Improving TCP Performance over Long Delay Satellite Links

Jing Peng, Peter Andreadis, Claude Bélisle
Communications Research Centre Canada
Ottawa, ON K2H 8S2 Canada
E-mail: {jing.peng, peter.andreadis, claude.belisle}@crc.ca

Michel Barbeau
Carleton University
Ottawa, ON K1S 5B6 Canada
E-mail: michel.barbeau@scs.carleton.ca

Abstract
TCP is robust and flexible when operated on wired terrestrial networks. There are problems, however, when TCP is used on long delay satellite links. In this paper, we first study the effects that these long delays have on TCP, and then we present a transport layer solution that is implemented in OPNET 7.0. This solution enables TCP to share congestion control information among connections between the same host-pair. The shared information is used to speed up new connections to the same destination, and to coordinate concurrent connections in order to reduce the possibility of congestion losses. Simulation results show that, with the presence of information about the same channel discovered by previous connections, this solution performs better than standard TCP. As the delay increases, the degree of performance improvement also increases, making this approach particularly well-suited for long delay satellite links.

1. Introduction
There is increasing need to use communications satellites to carry Internet traffic. Satellite communications offer many benefits, such as wide coverage areas, natural broadcast capabilities, and the ability to reach remote and geographically adverse locations at relatively low cost. However, many studies show that satellite links can cause some problems to the reliable end-to-end data transmission on the Internet, which is realized by the Transmission Control Protocol (TCP) [1][2].

Long propagation delay is one of the main causes of negative effects on TCP's performance [3]. Latency in a satellite environment is higher than the one in a terrestrial environment. The one-way propagation delays are 110 ~ 150 ms for medium earth orbit systems (MEO) and 250 ~ 280 ms for geostationary satellites (GEO). Intersatellite links, on-board processing and other network factors can increase one-way propagation delay by more than one second.

Sharing TCP state information is one of the solutions suggested in RFC 2760 [2] to better utilize the bandwidth in long-delay satellite environments. Sharing TCP state information was first used in TCP Extensions for Transactions (T/TCP) to accelerate data delivery prior to the completion of the three-way handshake [4]. Touch [5] extended this idea and provided a general guidance for sharing TCP state information in RFC 2140. Some researchers implemented this idea to provide efficient support for numerous, short concurrent transfers existing in Web applications [6][7]. Sharing TCP state information is particularly suitable for use in a satellite environment where many transmissions are short compared to the delay-bandwidth product.

We have implemented sharing TCP state information for the purpose of improving TCP performance over long delay satellite links. Sender side modification has been made to the TCP protocol. The resulting TCP is called Sharing TCP (STCP). In STCP, information about the channel between a host-pair is shared among sequential and concurrent connections. New connections to the same destination can start up more efficiently. Concurrent connections are under better control so that the possibility of congestion losses is reduced. In addition, STCP provides a new mechanism to allocate the network capacity among concurrent connections in a relatively fair manner. This allocation mechanism works in a distributed manner to ensure that minimal overhead is added to the TCP operation.

The rest of the paper is organized as follows. Section 2 reviews TCP and its problems over long delay satellite links. Section 3 presents the design and implementation of STCP. Section 4 discusses the simulation results. Section 5 concludes the paper.

2. TCP Review
TCP is a connection-oriented, end-to-end reliable protocol. It provides a set of congestion control mechanisms to ensure the reliable delivery of data, and to adjust the data transmission according to network conditions. The fundamental congestion control mechanisms are slow start, congestion avoidance, fast retransmit and fast recovery [8].

TCP uses a state variable, congestion window (*cwnd*) to represent the largest amount of data a TCP connection can transmit into the network without waiting for the acknowledgments. Another state variable, slow start threshold (*ssthresh*), is used to determine whether the slow start or congestion avoidance algorithm is used to control data transmission.

Slow start defines a mechanism when starting traffic on a new connection, or after repairing loss detected by the retransmission timer. A connection starts out by setting the

congestion window to one Maximum Segment Size (MSS) and sending out one segment. For each segment that is acknowledged successfully, the congestion window is increased by one MSS. That is, the congestion window is doubled for every Round Trip Time (RTT). Once the size of the congestion window reaches the slow start threshold, congestion avoidance is used to probe the network for additional capacity. During congestion avoidance, the congestion window is increased by approximately one MSS per RTT.

TCP retransmits a segment if an acknowledgment (ACK) is not received for the segment before a specific TCP clock times out. This timeout is called the Retransmit Timeout (RTO). Whenever a loss happens, the TCP sender always reduces its transmission rate because TCP assumes that all the losses are caused by network congestion. If a timeout occurs, the congestion window is reduced to one MSS, and slow start is used to control the data transmission.

Fast retransmit provides a more efficient way to detect a loss before the timer expires. It is based on duplicate ACKs that are sent from the data receiver to indicate receipt of out-of-order segments. When fast retransmit detects a loss, fast recovery is used to control data transmission during loss recovery. When loss recovery is finished, the sender cuts the congestion window to half and resumes with congestion avoidance, rather than going into slow start.

Long delays cause problems to TCP's congestion control mechanisms. The direct effect of long delays on TCP is that it takes longer for a TCP segment to reach the destination and for an acknowledgment to come back. Since TCP always starts with a congestion window size of one MSS and gradually increase its transmission rate upon receiving the acknowledgments, it takes much longer for TCP to fill the pipe on a long delay satellite network than on a terrestrial network [1]. If multiple TCP connections are sharing a satellite link, the group of concurrent connections increases the overall congestion window faster than a single TCP connection and is more likely to cause congestion losses [5]. The loss recovery over long delay satellite links is very costly because a large amount of time is required for retransmission and for reaching the optimum transmission rate after rate reduction. Furthermore, if one or more of the concurrent connections close, some bandwidth is released. The amount of time it takes for the remaining connections to make use of the released bandwidth can be significant [9].

3. Design and Implementation of STCP

STCP is an extension to TCP that aims to mitigate the problems described in Section 2. STCP is based on the idea of sharing TCP state information proposed in RFC 2140 [5]. The design goal of STCP is to speed up similar connections and to coordinate concurrent connections by sharing the information about the network condition. Current implementation of STCP focuses on the information sharing of the slow start threshold and the congestion window among sequential and concurrent connections between the same host-pair.

In this section, we present the design and implementation of STCP in OPNET 7.0. First we introduce a new structure added to the TCP layer to store the shared information. Then we discuss how the information of the slow start threshold and the congestion window is shared among similar connections.

3.1 Data Structure

The new structure used to store the shared information is called Ensemble Control Block (ECB), a name borrowed from Eggert et al. [6]. An ECB contains the information about the channel between a pair of hosts. This channel is shared by multiple connections on the same host with the same destination IP address. We call such a group of concurrent connections an *ensemble*. The ECB structure definition is shown in Figure 1.

```
Typedef struct
{
  IP_Addr       rem_addr;
  TcpT_Size     n_ssthresh;
  TcpT_Size     n_cwnd;
  TcpT_Size     used_cwnd;
  int           num_connections;
  Boolean       add_new_flag;
  TcpT_Size     quota;
  TcpT_Size     contribution;
} ECB;
```

Figure 1: ECB Structure Definition

In an ECB, field *rem_addr* is the destination IP address. Field *n_ssthresh* represents the slow start threshold for that channel. Field *n_cwnd* represents the aggregate congestion window, which specifies the largest amount of data the group of connections can put on that channel without waiting for the acknowledgments. Each connection obtains a share of *n_cwnd* and stores the share in its congestion window *cwnd*, which is defined in standard TCP to limit the largest amount of outstanding data a connection can put on that channel. Field *used_cwnd* is defined as the sum of the congestion windows of the connections sharing the ECB.

Field *num_connections* is the number of member connections currently associated with this ECB. Fields *add_new_flag, quota* and *contribution* are used for congestion window reallocation when opening a new connection. The use of the three fields will be discussed in Section 3.3.

OPNET comprises two models related to the TCP implementation: TCP manager and TCP connection. They

provide the most widely used functionalities of TCP [10]. TCP manager represents the root process of the TCP protocol. It creates and invokes the appropriate connection processes upon the requests from other layers. A TCP connection process is spawned by the TCP manager to handle the activities related to an individual connection.

In STCP, TCP manager maintains a list of ECBs. When opening a new connection, TCP manager searches the list for an existing ECB associated with the destination IP address. If no ECB is found, a new ECB is created for that destination IP address. The new connection is then associated with the ECB. *n_ssthresh* and *n_cwnd* in the new ECB are initialized to the values that the slow start threshold and congestion window would have for a new connection in standard TCP, as specified in RFC 2581 [8]. For example, *n_ssthresh* can be initialized to the advertised receive window and *n_cwnd* can be initialized to one MSS. If an ECB is found and no connection is currently associated with the ECB, the new connection is associated with the ECB and starts with a congestion window size equal to *n_cwnd*. If an ECB is found and there is at least one connection associated with the ECB, the new connection is associated with the ECB and obtains a share of the aggregate congestion window. Allocation of the aggregate congestion window to the connections sharing the same ECB will be discussed in more detail in section 3.3.

3.2 Sharing Slow Start Threshold

There is a single slow start threshold for the connections sharing the same ECB, denoted as *n_ssthresh*. An individual slow start threshold for each connection is no longer necessary. All the operations on *ssthresh* defined in the congestion control mechanisms in standard TCP [8] are applied to *n_ssthresh* in STCP.

In standard TCP, the initial value of the slow start threshold may be arbitrarily high, and the congestion window keeps increasing until there is a loss. Then the slow start threshold is updated to a more reasonable value, half of the previous congestion window. However, this information is destroyed when the connection is closed. Another connection using the same channel has to go through the same procedure, causing loss first, and then reducing its transmission rate and updating the information.

By reusing information of the slow start threshold in STCP, another connection using the same channel is able to change from slow start to congestion avoidance before causing a loss. In this way, unnecessary retransmissions and rate cuts can be reduced.

3.3 Sharing Congestion Window

As discussed in section 3.1, there are three types of congestion window variables in STCP: *n_cwnd* is the aggregate congestion window for the connections associated with the same ECB; $cwnd_i$ is the congestion window of connection i ($i = 1 \ldots num\_connections$); *used_cwnd* is defined as the sum of the congestion windows of the connections. In other words, *n_cwnd* represents the available network capacity discovered so far, $cwnd_i$ represents the share of network capacity allocated to connection i, and *used_cwnd* represents the part of network capacity being used by the connections.

After receiving an acknowledgment, a connection increases the aggregate congestion window *n_cwnd* according to the congestion control mechanisms defined in standard TCP [8]. If every segment is acknowledged successfully, *n_cwnd* is doubled per RTT during slow start and is increased by one MSS per RTT during congestion avoidance. The decision of whether to change from slow start to congestion avoidance depends on the values of *n_cwnd* and *n_ssthresh*.

STCP applies a distributed window allocation algorithm to assign a share of *n_cwnd* to each member connection. A connection invokes the window allocation algorithm to update its share *cwnd* whenever it updates the value of the aggregate congestion window or it needs to send out data.

To prevent connections from transmitting more data than allowed by the network and to fully utilize the available network capacity, the window allocation algorithm targets satisfaction of Equation 1:

$$n\_cwnd = used\_cwnd \qquad (1)$$

Originally *used_cwnd* is equal to *n_cwnd*. When a member connection receives acknowledgments or detects losses, *n_cwnd* is updated, resulting in a difference between *n_cwnd* and *used_cwnd*. When invoked by connection k, the window allocation algorithm calculates a new share for the connection as follows:

$$\begin{aligned} new\_cwnd_k &= n\_cwnd - \sum_{i=1,\, i\neq k}^{n} cwnd_i \\ &= n\_cwnd - (used\_cwnd - cwnd_k) \\ &= cwnd_k + (n\_cwnd - used\_cwnd) \qquad (2) \end{aligned}$$

where $cwnd_k$ is the congestion window of connection k before invoking the window allocation algorithm, new_cwnd_k is the new congestion window assigned to connection k, $cwnd_i$ is the congestion window of connection i which is associated with the same ECB as connection k, and n is the number of connections sharing the ECB with connection k. Connection k obtains an increased share if the difference between *n_cwnd* and *used_cwnd* is positive, and gets a reduced share otherwise.

If the result of Equation 2 is negative, which is possible when *n_cwnd* is reduced to a very small value, then the following adjustment is performed:

If ($new\_cwnd_k \leq 0$)

$$new\_cwnd_k = MSS$$

Finally the window allocation algorithm updates *used_cwnd* as follows:

$$used\_cwnd = used\_cwnd - cwnd_k + new\_cwnd_k \qquad (3)$$

After the above update is performed, *used_cwnd* becomes equal to *n_cwnd* again. If all the member connections progress at the same speed and they all make full use of their congestion windows, each connection can get a fair share of the aggregate congestion window.

If a loss is detected by duplicate ACKs on a connection, the connection reduces the aggregate congestion window according to Equation 4:

$$n\_cwnd = n\_cnwd - cwnd_k / 2 \qquad (4)$$

where $cwnd_k$ is the congestion window of the connection that detects the loss. Then the connection invokes the window allocation algorithm and obtains a new congestion window which is equal to half of its previous congestion window. The other connections remain unaffected. If all the connections detect losses, then *n_cwnd* is reduced to approximately half of its previous value when the first loss is detected.

If a loss is detected by a timeout event, a single TCP connection will reduce the congestion window to one MSS. For a group of independent TCP connections, only the connections that detect a timeout will reduce the congestion window to one MSS, while the others keep increasing the transmission rate. Because a timeout event usually indicates severe network congestion, STCP reduces the aggregate congestion window in a way similar to a single TCP connection in order to prevent deteriorating network congestion and causing more losses. If a timeout occurs on a connection, the connection reduces the aggregate congestion window according to Equation 5:

$$n\_cwnd = num\_connections * MSS \qquad (5)$$

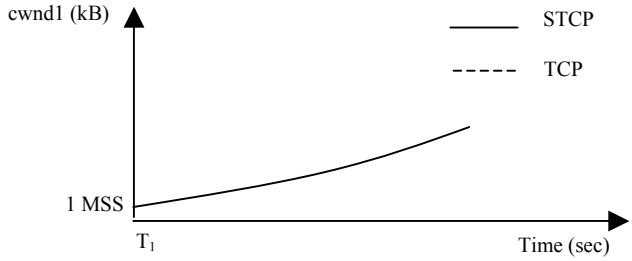
Any connection sharing the same ECB will obtain a congestion window of one MSS when it invokes the window allocation algorithm.

When opening a new connection, STCP intends to give the new connection a fair share of the network capacity as well as to keep the increase of the overall congestion window of an ensemble like the one of a single TCP connection. The share of the new connection comes from contributions of existing connections associated with the same ECB. First, TCP manager sets the field *add_new_flag* in the ECB to true, indicating that there is a new connection trying to get a share of the network capacity. Then it calculates the amount of share that each existing connection is expected to give to the new connection. The result of this calculation is stored in the field *quota* in the ECB. Field *contribution* represents the amount of the share that has been granted so far by existing connections and is available to the new connection. When invoked by an existing connection, the window allocation algorithm checks the value of *add_new_flag*. If the value is true, it then reduces the new share of the connection by *quota* and increases *contribution* by *quota*. *add_new_flag* remains true when the reallocation process is going on. The congestion window of the new connection is initialized to zero. Before it sends out the first segment, it invokes the window allocation algorithm, and obtains a new congestion window equal to *contribution*. Thus, the window reallocation process is completed and *add_new_flag* is set to false by the new connection.

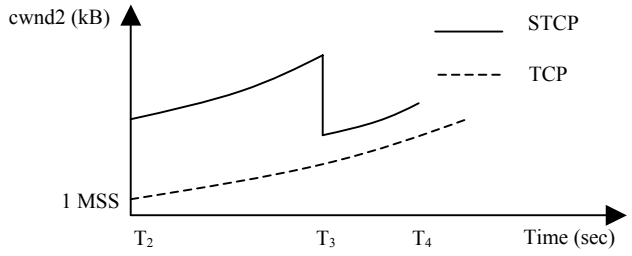
When a connection is closed, the released share from the closed connection will be utilized by remaining connections once the window allocation algorithm is invoked.

The following example illustrates how connections share the congestion window information in STCP. This example involves three connections between the same host-pair. The first connection runs alone and is closed before the second connection is opened. The third connection is opened later than the second connection and runs concurrently with the second connection for some time. Finally the second connection and the third connection are closed sequentially. Figure 2 shows the congestion windows of the three connections with comparison between TCP and STCP.

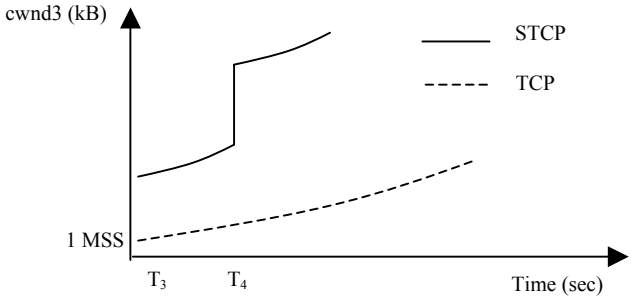
In Figure 2, the solid lines represent the congestion windows when STCP is used, and the dashed lines represent the congestion windows when standard TCP is used. For the first connection, TCP and STCP perform the same, thus the congestion windows of the first TCP connection and the first STCP connection are identical. The second and third TCP connection also perform the same as the first TCP connection, therefore the congestion windows of the three TCP connections are identical. When the second STCP connection is opened at time T_2, it reuses the knowledge about the network capacity discovered by the first STCP connection and starts with an initial congestion window larger than one MSS. At time T_3, the second STCP connection gives part of its share of the network capacity to the third STCP connection and cuts its congestion window to half. Thus, the third STCP connection starts with an initial congestion window equal to the congestion window of the second STCP connection. When the second STCP connection is closed at time T_4, the third STCP connection obtains the released network capacity and almost doubles its congestion window size.



(a) Congestion Window of Connection 1



(b) Congestion Window of Connection 2



(c) Congestion Window of Connection 3

Figure 2: Sharing Congestion Window Information

4. Simulation Results

In this section, simulation results of STCP with comparison to standard TCP are presented. Simulations are conducted for both FTP and HTTP applications. The performance of STCP in various one-way delays is also simulated and discussed.

4.1 FTP Simulation

In this simulation, we use two similar scenarios to transfer six 1MB files from a server to a client. The first scenario, called Basic TCP, uses standard TCP, while the second scenario uses STCP. The files are sent in two consecutive groups, with three concurrent transfers occurring in each group. The file transfer times are shown in Figure 3.

In Figure 3, the first group of STCP connections performs slightly slower than the first group of TCP connections. A group of STCP connections increases the aggregate congestion window like a single TCP connection. However, a group of independent TCP connections increases the aggregate congestion window N times faster than a single TCP connection, where N is the number of connections. So the total transfer time for the first STCP group is about 5% longer than that of the first TCP group. Once STCP has obtained some information about the network condition from the first group, the second group of STCP connections starts up more quickly than the first STCP group, while the second group of TCP connections performs the same as the first TCP group. Figure 3 shows that the second STCP group is about 3 times faster than the second TCP group.

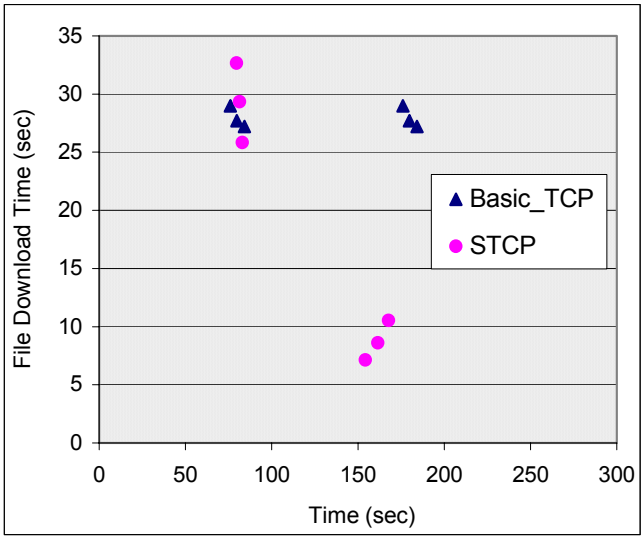


Figure 3: FTP File Download Time

4.2 HTTP Simulation

In this simulation, we compare the STCP scenario with the Basic TCP scenario in carrying HTTP traffic. The two scenarios transfer three consecutive Web pages from the server to the workstation. The pages consist of the same number of objects. Each page is 100kB. The page download times are shown in Figure 4.

Because HTTP 1.1 uses a persistent connection to transfer all the data in one Web page, TCP performs the same when downloading each of the three pages. STCP has the same performance as TCP when downloading the first page. Using the knowledge about the network condition obtained in the first transfer, the second connection established to download the second page starts with a congestion window size larger than one MSS. Therefore, the second page in the STCP scenario is downloaded in a shorter time than that of the first page. The third transfer is even faster. Comparing STCP with standard TCP for the third page, the download time for STCP is almost 3 times faster.

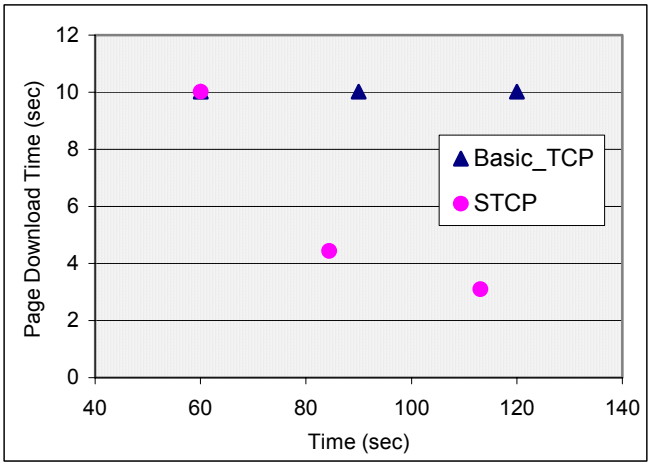


Figure 4: HTTP Page Download Time

4.3 Various Delays

By comparing standard TCP and STCP under various values of delay, we can see how much STCP can improve TCP's inefficiency over long delay satellite links.

We set up two scenarios, each transferring two files sequentially. The file size is fixed to 1MB. For scenario Basic TCP, the two files are transferred in the same amount of time. For STCP, the first transfer is the same as TCP, while the second transfer is faster than the first transfer in the same scenario, thus faster than the second transfer in the TCP scenario. We vary the one-way delay from 0.05 to 0.67 seconds for both scenarios, and observe that the performance improvement in transferring the second file using STCP is significant, as shown in Figure 5. As the delay increases, the degree of performance improvement also increases.

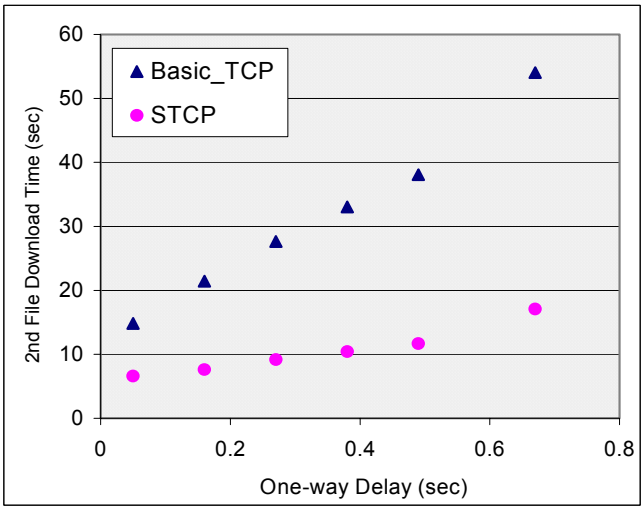


Figure 5: Download Time of the 2nd File under Various Delays

5. Conclusion

In this paper, we have presented a sender-side TCP modification, STCP, to improve TCP performance over long delay satellite links by sharing congestion control information. Simulations for FTP and HTTP applications have been conducted. Simulation results show that this solution performs better than standard TCP when there exists information about the same channel discovered by previous connections.

Future work will focus on how to maintain the state information discovered by previous connections when the information is not currently used by any connection, and how to share the state information among connections on the same subnet.

References

[1] Allman, M., Glover, D., Sanchez, L., Enhancing TCP over Satellite Channels using Standard Mechanisms, IETF RFC 2488, January 1999.

[2] Allman, M., et al., Ongoing TCP Research Related to Satellites, IETF RFC 2760, February 2000.

[3] Ghani, N., Dixit, S., TCP/IP Enhancements for Satellite Networks, IEEE Communications Magazine July 1999.

[4] Braden, R., T/TCP -- TCP Extensions for Transactions, IETF RFC 1644, July 1994

[5] J.Touch, TCP Control Block Interdependence, IETF RFC 2140, April 1997.

[6] Eggert, L., Heidemann J., and Touch, J., "Effects of Ensemble-TCP", SIGCOMM Computer Communication Review, December 20, 1999.

[7] Balakrishnan, H., Padmanabhan, , V.N., Seshan, S., Stemm, M., Katz, R.H., TCP Behavior of a Busy Internet Server: Analysis and Improvements, Proc.IEEE Infocom, March 1998

[8] Allman, M., et al., TCP Congestion Control, IETF RFC 2581, April 1999.

[9] Stadler, J. Scott, and Gelman, Jay, "Performance Enhancement for TCP/IP on a Satellite Channel", MIT Lincoln Laboratory, IEEE Communications, January 1998.

[10] OPNET Modeler Documentation, Model Descriptions, Transport Layer Protocols, TCP Model, version 7.0.B.