SaTPEP: A TCP Performance Enhancing Proxy for Satellite Links

Dimitris Velenis, Dimitris Kalogeras, and Basil Maglaris

Department of Electrical and Computer Engineering, Network Management and Optimal Design (NETMODE) Laboratory, National Technical University of Athens, Heroon Politechniou 9, Zographou, 157 80, Athens, Greece
{dbelen,dkalo,maglaris}@netmode.ece.ntua.gr

Abstract. Satellite link characteristics cause reduced performance in TCP data transfers. In this paper we present SaTPEP, a TCP Performance Enhancing Proxy which attempts to improve TCP performance by performing connection splitting. SaTPEP monitors the satellite link utilization, and assigns to connections window values that reflect the available bandwidth. Loss recovery is based on Negative Acknowledgements. The performance of SaTPEP is investigated in terms of goodput and fairness, through a series of simulation experiments. Results obtained in these experiments, show significant performance improvement in presence of available bandwidth and at high error rates. [1]

1 Introduction

Satellite link characteristics, namely long propagation delays, large *bandwidth* $\cdot$ *delay* products, and high bit error rates, affect the performance of TCP, the dominant transport layer protocol in the Internet. In network paths with large *bandwidth* $\cdot$ *delay* products, TCP needs a considerable amount of time to set its congestion window, *cwnd*, to the appropriate value [1]. Furthermore, TCP reacts to segment drops by lowering *cwnd* [2]. When drops are caused by transmission errors, TCP unnecessarily reduces its transmission rate.

Several methods to overcome those problems are listed in [3] and [4]. Many of them employ end-to-end mechanisms. Others try to increase performance by mechanisms implemented at certain points in the path between the TCP endpoints [5], [6]. The Satellite Transport Protocol, STP [7], may be used either in a split TCP connection over the satellite part of a network, or as a transport layer protocol within a satellite network. TCP-Peach [8] attempts to improve end-to-end TCP performance in a priority aware environment.

In this paper we introduce SaTPEP, a TCP Performance Enhancing Proxy that aims at increasing the performance of TCP over single-hop satellite links. SaTPEP's flow control is based on link utilization measurements, and segment loss is handled with Negative Acknowledgements (NACKs). The remainder of the paper is organized as follows: In Section 2 we describe the design of SaTPEP. In Section 3 we present simulation results obtained by a SaTPEP model in the ns [9] simulator. Section 4 concludes the paper.

[1] This work is partially supported by OTE S.A. R&D laboratories

2 Satellite TCP Performance Enhancing Proxy – SaTPEP

SaTPEP consists of the two gateways at each end of a bidirectional satellite link (or a unidirectional satellite forward link and a reverse terestrial link). Every TCP connection traversing the link is split as follows: One connection is established between the TCP sender and the Uplink Gateway (UG), another one between the two gateways, called the *SaTPEP connection*, and a third one between the Downlink Gateway (DG) and the TCP receiver.

In order to improve performance over the satellite hop, the SaTPEP connection performs flow control based on link utilization measurements, and error recovery with Negative Acknowledgements (NACKs).

2.1 Flow Control

A SaTPEP connection begins with the standard TCP three-way handshake. The SaTPEP sender (UG) does not perform any *cwnd* calculations. It just sets *cwnd* to *rwnd*, the window value advertised by the SaTPEP receiver (DG). SaTPEP measures window values in MSS-sized segments, rather than in bytes. At the beginning of a SaTPEP connection, the sender sets *rwnd* to 1. On receipt of the SYN-ACK segment, *rwnd* is set to the value in the window field of the TCP header. This value is calculated by DG as the minimum of the available buffer space for incoming data, and a window value calculated using the link utilization measurement. Given the link capacity, DG can measure its utilization by measuring the incoming data throughput. This throughput measurement is based on the Packet Pair algorithm [10], and it is performed over all received IP traffic. A timer, the *idle timer*, is used to handle periods of link inactivity. When it expires the throughput measurement is set to zero.

Whenever a measurement is completed, DG calculates the available bandwidth by subtracting the throughput measurement from the total bandwidth of the link. The available bandwidth multiplied by the link RTT denotes how much more data the link can attain. We call this value *Available Window*, AW. AW is distributed to the connections as an increment to their *rwnd* values. DG may use a wide variety of criteria to distribute AW to its connections. It might implement policies that favor certain types of traffic, or certain hosts over others.

In the present paper we propose an algorithm for distributing AW in a fair manner to all *active* connections. A connection is characterized as active, if it has transmitted at least one data segment during the last throughput measurement. Non-active connections do not get a share from AW. Assuming n active connections, the *rwnd* value of the k-th connection is incremented by a value $drwnd_k$, defined in equation (1). Note that the $drwnd_k \cdot MSS_k$ products, of all active connections, sum up to AW.

$$drwnd_k = \frac{AW}{MSS_k} \cdot \frac{2 \cdot \sum_{i=1}^{n} rwnd_i - n \cdot rwnd_k}{n \cdot \sum_{i=1}^{n} rwnd_i} \; . \tag{1}$$

When AW is distributed to the active connections, connections with smaller *rwnd* values receive larger *drwnd*, leading to a steady state of fair bandwidth distribution among all active connections. The $rwnd_k$ value is limited by a maximum value, max_rwnd_k, defined in equation (2), where $c \leq 1$, and *del* the link round-trip propagation delay. Constant c accounts for data that is released from the SaTPEP socket buffer to the IP layer and has not yet been transmitted.

$$max\_rwnd_k = \frac{(1+c)}{n} \cdot \frac{bw \cdot del}{MSS_k} \ . \qquad (2)$$

Whenever a connection becomes idle, its *rwnd* is reset to 1 segment. By setting its *cwnd* to *rwnd*, the SaTPEP sender transmits data bursts much larger than a TCP sender. With these bursts transmitted over a one-hop path, a buffer size of at least the $bandwidth \cdot delay$ product of the satellite link is enough to assure that the link will not experience congestion.

2.2 Loss Recovery

The SaTPEP flow control mechanism guaranties that the link will not experience congestion. Therefore, segment drops are only caused by errors, and SaTPEP does not reduce *cwnd* when loss is detected. The SaTPEP sender enters recovery mode when the first duplicate acknowledgement, *dupACK*, is received, since there can be no segment reordering on a single-hop connection. While in recovery mode, *cwnd* is inflated by the amount of dupACKs, *dupwnd*, received. Recovery ends when *recover*, the highest sequence number transmitted when the first dupACK arrived, is acknowledged.

The SaTPEP receiver notifies the sender of missing segments by means of *Negative Acknowledgements*, *NACKs*. NACKs are included in dupACKs as a TCP option, describing a contiguous missing part of the receiver data stream with sequence numbers lower than the maximum sequence number received. The receiver transmits increasingly sequenced NACKs in successive dupACKs, and repeats the same NACKs in a cyclic manner until the data they describe is received.

On receipt of a NACK, the SaTPEP sender retransmits the requested segment(s) along with as much new data as the inflated *cwnd* allows. The sender will not respond to repeated NACKs until a counter, *rtx_count*, expires. *rtx_count* is set to $rwnd - 1$ on receipt of the first dupACK and is decreased by 1 for every dupACK received. It is an estimate of the expected number of dupACKs that will be received before the retransmission reaches the receiver. As long as $rtx\_count > 0$, the retransmitted segment cannot have reached the receiver, and there is no point in repeating the retransmission. When *rtx_count* expires, *rtx_count* is reset and the sender repeats a complete cycle of all retransmissions still requested by incoming NACKs.

SaTPEP also utilizes TCP's Retransmission Timer. Whenever the timer expires the SaTPEP sender sets *rwnd* to 1, *dupwnd* to 0 and retransmits the segment requested by the last ACK received. Figure 1 describes more formally the loss recovery algorithm implemented at the SaTPEP sender.

```
Initially: $dupwnd = 0$, $recover = null$, $hinack = null$, $hiack = null$
           $rtx\_count = null$, $rtx\_allow = 0$, $rtx\_stop = null$
On arrival of 1st dupACK:
  – $hiack \leftarrow$ dupACK's ack_no, $recover \leftarrow$ highest transmitted seq_no, $dupwnd \leftarrow 1$
  – if $rtx\_count$ is $null$ or 0: $rtx\_count \leftarrow rwnd - 1$
  – $hinack \leftarrow$ NACK's highest seq_no
  – retransmit segment(s) requested in NACK option
  – reduce $dupwnd$ by number of segments retransmitted
  – transmit new data (as much as $cwnd$ allows)
On arrival of any dupACK:
  – $dupwnd \leftarrow dupwnd + 1$, $rtx\_count \leftarrow rtx\_count - 1$
  – if NACK's highest seq_no > $hinack$ and $rtx\_allow$=0:
      • retransmit what NACK requests, $hinack \leftarrow$ NACK's highest seq_no
  – else if $rtx\_allow$=1 and NACK does not contain $rtx\_stop$:
      • retransmit segment(s) requested in NACK option
      • if $hinack >$ NACK's highest seq_no: $hinack \leftarrow$ NACK's highest seq_no
  – else if $rtx\_allow$=1 and NACK contains $rtx\_stop$:
      • $rtx\_allow \leftarrow 0$, $rtx\_stop \leftarrow null$, $rtx\_count \leftarrow rwnd$
  – reduce $dupwnd$ by amount of data retransmitted
  – if $rtx\_count = 0$: $rtx\_allow \leftarrow 1$, $rtx\_stop \leftarrow$ NACK's highest seq_no
  – transmit new data (as much as $cwnd$ allows)
On arrival of Partial ACK:
  – reduce $dupwnd$ by Partial ACK's ack_no - $hiack$
  – $hiack \leftarrow$ Partial ACK's ack_no, Perform actions for any dupACK
On arrival of New ACK:
  – $dupwnd \leftarrow 0$, $recover \leftarrow null$, $hiack \leftarrow$ New ACK's ack_no,
    $rtx\_count \leftarrow null$, $rtc\_allow \leftarrow 0$
```

Fig. 1. SaTPEP sender reaction to Duplicate Acknowledgements

3 Simulation Experiments

We evaluate the performance of SaTPEP in comparison to SACK-TCP, in a series of simulation experiments using the *ns* simulator [9]. A bi-directional GEO satellite link is used to establish communication between N data senders and N receivers. The data senders are connected to the Uplink Gateway, and the receivers to the Downlink Gateway. They perform bulk data transfers of various file sizes. The packet size is set to 1500 bytes. The propagation delay of the satellite link is set to $275ms$, resulting in a RTT of $550ms$ between UG and DG. Link capacity values range from 2 to $10Mbps$. All other links have a propagation delay of $1ms$, and their capacity is set to $10Mbps$, or $100Mbps$ in the case of a $10Mbps$ satellite link. The packet loss probability, P_{loss}, of the satellite link, ranges from 10^{-6} to 10^{-2}. All other links are error-free. Queue sizes are set to 600 packets for all links, so that end-to-end TCP transfers do not experience congestion loss. The focus of our comparison is on goodput ($file\ size/connection\ duration$), as perceived by the receiver hosts, and on fairness between multiple simultaneous connections.

In the first series of experiments N is set to 1. All other parameters cover the full ranges already mentioned. In order for TCP to be able to eventually fully utilize the satellite link, we have set the TCP *rwnd* to rather high values, from 100 to 500 segments. Figure 2 depicts goodput achieved by SaTPEP and TCP for different P_{loss} values. The file size is $1Mbyte$ and the link capacity $6Mbps$. SaTPEP performs significantly better than TCP because it is able to fully utilize the link after the first RTT of the connection. Frequent losses cause

TCP's *cwnd* to remain low, while SaTPEP still raises *cwnd* high enough to fully utilize the link. Figure 2 also depicts the goodput ratio for SaTPEP to TCP, which rises significantly for $P_{loss} = 10^{-2}$. For a given P_{loss} value, SaTPEP's performance increases for higher file sizes and link capacities, as shown in figure 3. Both graphs are obtained for $P_{loss} = 10^{-3}$.

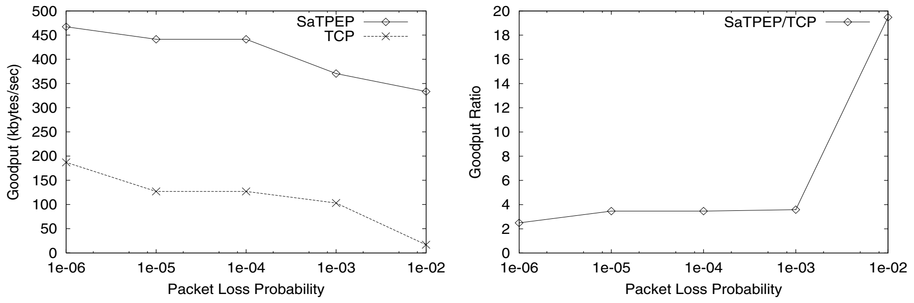


Fig. 2. Goodput and Goodput Ratio for different P_{loss} values

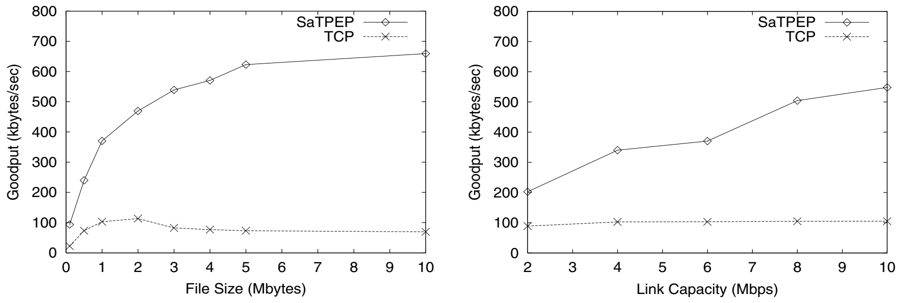
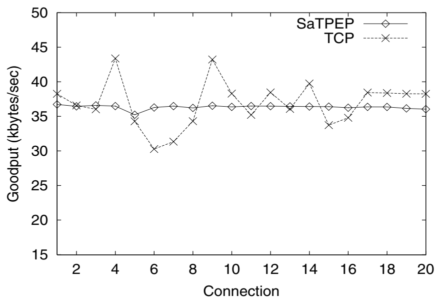


Fig. 3. Goodput for different file size and link capacity values

In the second series of experiments, we set N to 21 and the link capacity to $6Mbps$. At time $t_1 = 1sec$ twenty senders begin transmission of a $2Mbyte$ file each. At time $t_2 = 10sec$ the 21st sender begins transmission of a $500kbyte$ file. The *rwnd* value for TCP connections is set to 25 segments, high enough to result in full utilization of the link, without causing congestion during the initial Slow Start phase. Figure 4 depicts goodput achieved by each of the initial twenty connections for $P_{loss} = 10^{-3}$. It is clear that SaTPEP distributes the link capacity in an even more fair manner than TCP does. The average goodput achieved by the twenty initial connections, along with the goodput of the 21st connection, is shown in figure 4 for different P_{loss} values.



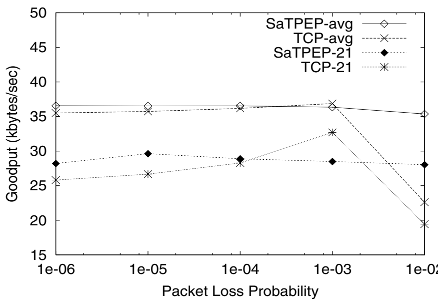


Fig. 4. Goodput of 20 simultaneous connections. Average Goodput of 20 connections, and Goodput of connection 21 for different P_{loss} values

4 Conclusion

In this paper we introduced SaTPEP, aiming at improving TCP performance over satellite links. SaTPEP's flow control is based on link utilization measurements. Loss recovery is based on Negative Acknowledgements. Simulation experiments show significant performance improvement over TCP, in presence of available link capacity, and under high error rates. Under heavy traffic, SaTPEP exhibits remarkable fairness between simultaneous connections.

References

1. C. Partridge and T. Shepard, "TCP/IP Performance over Satellite Links," *IEEE Network Mag.*, pp. 44–49, Sept. 1997.
2. V. Jacobson, "Congestion Avoidance and Control," in *Proc. ACM SIGCOMM*, Stanford, CA USA, Aug. 1988.
3. M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP over Satellite Channels using Standard Mechanisms," RFC 2488, Jan. 1999.
4. M. Allman, S. Dawkins, D. Glover, J. Griner, D. Tran, T. Henderson, J. Heidemann, J. Touch, H. Kruse, S. Ostermann, K. Scott, and J. Semke, "Ongoing TCP Research Related to Satellites," RFC 2760, Aug. 2000.
5. J. Border, M. Kojo, J.Griner, G. Montenegro, and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135, June 2001.
6. I. Minei and R. Cohen, "High-Speed Internet Access Through Unidirectional Geostationary Satellite Channels," *IEEE JSAC*, vol. 17, no. 2, pp. 345–359, Feb. 1999.
7. T. Henderson and R. Katz, "Transport Protocols for Internet-Compatible Satellite Networks," *IEEE JSAC*, vol. 17, no. 2, pp. 326–344, Feb. 1999.
8. I. Akyildiz, G. Morabito, and S. Palazzo, "TCP-Peach: A New Congestion Control Scheme for Satellite IP Networks," *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, pp. 307–321, June 2001.
9. "NS (Network Simulator)," http://www.isi.edu/nsnam/ns/.
10. S. Keshav, "A Control-Theoretic Approach to Flow Control," in *Proc. ACM SIGCOMM*, Zurich, Switzerland, Sept. 1991.