To Appear in INFOCOM 2004

Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks

Lisong Xu, Khaled Harfoush, and Injong Rhee
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534
lxu2, harfoush, rhee@csc.ncsu.edu

Abstract—**High-speed networks with large delays present a unique environment where TCP may have a problem utilizing the full bandwidth. Several congestion control proposals have been suggested to remedy this problem. The existing protocols consider mainly two properties: TCP friendliness and bandwidth scalability. That is, a protocol should not take away too much bandwidth from standard TCP flows while utilizing the full bandwidth of high-speed networks. This paper presents another important constraint, namely, *RTT (round trip time) unfairness* where competing flows with different RTTs may consume vastly unfair bandwidth shares. Existing schemes have a severe RTT unfairness problem because the congestion window increase rate gets larger as the window grows – ironically the very reason that makes them more scalable. RTT unfairness for high-speed networks occurs distinctly with drop tail routers for flows with large congestion windows where packet loss can be highly synchronized.**

After identifying the RTT unfairness problem of existing protocols, this paper presents a new congestion control scheme that alleviates RTT unfairness while supporting TCP friendliness and bandwidth scalability. The proposed congestion control algorithm uses two window size control policies called *additive increase* and *binary search increase*. When the congestion window is large, additive increase with a large increment ensures square RTT unfairness as well as good scalability. Under small congestion windows, binary search increase supports TCP friendliness. The simulation results confirm these properties of the protocol.

Keywords – Congestion control, High-speed networks, RTT unfairness, TCP friendliness, Scalability, Protocol Design.

I. INTRODUCTION

The Internet is evolving. The deployment of wide-area high-speed networks like Abilene and ESNet has heightened demand for data-intensive, high-performance computing. Applications like scientific collaboration, telemedicine, and real-time environment monitoring benefit from this deployment. These applications require access to high-bandwidth real time data, images, and video captured from remote sensors such as satellite, radars, and echocardiography. They, in addition, require predictable, low-latency access to this data, in real time.

TCP has been widely adopted as a data transfer protocol for these networks. TCP, however, substantially underutilizes network bandwidth over high-speed connections with long RTTs [1, 2, 4, 7]. TCP increases its congestion window by one packet in every round trip time (RTT) and reduces it by half at a loss event. A *loss event* is defined to be a packet loss that causes TCP to reduce its congestion window. Over 83,333/2 RTTs are required for TCP to increase its window from half utilization to full utilization of 10Gbps with 1500-byte packets — approximately 1 hour with 100ms RTT. This requires that no packet is lost within 1 hour. That is, the loss rate cannot be more than 1 loss event per 2,600,000,000 packets, which is less than the theoretical limit of the network's bit error rates [1, 2].

Fine-tuning TCP parameters such as receiver windows and network interface buffers [9, 10, 11, 12] may mitigate this problem. One straightforward solution is to increase the packet size by using the Jumbo packet option (up to 8KB) and use multiple TCP connections [13, 14, 15]. This scheme is similar to Additive Increase and Multiplicative Decrease (AIMD) where the window increases with a fixed increment per RTT and decreases by a multiplicative factor under losses. Although these approaches enhance utilization by having a larger increment per RTT than TCP, the bandwidth ratio between these flows and standard TCP flows is always fixed. Therefore, even in the TCP's "well-behaving" operating range (between loss rates of 10^{-2} to 10^{-4}), they always use proportionally a larger bandwidth share compared to standard TCP. Guaranteeing both TCP friendliness and bandwidth scalability with one fixed increase rate of window is challenging. It calls for adaptive schemes that vary the window growth rate depending on network conditions.

The research community has responded quickly to the need to address TCP's limitations. Several promising new protocols have been put forward including XCP [5], SABUL [6], FAST [7], High Speed TCP (HSTCP) [1, 2, 3], and Scalable TCP (STCP) [4]. Except XCP, these protocols adaptively adjust their increase rates based on the current window size. These protocols are claimed to be TCP friendly under high loss rate environments as well as highly scalable under low loss environments.

The work presented here began as an attempt to assess the performance of high-speed congestion control protocols in under "realistic" network operation: i.e., to assess their impact on overall network performance in the presence of heterogeneous, competing flows. We focus on HSTCP and STCP, which are window-based, self-clocking protocols known for safer incremental deployment [16]. Other protocols such as

The work reported in this paper is sponsored in part by NSF CAREER ANI-9875651, NSF ANI- 0074012

SABUL and FAST are not studied because SABUL is a rate-based protocol and FAST has not yet been described in detail.

Lakshman and Madhow[19] studied RTT unfairness of TCP in networks with high bandwidth delay products. It was reported that under a FIFO queue (i.e., drop tail) that TCP throughput is inversely proportional to RTT^{α} where $1 \le \alpha \le 2$. This paper extends their work by investigating RTT fairness for new high-speed protocols. Our study reveals that notwithstanding their scalability and TCP friendliness properties, HSTCP and STCP have a serious RTT unfairness problem when multiple flows with different RTT delays are competing for the same bottleneck bandwidth. We define the *RTT unfairness* of two competing flows to be the throughput ratio in terms of their RTT ratio. Under a completely synchronized loss model, we show that two HSTCP flows with RTT_1 and RTT_2 have RTT unfairness $(RTT_2/RTT_1)^{5.56}$ and those of STCP have RTT unfairness even $(RTT_2/RTT_1)^{\infty}$. That is, the shorter RTT flow eventually starves off the longer RTT flow. Note that TCP and AIMD have RTT unfairness $(RTT_2 / RTT_1)^2$ [19]. We report that this problem commonly appears for drop tail routers while the severity reduces for RED, and therefore, greatly impairs the ability to incrementally deploy the protocols, without adequate support from active queue management (such as RED and XCP).

The severe RTT unfairness of HSTCP and SCTP comes from their adaptability– ironically the very reason that makes them more scalable to large bandwidth. In HSTCP and STCP, a larger window increases faster than a smaller window. Compounded with delay differences, RTT unfairness gets worse as the window of a shorter RTT flow grows faster than that of a longer RTT flow.

Another source of RTT unfairness is *synchronized loss*: simultaneous loss events for multiple competing flows. Under the assumption that packet losses are uniformly distributed across all flows, a flow with a larger window gets more loss events than a flow with a smaller window. However, this assumption does not hold true in practice, especially in networks with drop tail routers. A short-term loss rate during the period around the overflow of router buffers can be much higher than the average long-term loss rate. Therefore, it is very possible that both flows lose packets at the same time when the buffer overflows; otherwise none of them lose packets. Assuming a random packet loss probability p with the uniform distribution, the probability that a flow with window size w loses at least one packet within an RTT is $1-(1-p)^w$, which increases exponentially as window size w increases. That is, many high-speed connections with relatively large windows more likely lose packets simultaneously at the time of buffer overflow.

Synchronized loss encourages RTT unfairness. Since loss events are more synchronized across different window sizes, larger windows with a short RTT can always grow faster than smaller windows with a long RTT. Furthermore, since synchronization can prolong convergence and cause a long-term oscillation of data rates, it can hurt bandwidth fairness in short-term time scales. In our simulation study, we observe the short-term unfairness of STCP and HSTCP over both drop-tail and RED routers.

Designing a high-speed congestion control that supports all three properties of RTT fairness, scalability, and TCP friendliness is challenging. For instance, AIMD, which provides square RTT unfairness and scales its bandwidth share by increasing its additive increase factor, is not TCP friendly. HSTCP and STCP, which are extremely scalable under low loss rates and TCP friendly under high loss rates, are not RTT fair. Recognizing these challenges, we propose, in the second part of this paper, a new protocol called *Binary Increase Congestion Control (BIC)* satisfies all these criteria:

1. ***Scalability***: BIC can scale its bandwidth share to 10 Gbps around 3.5e-8 loss rates (comparable to HSTCP which reaches 10Gbps at 1e-7).
2. ***RTT fairness***: for large windows, BIC's RTT unfairness is proportional to the inverse square of the RTT ratio as in AIMD.
3. ***TCP friendliness***: BIC achieves bounded TCP fairness for all window sizes. Around high loss rates where TCP performs well, its TCP friendliness is comparable to STCP's.
4. ***Fairness and convergence***: compared to HSTCP and STCP, BIC achieves better bandwidth fairness over both short and long time scales, and faster convergence to a fair bandwidth share.

The paper organized as follows: Section II describes our simulation setup. Section III discusses evidence for synchronized loss. In Sections IV and V, we discuss the behavior of HSTCP and STCP. In Section VI, we describe BIC and its properties. Section VII gives details on simulation results. Related work and Conclusion can be found in Sections VIII and IX.

II. SIMULATION SETUP

Fig. 1 shows the NS simulation setup that we use throughout the paper. Various bottleneck capacity and delays are tested. The buffer space at the bottleneck router is set to 100% of the bandwidth and delay product of the bottleneck link. All connections pass through the bottleneck link. Each link is configured to have different RTTs and different starting and finishing times to reduce the phase effect [17]. Three kinds of background traffic are simulated. Substantial web traffic—consuming a minimum of 20% of bottleneck bandwidth, and up to 50% of bottleneck bandwidth when no other flows are present—is generated in both directions to remove synchronization in TCP feedback. Twenty-five small TCP flows whose congestion window sizes are restricted to be less than 64 are transmitted in each direction. Their starting and finishing times are set randomly. And two to four long-lived TCP flows (with no window size restriction) are created for both directions. These three kinds of simulated background traffic totally consume a minimum of 20% of the backward bandwidth.

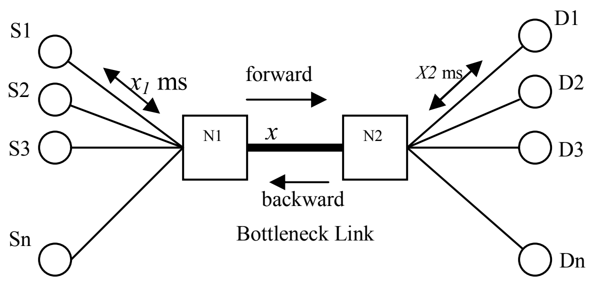

Fig. 1. Simulation network topology

This simulation topology is roughly comparable to that of the real high-speed networks. High-speed networks of our interest are different from the general Internet where a majority of bottlenecks are located at the edges. High-speed networks with high-speed edges can create a bottleneck at locations where much high-speed traffic meets, such as Startlight in Chicago that connects CERN (Geneva) and Abilene.

We make no claim about how realistic our background traffic is. However, we believe that the amount of background traffic in both directions, and randomized RTTs and starting and finishing times are sufficient to reduce any simulation anomaly, e.g., the phase effect and synchronized feedback. We also paced TCP packets so that no more than two packets are sent in burst. A random delay between packet transmissions is inserted to avoid the phase effect. We test both RED and drop tail routers at the bottleneck link. For RED, we use adaptive RED with the *bottom* of *max_p* set to 0.001.[1]

III. SYNCHRONIZED PACKET LOSS

We use a synchronized loss model for the theoretical analysis of RTT fairness. Before delving into the analysis, we provide some evidence that synchronized loss is common in high-speed networks.

To measure the extent of the synchronization, we ran a simulation involving 20 high-speed connections of HSTCP with RTTs varying from 40 ms to 150 ms in the network shown in Figure 1 with bottleneck link bandwidth varying from 100 Mbps to 2.5 Gbps. To measure synchrony in packet losses, we first compute the average time interval between two consecutive loss events of a connection. We define *measurement interval* to be the duration corresponding to one-tenth of the average interval. We consider all the flows having at least one loss event within the same synchronization interval undergo synchronized losses. To measure the degree of synchronized losses, we divide the total simulation time by the unit of the measurement intervals, and then in each interval count the number of unique high-speed flows that have at least one packet loss. *Synchronization ratio* is the ratio of the number of measurement intervals containing at least *x* flows, where *x* is a predefined threshold, to the number of measurement intervals containing at least one flow. Figure 2 shows synchronization ratios of RED and drop tail routers when

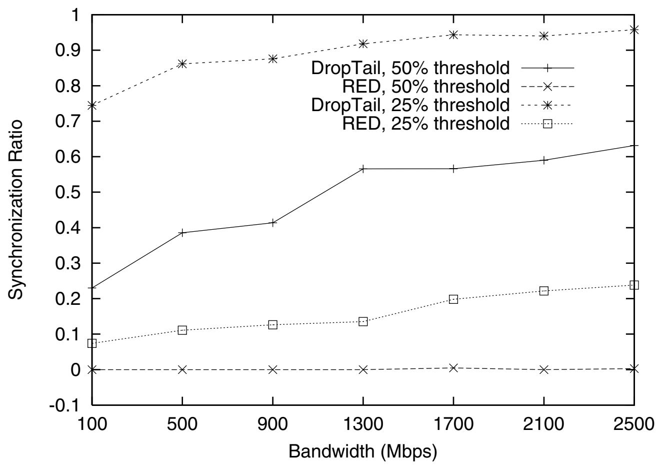

Fig. 2: Synchronization Ratio under RED and drop tail routers.

the threshold is set to 25% and 50% of the total number of high-speed flows respectively.

We observe that the synchronization ratio increases as the bandwidth increases, especially under drop tail routers. As the bandwidth increases from 100 Mbps to 2.5 Gbps, the synchronization ratio with 50% threshold (i.e., 10 flows) increases from about 0.23 to about 0.6. That implies whenever a flow loses a packet in the drop tail router, the probability that at least half of the total flows experience packet losses during a period of measurement interval increases from 23% to 60%. On the other hand, RED does not incur as much synchronized loss. The synchronization ratio with 50% threshold is always close to zero. However, there still exists some amount of synchronization; the synchronization ratio for RED with 25% threshold under 2.5 Gbps is about 23%, which means the probability that at least a quarter of the total flows have a synchronized loss event is around 23%.

The result implies that the number of synchronized loss can be quite substantial in drop tail. Although it requires real network tests to confirm this finding, we believe that our simulation result is not difficult to recreate in real networks. We leave that to future study.

IV. RTT UNFAIRNESS OF HSTCP AND STCP

In this section, we analyze the effect of synchronized loss on the RTT fairness of HSTCP and STCP. We use a synchronized loss model where all high-speed flows competing on a bottleneck link experience loss events at the same time. By no means, we claim that this model characterizes all the aspects of high-speed networks. We use this model to gain insights into RTT unfairness that we observe in our simulation experiment described later.

Let w_i and RTT_i denote the average window size and the RTT of flow *i* (*i*=1, 2) respectively. Let *t* denote the interval between two consecutive loss events of a flow during steady state.

In a network with a uniformly distributed packet loss, different connections, regardless of their RTTs, get the same packet loss rate. But with a synchronized loss model, different connections may see different packet loss rates. Suppose that the

[1] The use of a smaller bottom value is recommended by [1] to reduce synchronized loss – normally this value is set to 0.01.

loss events of flow i are uniformly distributed with rate p_i. The total number of packets sent by flow i between its two consecutive loss events is $1/p_i$. The total number of RTTs between two consecutive loss events is t/RTT_i. So, the average window size can be obtained as follows.

$$w_i = \frac{1/p_i}{t/RTT_i} = \frac{RTT_i}{t \cdot p_i} \qquad (1)$$

Let $R(p)$ denote the *response function* of a protocol, which is the average sending rate of the protocol in the unit of packets per RTT, in terms of a loss event rate p. For HSTCP, STCP, and AIMD, their response functions are of form $R(p) = \frac{1}{RTT}\frac{c}{p^d}$, where c and d are protocol-dependent constants. The value of d is between 0.5 and 1 [2]. The value of d for AIMD [18], HSTCP [1], and STCP [4] is 0.5, 0.82, and 1, respectively. Therefore, the average sending rate of flow i is calculated as follows.

$$\frac{w_i}{RTT_i} = \frac{1}{RTT_i}\frac{c}{p_i^{\,d}} \Rightarrow p_i = \sqrt[d]{c/w_i}$$

Substitute the above equation into (1), we have

$$w_i = \frac{RTT_i}{t \cdot \sqrt[d]{c/w_i}} \Rightarrow w_i = \left(\frac{RTT_i}{t \cdot \sqrt[d]{c}}\right)^{\frac{d}{d-1}}$$

Therefore the RTT unfairness of the two flows, the ratio of their average throughputs, can be calculated as

$$\frac{(w_1/RTT_1)(1-p_1)}{(w_2/RTT_2)(1-p_2)} \approx \frac{w_1/RTT_1}{w_2/RTT_2} = \left(\frac{RTT_2}{RTT_1}\right)^{\frac{1}{1-d}}$$

where $(1-p_i)$ is approximated by 1, since p_i is usually far less than 1.

Substitute the value of d for each protocol, we get the exponent $1/(1-d)$ for AIMD, HSTCP, and STCP, which is 2, 5.56, and ∞, respectively.

Table 1 presents a simulation result that shows the bandwidth shares of two high-speed flows with different ratios of RTTs running in drop tail (RED does not show significant RTT unfairness). The inverse ratio of RTTs is varied to be from 1 to 6 with base RTT 40 ms.

AIMD shows a quadratic increase in the throughput ratio as the inverse RTT ratio increases. STCP shows almost 400 times throughput ratio for inverse RTT ratio 6. HSTCP shows about 100 times throughput ratio for inverse RTT ratio 6. Clearly the results indicate extremely unfair use of bandwidth by a short RTT flow in drop tail.

The result shows better RTT unfairness than predicted by the analysis. This is because while the analysis is based on a completely synchronized loss model, the simulation may involve loss events that are not synchronized. Also when the window size is very small, the occurrence of synchronized loss is substantially low. Long RTT flows continue to reduce their windows up to a limit where synchronized loss does not occur much. Thus, the window ratio does not get worse.

Table 1: The throughput ratio of two high speed flows over various RTT ratios in 2.5Gbps networks.

Inverse RTT Ratio	1	3	6
AIMD	1.11	6.68	22.03
HSTCP	1.01	29.19	107.90
STCP	1.01	127.23	389.13

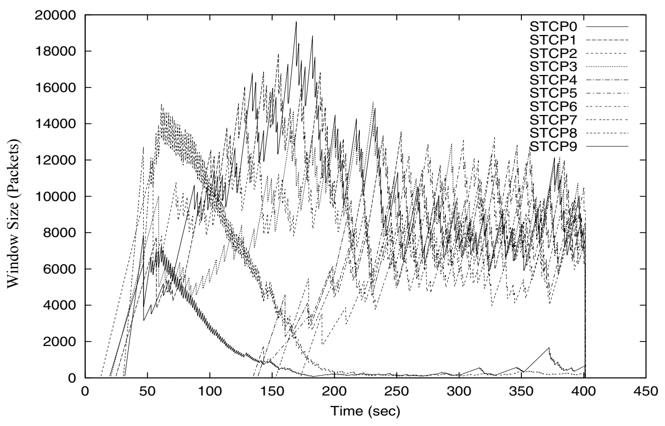

Fig. 3: 10 scalable TCP flows; 2.4 Gbps, Drop tail.

Figure 3 shows a sample simulation run with ten STCP flows in drop tail. All flows are started at different times. Eight flows have 80 ms RTT, and two flows have 160 ms RTT. It exhibits a typical case of RTT unfairness. The two longer RTT flows slowly reduce their window down to almost zero while the other flows merge into a single point. Note that STCP does not converge in a completely synchronized model because of Multiplicative Increase and Multiplicative Decrease (MIMD) window control [20]. However, in this simulation, eight flows with the same RTT do converge (despite high oscillation). This indicates that there exists enough asynchrony in packet loss in our simulation to make the same RTT flows converge, but not enough to correct RTT unfairness.

V. RESPONSE FUNCTION

The response function of a congestion control protocol can show many characteristics of the protocol including RTT fairness, TCP friendliness, and scalability. This section briefly discusses how these properties can be deduced from response functions.

Figure 4 draws the response functions of HSTCP, STCP, AIMD, and TCP in a log-log scale. AIMD uses the increase factor 32 and the decrease factor 0.125.

From the previous section, we showed that for a protocol with a response function c/p^d where c and d are constants and p is a loss event rate, its RTT unfairness could be modeled as $(RTT_2/RTT_1)^{1/(1-d)}$. As d increases, the slope of the response function and RTT unfairness increase. That is, the slope of a response function in a log-log scale determines its RTT unfairness. Since TCP and AIMD have the same slope, the RTT unfairness of AIMD is the same as TCP – square RTT unfairness. The RTT unfairness of STCP is infinite while that of HSTCP falls somewhere between TCP's and STCP's.

The ability that a protocol utilizes the high amount of

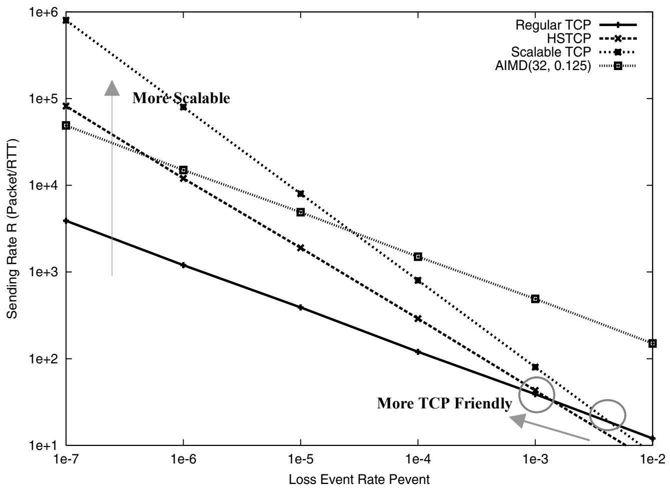

Fig. 4: Response Functions of various protocols.

bandwidth under high-speed networks is determined by its high sending rates under low loss rates. Thus from Figure 4, a protocol becomes more scalable as its sending rate gets higher under lower loss rates. STCP shows the best scalability followed by HSTCP and AIMD.

The TCP friendliness of a protocol is indicated by the point where the response function of the protocol crosses that of TCP. This is because high-speed congestion control protocols act the same as standard TCP over a higher loss rate than the cross point. Under lower loss rates below this point, the protocols run their own "scalable" schemes. Under this strategy, it becomes more TCP friendly if a protocol crosses TCP as low a loss rate as possible (to the left of the X axis) since the protocol follows TCP below that point. However, moving the cross point to the left while maintaining the scalability increases the slope of the response function, hurting RTT fairness.

An ideal protocol would be that (1) its response function crosses TCP's as low a rate as possible, and at the same time (2) under lower loss rates, its slope is as close as that of AIMD. Note that the function need not be a straight line, i.e., the slope can vary depending on loss rates. But its slope at any loss rates should not exceed that of STCP although it can be much more forgiving under a high loss rate where window size is small enough to avoid frequent synchronized loss.

VI. BINARY INCREASE CONGESTION CONTROL

It is challenging to design a protocol that can satisfy all three criteria: RTT fairness, TCP friendliness, and scalability. As alluded earlier, these criteria need not be satisfied simultaneously for all loss rates (for instance, it does not make sense to be TCP friendly for low loss rates). A protocol should adapt its window control depending on the size of windows. Below, we present such a protocol, called *Binary Increase Congestion Control (BIC)*. BIC consists of two parts: *binary search increase* and *additive increase*.

Binary search increase: We view congestion control as a searching problem in which the system gives yes/no feedback through packet loss as to whether the current sending rate (or window) is larger than the network capacity. The starting points for this search are the current *minimum window size* W_{min} and *maximum window size* W_{max}. Usually, W_{max} is the window size just before the last fast recovery (i.e. where the last packet loss occurred), and W_{min} is the window size just after the fast recovery. The algorithm repeatedly computes the midpoint between W_{max} and W_{min}, sets the current window size to the midpoint; and checks for feedback, in the form of packet losses. Based on this feedback, the midpoint is taken as the new W_{max} if there is a packet loss, and as the new W_{min} if not. The process repeats, until the difference between W_{max} and W_{min} falls below a preset threshold, called *the minimum increment* (S_{min}).

This technique, which we call *binary search increase,* allows bandwidth probing to be more aggressive initially when the difference from the current window size to the target window size is large, and become less aggressive as the current window size gets closer to the target window size. A unique feature of the protocol is that its increase function is logarithmic; it reduces its increase rate, as the window size gets closer to the saturation point. The other scalable protocols tend to increase their rates at the saturation point so that the increment at the saturation point is the maximum in the current epoch (defined to be a period between two consecutive loss events). Typically, the number of lost packets is proportional to the size of the last increment before the loss. Thus binary search increase can reduce packet loss. As we shall see, the main benefit of binary search is that it gives a concave response function, which meshes well with that of additive increase described below. We discuss the response function of BIC in Section VI-B.

Additive Increase: In order to ensure faster convergence and RTT-fairness, we combine binary search increase with an additive increase strategy. When the distance to the midpoint from the current minimum is too large, increasing the window size directly to that midpoint may add too much stress to the network. When the distance from the current window size to the target in binary search increase is larger than a prescribed maximum step, called *the maximum increment* (S_{max}), we increase window size by S_{max} until the distance becomes less than S_{max}, at which time window increases directly to the target. Thus, after a large window reduction, the strategy initially increases the window linearly, and then increases logarithmically. We call this combination of binary search increase and additive increase *binary increase*.

Combined with a multiplicative decrease strategy, binary increase becomes close to pure additive increase under large windows. This is because a larger window results in a larger reduction in multiplicative decrease, and therefore, a longer additive increase period. When the window size is small, it becomes close to pure binary search increase – a shorter additive increase period.

Slow Start: When the current window size grows past the current maximum window W_{max}, the binary search algorithm switches to probing the new maximum window, which is unknown (i.e., the window size where loss can occur is unknown).

We run a "slow start" strategy to probe for a new maximum, once the current window size is greater than W_{max}, but smaller than $W_{max}+S_{max}$. The congestion window increases in each RTT round in steps $W_{max}+S_{min}$, $W_{max}+2*S_{min}$, $W_{max}+4*S_{min}$, ..., $W_{max}+S_{max}$. The rationale is that the current window size is likely to be at the saturation point because the last losses occur around the current window size. BIC probes for available bandwidth in a "slow start", until it is safe to increase the window by S_{max}. After slow start, BIC switches to additive increase with fixed increment S_{max} to probe a new W_{max}.

Fast convergence: Under a completely synchronized loss model, binary search increase combined with multiplicative decrease converges to a fair share [8]. Suppose there are two flows with different window sizes, but with the same RTT. Since the larger window reduces more in multiplicative decrease (with a fixed factor β), the time to reach the target is longer for a larger window. However, its convergence time can be very long. Assuming infinitely large S_{max}, binary search increase takes $\log(\beta W_{max})-\log(S_{min})$ RTT rounds to reach the maximum window after a window reduction of βW_{max}. Since window size increases logarithmically, both larger and smaller windows can return to their respective maxima very fast almost at the same time (although the smaller window flow gets to its maximum slightly faster). Thus, the smaller window flow ends up taking away only a small amount of bandwidth from the larger flow before the next window reduction, prolonging the convergence time.

To remedy this behavior, we modify the binary search increase as follows. In binary search increase, after a window reduction, new maximum and minimum are set. Suppose these values are $W_{max,i}$ and $W_{min,i}$ for flow i (i=1, 2). If the new $W_{max,i}$ is less than the previous, this window is in a downward trend (so likely to have a window larger than the fair share). Then, we readjust the new maximum to be the same as the new target window which is the midpoint (i.e., $W_{max,i} =(W_{max,i} + W_{min,i})/2$), and then readjust the target and apply binary search. This has an effect of reducing the increase rate of the larger window which allows the smaller window to catch up. We call this strategy *fast convergence.*

Figure 5 shows a sample run of two BIC flows. Their operating modes are marked by circles and arrows.

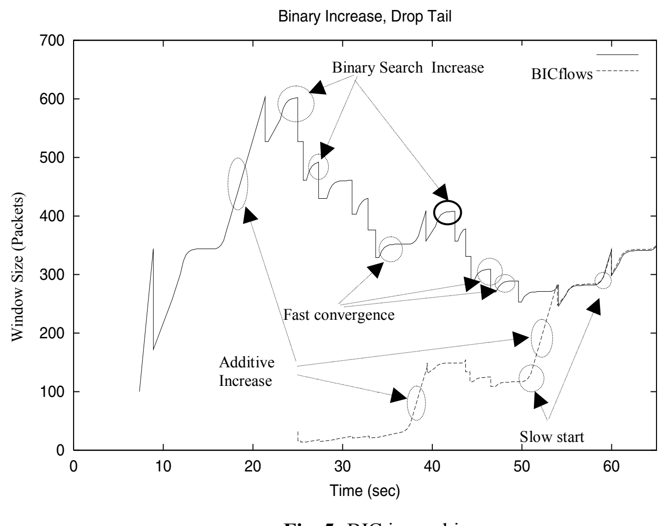

Fig. 5: BIC in working.

A. Protocol Implementation

Below, we present the pseudo-code of BIC implemented as a modification of TCP-SACK.

The following preset parameters are used:

`low_window`	if the window size is larger than this threshold, BIC engages; otherwise normal TCP increase/decrease.
S_{max}	the maximum increment.
S_{min}	the minimum increment.
β	multiplicative window decrease factor.

The following variables are used:

W_{max}	the maximum window size
`cwnd`	congestion window size;
`bic_inc`	window increment per RTT

When entering faster recovery:

```
if (cwnd < low_window){ //normal TCP
  cwnd = cwnd *0.5;
  return;
}
 if (cwnd < $W_{max}$) //fast convergence
  $W_{max}$ = cwnd * (2-$\beta$) / 2;
else
  $W_{max}$ = cwnd;
cwnd = cwnd * (1-$\beta$);//multiplicative decrease
```

When not in fast recovery and an acknowledgment for a new packet arrives:

```
if (cwnd < low_window){//normal TCP
  cwnd = cwnd + 1/cwnd;
  return;
}

if (cwnd < $W_{max}$) //binary search or additive increase
  bic_inc = ($W_{max}$ – cwnd)/2;
else              //slow start or additive increase
  bic_inc = cwnd - $W_{max}$;
if (bic_inc > $S_{max}$) //additive increase
  bic_inc = $S_{max}$;
else if (bic_inc < $S_{min}$)//binary search increase
                              //or slow start
  bic_inc = $S_{min}$;

cwnd = cwnd + bic_inc/cwnd;
```

B. Characteristics of BIC

In this section, we analyze the response function and RTT fairness of BIC. An analysis on the convergence and smoothness of the protocol can be found in [8].

1) Response function of BIC

In this section, we present a deterministic analysis on the response function of BIC.

We assume that a loss event happens at every $1/p$ packets. We define a *congestion epoch* to be the time period between two consecutive loss events. Let W_{max} denote the window size just before a loss event. After a loss event, the window size decreases to $W_{max}(1-\beta)$.

BIC switches from additive increase to binary search increase when the distance from the current window size to the target window is less than S_{max}. Since the target window is the midpoint between W_{max} and the current window size, it can be said that BIC switches to binary search increase when the distance from the current window size to W_{max} is less than $2S_{max}$. If the distance between the current window and W_{max} is less than $2S_{max}$, there is no additive increase. Let N_1 and N_2 be the numbers of RTT rounds of additive increase and binary search increase, respectively. We have

$$N_1 = \max\left(\left\lceil \frac{W_{max}\beta}{S_{max}} \right\rceil - 2, \quad 0\right)$$

Then the total amount of window increase during binary search increase can be expressed as $W_{max}\beta - N_1 S_{max}$. Assuming that this quantity is divisible by S_{min}, then N_2 can be obtained as follows.

$$N_2 = \log_2\left(\frac{W_{max}\beta - N_1 S_{max}}{S_{min}}\right) + 2$$

In the above equation, 2 corresponds to the first and the last RTTs of the binary search increase.

During additive increase, the window grows linearly with slope $1/S_{max}$. So, the total number of packets during additive increase, Y_1, can be obtained as follows.

$$Y_1 = \frac{1}{2}\left(W_{\max}(1-\beta) + W_{\max}(1-\beta) + (N_1 - 1)S_{\max}\right)N_1 \qquad (2)$$

During binary search increase, the window grows logarithmically. So, the total number of packets during binary search increase, Y_2, can be expressed as follows.

$$Y_2 = W_{\max} N_2 - 2(W_{max}\beta - N_1 S_{max}) + S_{min} \qquad (3)$$

The total number of RTTs in an epoch is $N = N_1 + N_2$, and the total number of packets in an epoch is $Y = Y_1 + Y_2$. Since a loss event happens at every $1/p$ packets, Y can be expressed as $Y = 1/p$. Using (2) and (3), we may express W_{max} as a function of p. Below, we give the closed-form expression of W_{max} for two special cases.

First, we assume that $W_{max}\beta > 2S_{max}$, and $W_{max}\beta$ is divisible by S_{max}. Then $N_1 = W_{max}\beta/S_{max} - 2$. Now we can get

$$W_{\max} = \frac{-b + \sqrt{b^2 + 4a\left(c + \frac{1}{p}\right)}}{2a} \qquad (4)$$

where $a = \beta(2-\beta)/(2S_{max})$, $b = \log_2(S_{max}/S_{min}) + (2-\beta)/2$, and $c = S_{max} - S_{min}$. The average sending rate, R, is then,

$$R = \frac{Y}{N \cdot RTT} = \frac{\frac{1}{RTT}(2-\beta)\frac{1}{p}}{\sqrt{b^2 + 4a\left(c + \frac{1}{p}\right)} + (1-\beta)b + \frac{\beta(2-\beta)}{2}} \qquad (5)$$

In case that $W_{max}\beta >> 2S_{max}$, for a fixed S_{min}, $N_1 >> N_2$. Therefore, the sending rate of BIC mainly depends on the linear increase part, and for small values of p, the sending rate can be approximated as follows:

$$R \approx \frac{1}{RTT}\sqrt{\frac{S_{\max}}{2}\frac{2-\beta}{\beta}\frac{1}{p}} \quad \text{when } W_{max}\beta >> S_{max} \qquad (6)$$

Note that for a very large window, the sending rate becomes independent of S_{min}. Eqn. (6) is very similar to the response function of AIMD [18] denoted as follows.

$$R_{AIMD} \approx \frac{1}{RTT}\sqrt{\frac{\alpha}{2}\frac{2-\beta}{\beta}\frac{1}{p}}$$

For a very large window, the sending rate of BIC is close to the sending rate of AIMD with increase parameter $\alpha = S_{max}$

Next, we consider the case when $W_{max}\beta \le 2S_{max}$, then $N_1 = 0$, Assuming $1/p >> S_{min}$, we get W_{max} as follows.

$$W_{\max} \approx \frac{1}{\left(\log_2\left(\frac{W_{\max}\beta}{S_{\min}}\right) + 2(1-\beta)\right)p}$$

By solving the above equation using function *LambertW(y)* [21], which is the only real solution of $x \cdot e^x = y$, we can get a closed-form expression for W_{max}.

$$W_{\max} = \frac{\ln(2)}{LambertW\left(\frac{4\ln(2)\beta e^{-2\ln(2)\beta}}{pS_{\min}}\right)p}$$

then,

$$R = \frac{Y}{N \cdot RTT} = \frac{\frac{1}{RTT}\frac{1}{p}}{\log_2\left(\frac{w_{\max}\beta}{S_{\min}}\right) + 2} \approx \frac{W_{\max}}{RTT}\left(1 - \frac{2\beta}{\log_2\left(\frac{W_{\max}\beta}{S_{\min}}\right) + 2}\right)$$

When $2\beta << \log(W_{max}\beta / S_{min}) + 2$,

$$R \approx \frac{W_{\max}}{RTT} \qquad (7)$$

Note that when $W_{max}\beta \le 2S_{max}$, the sending rate becomes independent of S_{max}.

In summary, the sending rate of BIC is proportional to $1/p^d$, with $1/2 < d < 1$. As the window size increases, d decreases from 1 to 1/2. For a fixed β, when the window size is small, the sending rate is a function of S_{min} and when the window size is large, a function of S_{max}. Our objective is that when window is small, the protocol is TCP-friendly; and when the window is large, it is more RTT fair and gives a higher sending rate than TCP. We can now achieve this objective by adjusting S_{min} and S_{max}. Before we give details on how to set these parameters, let us examine the RTT fairness of BIC.

2) *RTT unfairness of BIC*

As in Section IV, we consider the RTT unfairness of a protocol under the synchronized loss model. Recall that the RTT unfairness of a protocol with a response function $\frac{1}{RTT}\frac{c}{p^d}$ is $(RTT_2 / RTT_1)^{1/(1-d)}$

From the previous section, we know that as the window size increases, the value of *d* of BIC decreases from 1 to 1/2. That is, with a high bandwidth, BIC has the same RTT unfairness as the TCP; while with a low bandwidth, BIC has the same RTT unfairness as STCP. However, since under small windows, synchronized loss is less frequent, we believe that this unfairness can be managed to be low. We verify this in Section VII.

3) Setting the parameters

In this section, we discuss a guideline to determine the preset parameters of BIC: β, S_{min}, S_{max} and *low_window* in Section VI-A.

From (6) and (7), we observe that reducing β increases the sending rate. Reducing β also improves utilization. However it hurts convergence since larger window flows give up their bandwidth slowly. From the equations, we can infer that β has a much less impact on the sending rate than S_{max}. So it is easier to fix β and then adjust S_{min} and S_{max}. We choose 0.125 for β. Under steady state, this can give approximately 94% utilization of the network. STCP chooses the same value for β.

For a fixed $\beta = 0.125$, we plot the response function as we vary S_{max} and S_{min}. Figure 6 shows the response function for different values of S_{max}. As S_{max} increases, the sending rate increases only for low loss rates using 1e-4 as a pivot. S_{max} allows us to control the scalability of BIC for large windows. We cannot increase S_{max} arbitrarily high since it effectively increases the area of RTT unfairness (the area where the slope is larger than TCP's). Recall that when $W_{max}\beta \leq 2S_{max}$, the protocol is less RTT fair. This area needs to be kept small to reduce the RTT unfairness of the protocol.

Figure 7 plots the response function for various values of S_{min}. As we reduce S_{min}, the sending rate reduces around high loss rates (i.e., small windows) and the cross point between TCP and BIC moves toward a lower loss rate. Since we can set *low_window* to the window size corresponding to the cross point, reducing S_{min} improves TCP friendliness. However, we cannot reduce S_{min} arbitrarily low because it makes the slope of the response function steeper before merging into the linear growth area, worsening RTT unfairness. HSTCP crosses TCP at window size of 31, and STCP at 16.

For a fixed $\beta = 0.125$, we plot the response function of BIC with $S_{max} = 32$ and $S_{min} = 0.01$ in Figure 8. For comparison, we plot those of AIMD ($\alpha = 32$, $\beta = 0.125$), HSTCP, STCP, and TCP. We observe that BIC crosses TCP around p = 1e-2 and it also meets AIMD around p=1e-5 and stays with AIMD. Clearly, BIC sets an upper bound on TCP friendliness since the response functions of BIC and TCP run in parallel after some point (at p=1e-5). BIC's TCP-friendliness under high loss rates is comparable to STCP's, but less than HSTCP's. BIC crosses TCP at the window size of 14 (lower than STCP and HSTCP). The sending rate of BIC over extremely low loss rates (1e-8) is less than that of STCP and HSTCP.

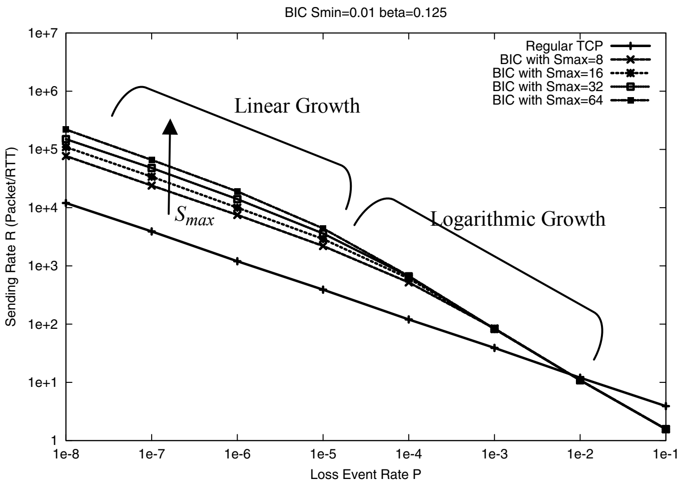

Fig. 6: Response functions of BIC for different values of S_{max}.

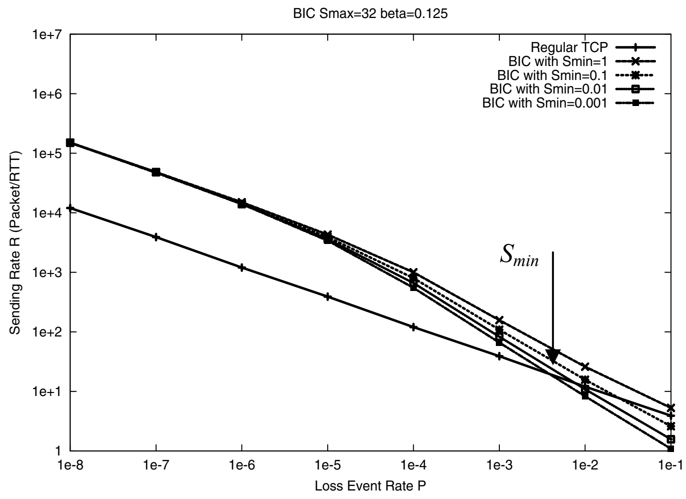

Fig 7: Response functions of BIC for different values of S_{min}.

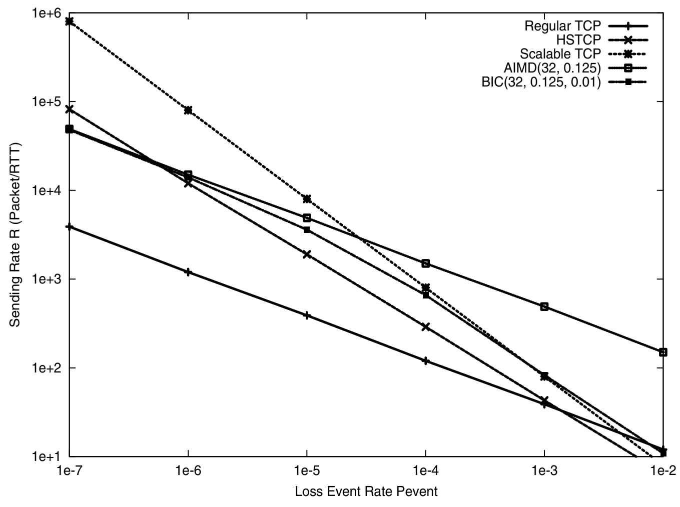

Fig. 8: Response functions of various protocols

VII. Simulation Study

In this section, we compare the performance of BIC using simulation with that of HSTCP, STCP, and AIMD. Every experiment uses the same simulation setup described in Section II. Unless explicitly stated, the same amount of background traffic is used for all the experiments. In order to reduce noise in data sampling, we take measurement only in the second half of each run. We evaluate BIC, AIMD, HSTCP, and STCP for the following properties: bandwidth utilization, TCP friendliness, RTT unfairness, and convergence to fairness. For BIC, we use S_{max}=32, S_{min}=0.01, and $\beta = 0.125$, and for AIMD, $\alpha = 32$ and $\beta = 0.125$. All high-speed protocols are implemented by modifying TCP/SACK.

Utilization: In order to determine whether a high-speed protocol uses available bandwidth effectively under high-bandwidth environments, we measure bandwidth utilization for 2.5 Gbps bottleneck. Each test consists of two high-speed flows of the same type and two long-lived TCP flows. In each test, we measure the total utilization of all the flows including background traffic. In drop tail, all protocols give approximately 97% utilization, and in RED, AIMD and BIC consume about 95% of the total bandwidth while HSTCP and STCP consume about 92%. The drop tail experiment consumes more bandwidth because drop tail allows flows to fill up the network buffers. Note buffers are part of the capacity.

RTT Fairness: In this experiment, two high-speed flows with a different RTT share the bottleneck. The RTT of flow 1 is 40 ms. We vary the RTT of flow 2 among 40ms, 120ms, and 240ms. The bottleneck link delay is 10ms. We run two groups of simulation, each with different values of bottleneck bandwidth: 2.5 Gbps, and 100 Mbps. This setup allows the protocols to be tested for RTT fairness for different window sizes. According to our analysis in Section VI, around small window sizes, BIC shows the worst RTT unfairness. BIC has window sizes about 7000 (loss rate 0.0003) for 2.5Gbps and 300 (loss rate 0.004) for 100Mbps.

Tables 2 and 3 show the results for the runs in 100Mbps and 2.5Gbps, respectively. We show the results of drop tail only. The RTT unfairness under RED is close to the inverse of RTT ratio for every protocol. We omit the RED figures. It can be seen that the RTT unfairness of BIC is relatively comparable to AIMD. This result verifies our analysis in Section VI. In Table 2, the performance of BIC does not deteriorate much while HSTCP and STCP have improved RTT unfairness. This is because in 100 Mbps, the window size of all the flows is much smaller than in 2.5Gbps run. Therefore, the degree of synchronized loss is very low. Although the RTT unfairness of BIC gets worse around this window size by the analysis, this deficiency gets compensated by lack of synchronized loss so that it does not have much performance degradation. Overall, its RTT unfairness is much better than HSTCP and STCP. HSTCP and STCP tend to starve long RTT flows under high bandwidth environments.

Table 2: The throughput ratio of protocols under 100 Mbps

Inverse RTT Ratio	1	3	6
AIMD	1.04	7.08	25.77
BIC	0.99	11.78	27.48
HSTCP	1.06	8.86	39.12
STCP	0.95	19.05	64.97

Table 3: The throughput ratio of protocols under 2.5Gbps

Inverse RTT Ratio	1	3	6
AIMD	1.11	6.68	22.03
BIC	0.88	11.96	40.04
HSTCP	1.01	29.19	107.90
STCP	1.01	127.23	389.13

TCP-friendliness: We run four groups of tests, each with different bottleneck bandwidth. Each group consists of four independent runs, each with a different type of high-speed flows. In every run, the same number of network flows (including background) is used.

Figure 9 shows the percentage of bandwidth share by each flow type under drop tail. (RED gives approximately similar results as drop tail.) Three flow types are present: background (web traffic and small TCP flows), long-lived standard TCP/SACK flows, and high-speed flows.

Under bandwidth below 500Mbps, the TCP friendliness of BIC is comparable to that of STCP. At 20Mbps, long-lived TCP and background flows with BIC flows consume slightly less bandwidth than those with HSTCP and STCP. However, in HSTCP and STCP simulations, the unused bandwidth is slightly more than that in BIC. That is, while BIC consumes more bandwidth than HSTCP and STCP, it takes bandwidth not only from TCP, but also from unused bandwidth. However, AIMD is too aggressive, and it takes too much bandwidth from TCP.

For 500Mbps and 2.5Gbps, the amount of shares by background and long-lived TCP flows substantially reduce due to TCP's limitation to scale its bandwidth usage in high-bandwidth. Under 500Mbps, STCP, BIC, and AIMD use approximately the same share of bandwidth. Under 2.5Gbps, the bandwidth share of background TCP traffic is very small. STCP becomes most aggressive. BIC becomes friendlier to TCP.

To sum, BIC achieves good TCP fairness for all window sizes. Around high loss rates where TCP performs well, its TCP friendliness is comparable to STCP's. The result closely follows our analysis in Section VI-B.

Fairness: Synchronized loss has impact also on bandwidth fairness and convergence time to the fair bandwidth share. In this experiment, we run 4 high-speed flows with RTT 100ms. Two flows start earlier randomly in [0:60] seconds, and the other two flows start later randomly in [100:160] seconds. The total simulation time is 600 seconds. The bottleneck link bandwidth is 2.5Gbps. For this experiment, we measured the fairness index [20] at each 50-second interval, and we take samples only after the first 100 seconds. This result gives an indication on (1) how fast it converges to a fair share, and (2) even after convergence to a fair share, how much it oscillates around the fair share.

Figure 10 shows the result for drop tail routers. At time around 150 seconds, the second set of two connections just start while the first two connections consuming almost all the bandwidth. Therefore, the difference between these four connections is the largest, and hence the fairness index is the lowest. As the latter two connections get more bandwidth, the fairness index increases. The protocol whose fairness index reaches 1 the fastest has the fastest convergence speed. BIC and AIMD give the best results, and they quickly converge to the fair share (where the fairness index is close to 1). STCP is the worst, and even after 600 seconds, the fairness index is still below 0.95. We observe that the fairness index of all protocols first increases fast, and then slowly reaches to 1. The reason is that the congestion windows of the latter two connections first increase exponentially in the slow start state; after the first loss event, they enter the congestion avoidance state, and increase slowly as specified by each protocol.

Figure 11 shows the result for RED routers. We observe much better convergence for HSTCP and STCP than in drop tail. This is because RED does not incur as much synchronized loss as drop tail. However, after convergence, HSTCP with RED shows a larger oscillation around the fair share than HSTCP with drop tail, and also than the other three protocols with RED. We note that the fairness index of HSTCP increases earlier than others in Figure 11. The reason is the latter two connections in HSTCP simulation start earlier than those in other simulations, since the simulation script randomly generates the starting time of each connection.

VIII. Related Work

As HSTCP and STCP have been discussed in detail in this paper, we examine other high-speed protocols that are not covered by this paper.

Recent experiment [9] indicates that TCP can provide good utilization even under 10Gbps when the network is provisioned with a large buffer and drop tail. However, the queue size of high-speed routers is very expensive and often limited to less than 20% of the bandwidth and delay products. Thus, generally, TCP is not suitable for applications requiring high bandwidth. FAST [7] modifies TCP-Vegas to provide a stable protocol for high-speed networks. It was proven that TCP can be instable as delay and network bandwidth increase. Using delay as an additional cue to adjust the window, the protocol is shown to give very high utilization of network bandwidth and stability. It fairness property is still under investigation. XCP [5] is a router-assisted protocol. It gives excellent fairness, responsiveness, and high utilization. However, since it requires XCP routers to be deployed, it cannot be incrementally deployed.

IX. Conclusion

The significance of this paper is twofold. First, it presents RTT fairness as an important safety condition for high-speed congestion control and raise an issue that existing protocols may have a severe problem in deployment due to lack of RTT

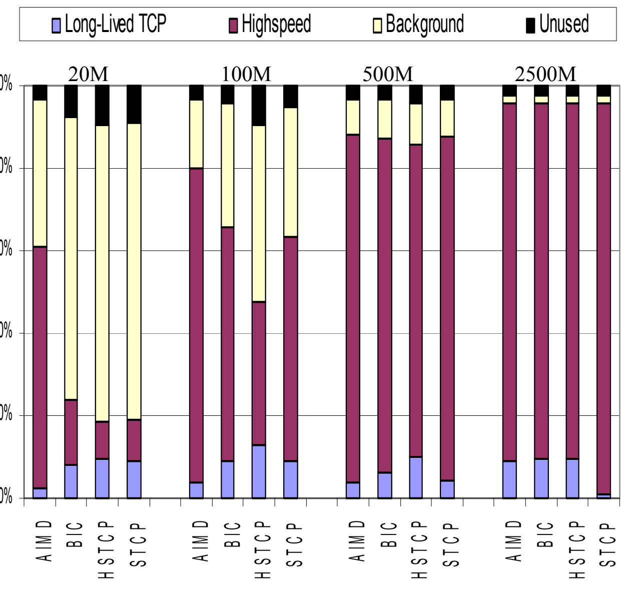

Fig 9: TCP friendliness for various bandwidth networks (DROPTAIL)

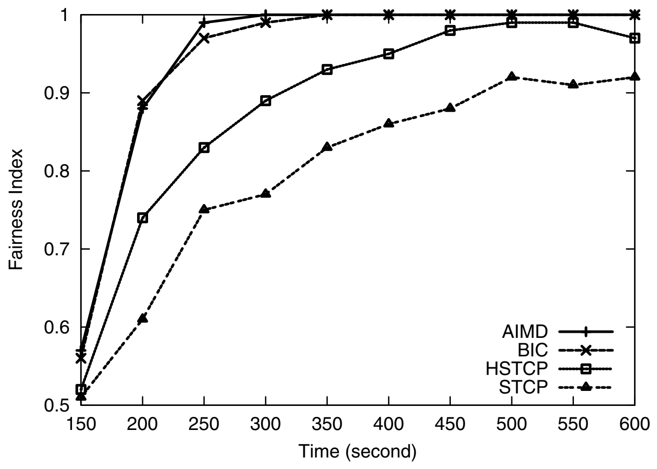

Fig. 10: Fairness index over various time scales (DROP TAIL)

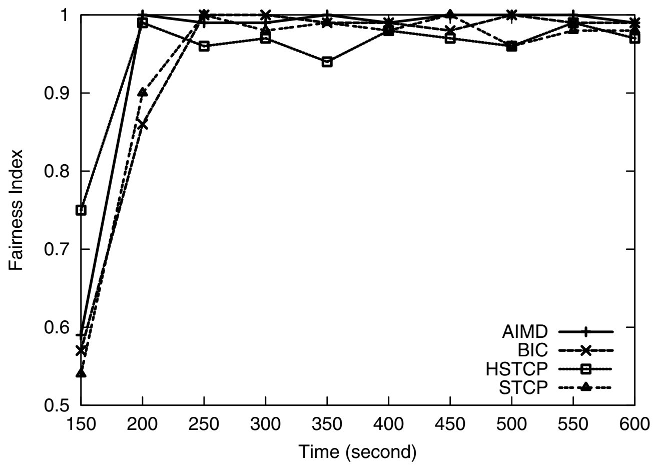

Fig. 11: Fairness index over various time scales (RED)

fairness under drop tail. RTT fairness has been largely ignored in designing high-speed congestion control. Second, this paper presents a new protocol that can support RTT fairness, TCP friendliness, and scalability. Our performance study indicates that it gives good performance on all three metrics.

We note that the response function of BIC may not be the only one that can satisfy the three constraints. It is possible that there exists a better function that utilizes the tradeoff among the three conditions. This is an area of more research. Another point of discussion is that high-speed networks can greatly benefit from the deployment of AQM. Our work supports this case since a well-designed AQM can relieve the protocol from the burden of various fairness constraints caused by synchronized loss.

One possible limitation of BIC is that as the loss rate reduces further below 1e-8, its sending rate does not grow as fast as HSTCP and STCP. This is because of the lower slope of its response function. Hence, it may seem that there is a fundamental tradeoff between RTT fairness and scalability. We argue it is not the case. Under such a low loss rate, most of loss is due to signal or "self-inflicted", i.e., loss is created because of the aggressiveness of the protocol. As a protocol gets more aggressive, it creates more loss and thus, needs to send at a higher rate for a given loss rate. The utilization of the network capacity is more important under such low loss rates, which is determined mostly by the decrease factor of congestion control. For TCP, the utilization is 75% and for STCP and BIC, around 94% (these numbers are determined from β). In addition, we believe that by the time that much higher bandwidth (in the order of 100 Gbps) becomes available, the network must have more advanced AQM schemes deployed so that we can use a higher slope response function. Then again, a less aggressive (i.e., lower slope) protocol, however, is less responsive to available bandwidth; so short file transfers will suffer. There are many facets to the problems, requiring more research efforts. We believe that fast convergence to efficiency requires a separate mechanism that detects the availability of unused bandwidth which has been an active research area lately ([22, 23]). We foresee that advance in this field greatly benefits the congestion control research.

REFERENCES

[1] S. Floyd, S. Ratnasamy, and S. Shenker, "Modifying TCP's congestyion control for high speeds", http://www.icir.org/floyd/hstcp.html, May 2002

[2] S. Floyd, "HighSpeed TCP for large congestion windows", IETF, INTERNET DRAFT, draft-floyd-tcp-highspeed-01.txt, 2003

[3] S. Floyd, "Limited slow-start for TCP with large congestion windows", IETF, INTERNET DRAFT, draft-floyd-tcp-slowstart-01.txt, 2001

[4] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks", Submitted for publication, December 2002

[5] Dina Katabi, M. Handley, and C. Rohrs, "Internet congestion control for high bandwidth-delay product networks." ACM SIGCOMM 2002, Pittsburgh, August, 2002

[6] Y. Gu, X. Hong, M. Mazzucco, and R. L. Grossman, "SABUL: A high performance data transport protocol", Submitted for publication, 2002

[7] C. Jin, D. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, H. Newman, F. Paganini, S. Ravot, and S. Singh, "FAST Kernel: Background theory and experimental results". Presented at the First International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet 2003), February 3-4, 2003, CERN, Geneva, Switzerland

[8] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast long-distance networks", Tech. Report, Computer Science Department, NC State University, 2004

[9] S. Ravot, "TCP transfers over high latency/bandwidth networks & Grid DT", Presented at First International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet 2003), February 3-4, 2003, CERN, Geneva, Switzerland

[10] T. Dunigan, M. Mathis, and B. Tierney, "A TCP tuning daemon", in Proceedings of SuperComputing: High-Performance Networking and Computing, Nov. 2002

[11] M. Jain, R. Prasad, C. Dovrolis, "Socket buffer auto-sizing for maximum TCP throughput", Submitted for publication, 2003

[12] J. Semke, J. Madhavi, and M. Mathis, "Automatic TCP buffer tuning", in Proceedings of ACM SIGCOMM, Aug. 1998

[13] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, S. Tuecke., "GridFTP protocol specification". GGF GridFTP Working Group Document, September 2002.

[14] PFTP : http://www.indiana.edu/~rats/research/hsi/index.shtml

[15] H. Sivakumar, S. Bailey, and R. L. Grossman, "PSockets: The case for application-level network striping for data intensive applications using high speed wide area networks", in Proceedings of SuperComputing: High-Performance Networking and Computing, Nov 2000

[16] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker, "Dynamic behavior of slowly responsive congestion controls", In Proceedings of SIGCOMM 2001, San Diego, California.

[17] S. Floyd, and E. Kohler, "Internet research needs better models", http://www.icir.org/models/bettermodels.html, October, 2002

[18] S. Floyd, M. Handley, and J. Padhye, "A comparison of equation-based and AIMD congestion control", http://www.icir.org/tfrc/, May 2000

[19] 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, July 1997

[20] D. Chiu, and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks", Journal of Computer Networks and ISDN, Vol. 17, No. 1, June 1989, pp. 1-14

[21] R. M. Corless, G. H. Gonnet, D.E.G. Hare, D. J. Jeffrey, and Knuth, "On the LambertW function", Advances in Computational Mathematics 5(4):329-359, 1996

[22] C. Dovrolis, and M.Jain, ``End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput". In Proceedings of ACM SIGCOMM 2002, August 2002.

[23] K. Harfoush, A. Bestavros, and J. Byers, "Measuring bottleneck bandwidth of targeted path segments", In Proceedings of IEEE INFOCOM '03, April 2003.