TECHNICAL RESEARCH REPORT

TCP over Satellite Hybrid Networks: A Survey

by Xiaoming Zhou, John S. Baras

CSHCN TR 2002-15
(ISR TR 2002-27)

The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CSHCN/

TCP over satellite hybrid networks: A Survey[1]

Xiaoming Zhou and John S. Baras
Institute for Systems Research and
Center for Satellite and Hybrid Communication Networks
University of Maryland
College Park, MD 20742

Abstract

Satellite is going to play an important role in the global information infrastructure. Satellite can provide direct to home Internet service (i.e. DirecPC from Hughes Network System) and it can also serve as traffic trunk in the middle of the network. About 98 percent of the Internet traffic is TCP traffic. TCP works well in the terrestrial fiber network. However, in the satellite hybrid networks, because of the long propagation delay, large bandwidth-delay product, high bit error rate and downstream/upstream bandwidth asymmetry, TCP performance degrades dramatically. This paper addresses the problems of TCP in satellite data networks and reviews the proposed solutions in the literature. For each solution, the advantages and disadvantages are pointed out. In addition, extensive simulations have been done for the proxy based scheme currently used in the industry to find out how and to what extent the enhancements, such as connection splitting, window scaling and selective acknowledgement, benefit the TCP throughput.

I. Introduction

A. Overview of TCP

TCP is a connection-oriented, end-to-end, process-to-process reliable transport protocol. TCP source and destination port combined with IP source and destination addresses uniquely identify each TCP connection. There are several mechanisms in TCP, such as flow control, congestion control and error control [1,2,3,4].

- ***Flow control***: TCP uses a sliding window to achieve flow control. The TCP receiver sets the receive window field (RCVWND) in the acknowledgement to its free buffer size so that the sender will never overflow the receiver's buffer.

- ***Congestion control***: The TCP sender maintains a state variable CWND for congestion window size. While RCVWND is used to guard that the sender will not overflow the receiver buffer, the CWND is used to guard the sender will not overload the network. The TCP sender can send at most the minimum of RCVWND and CWND window worth packets without receiving any acknowledgement. In the most popular TCP Reno, there are four algorithm used for congestion control which are slow start, congestion avoidance, fast

[1] The material is based upon work supported by NASA under award No NCC3528.

retransmit and fast recovery. Slow start is used upon the start of a new connection to probe the network bandwidth, it increases the CWND by one maximum segment size (MSS) when an acknowledgement is received, which results in increasing congestion window exponentially. TCP stays in slow start until its CWND is greater than the slow start threshold. After that TCP gets into congestion avoidance, it increases CWND about one MSS per round trip time (RTT). Fast retransmit algorithm is triggered when a fixed number of duplicate acknowledgements (usually 3) are received. TCP retransmits the potential lost packet indicated by the acknowledgement and cuts its CWND to half. After that, it inflates its CWND by one MSS when a duplicate acknowledgement is received. If there is one and only one packet lost in a single window, the inflation can increase the CWND to the original CWND before the loss after about half RTT. After that TCP can send a new packet when each duplicate acknowledgement is received if allowed by the RCVWND. Finally it will send half a window new packets when it receives the first non-duplicate acknowledgement. TCP Reno doesn't like TCP Tahoe, which does not have the fast recovery algorithm and sends half a window packets in burst after the loss has been recovered.

- ***Error control***: Error control is the main component of reliable protocols, which includes error detection and error recovery. TCP uses acknowledgement packet, timer and retransmission to achieve error control. TCP uses cumulative acknowledgement, which means when a packet gets lost, it prevents the acknowledgement from being advanced and the window cannot slide until the lost packet is recovered. The sliding window mechanism actually ties the flow control, congestion control and error control together and it becomes vulnerable when there are losses including congestion loss and packet corruptions in the network [29].

B. Overview of satellite hybrid network

For the home users or small enterprise, using dial-up modem to access the Internet is too slow. In order to provide broadband Internet service for these customers, satellite hybrid network was proposed to solve this last-mile problem (Figure 1). This kind of hybrid network exploits three observations [5]: 1) some rural area may not be reached by fiber network or it may be too expensive to do so 2) satellite hybrid network can provide higher bandwidth to a large geographical area and it is easy to deploy 3) home users usually consume much more data than they generate. So this asymmetric hybrid network fits in the need very well.

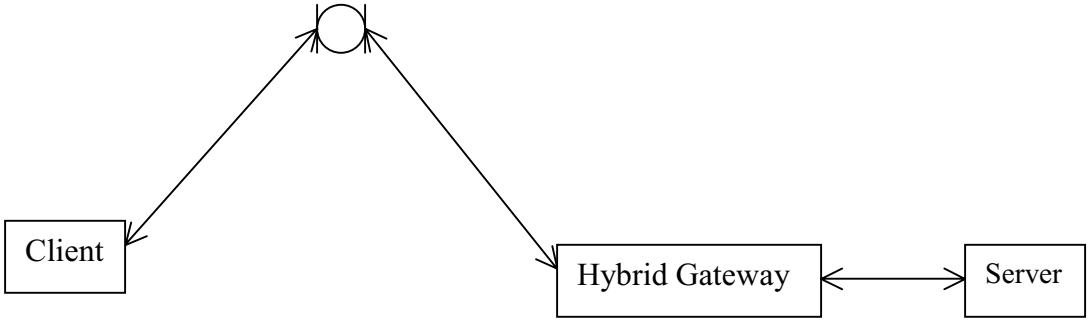


Figure 1: Direct to User satellite hybrid network

Satellites can also be in the middle of the data networks to provide communication between two gateways located geographically far away from each other [6] (Figure 2).

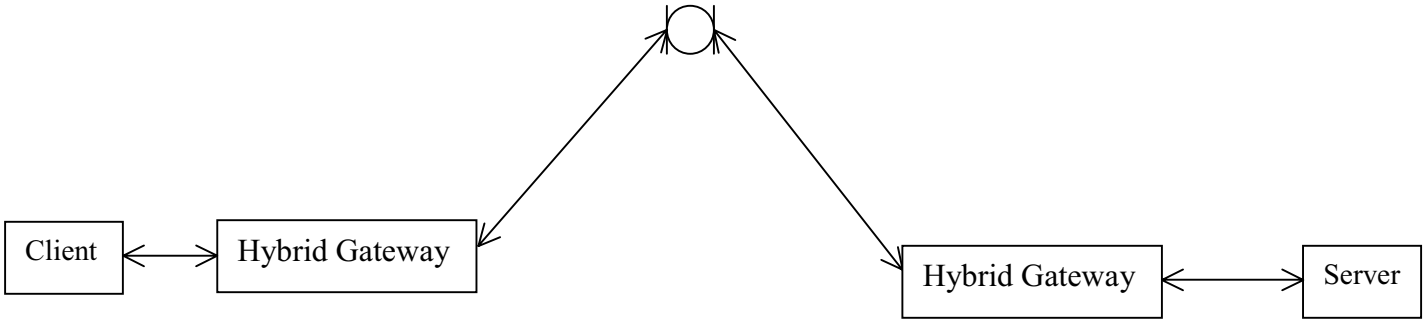


Figure 2: satellite in the middle of the network

II. TCP problems in satellite hybrid network

- ***Long propagation delay*** GEO satellite is about 36,000km above the earth. The propagation delay from the earth up to the satellite and from the satellite down to the earth is about 125ms. Therefore a typical round trip time for two-way system is about 580ms including about 80ms RTT for the terrestrial networks. The time TCP spends in slow start equals $RTT*\log_2 SSTHRESH$ when every segment is acknowledged and equals $RTT*\log_{1.5} SSTHRESH$ [7] when every other segment is acknowledged. For a connection with large RTT, it spends a long time in slow start before reaching the SSTHRESH. For short transfers, they could be finished in slow start, which obviously does not use the bandwidth efficiently. Some researchers propose to use a larger initial widow [8] (i.e. 4 MSS) rather than one for slow start. So files less than 4K bytes can finish its transfer in one RTT rather than 2 or 3. Another proposal [9] is to cancel the delay acknowledgement mechanism in the slow start so every packet get acknowledged and the sender can increase its CWND more quickly. For bulk transfer, TCP throughput is inverse proportional to RTT [10]. So TCP connection with larger RTT does not get its fair share of the bandwidth when it competes with the connections with shorter RTT. Using simulations, Henderson claims the 'Constant-rate' additive increase policy can correct the bias against connection with long RTT [11]. However it is difficult to implement this policy in a heterogeneous network.

- ***Large bandwidth-delay product*** The bandwidth delay product in this system is very large. In order to keep the pipe full, the window should be at least the bandwidth delay product [9,12]. However the receiver advertised window which is 16 bits in the TCP header cannot be more than 64k, which limits the two-way system throughput to 64k/580ms=903Kbps. Window scaling [18] is used to solve this problem. However when the window is large, it is quite possible for multiple losses in one window, which leads to poor performance.

- ***High Bit Error Rate*** Ka band satellite channel is noisier than fiber channel. Bit error rates of the order of 10E-6 and of 10E-4 as worst case are often observed [13]. Because TCP Reno treats all losses as congestion in the network. This kind of link layer corruption can cause TCP to drop its window to a small size and leads to poor performance. Forward error correction (FEC) coding is usually used in satellite communication to reduce the bit error rate. FEC can reduce the BER to 10E-10 for about 99.5% of the time for the terminals and 99.75% of the time for the gateways. However, FEC consumes some bandwidth by sending redundant information together with the data and transforms the original random error nature to one with bursty errors [14,15]. TCP SACK [16] was proposed to convey non-contiguous segments received by the receiver in the ACKs so that the sender can recover error much faster than TCP Reno, which well known can recover only one loss per RTT.

- ***Bandwidth Asymmetry*** The downstream bandwidth from the satellite to the earth terminals is much higher than the upstream bandwidth. The upstream link is shared by all terminals, and it could get congested. The congestion in upstream could lead to poor performance in the downstream link because TCP uses ACKs to clock out data. In the best case, the ACKs are not lost, but queued, waiting for available bandwidth. This has a direct consequence on the retransmission timer and slows down the dynamics of TCP window. Most of the time the upstream is transferring pure ACKs. However if the users are sending data (say email with large attachment or upload file using FTP), a lot of data packets could be queued in front of ACKs in a FIFO queue, which increases the ACKs delay dramatically. To alleviate these problems, ACK filtering was proposed to drop the ACKs in the front of the queue. In the two-way transfer, a priority queue can be used to schedule the ACK to be sent first [17].

III. Proposed solutions

A. End-to-end window based solutions

- ***TCP enhancements*** TCP enhancements include large initial window [8], byte counting, delayed ACKs after slow start [9], TCP for transaction [19], selective acknowledgement [16, 33] and forward acknowledgement [20]. Read the appendix for details.

- ***TCP Peach*** TCP peach [21] has two new algorithms sudden start and rapid recovery, which replace the slow start and fast recovery algorithm in TCP Reno respectively. Essentially TCP Peach has two channels, one is for the data transmission and another one is for bandwidth probing. TCP peach uses low priority dummy segments to probe the bandwidth. The dummy segment is the last transmitted data segment and does not carry any new information. All the routers along the path should implement some kind of priority mechanism. During the sudden start upon the connection establishment, the sender sends one data segment as in TCP Reno as well as RWND-1 dummy segments. The timing for sending these segments is uniformly distributed in one RTT. If there is no congestion along the path, the acknowledgements of the dummy segments will get back to the sender, which are used to increase the CWND so that the CWND could reach the RWND one RTT later. If one or more routers along the path are congested, the low priority dummy segments are

dropped so that they do not affect the bandwidth of the data channel. Rapid recovery can distinguish link error corruption from congestion loss using the same idea. After the fast retransmit, the sender sends dummy segments to probe the channel. If the loss is due to link layer, most of the dummy segments should arrive the destination and their ACKs are used to increase the CWND. If the loss is due to congestion, all the dummy segments will be dropped and rapid recovery just behaves as fast recovery. The problem with TCP peach is that dummy segments do not carry any information and they are overhead to the data. Another problem is that all the routers need to implement priority mechanism, which makes it difficult to deploy. Finally TCP peach just addresses the slow start and link layer error problem. The bandwidth asymmetry problem has not been addressed.

B. TCP connection splitting

- ***I-TCP***. The basic idea of indirect TCP [22] is that the end-to-end TCP connection is now divided into two connections, one is from the server to the base station and another one is from base station to the mobile users. The base station sends premature acknowledgements to the server and takes responsibility to relay the data to the mobile host reliably. The advantages are separation of flow control and congestion control of wireless and wired network, faster reaction to link layer loss. However this scheme violates the end-to-end semantics of TCP. In I-TCP, it is possible the sender receives an acknowledgement of a data packet while the data packet has not reached the destination rather is buffered at the base station. The authors argue that many applications such as FTP and HTTP use application layer acknowledgements in addition to end-to-end TCP acknowledgements. Using I-TCP for these applications does not comprise end-to-end reliability.

- ***M-TCP***. M-TCP [23] also uses connection splitting. One novel idea in M-TCP is that the base station may set the advertised receiver window size to zero so that it can choke the sender and force it into persist mode. While in this state, the TCP sender will not suffer from timeouts, retransmission timer back off and the congestion window stays open. This idea can be used as a flow control mechanism by the base station when there is congestion at the base station. TCP Connection splitting is always criticized for extra processing overhead and scalability problem. Using simulation, this papers shows that a 100 MHz Pentium PC can adequately handle numerous simultaneous connections with little performance degradation.

- ***Super TCP*** Because satellite channel is a FIFO channel, out-of-order routing and congestion on the satellite link are impossible. [24] proposes to use one duplicate ACKs to trigger the retransmission at the base station and to use a fixed window size for the satellite TCP connection. In order for the fast retransmit algorithm to recover the loss, the congestion window size has to be greater than four for single packet loss and has to be greater than ten for two consecutive losses in one window. While for three or more consecutive losses in one window, the TCP sender has to wait for timeout to recover the loss. The paper proposes a new sender algorithm using the same idea as in TCP new Reno [25,26]. It uses partial ACKs to calculate the bursty loss gap and sends all the potential loss packets beginning from the partial acknowledgement number. Although it is possible that the sender could retransmit packets that have already been correctly received by the

receiver, it was shown that this algorithm performs better than TCP SACK in recovering bursty errors.

C. Rate based solutions

- ***TCP Vegas*** TCP Vegas [27] addresses the congestion problem from another perspective. It does not use packet loss as the indication of congestion. Rather it uses the transmission rate for congestion control. Every round trip time, the sender calculates its transmission rate based on the sending window and the measured RTT. This rate is compared with the expected rate, which is Transmission window/Base RTT. Base RTT is the smallest RTT measured so far. The basic idea is that if there is no congestion in the network the measured rate should be closed to the expected rate. Two thresholds are used to trigger additive increase or decrease, depending of whether the channel is under-utilized or over-utilized. TCP Reno always increases its congestion window if there is no loss and periodically causes the packet dropped and window oscillation. TCP Vegas can decrease its window in congestion avoidance, it does not cause window oscillation once it reaches the equilibrium. The RTT measured by the sender may be caused by the congestion in the reverse path rather than the forward path. So TCP Vegas does not work well in the asymmetric path case.

- ***SCPS-TP*** Space communication protocol standards-transport protocol (SCPS-TP) [28] is a set of TCP extensions for space communications. This protocol adopts the Timestamps and window scaling options in RFC1323 [18]. It also uses TCP Vegas low-loss congestion control mechanism. SCPS-TP receiver doesn't acknowledge every data packet. ACKs are sent periodically based on the RTT. The traffic demand for the reverse channel is much lighter than in the traditional TCP. However it is difficult to determine the optimal acknowledgement rate and the receiver may not respond properly to congestion in the reverse channel. Because there is no regular acknowledgement-driven clock, it uses an open-loop rate control mechanism to meter out data smoothly. To transmit data continuously in the presence of link layer loss rather than congestion loss is especially important. SCPS-TP uses selective negative acknowledgement (SNACK) to address this problem. SNACK is a negative acknowledgement and it can specify a large number of holes in a bit-efficient manner.

- ***STP*** Satellite transport protocol (STP) [13] is very familiar to SCPS-TP. STP adapts an ATM-based protocol for use as a transport protocol in satellite data networks. STP can get comparable performance to TCP SACK in the forward path. However STP consumes significantly less bandwidth in the reverse path. The transmitter sends POLL packets periodically to the receiver, the receiver sends STAT packet as acknowledgements and the reverse path bandwidth requirement depends mainly on the polling period, not on the forward path data transmission rate. Therefore the bandwidth demand for the reverse path decreases dramatically. STP can be used as the proprietary protocol in the satellite connection.

- ***NETBLT.*** NETBLT [29] is a reliable transport layer protocol designed for bulk data transfer over satellite IP networks. Based on the observations that window and timers

perform poorly in synchronizing the end states and widow mechanism, which ties flow control and error control together, becomes vulnerable in the face of data loss, NETTBLT decouples flow control from error control. It uses rate rather window for flow control and the rate control parameters are negotiated during the connection setup and periodically updated. Data is divided into large fixed sized blocks called "buffers" not like TCP, which is a stream data protocol.

D. Link Layer solution

- ***Snoop***. Snoop TCP [30,31] was originally designed for last-hop wireless network. Wireless network has a lot of similarities to satellite network. Both of them have link layer corruption, wireless communication media and bandwidth asymmetry. However the satellite network has a much longer round trip delay than the wireless network and the satellite network does not have the handoff problem as in wireless network. Snoop essentially uses the TCP acknowledgements to trigger the link layer retransmission at the base station and suppresses the duplicate ACKs from propagating to the TCP sender therefore it can shield the link layer loss and does not drive the TCP sender to cut its window to half as end to end TCP does. Although Snoop does not have any TCP layer code running at the base station, it still need to access the TCP header to get the sequence number and acknowledgement number. It does not work if IPSEC is used. Snoop preserves the end-to-end semantics of TCP. However Snoop cannot be used for satellite network because the long propagation delay of the satellite link could cause fairness problem if the base station keep the ACKs to transmit end to end.

- ***Reliable link layer protocol*** Another solution to shield the link layer corruption from TCP is to implement a reliable link layer protocol. The problem with this solution is that it needs to be designed very carefully for it to work together with TCP. It is possible for the error to trigger the link layer retransmission while the duplicate ACKs of TCP propagate to the TCP source and cause TCP to halve its window and to retransmit the same packet. Another problem is that not all the up layers need reliable link layer service, e.g. real-time traffic using UDP does not need reliable data transmission. According to the end-to-end argument [32], the reliable service can completely and correctly be implemented only with the knowledge and help of the application at the end hosts. However link layer can implement an incomplete version of the reliable service for a performance enhancement.

- ***Adaptive forward error correction*** Usually FEC uses K symbols for information and N-K symbols for redundancy. When more redundancy is added or a strong error correction code is used, the packet error rate is decreased. However the effective bandwidth for useful information transmission is reduced given the total raw bandwidth is fixed. [10] gives a TCP throughput formula in which the throughput is proportional to $1/(RTT*\sqrt{p})$. The throughput in this formula is the maximum achievable throughput, which increases monotonically when the packet error rate decreases. The actual throughput is the minimum of the effective throughput and the achievable throughput. The basic idea in AFEC [15] is to find a code (K,N) so that the effective throughput is equal to the achievable throughput which is actually is maximum throughput of the TCP connection. The problem with this scheme is that it increases the coding and decoding overhead both at the base station and at

the mobile hosts. And like Snoop, it still has the unfairness problem when used for the satellite network.

Conclusion:

Because of the long propagation delay, large bandwidth-delay product, high bit error rate and upstream/downstream bandwidth asymmetry, TCP performance degrades significantly over satellite data networks. There are several proposals in the literature target to solve these problems, which include end-to-end window based, rate based, proxy-based protocols as well as link layer enhancements. All these proposals are not independent of each other. A better solution can combine some of them and comes up with a new protocol for satellite network. All in all, the new protocol should decouple the flow control from error control and fill the long-fat pipe with enough data packets to improve throughput as well as keep the upstream channel bandwidth requirement small. For short transfers, such as HTTP, new protocol can use a streamline handshake mechanism, which combines the handshake of application layer and transport layer together. This mechanism can save one or two round trip time, which is significant for short transfers considering the large end-to-end delay of satellite networks.

Reference：

1. J. Postel, "Transmission control protocol," *Internet RFC 793*, 1981.
2. W. Stevens, *TCP/IP Illustrated, Volume 1*. Reading, MA: Addison-Wesley, 1994.
3. Van Jacobson, "Congestion avoidance and control," Proceedings of SIGCOMM '88, Aug. 1988
4. M. Allman, V. Paxson, W. Stevens, "TCP congestion control", *Internet RFC 2581,* April 1999
5. V. Arora, N. Suphasindhu, J.S. Baras, D. Dillon, "Asymmetric Internet Access over Satellite-Terrestrial Networks", *CSHCN Technical Report 96-10* available at http://www.isr.umd.edu/CSHCN
6. V. G. Bharadwaj "Improving TCP Performance over High-Bandwidth Geostationary Satellite Links" *Master Thesis CSHCN M.S. 99-7* http://www.isr.umd.edu/CSHCN
7. C. Partridge and T. Shepard, "TCP performance over satellite links," *IEEE Network*, vol. 11, pp. 44–49, Sept. 1997.
8. M. Allman, S. Floyd, C. Partridge, "Increasing TCP's Initial Window," *Internet RFC 2414*, September 1998
9. M. Allman *et al.*, "Ongoing TCP research related to satellites," *RFC2760*, Feb. 2000.
10. Padhye, J.; Firoiu, V.; Towsley, D.F.; Kurose, J.F. "Modeling TCP Reno performance: a simple model and its empirical validation," *IEEE/ACM Transaction on Networking*, April 2000
11. T. Henderson, E. Sahouria, S. McCanne, and R. Katz, "On improving the fairness of TCP congestion avoidance," in *Proc. IEEE GLOBECOM'98 Conf.*, 1998
12. T. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss," *IEEE/ACM Trans. Networking*, vol. 5, pp. 336–350, June 1997.
13. T. R. Henderson and R. H. Katz, "Transport protocols for Internet-compatible satellite networks," *IEEE J. Select. Areas Comm.*, vol. 17, pp.326–344, Feb. 1999.

14. Chadi Barakat and Eitan Altman, “Bandwidth tradeoff between TCP and link-level FEC”, in *Proceedings of IEEE International Conference on Networking*, Colmar, France, Jul 2001.
15. Benyuan Liu, Dennis L. Goeckel, Don Towsley “TCP-Cognizant Adaptive Forward Error Correction in Wireless Networks,” *UMass CMPSCI Technical Report TR 01-30,* 2001
16. M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP selective acknowledgment options,” *Internet RFC 2018*, 1996.
17. H. Balakrishnan, V. Padmanabhan, and R. Katz, “The effects of asymmetry on TCP performance,” in *Proc. 3rd ACM/IEEE MobiCom Conf.*, Sept. 1997, pp. 77–89.
18. V. Jacobson, R. Braden, and D. Borman, “TCP extensions for high performance,” *Internet RFC 1323*, 1992.
19. R. Braden, “T/TCP—TCP extensions for transactions, functional specification,” *Internet RFC 1644*, 1994.
20. M. Mathis and J. Mahdavi, “Forward acknowledgment: Refining TCP congestion control,” in *Proc. ACMSIGCOMM*, Aug. 1996, pp. 281–291.
21. Akyildiz, I.F., Morabito, G., Palazzo, S., **"**TCP Peach: A New Congestion Control Scheme for Satellite IP Networks**,"** *IEEE/ACM Transactions on Networking*, Vol. 9, No. 3, June 2001
22. A. Bakre and B. R. Badrinath, “Implementation and performance evaluation of indirect TCP” *IEEE Transactions on Computers* Vol. 46 No. 3, March 1997
23. K. Brown and S. Singh, “M-TCP: TCP for mobile cellular networks,” *ACM Comput. Commun. Rev.*, vol. 27, pp. 19–43, Oct. 1997.
24. I. Minei and R. Cohen “High-speed internet access through unidirectional geostationary satellite channels”, *IEEE J. Select. Areas Commun.*, Vol. 17 Feb 1999
25. S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," *Internet RFC 2582 (Experimental),* April 1999.
26. J. Hoe, “Improving the start-up behavior of a congestion control scheme for TCP,” in *Proc. ACM SIGCOMM*, Aug. 1996, pp. 270–280.
27. L. Brakmo and L. Peterson, “TCP Vegas: End to end congestion avoidance on a global Internet,” *IEEE J. Select. Areas Commun.*, vol. 13, pp.1465–1480, Oct. 1995.
28. R. C. Durst, G. Miller and E. J. Travis, “TCP extensions for space communications,” *Proc. ACM Mobicom*, ’96, Nov 1996
29. D.D. Clark, M.L. Lambert, and L. Zhang. Netblt: A high throughput transport protocol. *1987 ACM SIGCOMM Conference*, pages 353--359, August 1987
30. H. Balakrishnan, S. Seshan, and R. H. Katz, “Improving reliable transport protocol and handoff performance in cellular wireless networks,” *ACM-Baltzer Wireless Networks J.*, vol. 1, no. 4, pp. 469–481, Dec.1995.
31. H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, “A comparison of mechanisms for improving TCP performance over wireless links,” *IEEE/ACM Trans. Networking*, vol. 5, Dec. 1997.
32. J.H. Salzer, D.P. Reed and D.D. Clark, "End-to-end arguments in system design", *ACM Transactions on Computer Systems*, Nov 1984, p. 277-288
33. K. Fall and S. Floyd, “Simulation-based comparisons of Tahoe, Reno, and SACK TCP,” *ACM Comput. Commun. Rev.*, vol. 26, pp. 5–21, July 1996.

Appendix:

TCP over Satellite Simulations

/* ********** */
Note:
SACK in OPNET 7.0 does not work.
It has been fixed in the following simulations.

Window Scaling does not work either.
It has also been fixed in the following simulations.

/* ********** */
All the following simulations are done with OPNET 7.0. The goal is to test which TCP enhancements will benefit the TCP performance over satellite. The TCP enhancements are connection splitting, window scaling, SACK etc.

The options we have in OPNET 7.0:

> Path MTU Discovery
NO. We can just set TCP payload Maximum Segment Size (MSS), which is usually equal to Maximum size MAC layer can handle, minus the size of TCP and IP headers.

> T/TCP
NO.

> Larger Initial Window
YES. We can set it to 1, 2, or 4 etc. We also can use the algorithm defined in RFC-2414 where the initial window is set to min [4*MSS, max (2*MSS, 4380)].

> Delayed ACKs after Slow Start
No. Delayed ACKs mechanism can be Clock Based or Segment/Clock based. However, there is no option to disable delayed ACKs while in slow start. The delayed ACKs mechanism in OPNET 7.0 is used for both slow start and congestion avoidance.

> Window Scaling
YES. As defined in RFC-1323.

> FEC
NO.

> SACK
YES. As defined in RFC-2018.

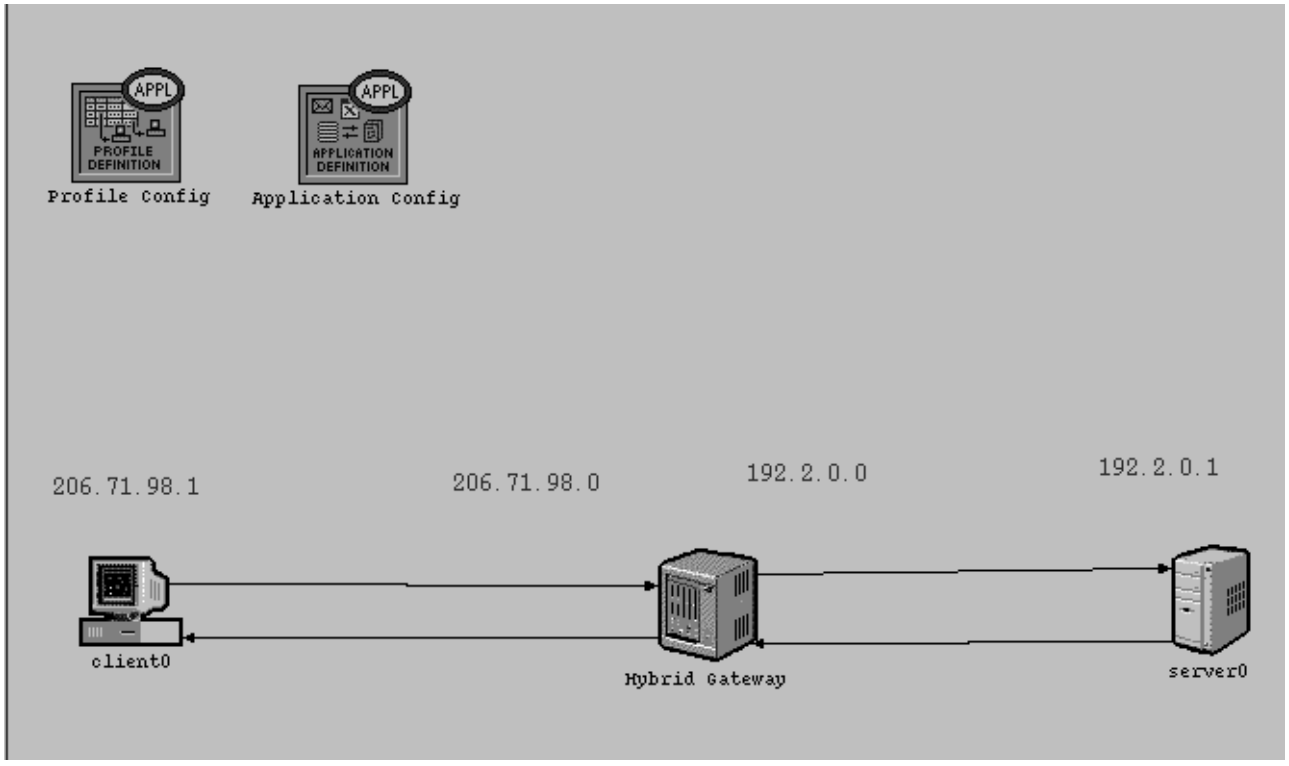
> TCP header compression
NO.

We can also choose different TCP flavors such as Tahoe, Reno. However there is no New Reno and Vegas. We also can set the Receiver Buffer size and receiver buffer usage threshold. These are important parameters that the application delay is sensitive to. It also affects the window size.

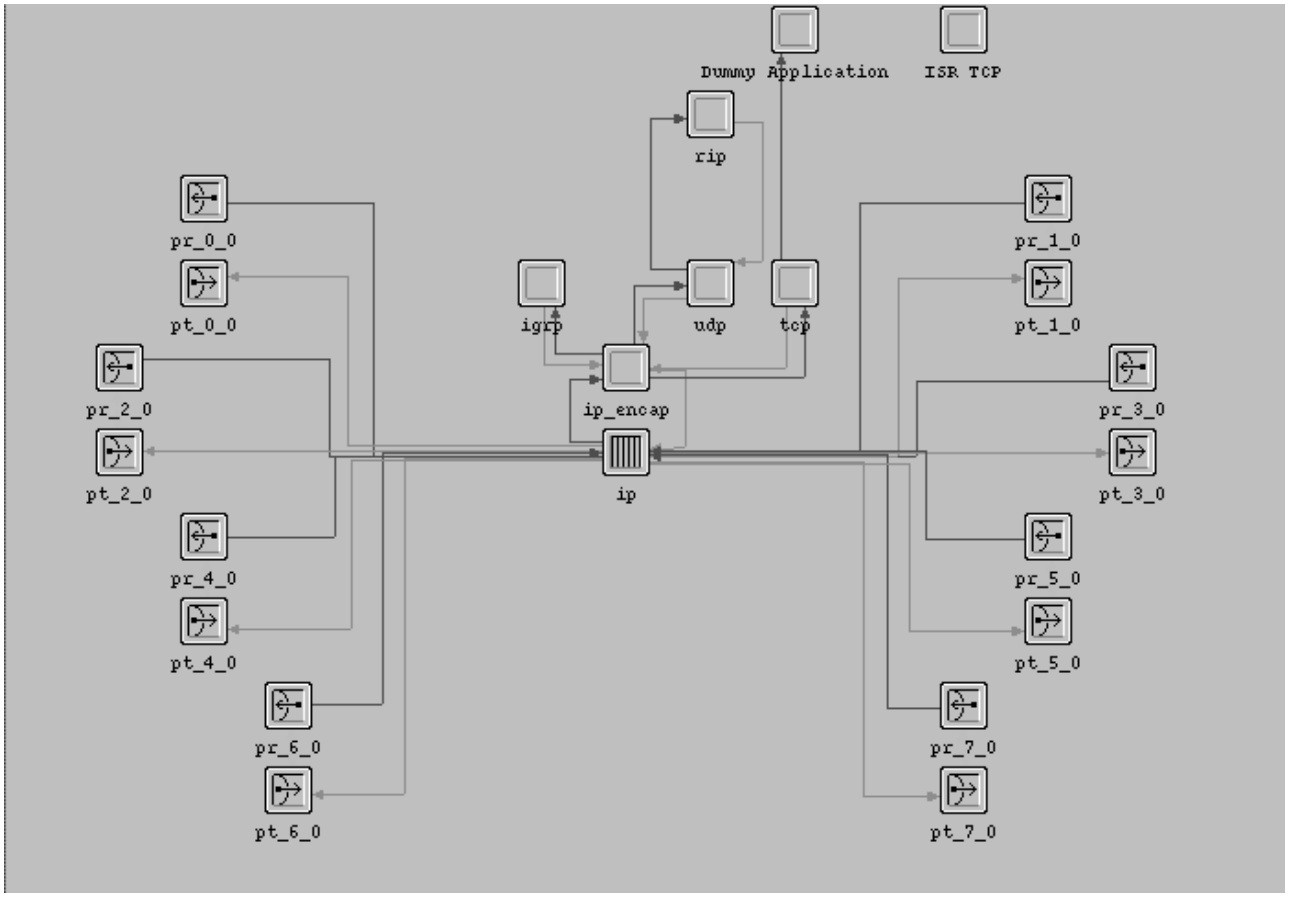
All the following protocol-specific algorithm can be either enabled or disabled in a given TCP implementation:

- Fast Retransmit
- Fast recovery
- Window Scaling
- SACK
- Nagle's Silly Window Syndrome Avoidance
- Karn's algorithm to avoid Retransmission ambiguity.

1. Simulation Topology



2. Enhanced hybrid gateway internal structure



3. Simulation parameters and baseline model

A. Application: An FTP application (16M bytes)

B. Transport protocol: TCP Reno

C. All the link bandwidth is DS1

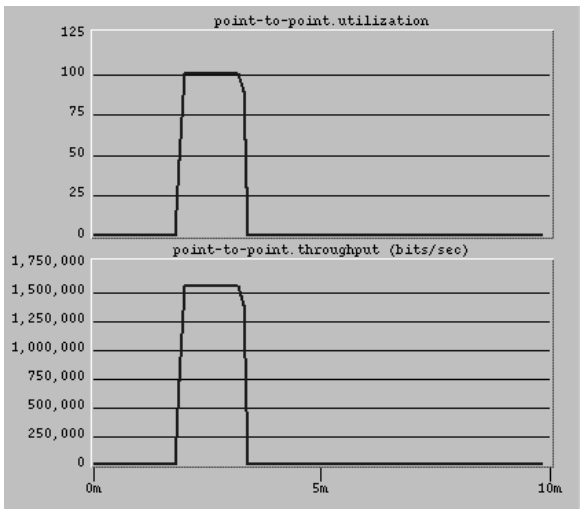
D. All receive buffer sizes are set to 64K bytes.

E. The link delay between the hybrid gateway and the client is 250ms and the link delay between server and hybrid gateway is 40ms. Therefore the RTT between server and client is 580ms, the RTT between hybrid gateway and client is 500ms.

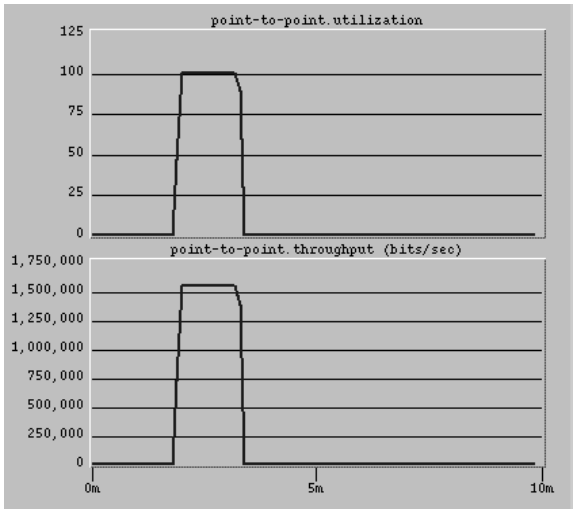
F. The statistics we collect is link utilization and link throughput of the two downstream links

In the baseline model, the link delay is distance based. So it is almost 0ms. It is the best throughput and end-to-end delay we can get. It is used to compare with other scenarios.

3.1 Without Connection Splitting

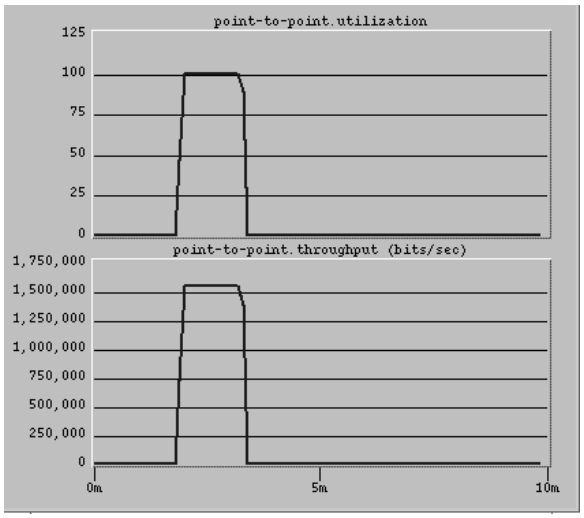


Gateway to Client

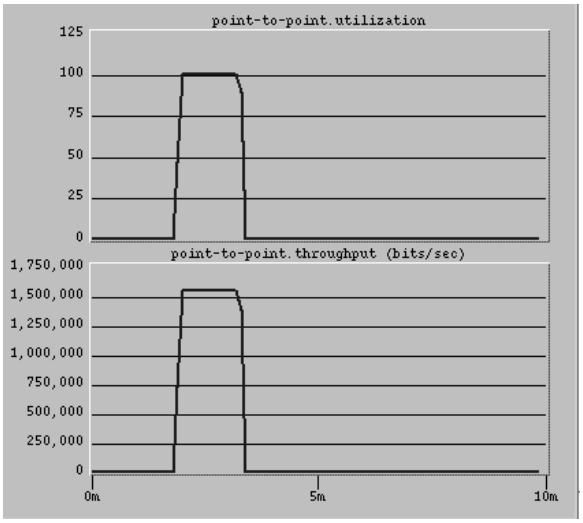


Server to Gateway

3.2 With Connection Splitting



Gateway to Client

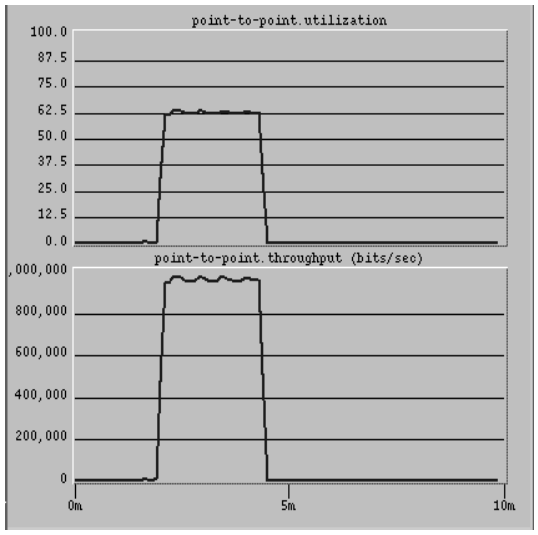


Server to Gateway

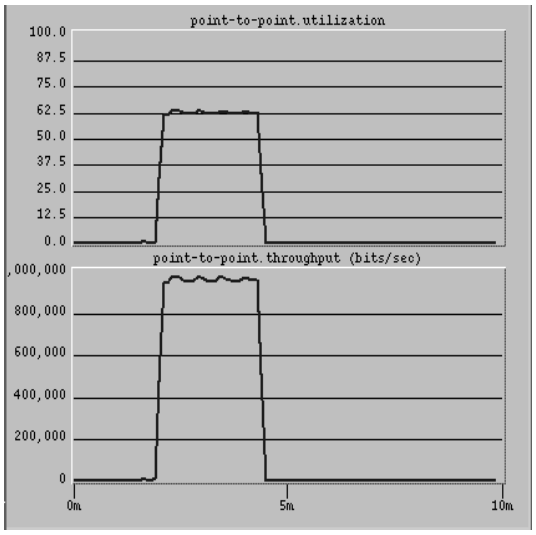
In the baseline model, we can see from the above figures that both links' throughput reaches DS1 and link utilizations are 100%. Because the file size is very large (16M bytes), the slow start phase, which does not utilize the bandwidth efficiently, is amortized by the congestion avoidance phase. Because we do not model the processing overhead of the connection splitting, we got the same results above. If the processing overhead is considered, the throughput is expected to be smaller.

4. Error free case

4.1 No connection Splitting and No window scaling



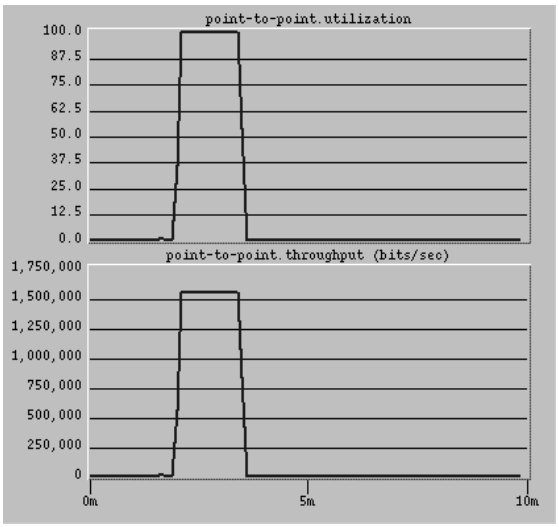
Gateway to Client



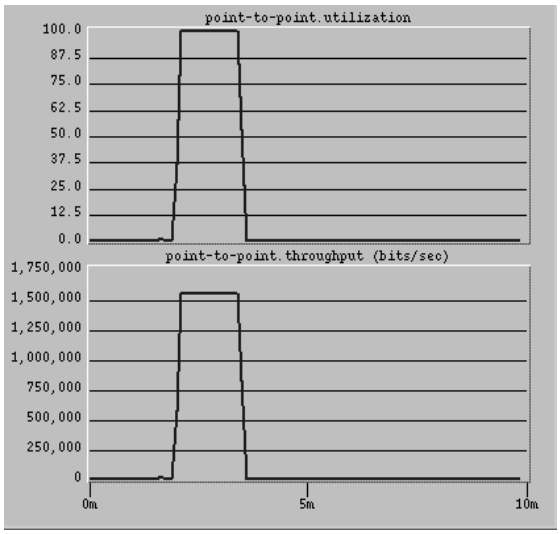
Server to Gateway

Because we set the receiver buffer size to 64K bytes, without window scaling, the maximum throughput we can get is 64kbytes/580ms=903,944bps. And DS1=1,544,000bps. So the maximum link utilization is 903,944/1,544,000=58.55%. This MSS is 512 bytes, the TCP header is 20 bytes and the IP header is 20 bytes, so the actual throughput is (512+20+20)/512*58.55% = 63.12%.

4.2. No connection Splitting and Window Scaling

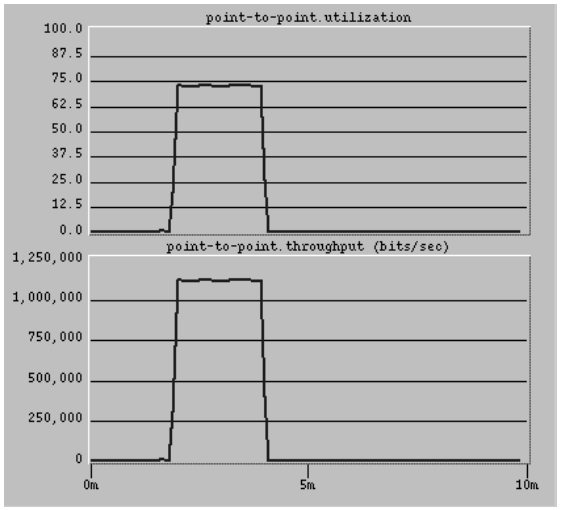


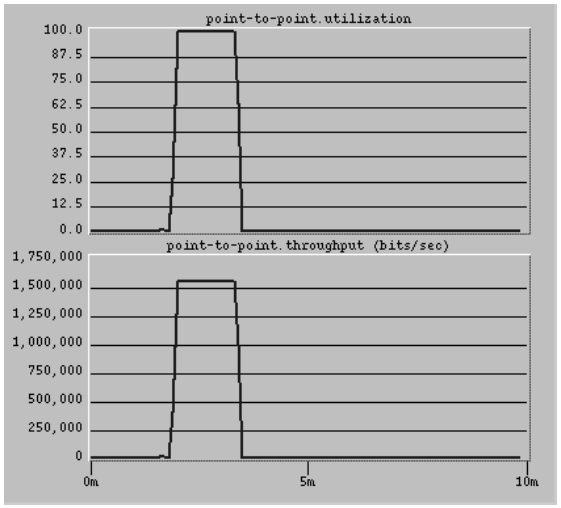
Gateway to Client



Server to Gateway

4.3 Connection Splitting and no window scaling

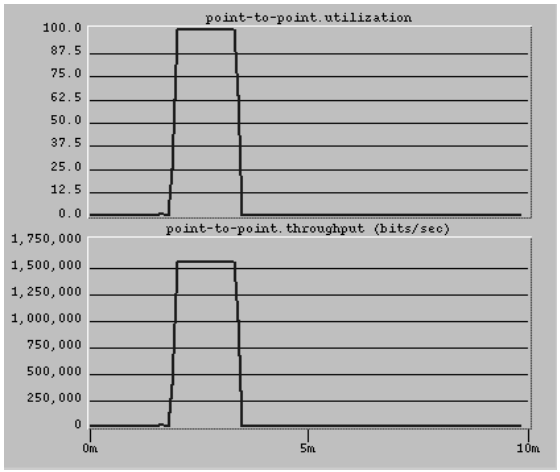


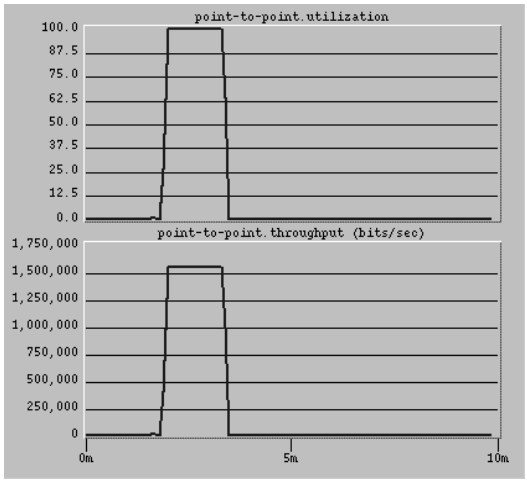


Gateway to Client Server to Gateway

Because we set receiver buffer size to 64K bytes, without window scaling, the maximum throughput we can get is 64kbytes/500ms=1,048,576bps. And DS1=1,544,000bps. So the maximum link utilization is 1,048,576/1,544,000=67.91%. This MSS is 512bytes, the TCP header is 20 bytes and the IP header is 20 bytes, so the actual throughput is (512+20+20)/512*67.91% = 73.21%.

4.4 Connection Splitting and window scaling

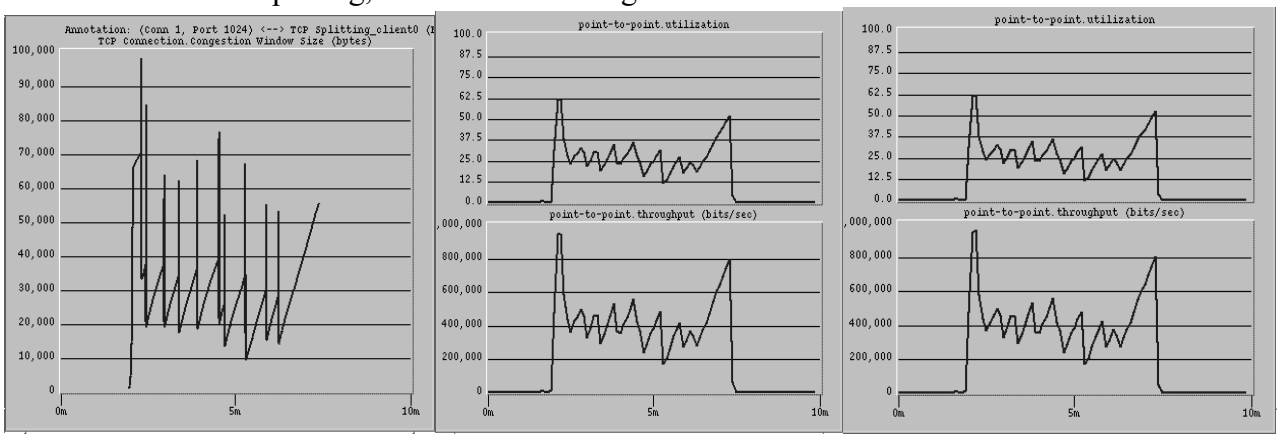




Gateway to Client Server to Gateway

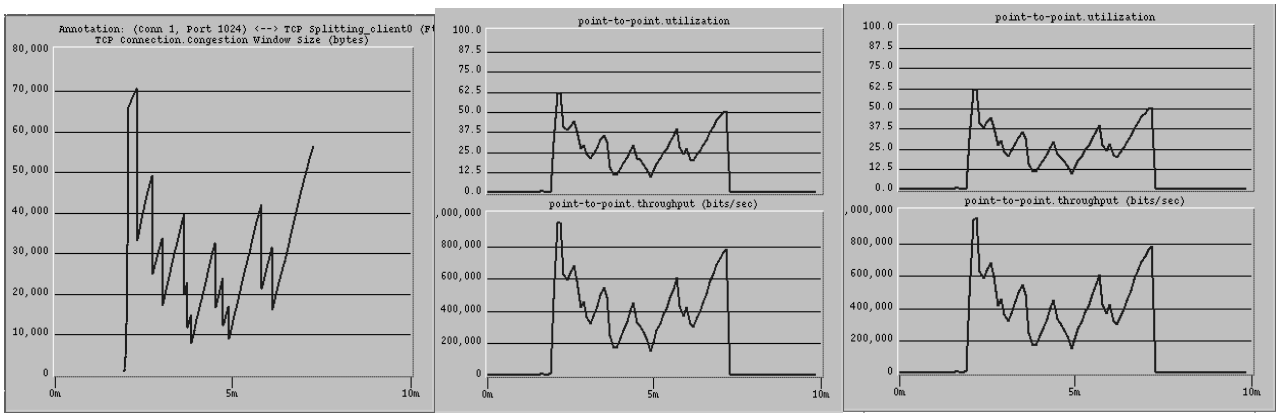
5. Introduce BER 10E-7 over the download between Hybrid gateway and client

5.1 No Connection Splitting, No Window scaling and No SACK



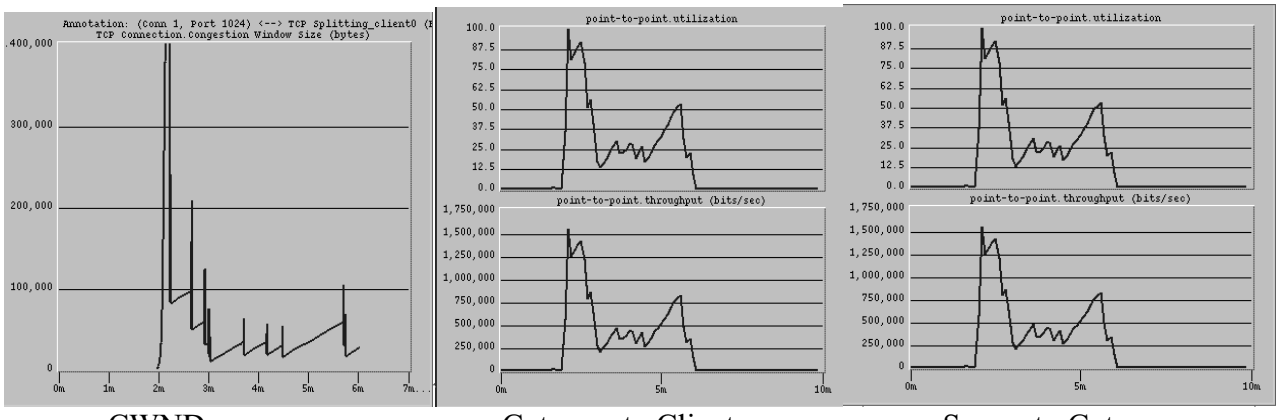
CWND Gateway to Client Server to Gateway

5.2 No Connection Splitting, No Window scaling and SACK



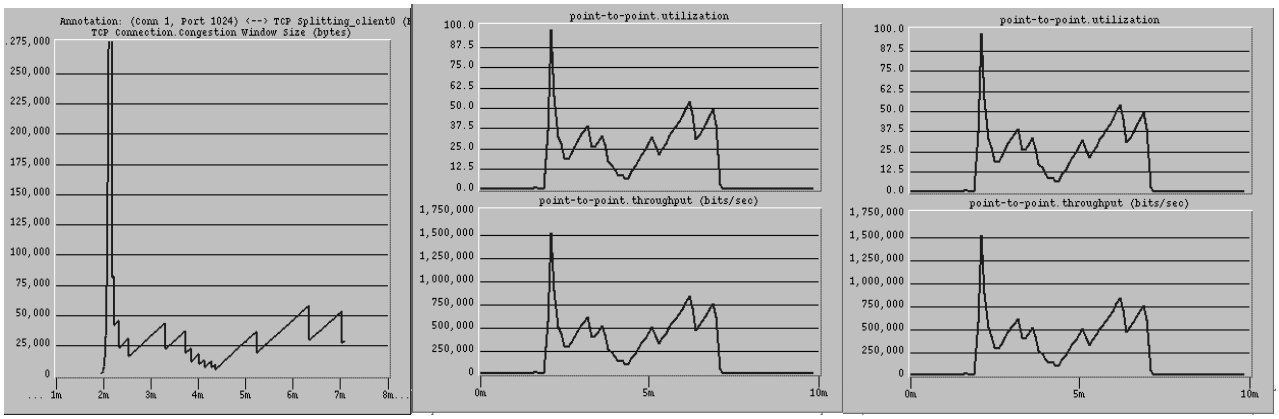
CWND Gateway to Client Server to Gateway

5.3 No Connection Splitting, Window scaling and No SACK (Rcv_buf = 160 k)



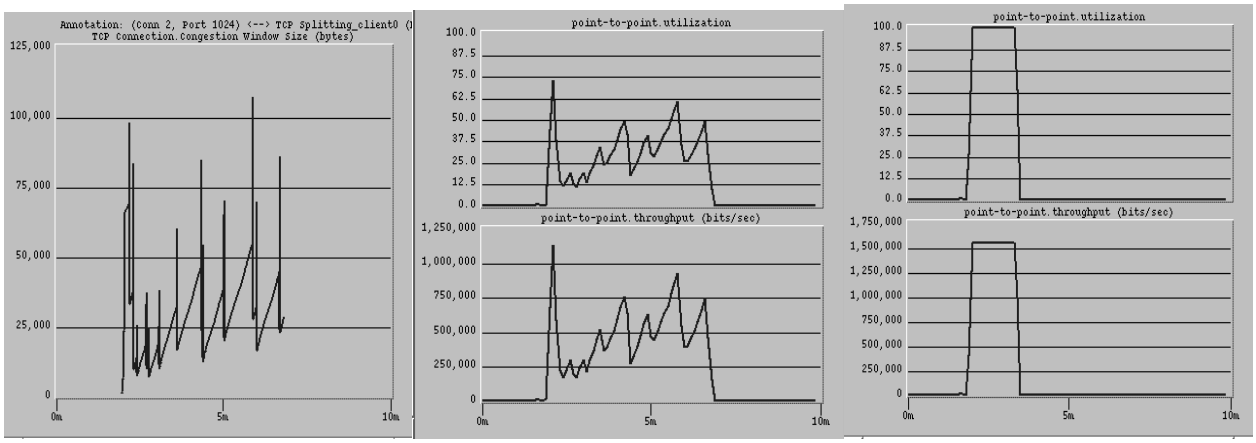
CWND Gateway to Client Server to Gateway

5.4 No Connection Splitting, Window scaling and SACK (Rcv_buf = 160 k)



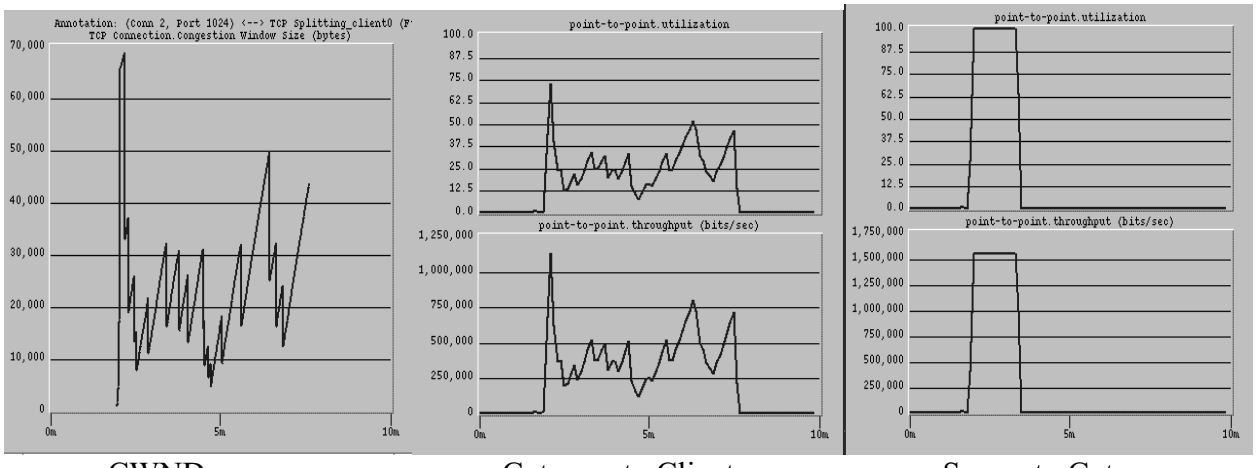
CWND Gateway to Client Server to Gateway

5.5 Connection Splitting, No Window scaling and No SACK



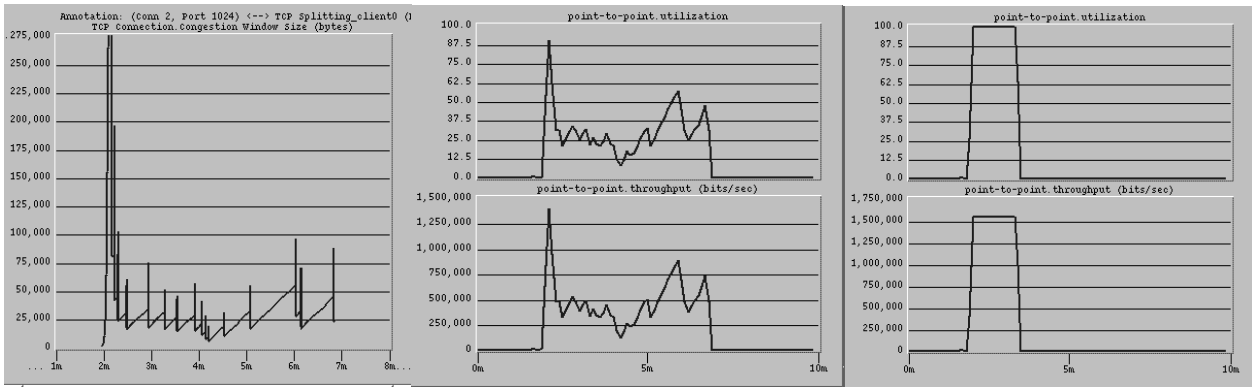
CWND Gateway to Client Server to Gateway

5.6 Connection Splitting, No Window scaling and SACK



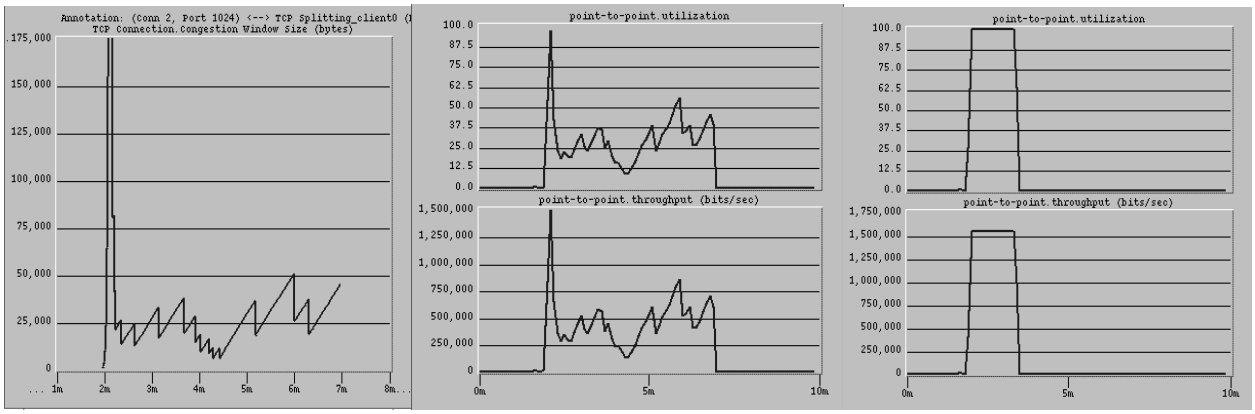
CWND Gateway to Client Server to Gateway

5.7 Connection Splitting, Window scaling and No SACK (rcv_buf = 160K)



CWND Gateway to Client Server to Gateway

5.8 Connection Splitting, Window scaling and SACK (rcv_buf = 160K)



CWND Gateway to Client Server to Gateway