LTCP: A Layered Congestion Control Protocol for 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 simple layering technique for the congestion window response of TCP to make it more scalable in highspeed networks. LTCP is a two dimensional congestion control framework - the macroscopic control uses the concept of layering to quickly and efficiently make use of the available bandwidth whereas microscopic control extends the existing AIMD algorithms of TCP to determine the per-ack behavior. We provide the general intuition and framework for the LTCP protocol modifications in this paper. Then, using a simple design, we illustrate the effectiveness of using layering for improving the efficiency, without sacrificing the convergence properties of TCP. Evaluation of the specified design is based on analysis, ns-2 based simulations and emulation on a Linux testbed. We 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 is easy to understand. The flexible framework opens a whole class of design options for improving the performance of TCP in highspeed networks.**

I. 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 congestion control algorithms of TCP [1]. The congestion control algorithms are designed such that bandwidth is shared fairly among flows with similar RTTs. However, these same congestion control algorithms hold TCP back from scaling to future networks with high-bandwidth links of the order of several Gbps.

The TCP congestion control algorithms use *additive increase multiplicative decrease*(AIMD) for moderating the congestion window. When there are no losses in the network, the window is increased by one for each RTT. Upon a loss of packet, the window is reduced by half. In other words, the AIMD parameter for increase is chosen such that the available bandwidth is probed conservatively, but the parameter for decrease upon a packet loss, is chosen to give up the bandwidth aggressively. While this has worked well for the networks in the past, for future networks that have capacity of the order of several hundreds of Mbps or Gbps, this could lead to highly degraded performance.

To illustrate the problem effectively, 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.

In this paper we propose a simple layering technique for the existing congestion response algorithms to make it scale in high-bandwidth networks. The idea of layering to probe and utilize the available bandwidth has been researched earlier in the context of video transmission on the Internet and in multicasting[27], [28]. The contribution of this paper is to extend this idea to the congestion control algorithms in TCP so that scalability can be achieved at the cost of minimal implementation overhead, while retaining many of the desirable characteristics that have made TCP the protocol of choice.

The rest of the paper is organised as follows - In

This work is supported in part by a grant from The Texas Higher Educatio n Board, by an NSF grant ANI-0087372 and by Intel Corp.

Section II we present a brief overview of the related work. This is followed by Section III where we provide the general framework for the LTCP scheme. Section IV discusses one possible design choice and presents analyses pertaining to it. Results of the evaluation of the chosen design using ns-2 simulations and emulation on a Linux testbed are presented in Section V. We conclude the paper in Section VI by summarising our experiences and taking a look at the future work.

II. Related Work

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 solution 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 TCP congestion control d) Modifications to the network infrastructure or use of non-TCP transport protocol

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 of the applications that use this approach or allow it as an option are 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 for improving the performance of TCP in highspeed networks has been to modify the congestion response function of TCP itself. 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. H-TCP [21] aims to identify whether the congestion window is operating in the low speed mode or highspeed mode and uses two different values for the increase/decrease parameters accordingly.

Several other schemes that go beyond modifications only to TCP have also been proposed. 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.

In this paper, we propose the Layered TCP scheme which modifies the congestion response function of TCP at the sender-side and requires no additional support from the network infrastructure or the receivers. In a sense, this scheme 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*. 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 algrithm of TCP with per-ack window adaptation* to provide efficient bandwidth probing in high bandwidth links while retaining fairness between multiple flows with similar RTTS.

III. Layered TCP : The Framework

The layered TCP scheme is a sender-side modification to the congestion response function of TCP for making it more scalable in high-speed networks. The congestion window response of the LTCP protocol is defined in two dimensions - (a) At the macroscopic level, LTCP uses the concept of layering to quickly and efficiently probe for available bandwidth (b) At the microscopic level, it extends the existing AIMD algorithms of TCP to determine the per-ack behavior. In this section, we present the intuition for the layering framework.

The primary goal behind designing the LTCP protocol is to make the congestion response function scale in highspeed networks under the following constraints:

1) the LTCP flows should be fair to each other (with similar RTTs)
2) the LTCP flows should be fair to TCP flows when the window is below a predefined threshold W_T.

The threshold W_T defines the regime in which LTCP is friendly to standard implementations of TCP. The optimal value of W_T is debatable and likely a topic of research by itself. 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 order to ensure that the new scheme with aggressive bandwidth probing is fair to TCP is slow networks, we choose a window threshold value W_T that is slightly greater than 44 packets. This imposes a constraint on LTCP to maintain proportional fairness to TCP flows in slow networks. Arguments aimed at keeping new protocols fair to TCP below a predefined window threshold, while allowing more aggressive behavior beyond the threshold have been put forth in [16], [17] as well.

In order to ensure that below the threshold W_T, LTCP is fair to TCP, all new LTCP connections start with only one layer and behave in all respects the same as TCP. The congestion window response is modified only if the congestion window increases beyond the threshold W_T. When the congestion window of the LTCP flow grows beyond the LTCP window threshold W_T, the increase behavior is modified to behave two dimentionally - (a) If congestion is not observed over a period of time, then number of layers is increased (b) The per-ack congestion window behavior of TCP is extended so that, when operating at higher layer a flow can increase its congestion window faster than when operating at a lower layer.

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, an LTCP flow increases the congestion window more aggressively than the standard implementation of TCP depending on the layer at which it is operating. At the microscopic level, when operating at some layer K, the LTCP protocol increases the congestion window as if it were emulating K virtual flows. That is, the congestion window is increased by $K/cwnd$ for each incoming ack, or equivalently, it is increased by K on the successful receipt of one window of acknowledgements. This is similar to the increase behaviour explored in [13].

Layers, on the other hand, are added if congestion is not observed over an extended period of time. To do this, a simple layering 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,

$$\begin{aligned} W_1 &= 0 \\ W_2 &= W_1 + \delta_1 \\ &\dots \\ W_K &= W_{K-1} + \delta_{K-1} \end{aligned} \tag{1}$$

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

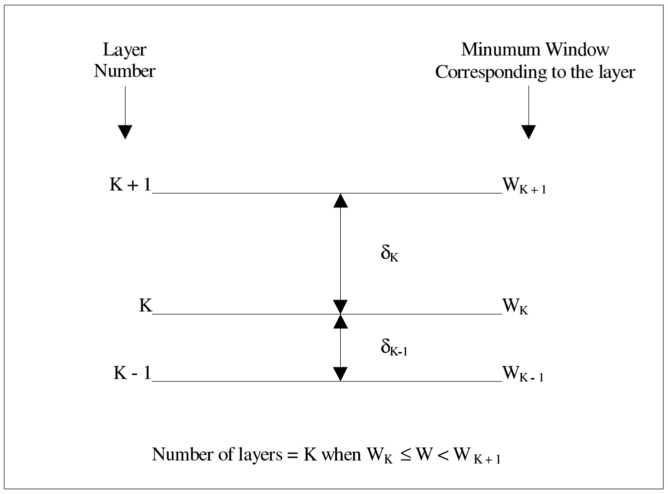
Layer Number
Minumum Window Corresponding to the layer
K + 1
W_{K+1}
δ_K
K
W_K
δ_{K-1}
K - 1
W_{K-1}
Number of layers = K when $W_K \leq W < W_{K+1}$

Fig. 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 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 WR_1 and WR_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 $\frac{WR_1}{K_1^{'}}$ and $\frac{WR_2}{K_2^{'}}$ RTTs respectively to regain the lost bandwidth. From the above reasoning, this gives us -

$$\frac{WR_1}{K_1^{'}} > \frac{WR_2}{K_2^{'}} \tag{3}$$

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.

This framework provides a simple, yet scalable design 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 architecture 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. The key factor for the framework is to determine an appropriate relationships 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.

IV. Design Choice

In order to evaluate the effectiveness of the LTCP protocol, we present the following design option. We support this design with extensive analysis to understand the protocol behavior. However, this is by no means the only possible or the best possible design for the LTCP scheme. We are currently persueing alternate design options and evaluation measures for identifying other options. The scheme presented here is a means for illustrating the effectiveness of the LTCP framework for efficiently claiming available bandwidth, without sacrificing the convergence and fairness properties.

For our design, we first choose the decrease behavior. Since we want to keep the design as close to that of TCP as possible, we choose a multiplicative decrease. However, the window reduction is based on a factor of β such that -

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

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

$$\frac{WR_1}{K_1^{'}} > \frac{WR_2}{K_2^{'}} \tag{5}$$

In order to allow smooth layer transitions, we stipulate that after a window reduction due to a packet loss, atmost one layer can be dropped i.e., a flow operating at layer K before the packet loss should operate at layer K or $(K-1)$ after the window reduction. Based on this stipulation, there are four possible cases -

1) $K_1^{'} = K_1, K_2^{'} = K_2$
2) $K_1^{'} = (K_1 - 1), K_2^{'} = K_2$
3) $K_1^{'} = K_1, K_2^{'} = (K_2 - 1)$ and
4) $K_1^{'} = (K_1 - 1), K_2^{'} = (K_2 - 1)$.

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 -

$$\frac{WR_1}{K_1} > \frac{WR_2}{(K_2 - 1)} \tag{6}$$

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

$$\frac{WR_1}{K} > \frac{WR_2}{K-2} \tag{7}$$

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,

$$\frac{W^{'}}{K} > \frac{W^{''}}{K-2}$$
$$\Rightarrow W^{'} > \frac{K}{K-2} W^{''}$$

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 -

$$W_{K+1} > \frac{K}{K-2} W_{K-1} \tag{8}$$

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

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

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 8 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.

Since layering starts at $W_2 = W_T$ we have,

$$W_K = \frac{K(K+1)(K-1)}{6} W_T \tag{10}$$

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

$$\delta_K = \frac{K(K+1)}{2} W_T \tag{11}$$

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.

Choice of β: The above presented analysis is hinged 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

$$\beta W_K < \delta_{K-1} \tag{12}$$

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

$$\beta < \frac{3}{K+1} \tag{13}$$

Thus, β should be chosen such that the above equation is satisfied. We use a value of $\beta = 0.15$ in this paper. The reason for this choice of β is explained in the next section.

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.

A. Analysis

As explained earlier, the primary goal for designing the LTCP protocol is to be able to utilize available link bandwidth aggressively in highspeed networks. In this section we provide quantitative analysis for time taken by an LTCP flow (in terms of RTTs) for claiming available bandwidth, the throughput analysis and the analysis to show that with this design, LTCP can potentially *reduce* the RTT unfairness problem inherent to TCP.

Time to claim bandwidth and Packet Recovery time: 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

$$T(\delta_1) + \frac{(K-2)(K+3)}{4} W_T \tag{14}$$

Note that the above analysis assumes that slowstart is terminated before layering starts.

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 choose $\beta = 0.15$ (corresponding to $K = 19$).

The table also 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. This column gives an idea of the number of virtual TCP flows emulated by an LTCP flow. For instance, a flow that evolves to layer 15, behaves similar to establishing 10 parallel flows at the beginning of the connection.

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
21	77000	13.50	66.67
22	88550	14.16	70.00
23	101200	14.82	73.33
24	115000	15.48	76.67
25	130000	16.14	80.00

Fig. 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 atleast $(K-1)$ packets per RTT (since we stipulate that a packet drop results in the reduction of atmost 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 $3.33 * (K-1)$.

Throughput Analysis: In order to understand the relationship between throughput of an LTCP flow and the drop probability p of the link, 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 atmost 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 $\frac{\beta W^{''}}{K^{'}}$ RTTs or $\frac{\beta W^{''}}{K^{'}} * RTT$ 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 -

$$N_D \simeq \frac{\beta (W^{''})^2}{K^{'}}(1-\frac{\beta}{2}) \tag{15}$$

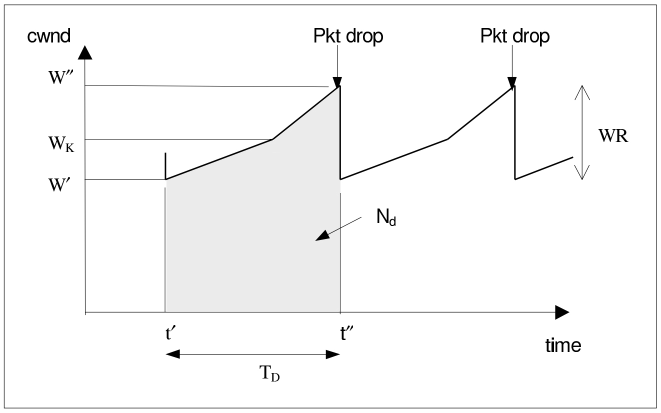

Fig. 3. Analysis of Steady State Behavior

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

$$BW = \frac{W''}{RTT}(1-\frac{\beta}{2}) \tag{16}$$

The number of packets sent between two losses N_D is nothing but $\frac{1}{p}$. By substituting this in Eq. 15, and solving for W_2 we have,

$$W^{''} = \sqrt{\frac{K^{'}}{\beta(1-\frac{\beta}{2})p}} \tag{17}$$

Substituting in Eq. 16 we have

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

Again if we consider the example above of the 2.4Gbps link with an RTT of 150ms and packet size of 1500 bytes, the window size can grow to 30,000. From, Fig. 2 we see that this window size corresponds to a layersize of $K = 15$. Substituting this value in the above equation, we notice that for the 2.4Gbps link mentioned above with $\beta = 0.15$, LTCP offers an improvement of a factor of about 8 for the achievable throughput compared to TCP.

Convergence: The inequality in the Eq. 3 of the LTCP framework ensures that flows will converge asymptotically to a fair share, since a larger flow takes longer time to recover lost throughput than a smaller flow. With the current design, we can show that this inequality is held after a congestion event irrespective of the layer transitions and hence asymptotic convergence is ensured.

Alternate Designs: In this section we presented one possible design for LTCP and provided the relevant analysis to understand the protocol behavior with this design. This design has several desirable properties such as efficient scaling of the window in high speed networks, quick convergence to fairness in the presence of multiple flows and alleviated RTT fairness problem. However, we would like to stress at this point that this is by no means the only possible or the best possible design choice. The aim of this design was to illustrate the effectiveness of using a simple concept like layering in the context of TCP congestion control to improve efficiency without sacrificing convergence properties. We are currently in the process of evaluating other designs. For instance by adding layers using the simple rule $W_K = \alpha W_{K-1}$ we can design a protocol that has very similar dynamics as TCP regarding window reduction and convergence but faster increase behavior. We have omitted the details of this design here because the window reduction similar to TCP results in highly fluctuating throughput, which may not be desirable in high speed networks. Details will be made available in a technical report.

B. Implementation Details

The LTCP protocol requires simple sender-side changes to the congestion window response function of TCP. For the chosen design, it uses two additional parameters - W_T and β. In our implementation, W_T and β are set to 50 and 0.15 respectively. Additionally, variables need to be used for saving the number of layers (K) and the windows corresponding to each layer K (W_K). When a new connection is established, the protocol is started with $K = 1$ and the slowstart algorithm of standard TCP. When slowstart is exited, the number of layers K is obtained based on the current cwnd. If K = 1, LTCP behaves in all respects similar to TCP. Otherwise (congestion window exceeds W_T), the following changes are made to the TCP congestion response function -

```
$if(newack)$
{
    $cwnd = cwnd + K/cwnd$
    $if(window() >= W_{K+1})$
        $K++$
}
$if(packet \quad loss)$
{
    $cwnd = cwnd(1-\beta)$
    $if(window() < W_K)$
        $K--$
}
```

The rest of the algorithms used in the traditional implementation of TCP - for instance the algorithms for RTT calculations, SACK processing, timer management etc, are not modified.

V. Results

To evaluate the LTCP protocol, we conducted experiments based on both simulations on the ns-2 simulator and emulations on a Linux testbed. The ns-2 simulations help us evaluate the behavior of LTCP under diverse network conditions. The emulations, on the other hand, help us evaluate a real implementation of LTCP on the Linux network stack. Third party evaluation of LTCP in comparison with other advanced TCP stacks such as HighSpeed TCP [16], Scalable TCP [17], FAST TCP [18], Bic-TCP [20] and H-TCP [21] and UDP based scheme UDT [25] is currently underway at the SLAC lab at Stanford. In this section we focus on some of the results of the simulation and the emulations comparing the performance of LTCP to unmodified TCP.

A. ns-2 Simulations

Fig. 4 shows the network topology used in the simulations. The topology is a simple dumbbell network. The bottleneck link bandwidth is set to 1 Gbps 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 70ms. The routers have the default queuesize of 6000 packets unless specified otherwise. 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.

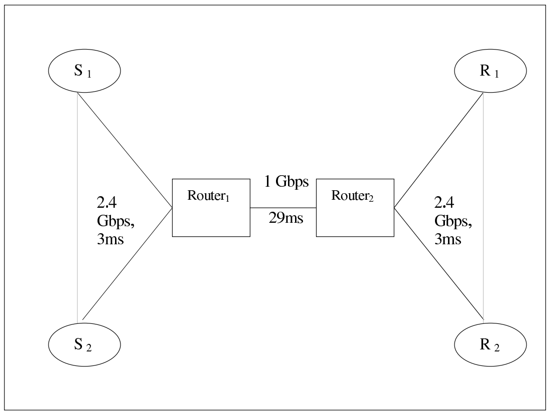

Fig. 4. Simulation Topology

1) Comparison of LTCP 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. 5 shows congestion window of LTCP in comparison with that of TCP, when the network consists of only one flow. As seen from the figure, the congestion window of the TCP flow takes over 600 seconds more than that of an LTCP flow to recover from a packet loss due to its drastic window reduction (factor of 1/2) followed by a conservative window increase (one per RTT). On the other hand, in case of LTCP, the window is reduced by a factor of 0.15 and after the window reduction, the window is increased at the rate of K per RTT, resulting in a more efficient window adaptation function.

The table in Fig. 6 shows the comparison between the long-term average steady state link utilization and average packet loss rates between a TCP flow and an LTCP flow for different link bandwidths. The throughput is calculated over a period of 2000 seconds after the flow reaches steady state. The buffersize at the router is set to the delayXbandwidth product. At large values of link bandwidth, the link utilization by the TCP flow is close to 75% , wheras the LTCP link utilization remians close to 96%. Since LTCP can operate close to the optimal value most of the time and keep the link utilization high, the congestion loss rate of the LTCP flow is larger than that of TCP, which owing to under-utilization of the link sees lower congestion losses.

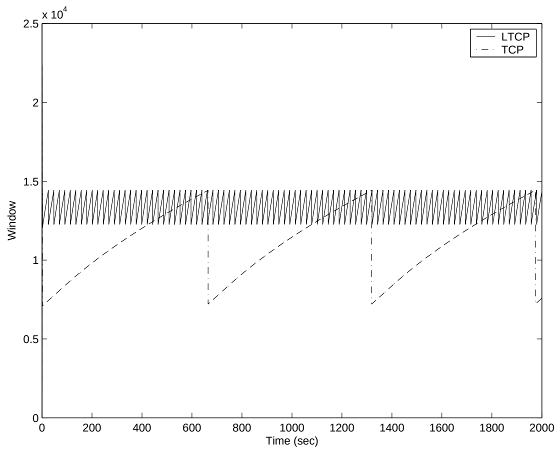

Fig. 5. Comparison Between LTCP and TCP Windows

Link Bandwidth	TCP		LTCP	
	Avg. Steady State Link Utilization (Mbps)	Packet Loss Rate (%)	Avg. Steady State Link Utilization (Mbps)	Packet Loss Rate (%)
10Mb	9.34	1.41E-02	9.62	1.33E-01
100Mb	96.15	2.08E-05	96.15	1.12E-03
1Gb	866.75	2.33E-06	961.54	4.49E-04
2.4Gb	1864.80	-	2307.69	2.01E-04

Fig. 6. Link Utilization and Packet Loss Rate for LTCP and TCP

2) Dynamic Link Sharing: Both the macroscopic and microscopic control of LTCP protocol respond to dynamic network conditions, to ensure that protocol converges quickly to its fair share. In this experiment, we evaluate how LTCP flows respond to dynamically changing traffic conditions created by multiplexing of several flows starting and stopping at different time. One LTCP flow is started at time 0, and allowed to reach steady state. A new LTCP flow is then added at 150 seconds and lasts for 1250 seconds. A third LTCP flow is added at 350 seconds and lasts for 850 seconds. Finally, the fourth flow starts the transfer at 550 seconds and sends data for 450 seconds. All these flows share the same bottelneck link. Fig. 7 shows the throughput of each flow. From the graph we see that, when a new flow is started and the available link bandwidth on the bottleneck link decreases, the existing LTCP flows quickly give up bandwidth until all flows reach the fair utilization level. The per-flow link utilizations remain stable at this value until some of the flows stop sending traffic. When that happens, the remaining LTCP flows quickly ramp up the congestion window to reach the new fair sharing level, such that the link is fully utilized.

3) 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 is

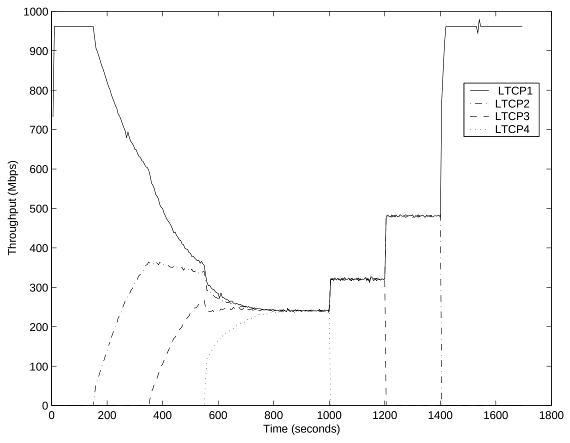

Fig. 7. Dynamic Link Sharing

designed to be more aggressive than TCP in high speed networks. So a single flow of TCP cannot compete with a single flow of LTCP and thus, for this simulation we compare the aggregate throughput of ten TCP flows to that of one LTCP flow at different link bandwidth. Also, to verify that an *established* LTCP flow gives up a share of its bandwidth to TCP flows, we first start the LTCP transfer and let it run for 300 seconds, so that it utilizes the link fully. At this point the TCP flows are started. The throughput is calculated for the flows after another 300 seconds. Fig.8 shows the results. As the link bandwidth increases, TCP flows become more inefficient in utilizing the available bandwidth, and as such, the percentage of the bandwidth used by TCP decreases. But, the aggregate throughput of TCP flows increases with higher link capacity, showing that inspite of the aggressive nature of LTCP congestion control, it still gives up a share of the bandwidth to competing TCP flows. For instance, on the 500Mbps link, the aggregate throughput of the TCP flows is 101.25Mbps but on the 1Gbps link it is 185.64 Mbps.

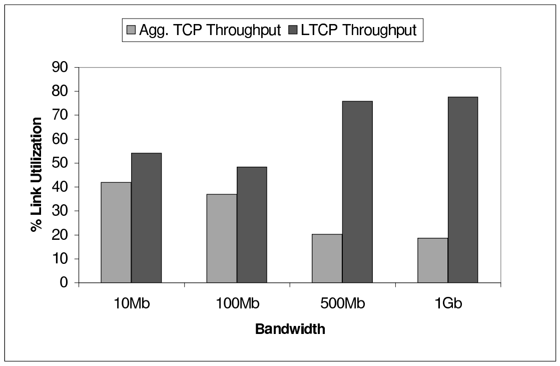

Fig. 8. Interaction of LTCP with TCP

B. Emulation Results

We have implemented LTCP in the network stack of the Linux 2.4.24 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 [33] aimed to provide good network performance. The LTCP modifications change only the congestion response behavior as discussed in section IV-B - no other code optimizations are made beyond those present int he Linux 2.4.24 network stack. Further work for improving the performance of the network stack by reducing the cost of SACK processing, modifications to the burst moderation algorithm etc, are currently underway in the Linux community. We expect that LTCP will see some performance benefits due to these optimizations as well, since it uses the same implementation as the TCP-SACK for all purposes other than the congestion window modifications.

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 Mbps. Setting one of these machines as a router further reduces the achievable throughput as both the incoming and outgoing traffic share the same PCI bus. Thus, instead of using NistNET or Dummynet for emulating LAN environments, we chose to use a packet delay scheduler patch made available on the Linux mailing list (by Stephen Hemminger). This patch modifies the FIFO queue such that every packet queued is delayed for a fixed amount. This allows us to use the link between the two machines to emulate links of different RTTs. The machine configured as the sender used the 2.4.24 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 [34] 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.

In all experiments we show results for three rtt values - 25ms, 70ms and 150ms - which are representative of links across closeby cities, across the country and transatlantic links. These values are chosen based on the pingER measurement history tables at SLAC [32].

1) Basic Performance Tests: In this experiment, we compare the performance of vanilla TCP with that of LTCP at different RTTs. 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.10 shows the average and standard deviation of the number of bytes transferred and the sustained 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 150ms, 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 17 GBytes. In contrast, a single LTCP flow transfers an average of about 49 GBytes, almost a factor of 3 improvement. Also, the deterioration in link uilitzation as the RTT is increased is more graceful in the case of LTCP compared to that of TCP.

	Total Bytes (GBytes)				Rate (Mbits/s)			
	TCP		LTCP		TCP		LTCP	
	Average	Standard Deviation	Average	Standard Deviation	Average	Standard Deviation	Average	Standard Deviation
Low RTT	75.40	0.08	77.85	0.06	719.25	0.50	742.75	0.50
Moderate RTT	50.33	5.73	71.38	0.55	480.25	54.52	681.50	5.00
High RTT	17.20	1.47	48.85	1.24	164.00	14.02	466.00	11.86

Fig. 9. Basic Performance Test

2) *Fairness Among Multiple Flows:* In order to evaluate the inter-protocol fairness among multiple flows of LTCP we repeated the experiment above with multiple flows starting at the same time and calculated the Jain Fairness Index [31]. Fig. 11 shows the results. The Jain Fairness Index for LTCP is high across different RTTs and different number of flows. Also, the standard deviation in the per-flow throughput of the LTCP flow is fairly low. This indicates that multiple flows of LTCP protocol share the available bandwidth equitably. Results for TCP are included to aid the comparison.

RTT	Number of flows	TCP			LTCP		
		Average per-flow goodput (Mbits/s)	Standard Deviation	Jain Fairness Index	Average per-flow goodput (Mbits/s)	Standard Deviation	Jain Fairness Index
Low RTT	2	373.50	50.20	0.991	373.00	12.73	0.999
	4	186.25	24.98	0.987	186.50	8.81	0.998
	6	124.00	12.63	0.991	124.17	2.32	1.000
	8	92.93	3.26	0.999	92.96	2.05	1.000
	10	74.26	4.13	0.997	74.28	2.69	0.999
Moderate RTT	2	299.50	64.35	0.977	353.50	9.19	1.000
	4	175.50	12.01	0.996	181.75	4.86	0.999
	6	116.77	20.40	0.975	121.50	6.53	0.998
	8	89.35	23.03	0.945	91.31	1.90	1.000
	10	70.53	10.33	0.981	73.04	4.66	0.996
High RTT	2	128.50	37.48	0.959	286.50	26.16	0.996
	4	123.73	30.21	0.957	166.00	7.53	0.998
	6	103.85	17.98	0.976	113.62	16.58	0.983
	8	78.06	39.80	0.815	86.68	8.72	0.991
	10	66.06	21.02	0.916	69.61	6.76	0.992

Fig. 10. Fairness Among Multiple Flows

3) *Interaction of LTCP with UDP-based Traffic:* In order to evaluate how LTCP responds to the presence of dynamically changing traffic that does not respond to congestion, we conducted the following experiment. In this experiment, an iperf stream of UDP sending at 350Mbps (about half the available link capacity) at intervals of 5 minutes was used as the source of non-responsive on-off traffic. UDP traffic consumes a lot of resources on the machines. In order to offset this, the txqueuelen and the backlog queue were set to 5000 in this experiment. Fig. 12 shows the results. It is clear from the graphs, that when the UDP stream is sending packets, the LTCP source reduces its congestion window so that the link is shared between the LTCP and the UDP traffic. When the UDP source, stops sending packets, the LTCP flow increases its sending rate close to the full link capacity. We verified this at different RTTs, but have included the results here for the medium RTT case.

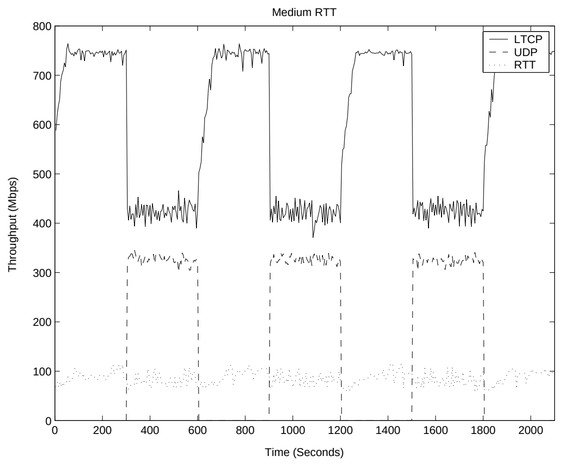

Fig. 11. Interaction of LTCP with non-responsive Traffic

VI. Conclusions and Future Work

In this paper we have proposed LTCP, a layered approach for modifying TCP for high-speed links. LTCP employs two dimensional control for adapting to available bandwidth. At the macroscopic level, LTCP uses the concept of layering to increase the congestion window when congestion is not observed over an extended period of time. Within a 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 one design option. With this design, the window reduction on a packet loss is multiplicative.

We have shown through analysis, simulations and emulations that a single LTCP flow can adapt to nearly fully utilizing the link bandwidth. Other significant features of the chosen design are - (a) it provides a significant speedup in claiming bandwidth and in packet loss recovery times (b) multiple flows share the available link capacity equitably (c) it could potentially alleviate the RTT unfairness inherent to TCP (d) requires simple modifications to TCP's congestion response mechanisms and is controlled only by two simple parameters - W_T and β.

As part of our future work, we plan to compare the performance of LTCP with the current design with other high speed TCP stacks and experiment with further benefits to be reaped by proposed kernel optimizations. Comparative third party evaluation of LTCP against other proposals is currently underway at SLAC, Stanford.

Our design is hinged on an early decision to use multiplicative decrease, and on the stipulation that atmost one layer is dropped after a congestion event. A number of other possibilities exist for alternate designs of the general LTCP approach. We plan to pursue these options in future.

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", *Proceedings 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 High-Speed 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] R. N. Shorten, D. J. Leith, J. Foy, and R. Kilduff, "Analysis and design of congestion control in synchronized communication networks", June 2003, submitted for publication.

[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", *Proceedings 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] Dah-Ming Chiu and Raj Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks", *Computer Networks and ISDN Systems, 17*, June 1989

[32] PingER History Tables at SLAC
$http : //www - iepm.slac.stanford.edu/cgi - wrap/table.pl?from = Country\&to = Country\&file = average_rtt\&date = 2004 - 04$

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

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