ON IMPROVING THE FAIRNESS OF TCP CONGESTION AVOIDANCE

Thomas R. Henderson, Emile Sahouria, Steven McCanne, Randy H. Katz
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720
E-mail: {tomh, emile, mccanne, randy}@cs.berkeley.edu

Abstract— **We report on a simulation-based study of the efficacy of modified congestion avoidance algorithms on TCP fairness. We first examined whether a change to that algorithm, as proposed by Floyd, that results in a "Constant-Rate" additive increase policy can correct the bias against connections with long round trip times (RTTs). Our first main result is that although the Constant-Rate policy can improve fairness dramatically, we found it difficult to successfully implement such a policy in a heterogeneous network. Next, we explored whether TCP connections with long RTTs could independently improve their own throughputs by becoming slightly more aggressive during the additive increase phase of congestion avoidance. Our second main result is that the judicious use of such a policy is beneficial for long RTT connections, without significantly affecting the performance of peer connections along the same path, thereby improving our metric of network fairness without compromising the utilization of the bottleneck link.**

Keywords— **TCP, Fairness, Congestion Control.**

I. Introduction

THE Transmission Control Protocol (TCP), widely used in the Internet for reliable end-to-end transport, incorporates a window-based congestion control algorithm to reduce congestion in end systems and network routers. TCP can have a window's worth of outstanding data, and it constantly probes for more bandwidth and reacts to congestion via a dynamic window adjustment algorithm [1]. This algorithm permits a new TCP connection to increase its sending window at an exponential rate ("slow start"), until the connection either causes congestion somewhere in the network or is limited by the offered window of the receiver. After congestion is inferred via packet loss, TCP halves its sending window and begins a phase called "congestion avoidance." During this phase, the window is additively increased during periods of non-congestion and multiplicatively decreased upon congestion detection. This algorithm is commonly known as "additive increase and multiplicative decrease," and Chiu and Jain have shown that it provides fair allocations of network bandwidth even though it operates in a distributed manner [2].

However, the analysis in [2] implicitly presumes that all connections in the network share the same additive increase rate and multiplicative decrease factor, and that control actions are taken in a synchronous fashion. In TCP, the multiplicative decrease factor (1/2) is the same for all connections, but the additive increase factor is roughly one segment per round trip time (RTT), and this does not provide a uniform increase in the rates of TCP connections with different RTTs. As observed by many researchers [3]-[6], connections with long RTTs open their windows more slowly after reacting to congestion compared to those with short RTTs; the bias against long RTT connections is on the order of RTT^{α}, where $\alpha < 2$ [6]. And if a mixture of short and long RTT connections share a bottleneck link, severe unfairness is likely as the short RTT connections grab the available bandwidth well before the long RTT connections have a chance.

Table 1 illustrates an example of this problem by presenting the average of 100 simulation runs of 5 "large-window" TCP connections using a bottleneck 1.5 Mb/s link with a buffer of 20 packets. Instead of the bottleneck link being split roughly evenly, the long RTT connections obtain an allocation far from their fair share of 20% each.

TABLE I
Average relative throughputs obtained by 5 TCP connections sharing a bottleneck link.

RTT (ms)	10	100	200	300	600
Usage (%)	40.3	28.1	15.6	10.6	5.4

In this paper, we investigate whether this fairness problem is correctible via changes to the policies implemented in TCP congestion avoidance. We first investigate a suggestion, due to Floyd, that TCP's additive increase phase be modified to ensure that all connections, regardless of their individual RTTs, accelerate their sending rates similarly [5]. Floyd refers to this as the "Constant-Rate" window increase algorithm. We next study how much an individual, long RTT TCP connection can improve its throughput by using a faster rate of increase during its window growth phase. Particularly important is the impact that such connections may have on their peer connections, because protocol changes should be well behaved. Using simulation, we quantify the changes in performance and network fairness due to these modifications by examining a few candidate network topologies. The following are our key results:

- For networks that contain large queues, particularly those that implement a congestion avoidance mechanism known as Random Early Detection (RED) [8], the fairness properties of the network can be vastly improved by using the Constant-Rate (CR) algorithm. However, if the queues are traditional first-in first-out (FIFO) queues, or are not sufficiently large, the performance of CR is much more inconsistent and in some cases worse than that of the standard algorithm. Furthermore, if some connections in the network do not use CR and instead use the standard policy, they can undo the positive effects of CR. For these reasons, we conclude that the CR algorithm would be difficult to successfully deploy in an operational network.

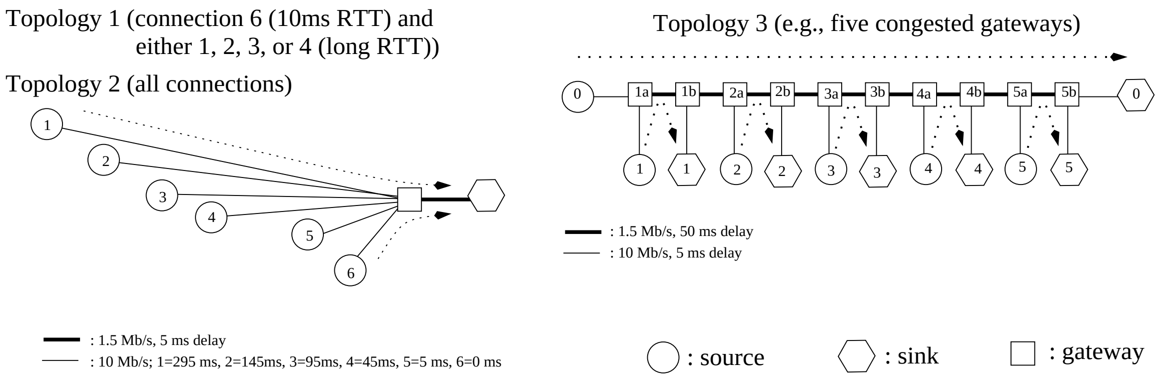


Fig. 1. Three simulation topologies.

• Long RTT connections can improve their own performance, without cooperation from other connections, by making a simple change to their additive increase policy. Such a change, which we refer to as an "increase-by-K" policy rather than the standard "increase-by-one" policy, allows long RTT connections to improve their own throughput, often in excess of 100%. Remarkably, this policy has a relatively benign effect on other competing connections, in the sense that we consistently observed that the bottleneck link utilization stayed relatively constant under our policy while our measure of network fairness improved. Because of the good fairness properties of our policy, we believe that it is a viable candidate for reducing the bias inherent in TCP congestion avoidance.

II. Experimental method

We used the UCB/LBNL Network Simulator "*ns*" [1] to evaluate TCP performance. In addition to using the standard *ns* modules, we ported the HTTP traffic generator module from Bruce Mah's INSANE simulator[2], which gave us the ability to add a mix of realistic background traffic to our simulations.

Our study focused on large file transfer performance. While short HTTP transfers and Telnet connections over long RTT paths are also subject to performance degradation during periods of congestion, this degradation is due more to the fundamental latency of long RTT connections than to problems with congestion avoidance. Additionally, HTTP protocol implementations are migrating towards "persistent-HTTP" and longer duration TCP connections. We also do not assume the implementation of fair scheduling and TCP-friendly buffer management that can isolate flows or classes of flows from one another (e.g., as discussed in [7]), or pricing structures that might give network providers incentives to protect the throughput of paying customers. In short, we assume an environment similar to the present day Internet, with the addition of Random Early Detection (RED) queues [8], and the latest in standardized TCP improvements (Selective Acknowledgments (SACK) [9] and large window enhancements [10]).

[1] http://www.mash.berkeley.edu/ns/

[2] http://http.cs.berkeley.edu/~bmah/Software/HttpModel/

A. Performance metrics

A number of metrics for quantifying fairness have been proposed but no single metric has common acceptance [5]. In this paper, we consider a "fair share per link" metric; i.e., if there are n flows through a bottleneck link, each flow has the right to $1/n$th of the capacity of that bottleneck link. Jain's metric of fairness [2] is applicable in this context. For n flows, with flow i receiving a fraction b_i on a given link, the fairness of the allocation is defined as:

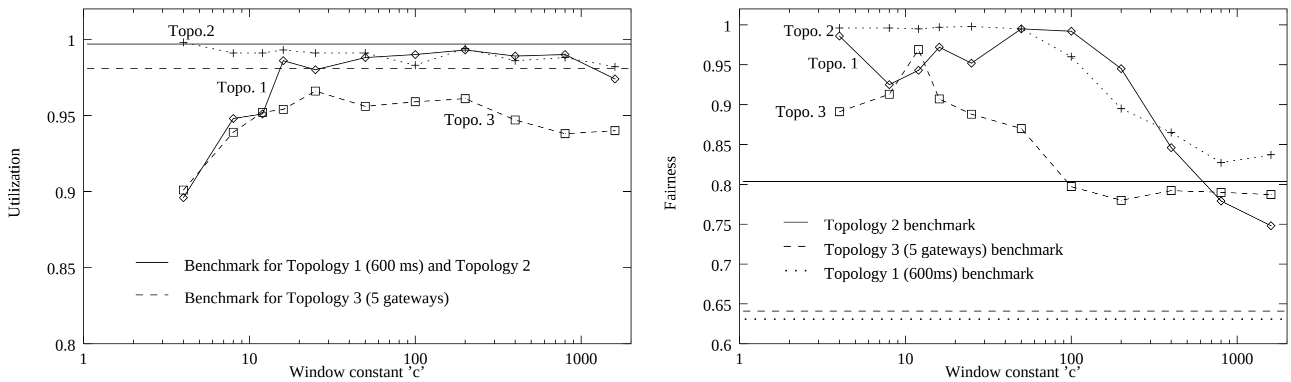
$$Fairness \equiv \frac{(\sum_{i=1}^{n} b_i)^2}{n * (\sum_{i=1}^{n} b_i^2)}.$$

This metric ranges continuously in value from $1/n$ to 1, with 1 corresponding to equal allocation for all users. Utilization is another important metric, since high fairness is of little use if the link capacity is grossly underutilized. Utilization is defined herein as the number of original bits (i.e., not counting retransmissions) successfully transferred over a link during some time interval divided by the product of link rate and that time interval; this is often called "goodput."

$$Utilization \equiv \frac{(segments\_acked) * (segment\_size)}{(link\_rate) * time}.$$

B. Topologies

We explored a number of test configurations that allowed us to isolate selected behavior of both the standard and our proposed window adjustment policies. We selected the topologies illustrated in Figure 1; similar test configurations have previously been used by the research community to study the effects of congestion. The first test configuration (Topology 1) was used to study the effects of two long duration connections, one with a short RTT (10ms) and one with a long RTT (100-600ms), sharing a single bottleneck link. The second test configuration (Topology 2) was used to examine the effects of many competing connections over a single bottleneck link. The six connections have RTTs of 10ms, 20ms, 100ms, 200ms, 300ms, and 600ms. The third configuration (Topology 3) was used to examine the effects of long RTT connections that must also traverse a number of network hops populated by short RTT connections. This topology is very similar to the one previously used by Floyd to study



(a) Utilization vs. window constant c (b) Fairness vs. window constant c

Fig. 2. Performance vs. window constant of TCP SACK with fine-grained RTT estimates over topologies with bottleneck RED queues.

the CR algorithm [5]; the number of congested gateways could vary between 1 and 10 (the figure illustrates 5 congested gateways).

C. Configuration details

We studied the performance of two different TCP variants: TCP "Reno," and TCP with Selective Acknowledgments [9], referred to as "SACK." We also examined two different queueing schemes: traditional "first-in, first-out" (FIFO) queueing, and Random Early Detection (RED) with packet discard.[3] In our simulations, data packet sizes were fixed at 1000 bytes, and the bottleneck link speed was 1.5 Mb/s. We examined a range of queue sizes from 4 to 50 packets, but in the data that follows, we concentrate on a RED queue size of 50 packets with a "minimum threshold" of 20 packets and a "maximum threshold" of 40 packets; all other RED parameters were set to the *ns* defaults. 20 packets in this case is approximately 100ms at our output line rate.

We also note the following simulation configuration details:

- In each case, the foreground TCP connections supported long-lived FTP connections that always had a packet to send. This models a large file transfer. The TCP sinks used delayed acknowledgments [11].
- In all of our simulations, we ran a number of simulated WWW clients and servers to provide a background traffic flow of approximately 20% of the bottleneck link in the direction we were measuring. The value of 20% was selected to provide a reasonable amount of random activity on the congested links while not dominating the congested link bandwidth.
- All simulations were run for 60 seconds, but the data from the first 20 seconds was not considered (transient removal).
- One potential limiting factor in TCP throughput is a small offered window by the receiver. To remove these effects from our data, we set the receive window for short RTT connections to 32 KB, and for long RTT connections (those with a RTT of at least 100ms) to at least three times the bandwidth-delay product of the connection.

D. Data analysis and presentation

TCP throughput in an environment containing random traffic can be quite variable, because small changes in initial conditions can cause wide variations in resulting behavior. Therefore, we computed the utilization and fairness of a particular configuration as follows. We first ran enough independent simulations such that the sample standard deviation of each connection's throughput was within 5% of its sample mean. We then used these sample means to compute the fairness and utilization of a given topology. We also established a set of benchmark performance results (using standard TCP congestion avoidance policies) for each of our test topologies, to compare against the results using the modified policies. We do not have the space to present all of our simulation data; instead, we focus on the most important data while leaving other points for discussion only. For example, we found that TCP SACK and RED queueing almost always outperformed TCP Reno and FIFO queueing; therefore, in the remainder of this paper we focus on TCP SACK and RED queueing.

III. Performance of the Constant-Rate policy

In this section, we describe the case in which each foreground and background TCP connection uses a Constant-Rate (CR) policy. In TCP implementations, an additive increase to the TCP variable *snd_cwnd* (send congestion window) of approximately one segment per RTT, assuming an acknowledgment (ACK) is received for each segment, is effected by executing the following pseudocode upon receipt of a new ACK:

```
snd_cwnd = snd_cwnd + 1/snd_cwnd.
```

In this manner, the TCP connection gradually adds to its congestion window at the rate of approximately

[3]RED queues operate by computing an exponentially weighted moving average of the queue size. When the average queue size is below some minimum threshold, the queue does not drop any packets. When the average queue size is between the minimum and maximum threshold, the queue probabilistically drops incoming packets according to an algorithm described in [8]. When the average queue size exceeds the maximum threshold, the queue drops every incoming packet. The instantaneous queue depth can exceed the maximum threshold if the average queue depth is below the maximum threshold.

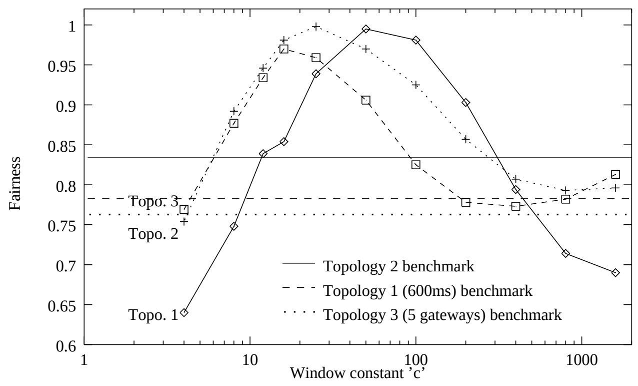


Fig. 3. The sensitivity of the proper selection of the constant c when the effects of a standard TCP connection are added to the performance shown in Figure 2.

one segment per RTT; this approach to building the congestion window reduces transmission burstiness [1]. To implement a CR policy, we can modify the window increase algorithm to account for the RTT bias:

```
snd_cwnd = snd_cwnd + (c*rtt*rtt)/snd_cwnd,
```

where c is the constant that controls the rate. This policy causes an additive increase in the throughput rate that is the same for all connections. After initial experiments, we observed that the second term of the above equation could lead to very bursty send patterns, which led to increased losses. For example, if the RTT is large and the value of *snd_cwnd* is small, each ACK can trigger the transmission of several segments. In order to avoid this behavior, we bounded the increase per ACK by 1 segment; i.e.:

```
snd_cwnd = snd_cwnd +
min((c*rtt*rtt)/snd_cwnd, 1 segment).
```

With this constraint on the sender's behavior, the TCP connection is never more bursty than a TCP connection in slow start. Another approach, with which we did not experiment, would be to smooth the sending of several segments across a longer time period.

One question previously raised by Floyd is how to pick the proper value for the constant c. One way to think of the value of c for CR connections is how the aggressiveness of the CR connection would compare to that of a standard TCP connection with a certain RTT. For example, if $c = 100$, the value of the RTT that makes the numerator equal to 1 in our pseudocode above is 100 ms. Therefore, an environment in which $c = 100$ would have connections that were about as aggressive as normal TCP connections with 100 ms RTTs. We chose to experiment with a range of values, between $c = 4$ (as aggressive as standard 500 ms connections) and $c = 1600$ (25 ms).

In addition to varying the constant c, we experimented with several other variations in an effort to identify which types of environments were suitable for the CR policy:

- TCP Reno vs. TCP SACK,
- RED vs. FIFO gateways,
- bottleneck queue lengths (or RED maximum queue thresholds) from 4 to 50 packets, and
- TCP RTT timer granularity of 500 ms (standard in many TCP implementations) vs. 10 ms.

As mentioned above, Topology 3 conforms closely to one with which Floyd experimented [5], and we were able, when using a similar value for the CR constant ($c = 4$), to confirm their results that CR can substantially improve the fairness of connections traversing multiple gateways when all connections use a CR policy. However, we also observed the following general trends in our data:

i) Deep RED queues appear to be a prerequisite for good performance of the CR policy. Performance when FIFO queues or short queues were used was very inconsistent, in the sense that there was often no value of c that simultaneously yielded high fairness and high utilization. Moreover, we could not determine strong correlations between the value of c and the fairness and utilization metrics we were using; i.e., the performance was highly sensitive to the particular simulation topology.

ii) TCP SACK and fine-grained RTT timers were the next most important indicators of good CR performance. The use of SACK helps TCP recover from losses more quickly, which leads to improved and more consistent performance. Also, many existing TCP implementations use a coarse estimate of the RTT, which impairs the ability of our modified congestion avoidance algorithm to determine the true RTT of the connection.

Figure 2 plots the utilization and fairness performance of TCP SACK over bottleneck RED queues when all connections, including background HTTP traffic, use the same CR policy, constant c, and RTT timer granularity of 10 ms. Topology 1 corresponds to the case in which the long RTT connection has a round trip propagation delay of 600 ms, while Topology 3 in this case corresponds to the topology with 5 congested gateways (the trace is taken from the first congested gateway). For comparison, we also plot as horizontal lines the utilization and fairness achieved when all TCP connections use the standard algorithm (i.e., benchmarks). The data indicates that the fairness can be substantially improved if all connections adopt the CR policy. However, the utilization suffered for small c when there were only two foreground connections in the topology (Topologies 1 and 3). When statistical multiplexing was in full effect (Topology 2), both fairness and utilization were near optimal for small values of c. Additionally, in Topology 3, although the fairness improved substantially, equal allocations were not obtained by the CR policy because the long RTT connection is also traversing multiple congested gateways.

In Figure 2, as c became larger, connections became more aggressive, to the point that the bound in our policy of adding 1 segment per ACK was in effect nearly all of the time. Consequently, since the CR policy was no longer being applied, the unfairness reappeared. In general, we observed that the fairness properties were best when the value of c was below 100. However, if too few connections are using the link, such as in Topologies 1 and 3, such a small value of c can lead to lower utilization. Because it is difficult in practice for a given connection to determine the number and type of connections against which it is competing, we conclude the following negative result: a good choice of the constant c cannot be determined with high confidence on an operational basis.

Not only does the CR policy appear difficult to manage in a distributed network, we also found it susceptible to the presence of TCP connections operating under the standard policy. For example, Figure 3 illustrates the fairness performance when a single additional connection using the standard window increase policy was introduced into each of the topologies (and also included in the fairness computation). Although this additional connection slightly improved link utilization, much of the fairness improvement due to CR was lost when this competing connection was introduced, as it not only used a disproportionate share of the bandwidth itself but also acted as a "trailblazer," improving the performance of short RTT connections that were using the CR policy by a disproportionate amount. We also observed similar performance degradation if no extra file transfers were introduced, but instead the HTTP background traffic (20% of the bottleneck link rate, on average) used the standard policy. Similar effects (passive connections competing with more aggressive connections using the standard congestion avoidance algorithm) are also responsible for the poor performance of TCP "Vegas" in a long RTT environment characterized by a mix of heterogeneous TCP implementations [12].

IV. Selectively modifying the additive increase policy

We next investigated whether the throughput of an individual long RTT connection could be improved by modifying the additive increase policy of only the long RTT connection. We were interested in two questions:

- Can an individual connection improve its own throughput by becoming more aggressive during additive increase?
- If so, how does the individual connection's more aggressive behavior affect the performance of other (unmodified) connections using the same path?

To study the first question, we experimented with an "increase-by-K" (IBK) policy rather than the standard "increase-by-one," again limited by a maximum increase of one segment per ACK. In other words, we used the following pseudocode in our implementation:

```
snd_cwnd = snd_cwnd +
min((K/snd_cwnd),1 segment).
```

For example, by setting $K = 2$, we built the window by roughly 2 segments per RTT.[4] Again, the increase is bounded by 1 segment per ACK, but this is the general trend.

Figure 4 illustrates fairness results, calculated over the entire simulated network, for the case in which only one connection in the simulated topology used the IBK policy. In particular, we enabled the IBK policy on the longest RTT connections in each of the three topologies, and then repeated the experiment by enabling the IBK policy on only the 300 ms connection in Topology 2. In the graph, the value of $K = 1$ (left-most data points) corresponds to the normal (benchmark) case.

[4]If delayed acknowledgments are being used, this is akin to correcting the window growth penalty that is due to delayed acknowledgments.

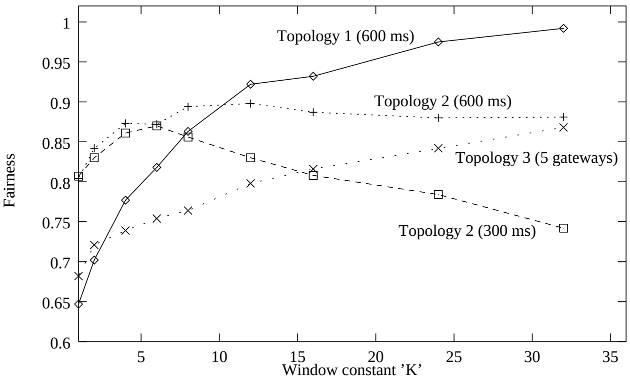


Fig. 4. Improvement in fairness vs. window constant due to the IBK policy.

We observed that the long RTT connection was able to steadily improve its performance over that of the benchmark case by increasing K across the range of values we considered. This resulted in improved fairness in all topologies for small values of K. For larger values, even though the throughput of the long RTT connection continued to improve, fairness actually decreased in some topologies as the more aggressive connection began to take more bandwidth than its fair share.

In Figure 4 we plotted the performance of TCP SACK over RED queues, and found that there was no limit to the improvement that a more aggressive connection could obtain for itself. We repeated the experiment for TCP Reno over FIFO queues, and found that connections could increase their own performance, independent of the topology, by using a value of K of up to 4 or so. However, for higher values of K, performance degraded, because the sending behavior became too bursty for the FIFO queues to successfully absorb.

Because the performance improvements result from increasing TCP's aggressiveness, we should be concerned that this can have a negative impact on other peer connections. Remarkably, we found that, in every case we examined, the average fairness index *always* improved, and the average utilization held relatively constant, when the more aggressive connection used a modest value of K (less than 8 or so). This improvement occurred regardless of whether TCP SACK or Reno was used, or whether FIFO or RED queues were present. In fact, the majority of the redistributed bandwidth came from connections that were already using more than their fair share. The effect on other connections was similar to what they would have experienced had the long RTT connection actually been a connection with a somewhat shorter RTT.

For example, Table 2 provides an example of the magnitude of the performance gains achievable. For the value $K = 4$, we tabulate the utilization, fairness, and throughput of the four longest RTT connections in Topology 2. In the first column are results from when each connection used the standard policy, the second column shows the results from when only the 300 ms connection used an IBK policy, and the third column is from when only the 600 ms connection used an IBK

TABLE II

Effect of the IBK policy on throughput (Topology 2, $K = 4$). 99% confidence intervals are shown in parentheses.

	benchmark	300ms	600ms
Utilization	0.99	0.99	0.99
Fairness	0.807	0.861	0.873
	Throughput (kb/s)		
600ms	**68.1(0.9)**	59.6(0.6)	**138.7(2.0)**
300ms	**116.2(1.1)**	**240.8(2.8)**	109.1(1.1)
200ms	156.6(1.7)	139.8(1.5)	151.6(1.1)
100ms	217.1(2.1)	188.1(1.9)	207.0(1.8)

policy. This table illustrates that the more aggressive connections did not seriously harm the throughput of the peer connections. The throughput gains (highlighted in bold font) are substantial; in many cases, throughput increases (and hence reductions in user-perceived latency) exceeded 100% when RED queues were used, and 50% when FIFO queues were traversed.

We next investigated whether the performance gains were sustainable when multiple connections become more aggressive by examining this case with Topology 2. We found that the aggressive connections were able to simultaneously improve their own performance, although their relative performance gains were not what they would have achieved had they been the only aggressive connection. Of course, if every connection adopted an IBK policy, the fairness situation would be back to the standard case, so there must be some RTT threshold beyond which connections can become more aggressive if such a policy is to work in practice. Finally, we experimented with the case in which congestion was induced in both directions of data transfer. This had the effect of disrupting the ACK stream to some extent, but did not significantly affect our main results.

V. Conclusion

We have investigated simple changes to TCP's congestion avoidance algorithm in an effort to improve its fairness properties. While we found that the Constant-Rate (CR) policy could improve fairness dramatically, we faced two practical difficulties that would likely prevent universal deployment of this scheme in its current form: i) the proper selection of a constant is dependent upon the network topology and the number of peer connections and is therefore difficult to determine in a distributed manner, and ii) the fairness benefits of the CR policy can be confounded by competing connections using standard congestion avoidance, thereby making it disadvantageous to deploy CR in an existing heterogeneous environment. However, when we instead made only certain long RTT connections slightly more aggressive, we were always able to improve network fairness while keeping bottleneck link utilization relatively constant by using an increase-by-K (IBK) policy. Interestingly, the effects on other unmodified connections that were sharing the bottleneck link were similar to what they would have experienced had the modified connection actually been a connection with a shorter RTT.

Our results indicate that it may be beneficial for long RTT connections to become slightly more aggressive during the additive increase phase of congestion avoidance. While we do not advocate a particular policy at this time, the IBK policy for small values of K (such as 2 or 4) may significantly improve the throughput while not significantly impacting other flows. Such a policy could be invoked in practice when the TCP implementation detects that the connection has an RTT above a certain threshold (for example, connections traversing a satellite link have a much larger RTT than terrestrial connections). A TCP receiver could even induce the sender into an IBK policy by acknowledging data in smaller chunks. Determining appropriate values for K as a function of RTT, as well as determining the accuracy and resolution required of TCP's RTT estimates, will be the focus of future work.

Acknowledgments

We thank Sally Floyd for sharing some unpublished research results with us and for commenting on an earlier draft of this manuscript. We also thank Bruce Mah for providing his HTTP traffic traces, and Giao Nguyen for assistance with the *ns* simulator. Finally, we thank Hari Balakrishnan and Venkata Padmanabhan for reviewing early drafts. This work was supported by DARPA contract DAAB07-95-C-D154, by the State of California under the MICRO program, and by Hughes Aircraft Corporation, Metricom, Fuji Xerox, Daimler-Benz, and IBM.

References

[1] V. Jacobson. Congestion Avoidance and Control. *Proceedings of ACM SIGCOMM '88 Conference*, pages 314–329, 1988.

[2] D. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems*, 17:1–14, 1989.

[3] E. Hashem. Analysis of Random Drop for Gateway Congestion Control. *Report LCS TR-465, Laboratory for Computer Science, MIT, Cambridge, MA*, 1989.

[4] A. Mankin. Random Drop Congestion Control. *Proceedings of ACM SIGCOMM '90 Conference*, pages 1–7, 1990.

[5] S. Floyd. Connections with Multiple Congested Gateways in Packet-Switched Networks, Part 1: One-way Traffic. *ACM Computer Communications Review*, 21(5):30–47, October 1991.

[6] T. Lakshman and U. Madhow. The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss. *IEEE/ACM Transactions on Networking*, 5(3):336–350, June 1997.

[7] B. Suter, T. Lakshman, D. Stiliadis, and A. Choudhury. Design Considerations for Supporting TCP with Per-flow Queueing. *Proceedings of IEEE Infocom '98 Conference*, pages 299–306, 1998.

[8] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 3(3):115–156, September 1993.

[9] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. *Internet RFC 2018*, 1996.

[10] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. *Internet RFC 1323*, 1992.

[11] W. Stevens. *TCP/IP Illustrated, Vol.1.* Addison Wesley, 1994.

[12] Y. Zhang, E. Yan, and S. Dao. A Measurement of TCP over Long-Delay Network. *Proceedings of 6th Int'l Conference on Telecommunication Systems, Modelling, and Analysis*, Nashville TN, March 1998.