LTCP: Improving the Performance of TCP in Highspeed Networks *

Sumitha Bhandarkar, Saurabh Jain and A. L. Narasimha Reddy
Dept. of Electrical Engineering,
Texas A & M University

{sumitha,saurabhj,reddy}@ee.tamu.edu

ABSTRACT

In this paper, we propose Layered TCP (LTCP for short), a set of simple modifications to the congestion window response of TCP to make it more scalable in highspeed networks. LTCP modifies the TCP flow to behave as a collection of virtual flows to achieve more efficient bandwidth probing. The number of virtual flows emulated is determined based on the dynamic network conditions by using the concept of *virtual layers*, such that the convergence properties and RTT-unfairness behavior is maintained similar to that of TCP. In this paper, we provide the intuition and the design for the LTCP protocol modifications and evaluation results based on ns-2 simulations and Linux implementation. Our results show that LTCP has promising convergence properties, is about an order of magnitude faster than TCP in utilizing high bandwidth links, employs few parameters and retains AIMD characteristics.

Categories and Subject Descriptors: C.2.2 [**Computer Communication Networks**]: Network Protocols - *TCP Congestion Control*.

General Terms: Design, Experimentation, Theory.

Keywords: AIMD, Congestion Control, Highspeed Networks, Protocol Design.

1. INTRODUCTION

Over the past few decades the traffic on the Internet has increased by several orders of magnitude. However, the Internet still remains a stable medium for communication. This stability has been attributed primarily to the wide-spread use of TCP [1] which responds to congestion such that multiple flows can co-exist on a bottleneck link and share the available bandwidth equitably (when RTTs are similar). At the heart of the TCP congestion control algorithms is the use of simple *additive increase multiplicative decrease*(AIMD) policy for moderating the sending rate, using a congestion window. When there are no losses in the network the congestion window is increased by one for each RTT, hence increasing the sending rate. Upon a loss of packet, the window and in turn the sending rate, is reduced by half. This simple policy has worked remarkably well in the networks of the past and to a large extent the present, because very high capacity links (greater than several hundreds of Mbps or Gbps) were available only at the core of the network where several thousands of flows got multiplexed. But in the recent past, there has been an increase in the use of high capacity links for connecting research labs end-to-end for fast exchange of large amounts of data. The availability of inexpensive gigabit NICs and fast computers indicate that an average end user could have access to high capacity networks, in the not-too-distant future. In such an environment, where the density of flows is low and the available per-flow capacity is high, the TCP mechanism of increasing by just one packet per RTT tends to be too conservative, while at the same time the window reduction by a factor of half tends to be too drastic. This results in inefficient link utilization.

In this paper we propose a simple scheme for making the AIMD algorithms used by TCP more efficient in probing for the available link bandwidth. The proposed scheme, which we call LTCP, uses the concept of *virtual layers* or *flows* to increase the pace at which the protocol probes for available bandwidth and is responsive to dynamic changes in the network traffic. Several other schemes have also been proposed in the recent past for modifying TCP to resolve this problem. What sets the LTCP scheme presented in this paper apart is that, unlike the other schemes, it retains the time-tested AIMD behavior of TCP while at the same time providing performance benefits similar to, if not better than the other proposed schemes.

The rest of the paper is organised as follows - In Section 2 we present a detailed explanation of problem that has motivated this research and a brief overview of the current research in this area. This is followed by Section 3 where we provide the intuition, design and analyses pertaining to the proposed scheme. Results of the evaluation using ns-2 simulations and the Linux implementation are presented in Section 4. We conclude the paper in Section 5 by summarising our experiences and taking a look at the future work.

2. BACKGROUND AND RELATED WORK

To illustrate the problem that motivated this work, consider the following popular example - The throughput of a TCP connection is given by $T \simeq \frac{1.2*S}{R\sqrt{p}}$, where S is the packet size, R is the round trip time for the connection and p is the packet loss rate [2]. This means that for a standard TCP connection using a packet size of 1500 bytes over a connection with round trip delay of 200ms and packet loss rate of $1.0X10^{-5}$, the maximum throughput that can be achieved is 23.2Mbps. If the packet loss rate were reduced to $1.0X10^{-7}$ the maximum throughput could be increased to 232.4Mbps. Conversely, to achieve a throughput of 1Gbps, the packet loss rate required should be $5.4X10^{-9}$ or lower and for 10Gbps it should be $5.4X10^{-11}$. These loss rates are unreasonable - for the 10Gbps link, the loss rate translates to a loss of at most one packet in $1.85X10^{10}$ packets or at most one loss for every six hours ! Clearly, the standard TCP connections do not scale in high capacity networks.

*This work is supported in part by a grant from the Texas Higher Education Board, by an NSF grant ANI-0087372 and by Intel Corp.

Over the past few years, several solutions have been put forth for solving the problem of performance degradation on highspeed networks due to the use of TCP. These solutions can be classified into four main categories - a) Tuning the network stack b) Opening parallel TCP connections between the end hosts c) Modifications to the network infrastructure or use of non-TCP transport protocol d) Modifications to the TCP congestion control.

The traditional solution to improve the performance of TCP on high-capacity networks has been to tune some of the TCP parameters. Several auto-tuning schemes have been proposed such as [3], [4], [5] and [6]. [7] presents a comparison of some of these auto-tuning schemes. Tuning the stack improves the performance of TCP in high-speed networks significantly and could be used in conjunction with other schemes to achieve the best possible performance.

A number of other proposals have employed *network striping*, where the *application* is network-aware and tries to optimize the network performance by opening parallel TCP connections. Some examples of this approach include XFTP [8], GridFTP [9], storage resource broker [10] and [11]. In [12] the authors provide a library called *PSockets* (Parallel Sockets) to make it easier to develop applications that use network striping. While most of this work has been at the application level, in the MulTCP scheme[13] the authors present a mechanism where a single TCP flow behaves as a collection of several virtual flows. In [14], the authors describe a scheme for using virtual round trip time for choosing a tradeoff between fairness and the effectiveness of network usage. In [15] the authors describe a scheme for managing the striped TCP connections that could take different network paths. However, all the above mentioned schemes use a *fixed* number of parallel connections and choosing the *optimal* number of flows to maximise the performance without effecting the fairness properties is a significant challenge.

The third category of research has proposed modifications to the network infrastructure or use of non-TCP protocols for addressing the issue. In the XCP[22] scheme, the authors propose changes to the TCP congestion response function as well as the network infrastructure. In schemes like Tsunami[23], RBUDP[24] and SABUL /UDT [25] reliable data transfer is achieved by using UDP for data and TCP for control information. GTP[26] is also a rate based protocol, but focuses on max-min fair rate allocation across multiple flows to support multipoint-to-point data movement.

The final category of research focuses on modifications to the congestion response function of TCP itself for improving its performance in highspeed networks. Highspeed TCP [16] uses a congestion window response function that has a higher slope than TCP. Scalable TCP [17], uses multiplicative increase/multiplicative decrease response, to ensure that the congestion window can be doubled in a fixed number of RTTs. FAST TCP [18] relies on the delay-based bandwidth estimation of TCP Vegas [19] and is optimised for Gbps links. Bic-TCP [20] focuses on the RTT fairness properties by modifying the congestion response function using binary search with additive increase and multiplicative decrease. HTCP [21] uses response function similar to Highspeed TCP but modifies the increase parameter based on time since last drop. None of these solutions retain the AIMD behavior of TCP.

Similar to these other schemes, the solution we propose, ie, LTCP, modifies the congestion response function of TCP at the sender-side and requires no additional support from the network infrastructure or the receivers. LTCP can be thought of as an emulation of multiple flows at the transport level, with the key contribution *that the number of virtual flows adapt to the dynamic network conditions*. It retains the AIMD behavior of TCP. Layering schemes for probing the available bandwidth have been studied earlier in the context of multicast and video transfer, for example [27, 28]. LTCP, in contrast to this earlier body of work, uses layering *within the congestion control algorithm of TCP with per-ack window adaptation* to provide efficient bandwidth probing in high bandwidth links while maintaining fairness between multiple flows with similar RTTS.

3. LAYERED TCP

LTCP is based on the very simple concept of *virtual layers* or *virtual flows*. To start out with, every LTCP flow has only one layer. If the sending rate of the flow increases, without observing any losses, then based on some criteria, it increases the number of layers and continues to do so until a loss event is observed. When operating at any given layer K, the flow behaves as if it were a collection of K virtual flows, increasing the aggressiveness of probing for bandwidth. Just like the standard implementations of TCP, the LTCP protocol is ack-clocked and the congestion window of an LTCP flow changes with each incoming ack. However, since the LTCP flow operating at layer K emulates K virtual flows, it increases the congestion window by K packets per RTT. This is similar to the increase behaviour explored in [13].

For determining the number of layers that a flow should operate at, the following scheme is used. Suppose, each layer K is associated with a step-size δ_K. When the current congestion window exceeds the window corresponding to the last addition of a layer (W_K) by the step-size δ_K, a new layer is added. Thus,

$$W_1 = 0, \quad W_2 = W_1 + \delta_1, \quad ..... \quad W_K = W_{K-1} + \delta_{K-1} \quad (1)$$

and the number of layers $= K$, when $W_K \leq W < W_{K+1}$. Figure 1 shows this graphically.

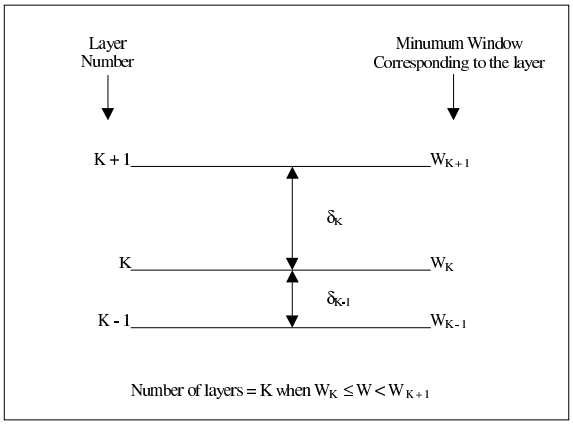


Figure 1: Graphical Perspective of Layers in LTCP

The step size δ_K associated with the layer K should be chosen such that convergence is possible when several flows share the bandwidth. Consider the simple case when the link is to be shared by two LTCP flows. Say, the flow that started earlier operates at a higher layer K_1 (with a larger window) compared to the later-starting flow operating at a smaller layer K_2 (with the smaller window). In the absence of network congestion, the first flow increases the congestion window by K_1 packets per RTT, whereas the second flow increases by K_2 packets per RTT. In order to ensure that the first flow does not continue to increase at a rate faster than the second flow, it is essential that the first flow adds layers at a rate

slower than the second flow. Thus, if δ_{K_1} is the stepsize associated with layer K_1 and δ_{K_2} is the stepsize associated with layer K_2, then

$$\frac{\delta_{K_1}}{K_1} > \frac{\delta_{K_2}}{K_2} \tag{2}$$

when $K_1 > K_2$, for all values of $K_1, K_2 \geq 2$.

The design of the decrease behavior is guided by a rather similar reasoning - in order for two flows starting at different times to converge, the time taken by the larger flow to regain the bandwidth it gave up after a congestion event should be larger than the time it takes the smaller flow to regain the bandwidth it gave up. Suppose the two flows are operating at layers K_1 and K_2 ($K_1 > K_2$), and ω_1 and ω_2 is the window reduction of each flow upon a packet loss. After the packet drop, suppose the flows operate at layers $K_1^{'}$ and $K_2^{'}$ respectively. Then, the flows take $\frac{\omega_1}{K_1^{'}}$ and $\frac{\omega_2}{K_2^{'}}$ RTTs respectively to regain the lost bandwidth. From the above reasoning, this gives us -

$$\frac{\omega_1}{K_1^{'}} > \frac{\omega_2}{K_2^{'}} \tag{3}$$

The window reduction can be chosen proportional to the current window size or be based on the layer at which the flow operates. If the latter is chosen, then care must be taken to ensure convergence when two flows operate at the same layer but at different window sizes.

Equations 2 and 3 provide a simple framework for the congestion response function of TCP for the congestion avoidance phase in highspeed networks. The congestion window response in slow start is not modified, allowing the protocol to evolve with experimental slowstart algorithms such as [30]. At the end of slowstart the number of layers to operate at can easily be determined based on the window size.

Asymptotic convergence to fairness among flows of similar RTT can be assured for competing flows if they satisfy the constraints in Equations 2 and 3. It must be noted however, that when the flows have different RTTs, the above scheme could make the RTT unfairness worse than that of TCP. Since LTCP is ack-clocked similar to TCP, the window of an LTCP flow with shorter RTT grows faster than that of an LTCP flow with larger RTT. In addition, since each layer K is associated with a step size δ_K, it takes exactly δ_K/K RTTs for a flow to increase the number of layers to $(K+1)$. When two flows with different RTTs share a bottleneck link, the flow with the short RTT will be able to add layers faster than the flow with the larger RTT making the RTT unfairness worse. In order to compensate for this dependence of aggressiveness on RTT, we introduce the RTT compensation factor K_R and modify the per-ack behavior such that, an LTCP flow at layer K will increase the congestion window at the rate of $K_R * K$ for the successful receipt of one window of acknowledgements. The RTT compensation factor is made proportional to the RTT. In order to ensure that the flows do not become aggressive as queues build up, we make K_R dependent only on the propagation delay of the link. In our experiments we use the lowest measured RTT sample for choosing the value of K_R. This uniformly scales up the rate at which a flow increase its window, with respect to RTT. Since the RTT compensation factor K_R is constant for a given RTT and multiplicative in nature, it does not alter the Equations 2 and 3 for the flows operating at same RTT.

The key then is to determine appropriate values for the step size δ (or equivalently, the window size W_k at which the layer transitions occur) and the window reduction that satisfy the conditions in Equation 2 and Equation 3. Based on this choice, the RTT compensation factor K_R can be determined to provide RTT fairness to satisfy a given requirement (eg. similar to TCP or similar to rate based flow etc.).

For determining the parameters, we start with the decrease behavior and analyse behavior for flows with similar RTTs. Since, the key requirement we have is to retain the AIMD properties of TCP, the decrease behavior is chosen to be multiplicative. The window reduction is based on a factor of β such that -

$$\omega = \beta * W \tag{4}$$

Based on this choice for the decrease behavior we determine the appropriate increase behavior such that the conditions in Equation 2 and Equation 3 are satisfied. To provide an intuition for choice of the increase behavior, consider Eq. 3

$$\frac{\omega_1}{K_1^{'}} > \frac{\omega_2}{K_2^{'}} \tag{5}$$

In order to allow smooth layer transitions, we stipulate that after a window reduction due to a packet loss, at most one layer can be dropped i.e., a flow operating at layer K before the packet loss should operate at either layer K or $(K-1)$ after the window reduction. Based on this stipulation, there are four possible cases - (a) $K_1^{'} = K_1, K_2^{'} = K_2$ (b) $K_1^{'} = (K_1 - 1), K_2^{'} = K_2$ (c) $K_1^{'} = K_1, K_2^{'} = (K_2 - 1)$ and (d) $K_1^{'} = (K_1 - 1), K_2^{'} = (K_2 - 1)$. It is most difficult to maintain the convergence properties, when the larger flow does not reduce a layer but the smaller flow does, ie, $K_1^{'} = K_1, K_2^{'} = (K_2 - 1)$.

With this worst case situation, Eq. 3 can be written as -

$$\frac{\omega_1}{K_1} > \frac{\omega_2}{(K_2 - 1)} \tag{6}$$

If this inequality is maintained for adjacent layers, we can show by simple extension, that it can be maintained for all other layers. So consider $K_1 = K, K_2 = (K-1)$. Then, the above inequality is

$$\frac{\omega_1}{K} > \frac{\omega_2}{K-2} \tag{7}$$

Suppose, the window for flow 1 is $W^{'}$ when the packet loss occurs and the window of flow 2 is $W^{''}$ then, substituting Eq. 4 in the above equation, we have,

$$\begin{aligned} \frac{W^{'}}{K} &> \frac{W^{''}}{K-2} \\ \Rightarrow W^{'} &> \frac{K}{K-2} W^{''} \end{aligned} \tag{8}$$

In order for the worst case behavior ($K_1^{'} = K, K_2^{'} = (K-2)$) to occur, the window $W^{'}$ could be close to the transition to the layer $(K+1)$ and the window $W^{''}$ could have recently transitioned into layer $(K-1)$. In order to get the estimate of the worst case we substitute these values in the above equation to get -

$$W_{K+1} > \frac{K}{K-2} W_{K-1} \tag{9}$$

Based on this, we conservatively choose the increase behavior to be

$$W_K = \frac{K+1}{K-2} W_{K-1} \tag{10}$$

Note that alternate choices are possible. This is essentially a tradeoff between efficiently utilizing the bandwidth and ensuring convergence between multiple flows sharing the same link. While

it is essential to choose the relationship between W_K and W_{K-1} such that the condition in equation 9 is satisfied to ensure convergence, a very conservative choice would make the protocol slow in increasing the layers and hence less efficient in utilizing the bandwidth.

Now suppose we choose to add the second layer at threshold $W_2 = W_T$. Then, by recursively substituting, we have

$$W_K = \frac{K(K+1)(K-1)}{6} W_T \tag{11}$$

By definition, $\delta_K = W_{K+1} - W_K$ and hence we have,

$$\delta_K = \frac{K(K+1)}{2} W_T \tag{12}$$

By simple substitution, we can show that the inequality in Eq. 2 is satisfied. Also, since this scheme was designed with the worst case for the inequality in Eq. 3, that condition is satisfied as well, when two competing flows are at adjacent layers. The result for adjacent layers can then be easily extrapolated for non-adjacent layers. It can also be shown that when two flows operate at the same layer, the inequality in Eq. 3 is satisfied.

3.1 Choice of W_T and β

The choice of the threshold W_T, the window size at which the LTCP flows starts to increase the number of layers, determines the region where the increase of LTCP has similar increase behavior as TCP. We choose a value of 50 packets for W_T. This value is motivated by the fact that when the window scale option [29] is not turned on, the maximum window size allowed is 64Kb which is about 44 packets (of size 1500 bytes). The window scale option is used in highspeed networks, to allow the receiver to advertise large window size. In slower networks, when the window scale option is not turned on, the actual sending rate is capped by this window value. We choose to begin the aggressive bandwidth probing of LTCP beyond this threshold.

The relationship between W_K and K has been derived based on the stipulation that after a window reduction due to packet drop, at most one layer is dropped. In order to ensure this, we have to choose the parameter β carefully. The worst case for this situation occurs when the flow has just added the layer K and the window $W = W_K + \Delta$, when the packet drop occurs. In order to ensure that the flow does not go from layer K to $(K-2)$ after the packet drop, we need to ensure that

$$\beta W_K < \delta_{K-1} \tag{13}$$

(Ignoring the reduction due to Δ since we are computing the worst case behavior.) On simple substitution, this yields,

$$\beta < \frac{3}{K+1} \tag{14}$$

Thus, β should be chosen such that the above equation is satisfied. The first two columns in the Table in Fig. 2 shows the number of layers corresponding to the windowsize at layer transitions (W_K) with $W_T = 50$. For a 2.4Gbps link with an RTT of 150ms and packet size of 1500 bytes, the window size can grow to 30,000. The number of layers required to maintain full link utilization is therefore $K = 15$. Based on this, we conservatively choose $\beta = 0.15$ (corresponding to $K = 19$).

With this design choice, LTCP retains AIMD behavior. At each layer K, LTCP increases the window additively by K, and when a packet drop occurs, the congestion window is reduced multiplicatively by a factor of β.

3.2 Time to claim bandwidth and Packet Recovery time

The primary goal for designing the LTCP protocol is to be able to utilize available link bandwidth aggressively in highspeed networks. Here, we provide quantitative analysis for time taken by an LTCP flow (in terms of RTTs) for claiming available bandwidth and the packet loss recovery time. For this analysis, we consider the case where the RTT compensation factor $K_R = 1$.

Suppose the maximum window size corresponding to the available throughput is W_K. Then, time to increase the window to W_K can be obtained as the sum of the time to transition from layer 1 to 2, 2 to 3 and so on until layer K. In other words, the time to increase the window to W_K is -

$$T(\delta_1) + T(\delta_2) + ... + T(\delta_{K-2}) + T(\delta_{K-1})$$

where $T(\delta_K)$ is the time (in RTTs) for increasing the window from layer K to $(K+1)$. When the flow operates at layer K, to reach to the next layer, it has to increase the window by δ_K and the rate of increase is K per RTT. Thus $T(\delta_K)$ is given by $\frac{\delta_K}{K}$. Substituting this in the above equation and doing the summation we find that the time to reach a window size of W_K is

$$T(\delta_1) + \frac{(K-2)(K+3)}{4} W_T \tag{15}$$

Note that the above analysis assumes that slowstart is terminated before layering starts. The third column in the Table in Fig. 2 shows the speedup in claiming bandwidth compared to TCP, for an LTCP flow with $W_T = 50$, with the assumption that slowstart is terminated when window = W_T.

K	W_K	Speedup in Claiming Bandwidth	Speedup in Packet Loss Recovery Time
1	0	-	-
2	50	1.00	1.00
3	200	2.00	6.67
4	500	2.57	10.00
5	1000	3.17	13.33
6	1750	3.78	16.67
7	2800	4.40	20.00
8	4200	5.03	23.33
9	6000	5.67	26.67
10	8250	6.31	30.00
11	11000	6.95	33.33
12	14300	7.60	36.67
13	18200	8.25	40.00
14	22750	8.90	43.33
15	28000	9.56	46.67
16	34000	10.21	50.00
17	40800	10.87	53.33
18	48450	11.52	56.67
19	57000	12.18	60.00
20	66500	12.84	63.33

Figure 2: Comparison of LTCP (with W_T = 50 and β = 0.15) to TCP

An LTCP flow with window size W will reduce the congestion window by βW. It then starts to increase the congestion window at the rate of at least $(K-1)$ packets per RTT (since we stipulate that a packet drop results in the reduction of at most one layer). The packet loss recovery time then, for LTCP is $\frac{\beta W}{(K-1)}$. In case of TCP, upon a packet drop, the window is reduced by half, and after the

drop the rate of increase is 1 per RTT. Thus, the packet recovery time is $W/2$. The last column of Table in Fig.2 shows the speed up in packet recovery time for LTCP with $\beta = 0.15$ compared to TCP. Based on the conservative estimate that a layer reduction occurs after a packet drop, the speed up in the packet recovery time of LTCP compared to TCP is a factor of $3.33 * (K - 1)$.

3.3 Response Function

In order to understand the relationship between throughput of an LTCP flow and the drop probability p of the link, and provide the basis for determining the value of K_R, we present the following analysis. Fig. 3 shows the steady state behavior of the congestion window of an LTCP flow with a uniform loss probability model. Suppose the number of layers at steady state is K and the link drop probability is p. Let $W^{''}$ and $W^{'}$ represent the congestion window just before and just after a packet drop respectively. On a packet loss the congestion window is reduced by $\beta W^{''}$. Suppose the flow operates at layer $K^{'}$ after the packet drop. Then, for each RTT after the loss, the congestion window is increased at the rate of $K^{'}$ until the window reaches the value W'', when the next packet drop occurs. Since we stipulate that at most one layer can be dropped after the window reduction due to packet loss, the window behavior of the LTCP flow, in general, will look like Fig. 3 at steady state.

With this model, the time between two successive losses, say T_D, will be $\frac{\beta W^{''}}{K_R K^{'}}$ RTTs or $\frac{\beta W^{''}}{K_R K^{'}} * RTT$ seconds. The number of packets sent between two successive losses, say N_D, is given by the area of the shaded region in Fig. 3. This can be shown to be -

$$N_D \simeq \frac{\beta (W^{''})^2}{K_R K^{'}} (1 - \frac{\beta}{2}) \tag{16}$$

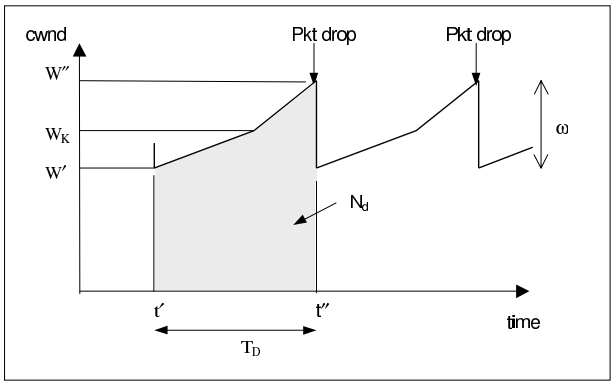


Figure 3: Analysis of Steady State Behavior

The throughput of such an LTCP flow can be computed as $\frac{N_D}{T_D}$. That is,

$$BW = \frac{W''}{RTT} (1 - \frac{\beta}{2}) \tag{17}$$

The expected number of packets sent between two losses N_D is $\frac{1}{p}$. By substituting this in Eq. 16, and solving for $W^{''}$ we have,

$$W^{''} = \sqrt{\frac{K_R K^{'}}{\beta (1 - \frac{\beta}{2}) p}} \tag{18}$$

$K^{'}$ is a discrete integer value, based on the Equation 11. Hence it can be approximated by $\lfloor (\frac{6W^{''}}{W_T})^{\frac{1}{3}} \rfloor$. Substituting this in Equation 18, we have,

$$W^{''} = (\frac{K_R (\frac{6}{W_T})^{\frac{1}{3}}}{\beta (1 - \frac{\beta}{2}) p})^{\frac{3}{5}} \tag{19}$$

Substituting in Eq. 17 we have

$$BW = \frac{C.K_R^{\frac{3}{5}}}{RTT.p^{\frac{3}{5}}} \tag{20}$$

$$where \quad C = \frac{(\frac{6}{W_T})^{\frac{1}{5}} (1 - \frac{\beta}{2})}{[\beta (1 - \beta)]^{\frac{3}{5}}}$$

Figure 4 shows the response function of LTCP when K_R is equal to 1. Response function of other highspeed proposals as well as that of unmodified TCP are shown for comparison. [21] states that the response function of H-TCP is similar to that of HS-TCP. From the figure, it can been seen that the slope of the response function of only LTCP is similar to that of regular TCP indicating that LTCP behaves in the AIMD fashion similar to TCP. At the same time, the curve is shifted along the Y-axis, indicating better scalability in highspeed networks, compared to TCP.

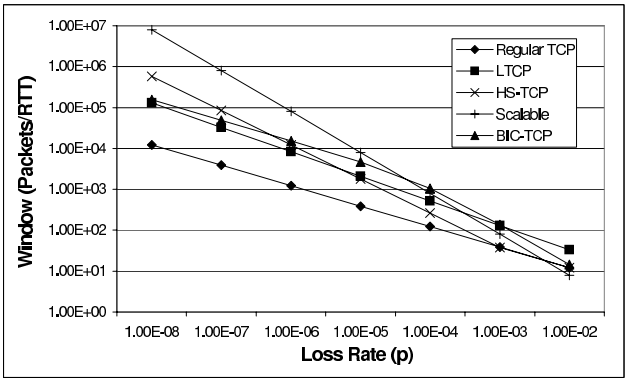


Figure 4: Response Function of Different Highspeed Protocols

3.4 RTT Unfairness and Choice of K_R

In this section we assess the RTT unfairness of LTCP under the assumptions of random loss model as well as synchronized loss models. Based on the discussion for the synchronized loss model, we derive the relationship between K_R and RTT to achieve RTT unfairness similar to that of TCP.

In [33], for a random loss model the probability of the packet loss λ is shown to be -

$$\lambda \propto \frac{A(w, RTT)}{A(w, RTT) + B(w, RTT)} \tag{21}$$

where $A(w, RTT)$ and $B(w, RTT)$ are the window increase and decrease functions respectively.

For LTCP $A(w, RTT) = K/w$ and $B(w, RTT) = \beta W$, when RTT compensation is not used. Substituting these values in the above equation and approximating $K \propto W^{1/3}$, we can calculate the loss rate λ as -

$$\lambda \propto \frac{1}{(1 + \beta W_s^{5/3})} \tag{22}$$

where W_s is the statistical equilibrium window.

It is clear from the above equation that the two LTCP flows experiencing the same loss probability, will have the same equilibrium

window size, regardless of the round trip time. However, throughput at the equilibrium point becomes inversely proportional to its round trip time since the average transmission rate r_s and is given by W_s/RTT.

The loss probability for TCP with similar assumptions is given by:

$$\lambda \propto \frac{1}{(1+0.5W_s)} \tag{23}$$

The equilibrium window size of the TCP flow does not depend on the RTT either. Therefore, for random losses the RTT dependence of window of an LTCP flow is same as TCP. Thus LTCP has window-oriented fairness similar to TCP and will not perform worse than TCP in case of random losses.

Because of the nature of current deployment of high bandwidth networks, it is likely that the degree of multiplexing will be small and as such, an assumption of synchronized loss model may be more appropriate. So we present here the details of the analysis with the synchronised loss model as well.

Following a similar analysis in [20], for synchronized losses, suppose the time between two drops is t. For a flow i with round trip time RTT_i and probability of loss p_i, the average window size is

$$W_i = \frac{\frac{1}{p_i}}{\frac{t}{RTT_i}} = \frac{RTT_i}{tp_i} \tag{24}$$

since the flow will send $\frac{1}{p_i}$ packets between two consecutive drop events and the number of RTTs between the two consecutive loss events $\frac{t}{RTT_i}$.

From Eq. 20, we have the bandwidth of an LTCP flow to be

$$BW = \frac{W_i}{RTT_i} = \frac{C.K_{R_i}^{\frac{3}{5}}}{RTT_i.p^{\frac{3}{5}}}$$
$$\Rightarrow p_i = \frac{C^{\frac{5}{3}}K_{R_i}}{W_i^{\frac{5}{3}}} \tag{25}$$

where C is a constant.

By substituting the above in Eq. 24 and simplifying we get,

$$W_i = (\frac{tK_{R_i}}{RTT_i})^{\frac{3}{2}}.C^{\frac{5}{2}} \tag{26}$$

When the RTT unfairness is defined as the throughput ratio of two flows in terms of their RTT ratios, the RTT unfairness for LTCP is -

$$\frac{(\frac{W_1}{RTT_1})(1-p_1)}{(\frac{W_2}{RTT_2})(1-p_2)} \simeq \frac{\frac{W_1}{RTT_1}}{\frac{W_2}{RTT_2}} = (\frac{RTT_2}{RTT_1})^{\frac{5}{2}}(\frac{K_{R_1}}{K_{R_2}})^{\frac{3}{2}} \tag{27}$$

(since $p << 1$).

The above equation shows that by choosing K_R appropriately, the RTT unfairness of LTCP flows can be controlled. For instance, choosing $K_R \propto RTT^{\frac{1}{3}}$, the RTT unfairness of the LTCP protocol will be similar to the AIMD scheme used in TCP. By choosing $K_R \propto RTT$, the effect of RTT on the scheme can be entirely eliminated and the LTCP protocol behaves like a rate controlled scheme independent of the RTT. By choosing an intermediate value such as, $K_R \propto RTT^{\frac{1}{2}}$, we can reduce the RTT unfairness of LTCP in comparison to TCP.

In general, suppose we choose, K_R proportional to RTT^{α}, where α is a constant. After a window reduction ω, suppose a flow operates at layer $K^{'}$. When RTT compensation is used it takes $\frac{\omega}{K_R * K^{'}}$ RTTs or $\frac{\omega * RTT}{K_R * K^{'}}$secs to regain the lost bandwidth. Suppose two flows with different RTTs are competing for the available bandwidth, Equations 3 can be re-written as

$$\frac{\omega_1 * RTT_1}{K_{R1} * K_1^{'}} > \frac{\omega_2 * RTT_2}{K_{R2} * K_2^{'}}$$

Substituting the value of ω and further solving it, we have,

$$\Rightarrow (\frac{RTT_1}{RTT_2})^{\alpha-1} < \frac{(K^{'}+1)}{K^{'}}$$
$$\Rightarrow (\alpha-1) < log(1+\frac{1}{K^{'}}) \quad \Rightarrow \alpha <= 1 \tag{28}$$

The above equation has been derived by assuming a worst case RTT ratio of 10 while taking the logarithm. It shows that when the RTT compensation factor is chosen based on the relationship $K_R \propto RTT^{\alpha}$ the value of α should be less that or equal to 1, to ensure asymptotic convergence. Since we aim to keep the behavior of LTCP similar to that of TCP we choose $K_R \propto RTT^{\frac{1}{3}}$.

3.5 Router Buffer Requirements

It has been conventional wisdom to set the router buffer size based on the classical rule of thumb of delay-bandwidth product of the link. For links with high bandwidth and high delays, this choice makes the required router buffer size very large. Research on sizing the router buffers [31] have shown that the classical rule of thumb is based on the desire to maintain high link utilization on the link, when a single flow tries to saturate the link. Since LTCP uses a less drastic decrease rule compared to TCP, the buffer size required by a single LTCP flow for keeping the link fully utilized all the time, is lower than that of TCP. Based on analysis similar to that in [31] it can be shown that the buffer size requirements for a single LTCP flow is $\frac{\beta}{(1-\beta)}(C * 2T_p)$ where C is the link capacity and T_p is the propagation delay of the link. Since we choose the value of β to be 0.15, the minimum buffer size required is $0.176 * C * 2T_p$, an 82% reduction compared to that of TCP.

4. RESULTS

To evaluate the LTCP protocol, we conducted simulations on the ns-2 simulator. Fig. 5 shows the network topology used in the simulations. The topology is a simple dumbbell network. The bottleneck link bandwidth is set to 1Gbps unless otherwise specified. The links that connect the senders and the receivers to the router have a bandwidth of 2.4Gbps. The end-to-end RTT is set to 120ms. The routers have the default queue size set to 5000 packets which is one third the delay-bandwidth product of the bottleneck link. DropTail queue management is used at the routers. The LTCP protocol is implemented by modifying the TCP/Sack1 agent. The unmodified TCP/Sack1 agent is used for TCP. The receiver advertised window is set to a large value to ensure that it does not interfere with the simulations. For the LTCP flows, the parameter W_T is set to 50 packets and the parameter β was set to 0.15. The traffic constitute of FTP transfer between the senders and receivers.

4.1 Basic Comparison with TCP

Since LTCP uses adaptive layering, it is capable of increasing its window size to the optimal value much faster than TCP. Also, when a packet loss occurs, the window reduction of LTCP is not as drastic as TCP. As a result the window adaptation of LTCP is much more efficient in utilizing the link bandwidth in highspeed networks. Fig. 6 shows congestion window of LTCP in comparison with that of TCP, when the network consists of only one flow.

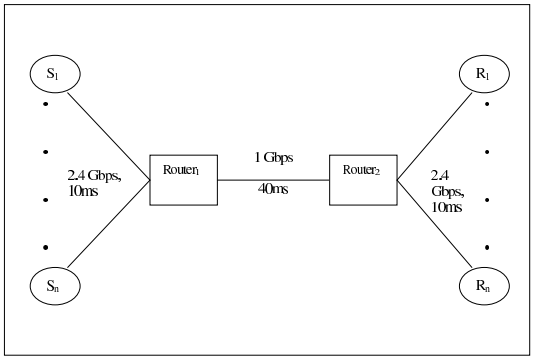
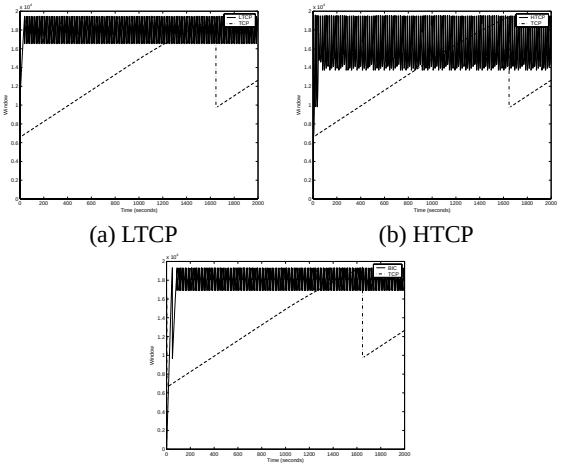


Figure 5: Simulation Topology

As seen from the figure, the congestion window of LTCP reaches the optimal value several orders of magnitude faster than the TCP flow. The comparison of windows for HTCP and BIC with TCP are included for reference. Since HTCP chooses the window reduction dynamically in the range (0.5, 0.8), the overall fluctuation in the HTCP window is slightly worse than that of LTCP and BIC.



(a) LTCP (b) HTCP

(c) BIC

Figure 6: Comparison of Congestion Window

The table in Fig. 7 shows the comparison between the goodput and average packet loss rates for different protocols at different bottleneck link bandwidths. The throughput is calculated over a period of 2000 seconds after the flow reaches steady state. Due to the large period for averaging the throughput and the buffer size of 5000 packets, TCP flow seems to be able to obtain reasonably high throughput. As seen from Fig.6, due to the large width of the TCP window cycle, using a smaller duration for measuring throughput would yield lower TCP throughput, depending on the window size during the measurement period. Since the high speed protocols operate close to the optimal value most of the time, the link utilization will be high even if the measurement duration is reduced. However, due to operating close to the optimal value, the congestion loss rate observed by highspeed flows is larger than that of TCP which due to under-utilization of the link sees lower congestion losses. Among the different highspeed protocols, LTCP and BIC have lower self-induced losses compared to HTCP while achieving similar goodput.

Bottleneck Link Bandwidth	TCP		LTCP		HTCP		BIC	
	Goodput (Mbps)	Packet Loss Rate (%)	Goodput (Mbps)	Packet Loss Rate (%)	Goodput (Mbps)	Packet Loss Rate (%)	Goodput (Mbps)	Packet Loss Rate (%)
100Mb	89.12	7.51E-05	96.15	4.07E-02	96.12	3.92E-02	95.24	6.39E-02
250Mb	222.91	8.89E-06	240.38	1.21E-02	240.37	1.77E-02	238.10	1.65E-02
500Mb	437.01	3.01E-06	480.77	4.83E-03	478.28	1.06E-02	476.19	5.22E-03
1Gb	817.58	7.47E-07	961.54	1.95E-03	952.10	6.32E-03	952.38	1.50E-03
2.4Gb	2103.07	1.18E-07	2307.69	6.10E-04	2277.29	3.15E-03	2277.45	2.93E-04

Figure 7: Goodput and Packet Loss Rate for Different Highspeed Protocols

4.2 Intra-protocol Fairness

In this experiment, we evaluate the fairness of LTCP flows to each other. Different number of LTCP flows are started at the same time (with random staggering to avoid synchronization) and the average per-flow bandwidth of each flow is noted. The table in Fig 8 shows that when the number of flows is varied, the maximum and the minimum per-flow throughputs remain close to the average, indicating that the per-flow throughput of each flow is close to the fair proportional share. This is verified by calculating the Fairness Index proposed by Jain et. al., in [32]. The Fairness Index being close to 1 shows that the LTCP flows share the available network bandwidth equitably. Similar behavior was observed with BIC and HTCP.

No. of Flows	Avg. per-flow Throughput (Mbps)	Min. per-flow Throughput (Mbps)	Max. per-flow Throughput (Mbps)	Standard Deviation	Jain Faimess Index
2	480.77	480.34	481.20	0.60	1.0000
4	240.39	240.28	240.55	0.12	1.0000
6	160.26	160.14	160.37	0.10	1.0000
8	120.19	120.16	120.31	0.05	1.0000
10	96.15	95.75	99.55	1.19	0.9999

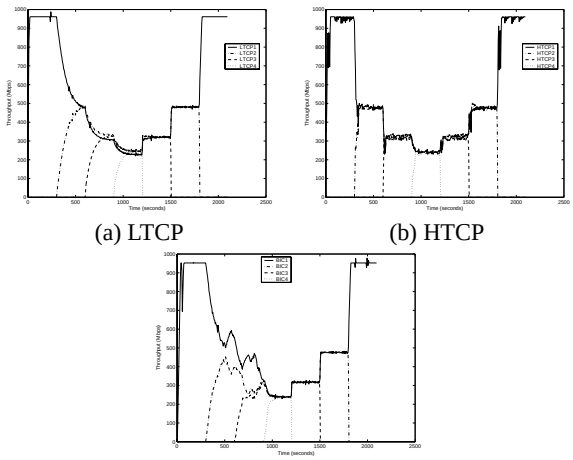
Figure 8: Fairness Among LTCP Flows

4.3 Dynamic Link Sharing

In the previous experiment with multiple flows, all the flows were started at about the same time and all different protocol flavors showed very good intra-protocol fairness. In this section, we evaluate the convergence properties when flows start and stop at different times, dynamically changing the available link bandwidth. The first flow is started at time 0, and allowed to reach steady state. A new flow is then added every 300 seconds. The flows last for 2100, 1500, 900 and 300 seconds respectively. Fig. 9 shows the throughput of each flow. From the graph we see that, the BIC flows take much longer to converge to fair share compared to LTCP and HTCP. When bandwidth becomes available because of the completion of a flow, all three protocols are capable of quickly increasing their sending rates and hence ensuring the link is fully utilized.

4.4 Effect of Random Losses

Fig. 4 shows the response curve of the different highspeed proposals. [21] states that the response function of H-TCP is similar to that of HS-TCP. In this simulation, we show the effect of the different response curves. We fix the capacity of the bottleneck link at 1Gbps and induce random losses on the link using a uniform loss model. Fig. 10 plots the throughput against the random loss rate. Note that the lossrate on the x-axis does not include the self-induced congestion losses. Packet loss rate of 10^{-7} due to channel



(a) LTCP (b) HTCP

(c) BIC

Figure 9: Dynamic Link Sharing

errors is comparable to the error rate in long haul fibre links [17]. As the random loss rate increases, the link utilization of the different protocols starts to deteriorate. The deterioration of LTCP and BIC are similar, whereas the deterioration of HTCP is slightly more drastic.

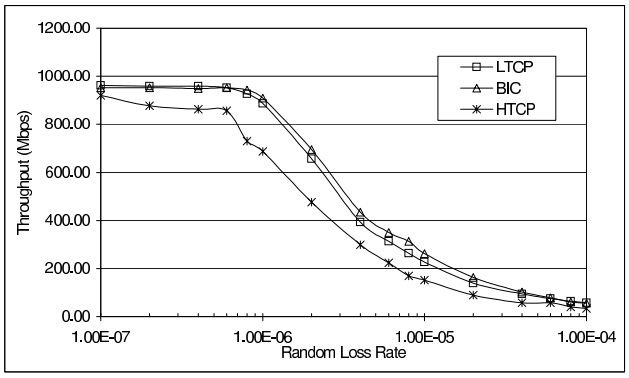


Figure 10: Effect of Random Losses on Different Highspeed Protocols

4.5 Impact of Bottleneck Link Buffer Size

In this experiment we study the impact of the bottleneck link buffer size on the performance of the different highspeed protocols. For the given topology, the bandwidth-delay product(BDP) is 15000 packets. We vary the bottleneck link buffer size from 1 times to 0.1 times the BDP. Fig. 11 shows the results. As seen from the graph, when the bottleneck link buffer size is at least 0.5 times the BDP, all the highspeed protocols maintain high link utilization. When the bottleneck link buffer size reduces below this, the throughput starts to degrade a little, with the degradation for HTCP being slightly worse than that of BIC or LTCP. As indicated in section 3.5, even when the buffer size at the bottleneck link router is low, LTCP can maintain high link utilization.

4.6 Interaction with TCP

In this section, we verify the effect of LTCP on regular TCP flows. It must be noted that the window response function of LTCP,

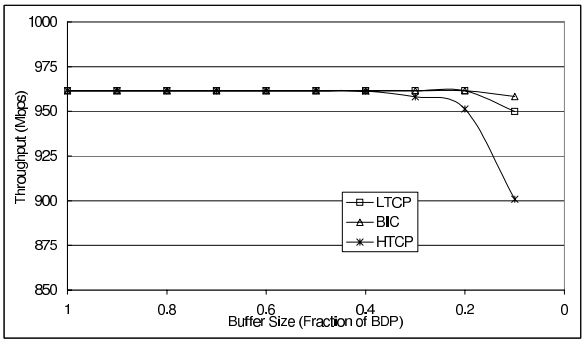


Figure 11: Impact of Bottleneck Link Buffer Size on Different Highspeed Protocols

BIC and HTCP are *designed* to be more aggressive than TCP in high speed networks. So a single flow of TCP cannot compete with a single flow of these highspeed protocols. We present these results here to show that the highspeed protocols do not starve the TCP flows for bandwidth. Fig.12 shows the results with one TCP flow sharing the bottleneck link with a highspeed flow. With an increase in the available bottleneck link capacity, the throughput achieved by the TCP flow slightly increases. The throughput obtained by the TCP flow is similar when competing with the different highspeed flows.

Bottleneck Link Bandwidth	LTCP		HTCP		BIC	
	TCP Throughput (Mbps)	LTCP Throughput (Mbps)	TCP Throughput (Mbps)	HTCP Throughput (Mbps)	TCP Throughput (Mbps)	BIC Throughput (Mbps)
100Mb	1.45	94.70	3.80	92.29	1.41	93.81
500Mb	4.50	476.27	7.53	471.82	3.77	472.42
1Gb	7.23	954.31	10.35	945.25	6.75	945.63

Figure 12: Interaction with TCP

4.7 RTT Unfairness

In our analyses we have showed that the RTT unfairness of LTCP can be tuned based on different requirements by modifying K_R. We choose K_R so that LTCP displays RTT unfairness similar to that of TCP, that is, the ratio of the throughput of two LTCP flows with different RTTs is proportional to the inverse square of the ratio of the RTTs. While, the choice of K_R can easily be modified to offer different levels of fairness, we present in this section the results with the chosen design. Each simulation consists of two flows with different RTTs competing for bandwidth on the bottleneck link. The RTT of the shorter link is fixed at 40ms, while varying the RTT of the larger link such that the RTTs have ratio 2, 3 and 4. The bottleneck link capacity is fixed at 1Gbps. Fig.13 shows the results in comparison with HTCP and BIC. Since the scaling factor for HTCP is chosen to provide linear unfairness (as opposed to the square unfairness of TCP), HTCP shows better performance. According to [20], the RTT unfairness of BIC is the same as that of TCP for high bandwidth, and the low RTT flows may starve the high RTT flows at low bandwidth. In this experiment, the RTT unfairness of BIC was observed to be a little worse than that of the inverse square unfairness of TCP.

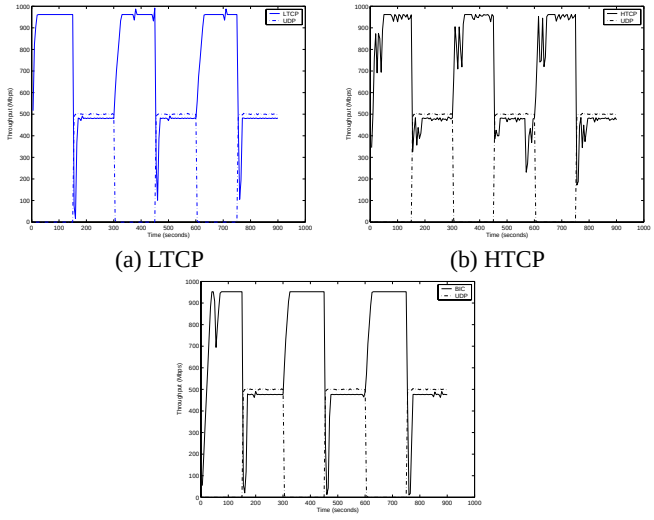
4.8 Interaction with Non-responsive Traffic

In order to evaluate how LTCP responds to the presence of traffic that does not respond to congestion, we conducted the following

RTT Ratio	LTCP	HTCP	BIC
2	3.36	1.85	9.74
3	7.63	2.78	16.61
4	13.65	3.68	26.96

Figure 13: RTT Unfairness

simulation. In this simulation, non-responsive on-off traffic was simulated using CBR/UDP source that sends data at half the bottleneck link capacity (500Mbps) for 150 seconds and then remains inactive for the next 150 seconds. Fig. 14 shows the throughput of the two flows computed over 5 second intervals. Results of similar experiments with BIC and HTCP are included for comparison. As seen from the graph, the response of all the three protocols is similar. In the presence of nonresponsive traffic all of them reduce their sending rate. When the non-responsive flows are not present, all of them quickly ramp up the sending rate.



(a) LTCP (b) HTCP

(c) BIC

Figure 14: Interaction with UDP Traffic

4.9 Emulation Results

We have implemented LTCP in the network stack of the Linux 2.4.25 kernel. The network stack in the 2.4.x kernel is quite sophisticated and supports several standards from the RFCs as well as features beyond those published in RFCs or IETF Drafts aimed to provide good network performance [34]. Our testbed consists of two off-the shelf Dell Optiplex GX260 workstations with Pentium 4 3.06GHz CPU, 512MB of RAM, Intel PRO/1000 MT gigabit NICs on to a 33MHz/32bit PCI bus. The two computers are connected using a copper Cat 6 cable. The 33MHz PCI bus limits the achievable throughput to around 750-800 Mbps. A packet delay scheduler patch made available on the Linux mailing list (by Stephen Hemminger) is used to generate uniform delay for all packets, so that larger RTTs can be emulated. This patch modifies the FIFO queue such that every packet queued is delayed for a fixed amount. This setup was used instead of the conventional sender-router-receiver setup, due to the limitation imposed by the PCI bus which in the router configuration would further reduce the bottleneck link capacity as it is shared by both incoming and outgoing interfaces.

The machine configured as the sender used the 2.4.25 kernel with the LTCP patch. The machine configured as the receiver used the 2.4.26-rc1 kernel with the packet delay scheduler patch. iperf [35] was used for generating traffic. We increased the socket buffers to allow maximum link utilization. The txqueuelen for the NIC was set to the default value of 1000. The backlog queue was modified to have a size of 1000.

We present here the results of the experiment comparing the performance of the standard Linux TCP (TCP-SACK) with that of LTCP at different RTTs for a 900 second transfer. The socket buffer size at both the sender and receiver is set to 32MB to ensure that a single flow can utilize all the available bandwidth. The experiment is run for 15 minutes (900 seconds) and is repeated four times. The table in Fig.15 shows the average number of bytes transferred and the average transfer rate. At low RTTs, both the TCP and LTCP flows manage to keep the link almost fully utilized and transfer about 75 GBytes of data. As the RTT increases, TCP takes longer to recover from packet losses and its performance starts to deteriorate. For an RTT of 120ms, which is comparable to that of the transatlantic links, the performance deterioration is significant and in 900 seconds, the TCP flow manages to transfer only about 22 GBytes. In contrast, a single LTCP flow transfers an average of about 65 GBytes.

	Low RTT (~20ms)		Medium RTT (~60ms)		High RTT (~120ms)	
	Bytes Sent (GB)	Average Sending Rate (Mbps)	Bytes Sent (GB)	Average Sending Rate (Mbps)	Bytes Sent (GB)	Average Sending Rate (Mbps)
TCP	75.08	716.25	54.58	520.50	22.55	215.00
LTCP	79.50	758.00	75.93	724.50	65.38	623.25

Figure 15: Linux Performance Test

We have conducted several other experiments to validate some of the results obtained in the ns-2 simulations. Currently, we are in the process of obtaining access to the Internet2 backbone so more experiments can be conducted with real traffic in the background.

5. CONCLUSIONS

In this paper we have proposed LTCP, a layered approach for modifying TCP for high-speed links. LTCP uses the concept of *virtual layers* to increase the congestion window when congestion is not observed over an extended period of time. When operating at layer K, LTCP uses modified additive increase (by K per RTT) and remains ack-clocked. The layered architecture provides flexibility in choosing the sizes of the layers for achieving different goals. This paper explored the design option that retains the AIMD characteristics of TCP.

We have shown through analysis and experimental evaluation that a single LTCP flow can adapt to nearly fully utilizing the link bandwidth on highspeed links. Other significant features of the chosen design are - (a) it provides a significant speedup in claiming bandwidth and in packet loss recovery times compared to TCP (b) multiple flows share the available link capacity equitably (c) RTT unfairness can be controlled by choosing K_R appropriately (d) requires only simple modifications to TCP's congestion response mechanisms and retains the AIMD behavior. Extensive experiments comparing LTCP to other proposals namely BIC and HTCP have shown that LTCP can produce similar or better performance than these schemes with the additional benefit that *the time-tested AIMD characteristics are retained*.

Currently we are in the process of evaluating the Linux implementation of LTCP on an Internet2 testbed and comparing its performance with BIC and HTCP. Comparative third party evaluation of LTCP against other proposals is also underway at SLAC, Stanford.

Our design is hinged on an early decision to use multiplicative decrease, and on the stipulation that at most one layer is dropped after a congestion event. In addition, we have chosen to use a constant value of 0.15 for the decrease factor β. A number of other possibilities exist for alternate designs of the general LTCP approach. We plan to pursue these options in future. LTCP improves on the loss-based congestion probing mechanism of TCP and makes it more aggressive. As part of our future work, we plan to investigate the behavior of loss-based mechanisms versus the rate-based mechanisms (such as FAST [18]) in high-speed environments and the possibility of a hybrid mechanism for reducing self induced losses while retaining the stability.

6. REFERENCES

[1] Sally Floyd, "Congestion Control Principles", *RFC 2914*, September 2000

[2] M. Mathis, J. Semske, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithms," *ACM Computer Communication Review, vol. 27(3)*, July 1997.

[3] Jeffrey Semke, Jamshid Mahdavi, and Matthew Mathis, "Automatic TCP Buffer Tuning", *Proceedings of ACM SIGCOMM*, October 1998.

[4] Eric Weigle and Wu-chun Feng, "Dynamic Right-Sizing: a Simulation Study", *Proceedings of IEEE International Conference on Computer Communications and Networks (ICCCN)*, October 2001.

[5] Brian L. Tierney, Dan Gunter, Jason Lee, Martin Stoufer and Joseph B. Evans, "Enabling Network-Aware Applications", *10th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, August 2001.

[6] Tom Dunigan, Matt Mathis and Brian Tierney, "A TCP Tuning Daemon", *SuperComputing (SC)* November, 2002.

[7] Eric Weigle and Wu-chun Feng, "A Comparison of TCP Automatic Tuning Techniques for Distributed Computing", *11th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, July 2002.

[8] Ostermann, S., Allman, M., and H. Kruse, "An Application-Level solution to TCP's Satellite Inefficiencies", *Workshop on Satellite-based Information Services (WOSBIS)*, November, 1996.

[9] J. Lee, D. Gunter, B. Tierney, B, Allcock, J. Bester, J. Bresnahan and S. Tuecke, "Applied Techniques for High Bandwidth Data Transfers Across Wide Area Networks", *Proceedings of International Conference on Computing in High Energy and Nuclear Physics*, September 2001.

[10] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC storage resource broker", *In Proc. CASCON'98 Conference*, Dec 1998.

[11] R. Long, L. E. Berman, L. Neve, G. Roy, and G. R. Thoma, "An application-level technique for faster transmission of large images on the Internet", *Proceedings of the SPIE: Multimedia Computing and Networking 1995*, Feb 1995.

[12] H. Sivakumar, S. Bailey and R. Grossman, "PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks", *Proc. of Super Computing*, November 2000.

[13] Jon Crowcroft and Philippe Oechslin, "Differentiated End-to-End Internet Services using a Weighted Proportional Fair Sharing TCP", ACM CCR, vol. 28, no. 3, July 1998.

[14] Thomas Hacker, Brian Noble and Brian Athey, "Improving Throughput and Maintaining Fairness using Parallel TCP", *Proceedings of IEEE Infocom 2004*, March 2004.

[15] Hung-Yun Hsieh and Raghupathy Sivakumar, "pTCP: An End-to-End Transport Layer Protocol for Striped Connections", *Proceedings of 10th IEEE International Conference on Network Protocols*, November 2002.

[16] Sally Floyd, "HighSpeed TCP for Large Congestion Windows", *RFC 3649* December 2003.

[17] Tom Kelly, "Scalable TCP: Improving Performance in HighSpeed Wide Area Networks", *ACM Computer Communications Review*, April 2003.

[18] Cheng Jin, David X. Wei and Steven H. Low, "FAST TCP: motivation, architecture, algorithms, performance", *IEEE Infocom*, March 2004.

[19] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance", *Proceedings of ACM SIGCOMM '94*, August 1994.

[20] Lisong Xu, Khaled Harfoush, and Injong Rhee, "Binary Increase Congestion Control for Fast Long-Distance Networks", To appear in *Proceedings of IEEE Infocom 2004*, March 2004.

[21] D.J. Leith and R. Shorten, "H-TCP Protocol for High-Speed Long Distance Networks", *PFLDNet 2004*, February 2004

[22] Dina Katabi, Mark Handley, and Chalrie Rohrs, "Congestion Control for High Bandwidth-Delay Product Networks", *Proceedings of ACM SIGCOMM 2002*, August 2002.

[23] README file of tsunami-2002-12-02 release. http://www.indiana.edu/ anml/anmlresearch.html

[24] Eric He, Jason Leigh, Oliver Yu and Thomas A. DeFanti, "Reliable Blast UDP : Predictable High Performance Bulk Data Transfer", *Proceedings of IEEE Cluster Computing*, September, 2002.

[25] H. Sivakumar, R. Grossman, M. Mazzucco, Y. Pan, and Q. Zhang, "Simple Available Bandwidth Utilization Library for High-Speed Wide Area Networks", to appear in *Journal of Supercomputing*, 2004.

[26] R.X. Wu and A.A. Chien, "GTP: Group Transport Protocol for Lambda-Grids", *4th IEEE/ACM International Symposium on Cluster Computing and the Grid*, April 2004.

[27] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast", *Proc. of ACM SIGCOMM '96*, August 1996.

[28] L. Vicisano, L. Rizzo, and J. Crowcroft, "TCP-like congestion control for layered multicast data transfer" *Proceedings of IEEE Infocom '98*, March 1998.

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

[30] S. Floyd, "Limited Slow-Start for TCP with Large Congestion Windows", *RFC 3742*, March 2004.

[31] G. Appenzeller, I. Keslassy and N. Mckeown, "Sizing router buffers", *Proceedings of ACM SIGCOMM'04*, August/September 2004.

[32] Dah-Ming Chiu and Raj Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks", *Computer Networks and ISDN Systems*, June 1989

[33] S. J. Golestani and K. K. Sabnani, "Fundamental observations on multicast congestion control in the Internet", In proc. of *IEEE INFOCOM'99*, March 1999.

[34] P. Sarolahti and A. Kuznetsov, "Congestion Control in Linux TCP", *Usenix 2002/Freenix Track, pp. 49–62*, June 2002.

[35] Iperf Version 1.7.0 $http://dast.nlanr.net/Projects/Iperf/$