe.

5 INTERNET MEASUREMENTS

In this section we discuss the results obtained from running TCP-BFA and Reno over the Internet. The FreeBSD 2.1.6 kernel (based on BSD 4.4) was instrumented to generate logs of various TCP state variables. The scenario is one of bulk data transfer (5MB FTP) from a 120 MHz Pentium machine running FreeBSD 2.1.6 to a DEC Alpha machine running OSF/1, across a transatlantic path of 20 hops with a 64 Kbps bottleneck link adjacent to the DEC Alpha machine. The receiver buffer size is 32KB, and the maximum segment size is 536 bytes. The round trip time for a segment of maximum size without queuing delay is approximately 320 ms. The bandwidth-delay product for this path is therefore 2.56KB, which is approximately five packets (this is the optimal window size). We note that when a (single) Reno source is running the round trip time rises to as much as 4 seconds. This implies a bandwidth-delay product of 32KB (about 60 packets), which is 12 times the optimal window size. Runs were made over a six month period at various times of day to obtain data under different network conditions.

We provide two types of plots for three different levels of congestion.* The first is a conventional congestion window (`cwnd`) vs. time graph, which also shows the slow-start-threshold (`ssthresh`) and the receiver's advertised window. The second is a scatter-plot of RTT vs. `cwnd`; it is the real-world counterpart of the ideal plot shown in Figure 2.

When congestion is low, the window size employed by TCP-BFA (Figure 13) is much smaller than Reno's (Figure 15). In fact, Reno's window could have

*These three levels of congestion represent a spectrum of network conditions from 'completely unloaded' (few or no competing flows) to 'high level of background traffic' (large number of competing flows and a high level of packet loss).

been even larger (possibly resulting in losses) if it were not for the limit imposed by the receiver's advertised window size. Despite the large difference in window size, the throughput is similar for both Reno and TCP-BFA, because the throughput of a TCP source is governed by *the rate at which the window is sliding*. The network power is much higher for TCP-BFA. The comparable throughput coupled with higher power means that TCP-BFA must have a lower delay. This is confirmed by the RTT vs. cwnd scatter-plots (Figures 14 and 16); TCP-BFA operates closer to the knee than Reno. The staircase pattern seen in Figure 13 is due to noise in RTT measurements causing the signed RTT variance to drop below σ_{off} momentarily.

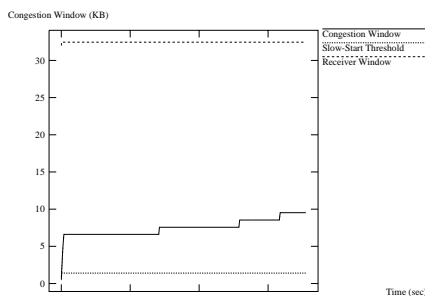


Figure 13 Congestion Window vs. Time: TCP-BFA, low congestion. (Throughput = 56.48 Kbps, Power = 54.68 Kb/s²)

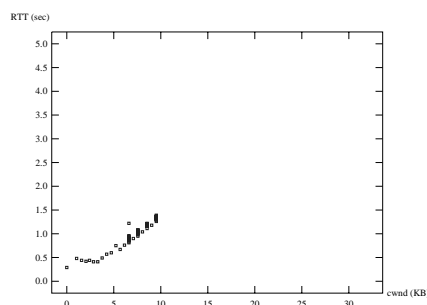


Figure 14 RTT vs. Congestion Window: TCP-BFA, low congestion.

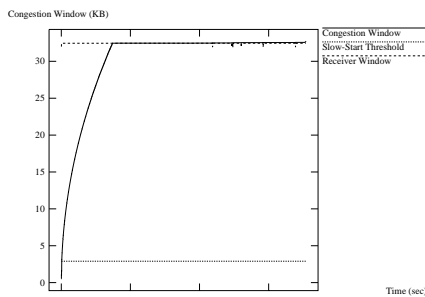


Figure 15 Congestion Window vs. Time: TCP Reno, low congestion. (Throughput = 56.64 Kbps, Power = 29.02 Kb/s²)

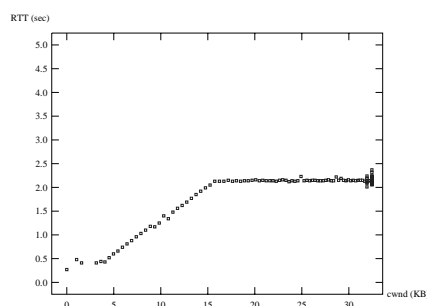


Figure 16 RTT vs. Congestion Window: TCP Reno, low congestion

For medium congestion (Figures 17 and 19), we note some packet losses* and more noise in the scatter-plots (Figures 18 and 20). TCP-BFA does not achieve high power in this case because there is competition with other sources at the bottleneck. In today's Internet, we can expect most of these sources to be Reno sources. It is important to note that in this situation TCP-BFA has adapted to behave more like Reno so that it does not lose throughput to competing Reno sources. α_{srv} can be increased to make TCP-BFA operate closer to the knee, though this will make it less aggressive and could cause it to yield throughput to competing Reno sources. Figures 21 and 22 show plots for TCP-BFA with $\alpha_{\text{srv}} = \frac{3}{4}$ (instead of the usual $\frac{1}{2}$). Figure 22 shows the tighter clustering of RTT vs. *cwnd* points around the knee.

TCP-BFA (Figure 23) and Reno (Figure 25) behave similarly during very high congestion. The scatter-plots (Figures 24 and 26) show that the network is not providing the sources with useful information, and the *cwnd* graphs show a large number of losses, with *cwnd* never rising over a few packets.

Unlike the simulation results described in the previous section, measurements for competing TCP-BFA sources show fair allocation of the bottleneck bandwidth. Runs were conducted with three TCP-BFA sources in competition at a time when there was little background traffic. The average value of Jain's Fairness Index for these runs was 0.9991 (the corresponding value for Reno sources was 0.9991), which is much higher than predicted by the simulations (0.89).

This results from the fact that natural losses and delay variations exist intrinsically in the real Internet. This prevents any TCP-BFA source from freezing its congestion window at a large size and monopolizing the bottleneck link. Measurements also revealed that TCP-BFA sources competing with Reno do not achieve larger throughput, but rather achieve throughputs slightly lower than Reno.

6 CONCLUSIONS AND FUTURE WORK

Both Reno's and TCP-BFA's congestion control mechanisms keep their congestion window size in the the region between the *knee* and *cliff* of Figure 1. The difference is that while Reno constantly drifts towards the *cliff*, TCP-BFA attempts to keep the window size as close to the *knee* as possible. Reno continues to increase the number of packets it sends into the network, even when the delay is rising. This continues until some buffer fills up and packets are lost. On the other hand, TCP-BFA will refrain from increasing its window size whenever it detects a sustained increase in delay. However, when TCP-BFA sources compete with Reno sources they are forced to operate closer to the *cliff*.

TCP-BFA retains the additive increase / multiplicative decrease mechanism

*The spikes in these plots are due to temporary *cwnd* inflation during fast recovery.

Conclusions and Future Work

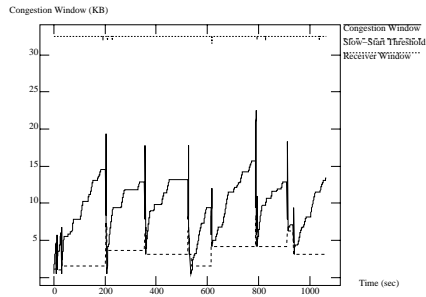


Figure 17 Congestion Window vs. Time: TCP-BFA, medium congestion. (Throughput = 37.68 Kbps, Power = 19.84 Kb/s²)

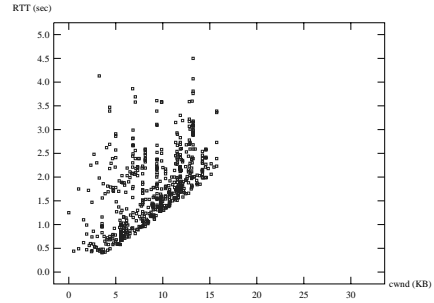


Figure 18 RTT vs. Congestion Window: TCP-BFA, medium congestion

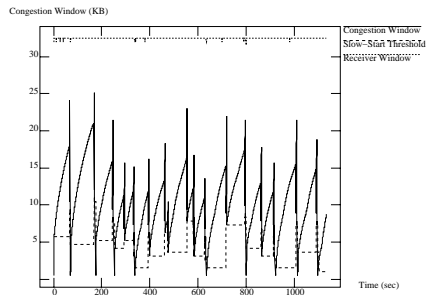


Figure 19 Congestion Window vs. Time: TCP Reno, medium congestion. (Throughput = 35.28 Kbps, Power = 17.49 Kb/s²)

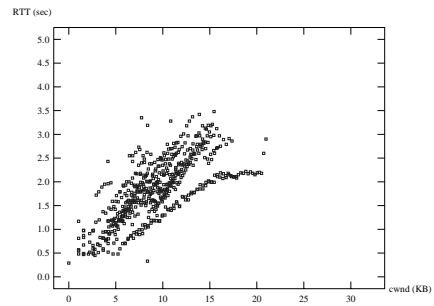


Figure 20 RTT vs. Congestion Window: TCP Reno, medium congestion

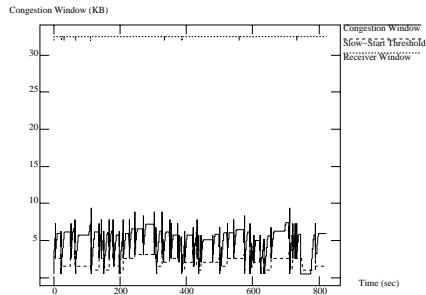


Figure 21 Congestion Window vs. Time: TCP-BFA with $\alpha_{srv} = \frac{3}{4}$. (Throughput = 49.69 Kbps, Power = 70.17 Kb/s²)

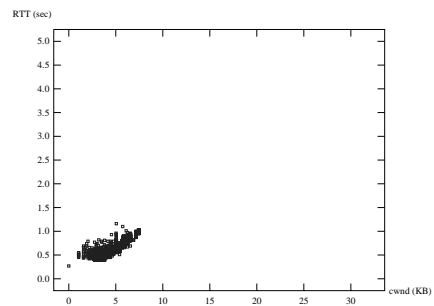


Figure 22 RTT vs. Congestion Window: TCP-BFA with $\alpha_{srv} = \frac{3}{4}$.

TCP-BFA: Buffer Fill Avoidance

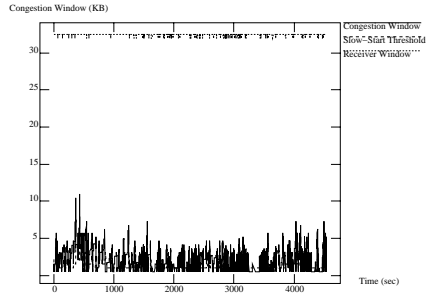


Figure 23 Congestion Window vs. Time: TCP-BFA, high congestion. (Throughput = 8.80 Kbps, Power = 6.01 Kb/s²)

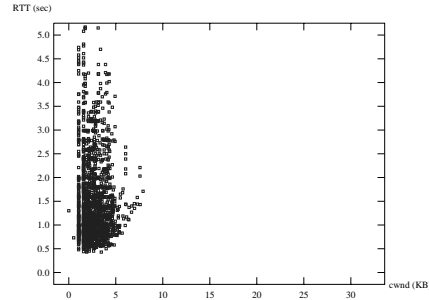


Figure 24 RTT vs. Congestion Window: TCP-BFA, high congestion

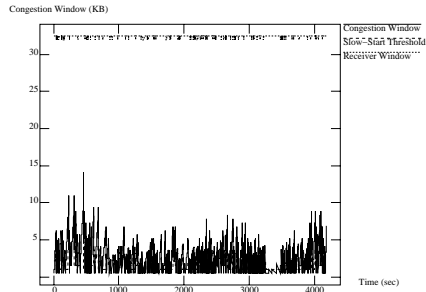


Figure 25 Congestion Window vs. Time: TCP Reno, high congestion. (Throughput = 9.52 Kbps, Power = 6.03 Kb/s²)

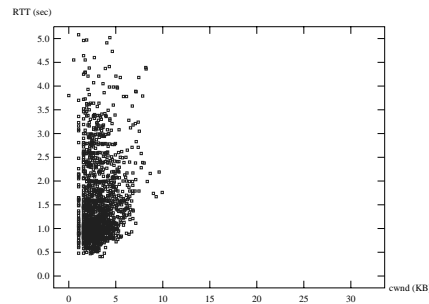


Figure 26 RTT vs. Congestion Window: TCP Reno, high congestion

of Reno, with the addition that it sometimes halts additive increase. If TCP-BFA reaches the cliff it reacts in the same manner as Reno (multiplicative decrease). This implies that if Reno is stable, TCP-BFA is stable.

The most important benefit of using TCP-BFA is that it leads to lower network buffer occupancies – which we have demonstrated by simulations. Both simulations and Internet measurements show that TCP-BFA sources are able to achieve a considerable improvement in network power over Reno sources, except if there is high congestion, when they perform just as well. The results also demonstrate that competing TCP-BFA sources achieve higher power and experience less losses than competing Reno sources. Simulations demonstrated that TCP-BFA sources get a higher share of throughput when competing with Reno sources. However, for our specific set of measurements, TCP-BFA sources achieved slightly lower throughput. Results also demonstrated that TCP-BFA has fewer losses, confirming that TCP-BFA sources avoid the cliff. We claim that if Reno is replaced with TCP-BFA, the end-

Conclusions and Future Work

station will get similar throughput and higher reactivity to network events due to lower delays, while the network will see lower buffer occupancies.

This work is different from Vegas in that it is simpler to implement and avoids some of Vegas' pitfalls, specifically: possible instability due to the use of finer granularity timeout values, yielding of bandwidth when competing with Reno sources, and incorrect behavior when there are route changes.*

Future research should include the investigation of a scheme which seeks the optimal window size more aggressively, for instance by *reducing* the congestion window when the signed variance is very high. The danger for a 'nice' scheme like this is that it can lose bandwidth when operating in competition with more aggressive sources. For example, Brakmo & Peterson (1995) demonstrated that Vegas loses throughput in head-to-head transfers against Reno.

Another aspect that deserves further investigation is the dynamic estimation of the thresholds σ_{off} and σ_{on} (depending on path characteristics such as bandwidth and delay). Techniques similar to those developed in Jacobson's (1997) `pathchar` can be used to estimate these path characteristics.

The signed RTT variance that we maintain can be used to improve TCP timeout behavior. In current Reno implementations, the RTO is obtained by adding the smoothed RTT estimate to 4 times the unsigned RTT variance. However, if the signed variance is negative, RTT is decreasing and this computation can lead to unnecessarily high RTO estimates. Using a multiplicative factor of less than 4 in this situation may lead to faster packet loss detection; the smoothing of the variance should help avoid spurious timeouts.

Since at moments of high congestion the source cannot make correct inferences about the network, we believe that schemes depending on network routers to assist in congestion avoidance are necessary (e.g., Floyd's (1995) Explicit Congestion Notification, Floyd & Jacobson's (1993) Random Early Detection). Routers have a unified view of the queuing behavior over time, and can therefore make better decisions about the level of congestion compared to the endpoints which have distorted and delayed information.

Since TCP-BFA sources cause fewer losses and generate smoother traffic, it would be interesting to investigate the behavior of TCP-BFA when it interacts with real-time multimedia streams as compared to Reno sources.

We think deploying fair queuing schedulers will provide TCP-BFA with an advantage that may allow it to outperform competing Reno sources. This is because fair queuing provides a separation between different flows, hence preventing ill-behaved Reno sources from interfering with the delay of TCP-BFA sources. Demers, Keshav & Shenker (1989) emphasize that fair queuing schedulers reward sources that use more sophisticated and responsive algorithms.

*TCP-BFA does not rely on a base value for RTT; instead it uses a moving average of the RTT variance.

7 ACKNOWLEDGMENTS

The authors would like to thank Professor Nick McKeown for his constant support and guidance without which this work would not have been possible. Much credit is due to Pablo Molinero-Fernández for initial contributions to the concepts behind Buffer Fill Avoidance. We are especially grateful to Michael Greenwald for many helpful discussions which helped shape this work. We would like to thank Nick McKeown, Craig Partridge, Yakov Rekhter and the anonymous SIGCOMM reviewers for their helpful comments on earlier drafts of the paper. We are grateful to Simon Crosby at Cambridge University, Balaji Prabhakar at MIT, and Samir Shaheen at Cairo University for providing us with access to the machines we used for Internet measurements. Finally, we would like to acknowledge the *ns* development team for their excellent network simulator.

APPENDIX 1 DEPLOYING TCP-BFA

Deploying TCP-BFA in the Internet is a rather straightforward task. Simple changes (on the order of a few lines of code) need to be made to the TCP source; no changes are required for the sink. A patch for FreeBSD 2.1.6 network servers that can be applied to `netinet/tcp_input.c` is available at <http://klamath.stanford.edu/~aaa/tcp-bfa>. The system administrator may tune TCP-BFA's parameters (σ_{off} , σ_{on} , α_{srv}) to modify the behavior of the TCP-BFA source as described in Section 3.

REFERENCES

- Ahn, J. S., Danzig, P. B., Liu, Z. & Yan, L. (1995), Evaluation of TCP Vegas: Emulation and experiment, *in* 'Proceedings of SIGCOMM '95'.
- Brakmo, L. S., O'Malley, S. W. & Peterson, L. L. (1994), TCP Vegas: New techniques for congestion detection and avoidance, *in* 'Proceedings of SIGCOMM '94', pp. 24–35.
- Brakmo, L. S. & Peterson, L. L. (1995), 'TCP Vegas: End to end congestion avoidance on a global internet', *IEEE Journal on Selected Areas in Communications* **13**(8), 1465–1480.
- Demers, A., Keshav, S. & Shenker, S. (1989), 'Analysis and simulation of a fair queueing algorithm', *IEEE/ACM Transactions on Networking* **9**(1), 1–12.
- Fendick, K. W., Mitra, D., Mitrani, I., Rodriguez, M. A., Seery, J. B. & Weiss, A. (1991), 'An approach to high-performance, high-speed data networks', *IEEE Communications Magazine* pp. 74–82.
- Floyd, S. (1995), 'TCP and explicit congestion notification', *ACM Computer Communication Review* **24**(5), 8–23.

- Floyd, S. & Jacobson, V. (1993), 'Random early detection gateways for congestion avoidance', *IEEE/ACM Transactions on Networking* **1**(4), 397–413.
- Hoe, J. C. (1996), Improving the start-up behavior of a congestion control scheme for TCP, in 'Proceedings of SIGCOMM '96', pp. 270–280.
- Jacobson, V. (1988), Congestion avoidance and control, in 'Proceedings of SIGCOMM '88', pp. 314–329.
- Jacobson, V. (1990), 'Modified TCP congestion avoidance algorithm', Email to end2end-interest mailing list. Obtain via <ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt>.
- Jacobson, V. (1994), 'Problems with Arizona's Vegas', Email to end2end-tf mailing list. Obtain via <ftp://ftp.ee.lbl.gov/email/vanj.94mar14.txt>.
- Jacobson, V. (1997), 'Pathchar - a tool to infer characteristics of internet paths', ftp directory. URL <ftp://ftp.ee.lbl.gov/pathchar>.
- Jacobson, V., Braden, R. T. & Borman, D. A. (1992), 'TCP extensions for high performance', RFC 1323. (37 pages).
- Jain, R. (1989), 'A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks', *ACM Computer Communication Review* **19**(5), 56–71.
- Jain, R. (1991), *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, Wiley-Interscience.
- Jain, R. & Ramakrishnan, K. K. (1988), Congestion avoidance in computer networks with a connectionless network layer: Concepts, goals, and methodology, in 'Proceedings of SIGCOMM '88'.
- Keshav, S. (1991), A control-theoretic approach to flow control, in 'Proceedings of SIGCOMM '91', pp. 3–15.
- Kleinrock, L. (1979), Power and deterministic rules of thumb for probabilistic problems in computer communications, in 'Proceedings of the International Conference in Communications'.
- Mathis, M. & Mahdavi, J. (1996), Forward acknowledgement: Refining TCP congestion control, in 'Proceedings of SIGCOMM '96', pp. 281–291.
- Mathis, M., Mahdavi, J., Floyd, S. & Romanow, A. (1996), 'TCP selective acknowledgement options', RFC 2018.
- McCanne, S. & Floyd, S. (1997), 'Ucb/lbnl/vint network simulator - ns (version 2.1b1)', Web page. URL <http://www-mash.cs.berkeley.edu/ns/ns.html>.
- Wang, Z. & Crowcroft, J. (1991), 'A new congestion control scheme: Slow Start and Search (Tri-S)', *ACM Computer Communication Review* **21**(1), 32–43.
- Wang, Z. & Crowcroft, J. (1992), 'Eliminating periodic packet losses in 4.3-Tahoe BSD TCP congestion control algorithm', *ACM Computer Communication Review* **22**(2), 9–16.

8 BIOGRAPHY

Amr A. Awadallah is currently a graduate student at Stanford University pursuing a PhD degree in Electrical Engineering. He received the B.S. and M.S. degrees in electrical engineering from Cairo University, Egypt, in 1992 and 1995, respectively. His current research interests are in the hardware and software aspects of high performance computer network systems.

Chetan Rai is a Ph.D. candidate in computer science at Stanford University. He received the B.Tech. degree in Computer Science and Engineering from the Indian Institute of Technology Bombay, India, in 1996. His research interests range over distributed systems and computer networking, and he hopes they never narrow any further.