TCP-Africa: An Adaptive and Fair Rapid Increase Rule for Scalable TCP

Ryan King, Richard Baraniuk, Rudolf Riedi
Departments of Electrical and Computer Engineering and of Statistics
Rice University, Houston Texas 77005-1892
{ryanking,richb,riedi}@rice.edu, www.spin.rice.edu

***Abstract*— High capacity data transfers over the Internet routinely fail to meet end-to-end performance expectations. The default transport control protocol for best effort data traffic is currently TCP, which does not scale well to 100Mbps and higher networks over long distances. In congestion avoidance TCP is not swift enough to fully utilize resources over paths with a high delay bandwidth product. First attempts to alleviate this problem by equipping TCP with increased aggressiveness have shown the disadvantage of poor fairness with the ubiquitous standard TCP-Reno, or in some cases, even among two connections running over the same path. We propose a new delay sensitive-congestion avoidance mode (TCP-Africa) that allows for scalable, aggressive behavior in large underutilized links, yet falls back to the more conservative TCP-Reno algorithm once links become well utilized and congestion is imminent. Through ns2 simulations we argue for the safety, efficiency, and fairness of TCP-Africa.**

***Index Terms*— Scalable TCP, high-bandwidth-delay products, TCP fairness**

I. Introduction

TCP-Reno has been used with great success in the Internet in general, since its proposal in 1988 [1]. The extent to which Reno has scaled is rather astonishing, but it is reaching its limit in the modern Internet, with its ever increasing bandwidth [2]. At the time when TCP was proposed, large Internet core links were on the order of 56Kbps, the speed of modern telephony modems [1].

The limitations of TCP-Reno are apparent in networks with *large bandwidth-delay-products*, which are becoming increasingly common in the modern Internet. For example, supercomputer grids, high energy physics, and large biological simulations require efficient use of very high bandwidth Internet links, often with the need to transfer data between different continents.

Support comes in part from NSF ANI-00099148, DARPA/AFRL F30602–00–2–0557, DOE Grant No. DE-FC02-01ER25462 and Texas Instruments.

The desired congestion window used by TCP-Reno is roughly equal to the bandwidth delay product of the connection. For high bandwidth-delay-product links, this desired congestion window is quite high, as high as 80,000 packets for a 10 Gbps link with a 100 ms RTT, or round trip time! TCP-Reno's combination of a slow linear increase and a fast multiplicative decrease requires an unreasonable amount of time for this desired window to be regained after a loss. Indeed, as pointed out by Floyd et al. [3], in such situations it can take a TCP-Reno stream over one hour to recover from a single congestion event. Under the random packet loss model, TCP-Reno can require an unreasonably low packet drop probability for these high bandwidth-delay-product links. Indeed, Reno's throughput scales with the inverse square root of the loss probability [4].

From recent approaches one is tempted to conclude that correcting TCP's deficiencies with a purely loss based protocol leads to super-aggressive protocols, which raises fairness and safety concerns. By contrast, delay based protocols, which make use of the wealth of information about the network state that is provided by packet delays, can indeed achieve excellent steady state performance, as well as minimal self-induced packet losses. In the presence of flows on shared links which are non-responsive to delay, such as TCP-Reno, however, delay sensitive protocols are often in the disadvantage of reacting earlier than the non-responsive ones. As a consequence, these delay-responsive protocols do not cause enough packet drops to the non-delay-responsive protocols to force them to maintain only their fair share [5].

In this paper, we propose *TCP-Africa*, a new delay sensitive two-mode congestion avoidance rule for TCP that promises excellent utilization, efficiency, and acquisition of available bandwidth, with significantly improved safety, fairness, and RTT bias properties. This new protocol uses an aggressive, scalable window increase rule to allow quick utilization of available band-

width, but uses packet round trip time measurements to predict eminent congestion events. Once the link is sensed to be highly utilized, the protocol reacts by resorting to the conservative congestion avoidance technique of standard TCP.

This paper is organized as follows. In Section II we start by looking at the time between induced congestion events for several purely loss based protocols, as well as an approximation of our new delay-sensitive protocol. We consider the effects of periodic congestion events (rather than just packet loss probability) on TCP-Reno to be an important clue to the backwards compatibility of new high speed protocols. We make the distinction that a rapid return to high utilization after a packet loss is desirable, but that the rapid creation of a new congestion event is not. Next, we review the factors involved in the limitations of most delay based protocols, such as TCP-Vegas [6], and what those factors suggest with regards to a safe high speed protocol. In Section III, we provide a detailed description of the protocol. In Section IV, we present a series of ns2 experiments, looking to address both the protocol's potential to utilize networks, as well as to assess its fairness and safety properties. Finally, in Section V, we address some of the concerns regarding the deployability of the protocol, including processing overhead, sensitivity to reverse path congestion, and assurances regarding congestion collapse. We close with conclusions and an outlook for future work.

II. Background: Existing protocols

Researchers have proposed several new high speed end-to-end protocols in the last few years. By and large, these protocols fall into two groups: purely loss based algorithms, such as STCP [7], HSTCP [3], and BIC-TCP [8], and delay based algorithms, such as FAST TCP [9]. XTCP [10], though it shows excellent performance, is not included in this analysis since we are considering only end-to-end congestion control.

A. Considerations for a high speed protocol

When one goes about designing a high speed transfer protocol, the most straightforward approach is to modify TCP's increase and decrease parameters to adjust its response function to provide a more desirable performance. However, there are several considerations that need to be taken into account when designing such a protocol.

In this paper, we decide to concentrate on the following four issues.

1) [Throughput] First, of course, one aims at improving the capability to utilize the network resources. This is intimately related to a better adapted increase and decrease behavior necessary and an improved loss rate function.
2) [Peer fairness] One must also consider fairness with other streams running the same protocol, which may have different round trip times [2]. A good protocol should be able to quickly converge to equilibrium with other flows of the same type.
3) [TCP-fairness] A protocol should also be fair with the older TCP-Reno standard. Indeed, any new protocol to be deployed which dominates either the myriad of existing Reno flows will not find acceptance due to the Internet's philosophy of providing best effort for all.
4) [Congestion collapse] We need also assurances that the protocol will not push networks into a state of congestion collapse. This is of particular concern for so called high speed protocols, since they are by nature more aggressive than TCP-Reno.

B. Aggressive loss based protocols

Let us start by discussing some immediate approaches to respond to the first requirement above. The main drawback of TCP-Reno over large delay-bandwidth-products lies in its linear increase mode in congestion avoidance. TCP-Reno reacts to packet loss by decreasing its congestion window by half, and then increasing its congestion window by one packet per round trip time while in congestion avoidance mode [11]. Operating around an ideal congestion window of, say, 1000 packets, a drop leads to a reduction of the window by 500 and requires roughly 500 round trip times to regain the size before the drop.

In order to keep most of the desirable qualities of TCP, the most immediate and simple change lies in a modification of the window update rules.

Simply tuning the window update rules of TCP-Reno can directly improve the protocols ability to utilize high speed links, but may impair its fairness properties. As pointed out in [8], protocols in the same class as HSTCP and STCP (protocols that make modifications to the increase and decrease parameters), can have undesirable RTT fairness properties, if they simply increase more aggressively when operating with larger windows.

A MIMD protocol, such as STCP [7] uses multiplicative increase and multiplicative decrease window adjustment rules. For every received packet, STCP increases its congestion window by 0.01 packets, and on a packet drop, the window is reduced to 0.875 times its current window. As a result, the recovery time from a drop is scale invariant, always requiring a constant

number of round trip times. MIMD is equivalent to an additive increase scheme where the increase step size grows proportionally to the congestion window size.

The TCP variant HSTCP [3] takes a similar approach to STCP, though it scales its drop parameter from 50% at low window sizes to 90% at higher windows. HSTCP then set's its increase parameter as necessary to achieve its desired packet loss response. The end result is that HSTCP's increase rate grows slightly slower than that of STCP, but still very rapidly as compared to TCP-Reno.

A more aggressive TCP version leads almost invariably to reduced fairness, lest care is taken. The authors of [8] point out that both HSTCP and STCP have undesirable fairness properties when flows with different round trip times are competing over a shared link. HSTCP has slightly improved RTT bias performance as compared to STCP [8]. Both HSTCP and STCP share a similar spirit with regards to their approach to adjusting for TCP-Reno's shortcomings, and can be considered to be members of the same class of high speed loss based protocols.

BIC-TCP [8], according to its authors, has desirable RTT bias properties. BIC-TCP makes use of a binary increase scheme to quickly approach an estimated safe window, then slowly increase above that window. The author's claim that its Reno-fairness properties are comparable to that of STCP, however, is not particularly strong; STCP is the most aggressive of the current well known TCP proposals. Nonetheless, BIC-TCP is still relatively new, and merits further investigation and analysis.

To address finally the risk of congestion collapse, let us mention that a common factor in each of these loss based protocols is that they increase their congestion windows by more than Reno's amount of 1 packet per RTT. While it is unlikely that any of the proposed congestion control protocols will cause another Internet congestion collapse, this more aggressive increase rate and drop behavior can cause increased burden on intermediate router buffers. With drop tail queues, the number of packets dropped per congestion event tends to be related to the number of additional packets introduced into the network in the last round trip time, that is, to the protocol's increase parameter. While it is possible that this is the price that one must pay for scalable congestion control, it remains a desirable goal to avoid such problems. With our window increase technique we have found evidence that it is indeed feasible to mitigate most of these problems.

Key to our discussion is a crucial assumption behind much of the analysis used to design and justify these protocols based on high speed loss, namely a simple independent random packet drop model. According to this assumption, losses occur at a certain rate independently from each other and from the protocol. We deem this assumption in many ways too simplistic and restrictive as we now argue. Indeed, packet losses in real networks tend to have periodic components and to involve correlated losses, meaning that many packets may be lost in the same event. Consequently, while informative, the drop rate response function is not sufficient to characterize the performance of a protocol. This is particularly true for any protocol that makes use of delay information, but must apply in general since it is insufficient to characterize a network condition by its packet loss rate solely.

In addition, the choice of protocols affects the loss rate for a link. Because of their more aggressive behavior, the above mentioned high-speed loss-based protocols induce congestion events at a much higher frequency than induced by TCP-Reno. In fact, due to STCP's choice of multiplicative increase, STCP must in steady state induce congestion events roughly every 13.4 round trip times, regardless of the link speed. HSTCP induces packet losses at a slower rate than STCP, but still much faster than TCP-Reno.

C. Induced congestion event frequency

To gain insight into the significance of frequent congestion events and exactly how often they occur for high speed protocols, we make a simple analysis looking at the cyclic period of induced congestion events for these protocols, for a given capacity link.

First, we make the assumption that there is some number $W_{\max}$ that is the maximum congestion window the flow can maintain over a path without a inducing a congestion event. For a given link, $W_{\max}$ can be approximated as $W_{\max} = C \times RTT/L + B$, where C is the link capacity in bits per second, RTT is the round trip time of the flow, L is the packet size in bits, and B is the maximum queue length of the bottleneck link. Since buffers are usually chosen to be proportional to a link's delay-bandwidth-product, we would expect $W_{\max}$ to scale linearly with both RTT and capacity. All that is needed for our following analysis, is that upon the congestion window reaching some $W_{\max}$, a self induced packet loss will occur. Finally, we make the assumption that the recovery mechanisms of the transport protocol will allow it to reduce its congestion window only once per loss event, regardless of the number of consecutive packets lost.

We begin by looking at the frequency of self induced congestion events for AIMD (additive increase, multiplicative decrease) protocols, such as TCP-Reno. The

relevant parameters for an AIMD protocol are a, the increase step size and b, the multiplicative decrease factor. For a general AIMD protocol, the induced congestion event period is

$$T = \frac{(1-b)W_{\max}}{a}. \quad (1)$$

For Reno, $a = 1$ and $b = 0.5$. Thus, we find, for TCP Reno, the induced congestion event period is

$$T = \frac{W_{\max}}{2}. \quad (2)$$

.

Multiplicative Increase Multiplicative Decrease (MIMD) protocols, which use a multiplicative increase parameter α and a multiplicative decrease parameter β, have a constant steady state induced congestion period that obeys the formula:

$$(\beta W_{\max})(1+\alpha)^T = W_{\max}. \quad (3)$$

which gives us:

$$T = \frac{-\log(\beta)}{\log(1+\alpha)}. \quad (4)$$

For STCP [7], $\beta = .875$ and $\alpha = 0.01$, thus giving us $T = 13.4$ round trip times between congestion events.

Unlike STCP, the time between congestion events for HSTCP grows slowly as $W_{\max}$ increases. The exact equations for HSTCP's growth curves do not lend themselves well to simple calculations such as this, but the results are easy to determine numerically. For $W_{\max}$ less than HSTCP's `low_window_` parameter of 38, HSTCP acts like normal TCP. However, above `low_window_`, HSTCP's congestion event frequency grows slowly, reaching roughly 120 round trip times between congestion events when $W_{\max}$ is around $83,000$. This is still extremely often when compared to what TCP-Reno would induce on these links.

We observe that it is this very rapid return to the maximum window that gives HSTCP and STCP their excellent scalability. We also note, however, that this scalability comes at the cost of very frequent self induced congestion events. In addition to the additional overhead and work involved for a protocol that must retransmit large numbers of packets, frequent congestion events can also have a negative impact on Reno streams that might be sharing the same bottleneck connection as the high speed flow. Indeed, from the AIMD equation above, we

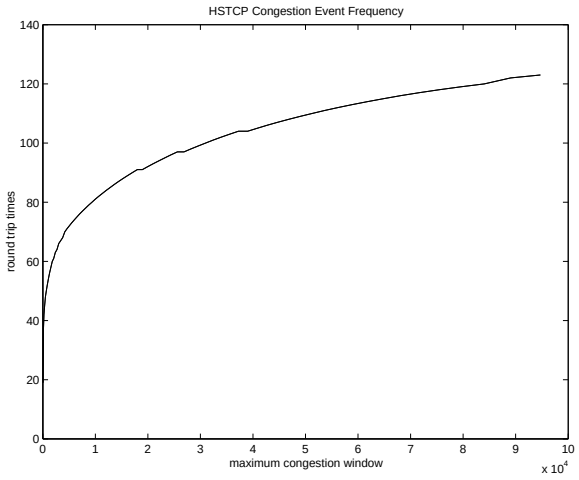

Fig. 1. Congestion Event Frequency for HSTCP

can calculate that for a given congestion event frequency, TCP-Reno will achieve a maximum window of

$$W_{\max} = \frac{Ta}{1-b}. \quad (5)$$

While losses in real networks are not completely synchronized, the implications of such frequent congestion events are worrisome, and raise possible fairness concerns. We maintain that while it is good for protocols to quickly reach nearly full capacity after a congestion event, actually reaching that full capacity will also quickly induce the next congestion event.

To illustrate this point, let us consider a hypothetical MIMD protocol that has some unspecified mechanism to detect when the network has reached a certain utilization, denoted G. G would ideally have a value in the range of 0.90 to 0.95. When the utilization surpasses G, the MIMD protocol then switches to a linear increase mode, increasing at one packet per round trip time. Thus, this hypothetical protocol would be able to return to a high level of utilization in constant time, but its period between self induced congestion events will still grow linearly with the window size, just as in TCP-Reno.

While actually detecting the utilization level of a link in practice is difficult for an end-to-end protocol, we find that we are able to achieve a similar result through the use of a delay metric, as described in section III.

D. TCP-Vegas-like protocols

Another significant group of high speed protocols are the Vegas-like delay based protocols, such as FAST TCP. These delay based protocols have many desirable properties: they are unlikely to cause significant queuing delay, can quickly converge to equilibrium, and they can

run in steady state without causing packet drops [12]. They also, however, can have the disadvantage of not being able to compete with greedy protocols such as Reno on congested links.

To understand our suggested solution for scalable TCP, it is instructive to recall the scenario behind TCP Vegas's deficiency when competing with greedy protocols such as TCP-Reno [5].

Imagine two streams starting up at the same time, one TCP-Reno, the other TCP Vegas, with no other traffic on the link. Every round trip time, each flow will increase its window by one packet. Thus, they will share the same rate, until the link approaches full utilization. Then, the delay on the link will increase, and the delay based protocol will cease to increase its window. The greedy TCP-Reno stream on the other hand will continue to increase its rate, injecting an additional packet into the network every round trip time. This will cause the delay on the link to increase further. The delay based protocol, since it tries to maintain only a small number of packets in the queue, will actually decrease its window in response to the aggressiveness of the TCP-Reno flow. This will continue until a packet is lost; the TCP-Reno stream will eventually overfill the queue.

Another way to picture the vulnerability of most delay based protocols is to imagine a protocol that tries to measure the available bandwidth of a link, and adjusts its rate such that there was always some small fixed N percent of the bandwidth free. When the link is underutilized, this flow would achieve excellent performance, however, in a link that is highly utilized (read: less than $N\%$ free bandwidth) by traditional packet loss congestion controlled flows, this available bandwidth sender would be completely starved of its fair share. It would decrease its sending rate repeatedly, but since there would always be less than $N\%$ bandwidth free, the protocol would end up sending at its minimum possible rate. Vegas-like delay based protocols end up in pitfalls similar to the above scenario, though they use estimated queue length rather than available bandwidth as a metric.

The Vegas-like protocols tend to avoid inducing packet drops, and as such, are unable to cause greedy flows to back off. This deficiency is one of the primary reasons preventing the widespread adoption of TCP-Vegas, despite its other benefits. FAST TCP is still theoretically vulnerable to this problem, as is virtually any protocol that decreases its congestion window based on delay information. Choosing a large value for the α parameter of these delay based protocols can lead to an improved ability to compete with Reno-like flows, but the correct choice of α will depend on the number of competing flows and the buffer capacity of the link. Since these parameters are inconsistent and unknown in real networks, choosing α can be very difficult. Furthermore, a choice of too large an α value can break the equilibrium behavior of these delay based protocols.

III. TCP-Africa

A. Design considerations

In this section we outline the steps leading to our proposal for a new, safe, high speed congestion avoidance mechanism for TCP. Let us start with a few observations.

It is interesting to note that the increase rate for HSTCP and STCP grows as the window grows. As a result, these protocols are in fact most aggressive just at the moment where they are sending at maximum capacity. Intuitively, this is the time when a protocol should be the least aggressive.

The excellent steady state behavior of the delay based protocols, on the other hand, is due to the fact that once the queue begins to fill, the delay will increase providing the protocol with early feedback. The delay based protocols are thus the least aggressive when sending at or near the capacity of the link. Let us now discuss how to use this early information more effectively and competitively than it is used by TCP Vegas, particularly for high bandwidth delay product links.

As a first step, we recognize the value of the Vegas congestion indicator but ignore its window adaptation rules as not competitive with Reno. Moreover, since no competitive protocol should be forced to back-off and reduce its window before Reno does, we find that the best way to use the Vegas-indicator is via a more aggressive increase *before* congestion. Intuitively, before congestion occurs, loss based schemes such as Reno should not see a performance penalty against protocols with more aggressive increase.

Second, to keep Reno competitive with such an enhanced protocol the latter should fall back to Reno strategies at the *onset* of congestion and well before the queue is filled.

These observations now lead us to our proposal for a safe high speed congestion control protocol: *TCP-Africa, an Adaptive and Fair Rapid Increase Congestion Avoidance mechanism for TCP*. TCP-Africa is a hybrid protocol that uses a delay metric to determine whether the bottleneck link is congested or not; in the absence of congestion, the *fast mode*, it uses an aggressive, scalable congestion avoidance rule. In the presence of congestion, the *slow mode*, it switches to the more conservative Reno congestion avoidance rule.

Thus, the protocol can exhibit scalable behavior without causing increased burdens on router buffers and other competing flows.

Under favorable delay conditions, which correspond to the presence of free available bandwidth, the protocol increases its window in an aggressive, scalable manner. Since our guidelines above give significant leeway in the exact increase curve used while the protocol is in "fast" mode, we have decided to use HSTCP's [3] increase and decrease parameters as TCP-Africa's scalable "fast" mode in our analysis and experiments to date. This choice was due in part to HSTCP's Reno compatibility at low congestion window sizes. We note, however, that one could just as easily implement a multiplicative increase technique as the "fast" mode for TCP-Africa.

TCP-Africa, in its scalable "fast" mode, quickly grabs available bandwidth. Once the delay begins to increase, and the protocol senses that the amount of available bandwidth is becoming small, the protocol switches to a conservative, linear increase mode of one additional packet per round trip time. The motivation behind such a slow growth mode for a high speed protocol is that once a flow is within a reasonable percent of its maximum bandwidth, it no longer has any reason to be growing at an increased rate. Indeed, as our earlier analysis from section II-C protocol shows, switching to a one packet per RTT rate can greatly decrease the frequency of self inflicted packet loss. Thus, TCP-Africa can be quick to utilize available bandwidth, but slow when it comes to inducing the next congestion event.

In our ns2 experiments to date, we find that the delay metric enables the protocol to run indeed in the scalable, aggressive "fast" mode in underutilized networks, but to act in a much more conservative manner in networks that are being sufficiently used by less aggressive protocols such as TCP-Reno, and where aggressive window increase behavior could cause performance degradation for other users.

B. Detailed description of TCP-Africa

An exponentially smoothed high accuracy round trip time estimate, $aRTT$ is kept by the flow. This delay information, along with the minimum delay observed on the path $minRTT$, is used to estimate the queuing delay on the link. W denotes the current congestion window of the flow.

This delay metric, then, is used to enable or disable an aggressive window increase mode. We note that the conditions in which a delay based protocol would decrease its window are the same conditions where aggressive window increase rules are not desirable.

There are several possible choices of a delay metric for this protocol, we have settled for one based on the TCP Vegas algorithm. TCP-Africa detects congestion using the following metric:

$$\frac{W(aRTT - minRTT)}{aRTT} \gtrless \alpha. \tag{6}$$

The quantity $(aRTT - minRTT)$ gives us an estimate of the queuing delay of the network. Since the overall round trip time is $minRTT + (aRTT - minRTT)$, the quantity $(aRTT - minRTT)/aRTT$ is the proportion of the round trip time that is due to queuing delay rather than propagation delay. Since TCP maintains an average sending rate of $W/aRTT$ packets per second, by extension of Little's Formula, $W(aRTT - minRTT)/aRTT$ is an estimate of the number of packets that the protocol currently has in the queue. The α parameter is a constant, usually set as a real number larger than one. The choice of α determines how sensitive the protocol is to delay.

By tracking this metric, our protocol can detect when its packets are beginning to enqueue at the bottleneck link, and thus, determine an opportune time to switch into slow growth mode.

The congestion avoidance steps followed by the protocol are thus:

if $(W(aRTT - minRTT) < \alpha \times aRTT)\{$
$\quad W = W + fast\_increase(W)/W$
$\}$ else $\{$
$\quad W = W + 1/W$
$\}$

The function *fast_increase(W)* is specified by a set of modified increase rules for TCP. Currently, we are using the congestion avoidance rules specified by HSTCP to specify the increase curve when the delay condition is met. Alternatively, a multiplicative increase scheme could be used, such as $fast\_increase(W) = 0.01 \times W$.

Another way of looking at the delay metric is

$$(aRTT - minRTT) \gtrless \alpha \frac{aRTT}{W}. \tag{7}$$

Thus, we are comparing the queuing delay of the bottleneck link to our expected time between packet sends. This helps explain why the delay metric scales well with link capacities, since both sides of this equation scale as link speeds increase.

This metric can also be considered to be an estimate of the total number of packets in the queue, given the current queuing delay and a link speed estimated as our current sending rate, $W/aRTT$. This suggests that an alternate congestion metric might be to use other techniques, such as packet pair probing, to estimate the maximum link capacity, and then estimate of the total number of packets in the queue. Alternatively, an available bandwidth estimation technique such as such as Pathchirp [13] could be used in parallel with

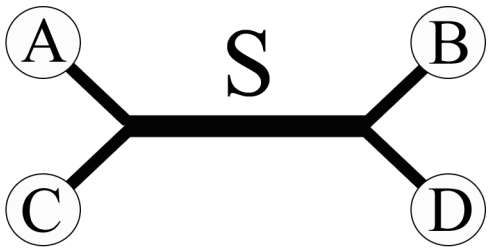

Fig. 2. Experiment Setup

the TCP data flow, to detect when it is appropriate to aggressively increase the congestion window. We are currently investigating these, and other possibilities.

One benefit of our current delay metric is that the switch-over point is always at the same rate for a given amount of queuing delay, independent of the round trip time of the flow. Thus, flows with a small round trip time do not gain a competitive advantage do to the delay metric. In fact, the dominating flows over a link, which have the smallest value for the expression $\frac{aRTT}{W}$ in 7, are the most sensitive to queuing delay, and thus, the first to switch into slow mode. This gives rise to improved RTT bias performance as compared to other high-speed loss-based TCP protocols. Indeed, this sometimes even leads to better than Reno performance, since the long RTT flow is using a scalable increase rule that allows faster than one packet per round trip time increase.

The choice of the α parameter can impact the performance of the protocol, but can never make it behave in a dangerous manner. In our experiments, we used an α value of 1.65, which provided good performance, but may not be optimal in all network conditions. We are currently investigating possible auto-tuning schemes for choosing alpha.

IV. Experimental Study

In this section, we perform an experimental study to compare the performances of TCP-Africa and HSTCP. Since TCP-Africa uses HSTCP's window adjustment rules as its high speed mode, this comparison reflects directly on the benefits of the presence of the two, delay sensitive modes of TCP-Africa.

The following series of experiments were performed using ns-2, version 2.27. The general setup for the following simulations is as follows:

The link capacities and delays were varied from experiment to experiment. A small amount of Poisson UDP traffic, roughly at 3% utilization, was added to the links to simulate a very lightly utilized link. All tests lasted 8 minutes. Since the router buffers were configured with drop tail queues, we found it necessary to use the ns parameter $overhead\_ = 8\times10^{-6}$ to mitigate the presence of phase effects, as was done in [8]. Phase effects in ns simulations are described in [14]. The α parameter of TCP-Africa was set to $\alpha = 1.65$ for these experiments. While this value seems to provide excellent behavior, more investigation into optimally choosing the α parameter needs to be done.

A. Safety

The first scenario that was simulated for TCP-Africa is one that may be a common experience for many researchers. We use the term *safety* to indicate that we are investigating whether or not a protocol hampers the performance of other flows in network conditions where they would normally be able to achieve acceptable performance. The purpose of this simulation was to look at the scenario where there are two classes of end hosts, those that are using 1 Gbps access links, and using a high speed protocol, and ones that are using traditional TCP with 100 Mbps access links.

Following the aforementioned dumbbell configuration, the speed of the shared link was set to 622 Mbps, with a delay of 80 ms. The access links for node's A and B were set to 1 Gbps, while the access links for links C and D were set to 100 Mbps. All access links had a path delay of 1 ms.

As one can see in Figure 3, HSTCP has a significant effect on the throughput achieved by the TCP-Reno flow. TCP-Africa, however, has a minimal effect on the TCP-Reno flow, as seen in Figure 4. The ratio of HSTCP traffic to Reno traffic in this experiment was roughly 25:1 and over 2700 non slow-start related packets were lost at the bottleneck link. For the TCP-Africa experiment, the ratio of TCP-Africa traffic to TCP-Reno traffic was roughly 6:1, and only 47 non slow-start related packets were lost at the bottleneck link over the course of the 8 minute experiment. Indeed, the TCP-Reno stream lost more packets at its access link than it did at the bottleneck link. At the same time, the overall utilization of the network was slightly higher in the TCP-Africa experiment.

B. Fairness with TCP-Reno

The next experiment we look at is very similar to the previous safety experiment. However, in this example we alter the access links so that all flows have 1 Gbps

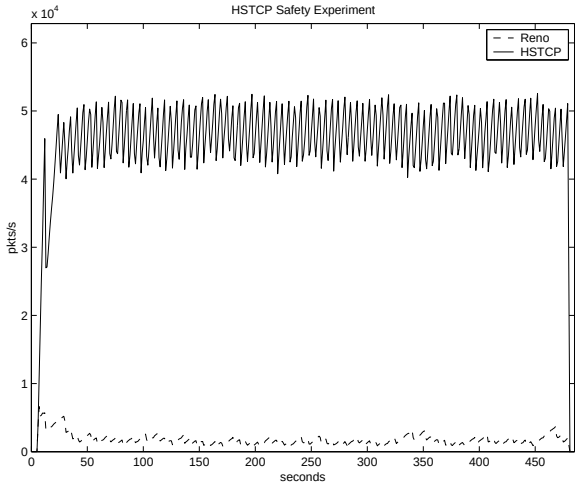

Fig. 3. Safety Experiment for HSTCP. The frequent congestion events induced by the HSTCP flow hamper the throughput of the TCP-Reno stream.

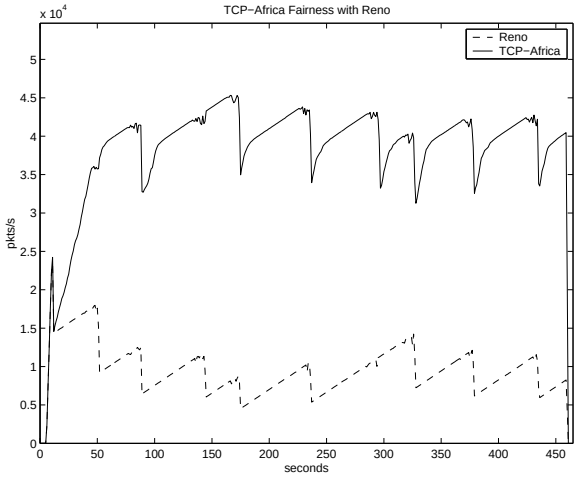

Fig. 5. Reno Fairness Experiment for TCP-Africa. Both TCP-Africa and TCP-Reno are bottlenecked at the common link. TCP-Africa's switch to slow growth mode is evident in each congestion event cycle.

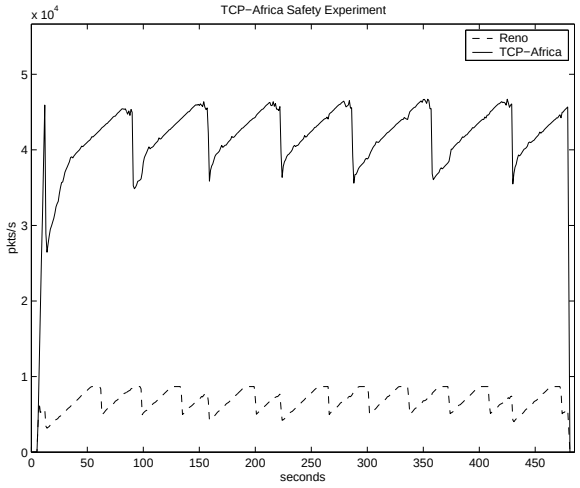

Fig. 4. Safety Experiment for TCP-Africa. TCP-Reno is effectively bottlenecked only at its access link. TCP-Africa's switch to slow growth mode is clearly visible.

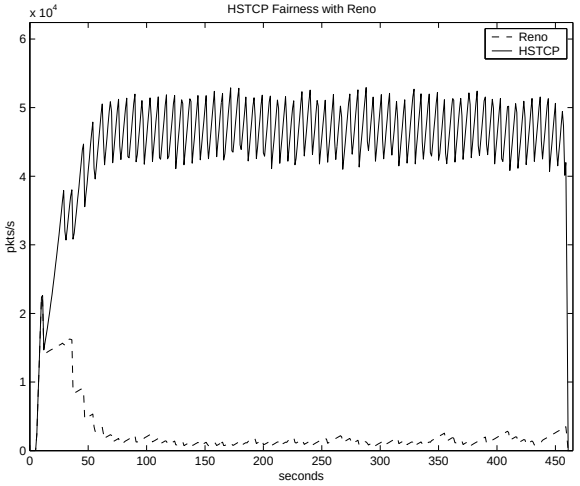

Fig. 6. Reno Fairness Experiment for HSTCP. As in figure 3, TCP-Reno is unable to achieve its fair share of the link bandwidth.

connections to the shared link. We thus expect to see exactly how much HSTCP and TCP-Africa impact the performance of a TCP-Reno flow.

In Figure 6, we see very similar results to what we found in the experiment in Figure 3, indeed since the TCP-Reno flow in that experiment was effectively bottlenecked at the shared link, we do not expect its performance against HSTCP to improve significantly by having its access link speed increased to one gigabit. Indeed most of the improvement is seen while the network is still stabilizing after slow start, before the HSTCP flow has pushed it back down. The ratio of HSTCP traffic to Reno traffic in this flow is roughly 17:1, and slightly over 2000 non slow start related packets were lost at the bottleneck link.

In the TCP-Africa safety experiment, Figure 4, we observed that the Reno flow was predominately bottlenecked at its access link. Thus, in this experiment, where both flows have the same access link speeds, we expect to see that the ratio of TCP-Africa traffic to TCP-Reno traffic improves. Indeed, the ratio between the two was nearly exactly $4:1$, and only 32 non slow-start related packets were lost on the link. In general, TCP-Africa sees an improvement in the number of packet losses both

TABLE I
AFRICA RTT BIAS EXPERIMENTAL RESULTS

RTT Ratio	Throughput Ratio	Packet Losses
1	1.0132	440
2	2.3552	581
3	3.6106	529
6	8.4616	336

TABLE II
HSTCP RTT BIAS EXPERIMENT RESULTS

RTT Ratio	Throughput Ratio	Packet Losses
1	0.9752	5050
2	31.7379	3881
3	93.7189	19603
6	284.4962	7614

from the decreased number of congestion incidents, and in the number of packets lost in each event.

C. RTT bias

As mentioned in Section II, another area where high speed protocols need to be evaluated is with regards to the fairness between two flows of differing round trip times. The authors of [8] demonstrated the undesirable RTT bias properties of HSTCP and STCP, and since TCP-Africa is based in part on HSTCP, we address this concern with the following experiment, which is similar to the simulations performed in [8].

The access link speeds are set to one gigabit, with the links from A-S and S-B set to have a delay of 5 ms. The shared link has a capacity of 622 Mbps, and a delay of 5 ms. Thus, the flow from A to B will have a round trip time of 30 ms. The links from C to S and from S to D are set so that the overall round trip time of the flow from C to D will be a desired multiple of the round trip time of the flow from A to B.

In Table II we see the results from the RTT bias simulation for HSTCP. As predicted in [8], there is a serious fairness problem with flows of different round trip times for HSTCP. The short round trip time flow quickly dominates the connection, starving out the other flow.

In Table I we see the results of the same set of experiments run for TCP-Africa. Here, the TCP-Africa flows share the bandwidth roughly proportional to their round trip times. This is actually an improvement over the throughput ratio experienced by TCP-Reno under such circumstances, where the throughput difference is quadratic with the round trip time ratio [8]. The packet losses in the experiments seem to follow and unusual pattern. This could be due to several factors. As the RTT ratio grows, one of the flows becomes less and less aggressive, this would tend to decrease the number of packet losses. At the same time, however, the other flow is able to increase its window significantly, and thus become more aggressive. These conflicting factors account for the packet loss behavior as the RTT ratio changes.

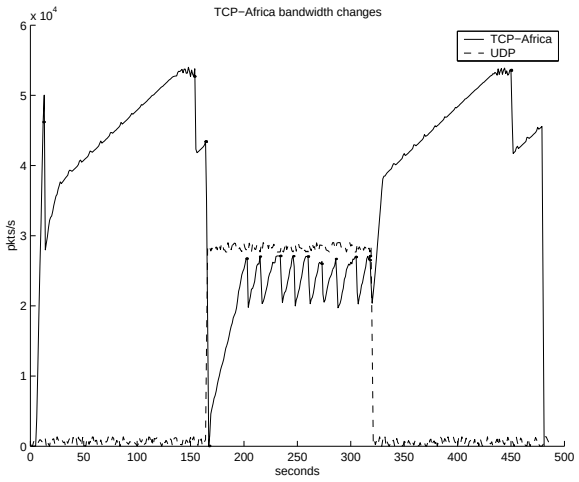

Fig. 7. Rapid Changes in Available Bandwidth Experiment

One reason why TCP-Africa can achieve such excellent RTT bias properties is the effect the delay metric has on limiting the aggressiveness of the dominant flows. These flows have a higher rate, and by equation 7, are more sensitive to the measured queuing delays.

Thus, they enter their slow growth mode sooner than the longer round trip time flow. While the short RTT flow is in slow mode, and the long RTT flow is in its fast mode, we are basically seeing the effects of a fast linear growth flow competing with the accelerated growth of a long round trip time HSTCP flow. Indeed, the long trip time flow is allowed to continue to use the higher increase rate of HSTCP while the dominant flow has fallen back to a one packet per round trip time increase rule.

D. Adapting to network conditions

In this experiment, seen in Figure 7, we look at how quickly TCP-Africa can adapt to changing network conditions. The bottleneck link has a capacity of 622 Mbps, and the flow is experiencing a minimum round trip time of 84 ms. After 160 s, a CBR UDP flow at 300 Mbps is started, then, at 320 s, the UDP flow is stopped. This allows us to see how well TCP-Africa adapts to changes in the available link capacity.

As is clear from Figure 7, and as is expected from all TCP variants, TCP-Africa quickly reduces its bandwidth in response to the UDP flow. As is clear from the increase slope of the TCP-Africa flow, the slow growth mode of the protocol is being entered as the flow approaches the maximum available bandwidth. Also, as is clear from the graph, after the UDP flow terminates, TCP-Africa quickly re-enters its high speed mode, and quickly utilizes the newly freed bandwidth.

V. Implementation Issues

A. Overhead

While this scheme does involve more potential overhead than TCP-Reno, this can be mitigated significantly. The protocol requires keeping a high accuracy round trip time estimate, which will probably involve a queuing action every packet, along with the multiplication and addition necessary for an exponentially smoothed estimate. Since TCP already uses round trip time estimates, this is not an unreasonable requirement. Although the TCP timestamp option probably won't provide sufficient resolution, the cycle counter registers on modern processors provide a very natural high resolution timer. If the overhead proves to be too significant, the round trip time estimate could be made using only a subset of packets. If the delay metric condition check every packet is objectionable, a state variable could be updated on the order of every round trip time. We are currently investigating what performance impact such changes would have.

B. Robustness

Another advantage of this window increase technique is that it is entirely robust to malfunction of the delay metric. In a network with unusually variable delay, the protocol would at worst perform as TCP-Reno with a modified drop rule. In a network where the delay never increases, that is, a bufferless network, the protocol will simply stay in fast mode, acting like a normal STCP or HSTCP sender. We do not expect a network with no buffers to be a common environment.

Sensitivity to reverse path congestion is also a concern for delay based protocols. As with Vegas, and possibly FAST [9], protocols that use round trip time estimates are sensitive to delay noise added to the returning stream of ACKs. In the case of TCP-Africa, this will result in falling back to more Reno-like behavior. Fortunately, since TCP-Africa never decreases its window solely due to delay information, return path congestion cannot cause it to surrender bandwidth. However, by forcing the protocol into slow growth mode, significant congestion in the reverse path may reduce the ability of the protocol to quickly acquire available bandwidth. We are still looking into how significant this effect is, and into ways of addressing it, perhaps by taking advantage of one way delay information from the TCP-Timestamp option.

C. Safety

The increase rules for TCP-Africa call for its increase rate to vary between that of TCP-Reno, and rates roughly equivalent to other high speed protocols. TCP-Africa is still ACK-clocked, as regular TCP is, and the window is still reduced multiplicatively on packet drop. It appears that such an adaptive increase algorithm is a conservative choice for the increase rule, and one that allows the protocol to run at high speeds while still being safe for the Internet at large. Furthermore, the proposed window increase scheme is entirely more conservative than the existing schemes. Since the innovation in TCP-Africa does not specify the exact form of the drop response while in fast mode, this conditionally fast increase technique can be combined with both STCP, and HSTCP. In our experiments, we used HSTCP [3] to specify the increase characteristics in fast mode, taking advantage of its RTT fairness benefits over STCP [8], as well as its smoother transition from the TCP-Reno drop rule. Since both HSTCP and STCP have been tested in the real Internet, and neither led to catastrophic failures, we do not expect TCP-Africa to pose a significant danger.

VI. Conclusions

To maintain a careful balance between the increased aggressiveness and the fairness and safety is a major concern when developing protocols for high bandwidth delay product links. We find that key towards succeeding at both challenges is to use delay information as an indicator for the appropriate level of aggressiveness. Unlike TCP-Vegas with its ideally loss-free steady state yet inherent disadvantage against greedy loss-based Reno, our protocol TCP-Africa exploits congestion indicators towards adaptive fair rapid increase, not giving in to Reno, yet not crushing it. In ns-2 simulations, the protocol has achieved excellent utilization, low induced packet loss rate, and excellent fairness properties, while at the same time exhibiting RTT bias performance equivalent to, or even exceeding that of TCP-Reno. We will continue to evaluate the performance of TCP-Africa in real network environments, first using UDP emulation, and then through an actual kernel implementation.

Finally, we note that there may be more than one way to use delay information to achieve safe, fast utilization

of network resources, while having acceptable performance against traditional greedy flows. Particularly, our use of two separate window increase curves can be considered to be a special case of the more general form, where the increase rate $a(w)$ is determined as some continuous function $a(w, delay)$. It is likely that certain choices for this function can provide even better performance than what we have achieved with our two mode rule, and we are currently investigating several such choices of functions. The performance results obtained with our two mode algorithm indicate exciting possibilities in this direction.

REFERENCES

[1] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM*, pp. 314–329, 1988.

[2] T. V. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss," *IEEE/ACM Transactions on Networking*, vol. 5, no. 3, pp. 336–350, 1997.

[3] S. Floyd, "Highspeed tcp for large congestion windows," *RFC 3649, Experimental*, Dec. 2003.

[4] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," *ACM SIGCOMM*, 1998.

[5] G. Hasegawa, K. Kurata, and M. Murata, "Analysis and improvement of fairness between TCP Reno and Vegas for deployment of TCP Vegas to the Internet," pp. 177–186, 2000.

[6] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," *ACM SIGCOMM*, pp. 24–35, 1994.

[7] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," *Computer Communication Review*, vol. 32, no. 2, Apr. 2003.

[8] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast long-distance networks," *IEEE INFOCOM*, 2004.

[9] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: motivation, architecture, algorithms, performance," *IEEE INFOCOM*, 2004.

[10] D. Katabi, M. Handley, and C. Rohrs, "Internet congestion control for future high bandwidth-delay product environments," *ACM SIGCOMM*, Aug. 2002.

[11] W. Richard Stevens, *TCP/IP Illustrated, Volume I*, Addison Wesley, 1994.

[12] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan, "Evaluation of TCP Vegas: Emulation and experiment," *ACM SIGCOMM*, 1995.

[13] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell, "pathChirp: Efficient available bandwidth estimation for network paths," *Passive and Active Measurement Workshop*, 2003.

[14] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," *Internetworking: Research and Experience*, vol. 3, no. 3, 1992.