Improvements Achieved by SACK Employing TCP Veno Equilibrium-Oriented Mechanism over Lossy Networks

Chung Ling Chi, Fu Chengpeng, and Liew Soung Chang.
Department of Information Engineering
Chinese University of Hong Kong, Shatin, N.T., Hong Kong.
lcchung9@ie.cuhk.edu.hk, cpfu7@ie.cuhk.edu.hk and soung@ie.cuhk.edu.hk

Abstract

TCP Veno[1] is a new TCP version that employs the refined AIMD (additive increase and multiplicative decrease) congestion control algorithm with its implementation only on the sending side. In addition to achieving significant throughput improvement over TCP Reno (particularly in wireless networks), Veno shows good compatibility and flexibility in heterogeneous networks. In this paper, we demonstrate the benefits of combining the retransmission strategy of SACK option with TCP Veno's congestion control mechanism. Our implementation is referred as SACK TCP$_{Veno}$, similar to the definition of SACK TCP in[4], which refines recovery strategy of Reno by using SACK option[3]. By means of simulation and real network experiments[1], we demonstrate that SACK TCP$_{Veno}$ can obtain the satisfactory throughput improvement over SACK TCP[2]. Also, we show SACK TCP$_{Veno}$'s compatibility with SACK TCP.

1. Introduction

Over past decades, many studies on TCP have been explored of how a TCP connection can recover from a single or multiple packet loss quickly. One common solution is to deploy a Selective Acknowledgement (SACK) option in today's TCP implementations since SACK TCP can recover from the multiple packet losses in a single window within one round trip time. Specifically, SACK TCP deals with bursty losses in an efficient way and avoids the unnecessary retransmit timeouts. However, like other recovery techniques (i.e. NewReno[5], NetReno[6]), SACK TCP or its variants (D-SACK[7], FACK[8], RH[9]) is regarded as one recovery-based algorithm. In fact, regardless of whichever of recovery-based techniques are employed, the sending rate of a TCP connection is halved as long as packet loss is detected.

Recently, a new implementation of TCP congestion control algorithm called TCP Veno [1] that examines the equilibrium point of the network capacity is proposed. It uses the detected equilibrium point as a coarse meter to smartly set the slow start threshold (*ssthresh*), and refines AIMD (additive increase and multiplicative decrease) algorithm of TCP Reno at the sending side. In addition to achieving significant throughput improvement over TCP Reno, particularly in wireless networks, Veno TCPs show good compatibility and flexibility with current TCPs over heterogeneous networks.

In this paper we demonstrate the benefits of combining the retransmission strategy of SACK option with TCP Veno's congestion control policy. We refer the implementation of SACK+TCP Veno as SACK TCP$_{Veno}$. By means of simulations and real network experiments, we compare SACK TCP$_{Veno}$ with SACK TCP in lossy networks. It is observed that a SACK TCP$_{Veno}$ connection has large improvement up to 72% in utilization over SACK TCP. Generally, SACK TCP$_{Veno}$ not only recovers packet losses quickly by using the inherent advantage of SACK option, but also maintains an appropriate window size to transfer data depending on its employed refined AIMD algorithm.

The rest of this paper is organized as follows. In Section 2 we briefly review the congestion control algorithm of SACK TCP and TCP Veno. In Section 2.3, SACK TCP$_{veno}$ are addressed in details. Then we investigate SACK TCP$_{Veno}$'s performance and compare it with the SACK TCP by simulations and experiments in real networks in Section 3. The conclusions are drawn in Section 4.

2. TCP algorithms

2.1. SACK TCP

TCP uses a cumulative acknowledgment scheme in which received segments[3] that are not at the left edge of the receiver window are not acknowledged. When multiple segments in a window are dropped, TCP seriously loses its ack-based clock. The ack with SACK option, which is about up to four noncontiguous blocks of data that have been received successfully by the receiver are proposed to remedy these problem. SACK TCP [4], making use of SACK option in Reno TCP sender, by refining the fast retransmit and fast recovery strategy of TCP Reno, generally can recover from multiple packet losses in a single window within one *RTT* (Round Trip Time). Since SACK is being implemented in most operating systems, a cascade of window reduction or the unnecessary retransmit timeouts due to multiple packet losses will be greatly reduced.

The original SACK option appears in RFC1072 [13] and standardized in RFC2018[3] and RFC2581[14] with some

[1] We have implemented SACK TCP$_{Veno}$ in NetBSD1.1, source code is available on http://www.broadband.ie.cuhk.edu.hk

[2] It is inappropriate for us to compare SACK TCP$_{Veno}$, with SACK TCP, since the former is based on the refined congestion control algorithm and the later on recovery phase of TCP. In fact, these different modifications, both of which are beneficial to TCP from different angles, has not any confliction between them.

[3] In this paper, segment and packet are interchanged without any explicit indication.

modifications. When three duplicate acks are received by a SACK TCP connection, the sender cuts the congestion window into half and an estimating outstanding data variable (*pipe*) is in function to control the number of segment sent during the fast recovery phase. Considering three duplicate acks to implicitly indicate that three outstanding packets are received in the receiver side, the pipe size of TCP connection (*pipe*) is estimated to be (*cwnd-3*) and then cwnd is halved. The variable *pipe* is increased by one segment when sending a packet or decreased by one segment when receiving a duplicate ack or decreased by two when receiving a partial ack[4][10]. Packets can be sent if *pipe* is smaller than *cwnd* in order to keep the outstanding data in a constant amount. SACK TCP retransmits lost segments according to the *scoreboard* information.

A retransmission queue (*scoreboard*) is set up to record which packet is received at the receiver buffer or is retransmitted by the sender. *scoreboard* contains a set of data entities, each of which consists of its sequence number of transmission packet as well as a flag bit, which indicates whether a segment has been "SACKed". A segment with the SACKed bit turned on is not retransmitted, but segments with the SACKed bit turned off and sequence number less than the highest SACKed segment are available for retransmission. Whether the SACKed segment is on or off, segments are only removed from the retransmission buffer when they have been cumulatively acknowledged. Additionally, if packet losses are so heavy and leads to timeout occurrence, the *scoreboard* is cleared. TCP then sets *ssthresh* to half the current congestion window *cwnd* and performs slow start action.

When SACK TCP ends its recovery phase, it enters additive increase phase, in which *cwnd* is increased by one segment per *RTT*. Otherwise, in slow start phase, *cwnd* is exponentially increased by one segment per received ack, as defined in [4], in this paper TCP composed of SACK option and above algorithms is called SACK TCP.

2.2. TCP Veno

TCP Veno operates with the objective of distinguishing loss type, thus performing an appropriate window reduction instead of a fixed drop of window in Reno, and forcing a TCP connection to stay longer at the equilibrium by employing a proactive congestion detection method and a reactive congestion detection method together. Veno differs from the conventional TCP in two ways [1]: 1) It dynamically adjusts the slow start threshold (*ssthresh*) based on the equilibrium estimation of a connection as opposed to using a fixed drop-factor window (e.g. ½ in Reno and Tahoe, ¾ in Vegas [15]) when packet loss is encountered; 2) It uses a refined linear increase algorithm, which employs both the proactive and reactive congestion detection schemes to adjust the congestion window size during the additive increase phase. Veno's better throughput is attributable to its more efficient use of the available bandwidth rather than "bandwidth-stealing" from other connections. It is compatible and can co-exist harmoniously with the current TCP in different networks. In addition, Veno only needs modification on the sending side of a TCP connection, thus it is easily deployed in real networks.

In details, Veno refines AIMD (additive increase and multiplicative decrease) algorithm of Reno. Its implementation on the sending side is simple, seeing following:

- during AI (linear increase) period,

```
if( DIFF*BaseRTT ≤ a)
  cwnd=cwnd+1/cwnd;          // for every received new ack
  equilibrium_arrival=false; // network equilibrium is not
                                reached
else if(DIFF*BaseRTT ≤ b)    // network equilibrium is
                                reached
     cwnd=cwnd+1/cwnd;       // every new ack received
     equilibrium_arrival=true;
else if(DIFF*BaseRTT > b)    //network is in congestion
  cwnd=cwnd+1/cwnd;          //for every other new ack
                                received
  equilibrium_arrival=true;
```

where *DIFF=cwnd/BaseRTT – cwnd/RTT, BaseRTT* is minimum of the calculated *RTT* and always reset after fast retransmit or timeout occurs, *equilibrium_arrival* is bullion variable, which is recalculated when an ack for the tagged data packet is received. α is set to one and β set to three in terms of buffers. By considering the coarse granularity of TCP clock (500ms) in real networks, in the implementation of TCP Veno, a millisecond resolution timestamp is recorded for each segment it transmits. Upon receiving an ack, Veno retrieves the corresponding segment's timestamp and calculates the segments' *RTT* in millisecond resolution. During the AI phase, Veno's window is increased by *1/cwnd* for **every other** new ack received other than for every new ack received when the equilibrium is reached (*equilibrium_arrival=true*). This subtle *distinguishing point* of the refined linear increase phase relatively extends the AI duration and thus has much positive impact on TCP performance.

- when packet loss is detected by fast retransmit:

```
if(equilibrium_arrival=false)
  ssthresh = $cwnd_{loss}$ * 4/5;   // random loss is most likely
                                      to occur
else
  ssthresh = $cwnd_{loss}$ /2;      // congestive loss is most
                                      likely to occur
```

- when packet loss is detected by timeout:

```
ssthresh is set to half the current window;
slow start is taken over;       //same action as Reno
                                  performs
```

where $cwnd_{loss}$ corresponds to the window size at which point packet loss is detected by fast retransmit or timeout. By the deduction of the *false* value for *equilibrium_arrival,*

[4]Acks which cover new data, but not all the data outstanding when loss was detected.

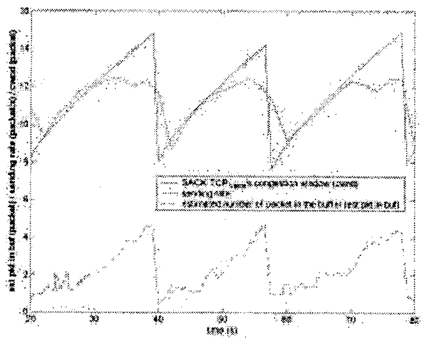
ssthresh is aggressively updated by $cwnd_{loss}$ * 4/5 based on the estimation that TCP connection probably has not achieved the available bandwidth. Otherwise, if packet loss occurs, *ssthresh* is set to $cwnd_{loss}/2$, same as multiplicative decrease of Reno with ½ drop factorsince the *true* value of *equilibrium_arrival* implicitly indicates that most likely the network has been in congestive state. Noted all above calculation are not executed when TCP are evolving in initial slow start stage. Veno keeps the same initial startup behavior as Reno in order no to add any extra load to hosts during the short file processing period.

Generally, TCP Veno is an intelligent congestion avoidance mechanism that adjusts *cwnd* in more aggressive way when network is under-utilized and in more conservative way when in fully utilized stage by its estimation on the connection state. However, TCP Veno has not been studied when combining with acknowledgement packet with SACK option. In fact, there are same problems as Reno in performing data retransmission when multiple packets are lost in a single of window. In the following, we propose to use SACK option to resolve this. We refer to SACK TCP plus TCP Veno as SACK TCP_{veno}.

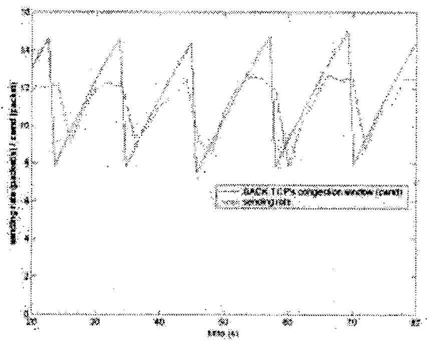
2.3. SACK TCP_{Veno}

SACK TCP_{Veno} combines the retransmission strategy of SACK TCP with the congestion control policy of TCP Veno. The initial slow start is same as Reno' s. During fast retransmit and fast recovery phase or when timeout occurs, SACK TCP_{Veno} takes corresponding actions, similar to SACK TCP' s, as described in Section 2.1. When three duplicate acks are received, SACK TCP_{Veno} reduces *cwnd* to the value of *ssthresh* (=$cwnd_{loss}$*4/5) if *equilibrium_arrival = false*, the purpose here is to try to match the network available bandwidth as much as possible. Otherwise, its *cwnd* is halved, same as the Reno' s multiplicative decrease algorithm when probing the onset of the network congestion (where *equilibrium_arrival = true*). During linear increase phase, SACK TCP_{Veno} take Veno' s additive increase algorithm: Veno' s window is increased by *1/cwnd* for **every other** new ack received other than for every new ack received when *DIFF*BaseRTT* > b . Noted that such conservative refining on linear increase phase will extend additive increase duration of a TCP connection without any sacrificing any utilization of the available bandwidth.

Actually, the main distinction between SACK TCP and SACK TCP_{Veno} is that change from SACK TCP' s AIMD (additive increase and multiplicative decrease) to TCP Veno' s RAIMD (refined AIMD). As seen in Fig.1 with the real experiments in our university, we use host A (Red Hat Linux) in Broadband Lab to dial up to modem pool in computer service center in order to connect host B (NetBSD1.1 with SACK TCP) and host C (NetBSD1.1 with SACK TCP_{veno}) situated in Broadband Lab. The round trip time from host B to host A or from C to A in this network configuration is about 150ms, detected by *traceroute*. The two files with each size of 512kBytes are downloaded from B to A and from C to A in sequence order. We repeat this experiments ten times during different daytimes. The results are same, seeing followings.



(a)



(b)

Figure 1 (a) congestion window, sending rate of SACK TCP_{Veno} and the number of estimated packets at the bottleneck buffer
(b) The congestion window, sending rate of SACK TCP.

Considering the modem speed of 33.6kbps, we set maximum segment size to 512kbytes in order to see window evolution clearly. Figure 1 separately shows congestion window (*cwnd*) evolution of SACK TCP_{Veno} and SACK over wired phone link between 20s and 80s interval. By taking refined additive increase algorithm, we see SACK TCP_{veno} has an extended congestion avoidance phase in contrast to SACK TCP' s. There are about three cycles happened in SACK TCP_{veno} and five in SACK TCP. At same time, the average sending rate of SACK TCP_{Veno} shows more smoothly than that of SACK TCP. For refined MD (multiplicative decrease) algorithm, we only see the window-halved evolution since *equilibrium_arrival* is set *true* (the number of estimated packets in the bottleneck buffer is more than b) when three duplicate acks received at the sender. Generally, SACK TCP_{Veno} maintains its congestion window at a higher average value with smoother fluctuation and has a better utilization of bottleneck bandwidth, particularly when SACK TCP_{veno} is running over lossy networks, as seen in the following Section.

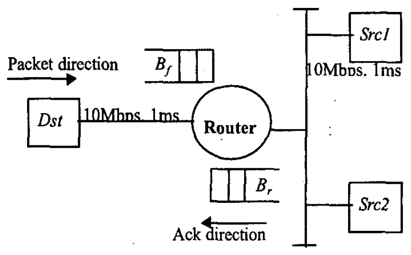
3. Experimental Evaluation

In this section, we describe the two experiment models in Section 3.1. Then in later subsections we present and compare the results for SACK TCP and SACK TCP_{Veno} running over lossy networks by using *dummynet* [11] embedded in FreeBSD 4.2 and *ns* simulator [12] developed at Lawrence Berkeley Laboratory. The throughput evaluation of SACK TCP and SACK TCP_{veno} is studied in Section 3.2. It is known random loss leads to significant throughput deterioration, particularly when the product of the loss probability and square of the bandwidth-delay product is larger than one [2]. Additionally, many users share public resources such as bottleneck' s bandwidth in the Internet, thus it is essential to obtain a better utilization and impartial sharing among the resources. In Section 3.3, we study fairness issue between SACK TCP and SACK TCP_{veno} over networks in hope of illuminating probable future of TCP Veno.

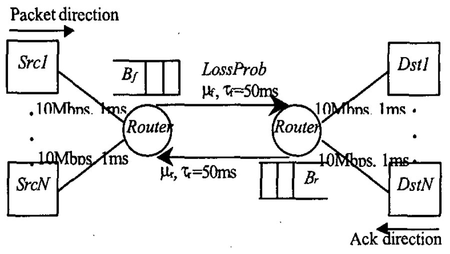
3.1. The Network Scenario

Figure 2(a) shows network for experiments by using *dummynet*. The circles indicate the drop-tail *router* of the network and the squares indicate data sources and destination hosts. *Src1* is NetBSD1.1 with SACK TCP sender, *Src2* is NetBSD1.1 with SACK TCP_{veno} sender and *Dst* is the TCP receiver of Red Flag Linux. The links are labeled with their bandwidth capacity and delay. Router is set up by FreeBSD4.2, which has embedded IPFW [16][19] command to configure the forward buffer size B_f and backward buffer size B_r, bandwidth and propagation delay and packet drop rate.

Figure 2(b) shows the network topology used for the simulations to observe TCP' s *cwnd* and the network utilization. The circles indicate the drop-tail *routers* of the network and the squares indicate data sources and destination hosts. *Src1 ..SrcN* are the TCP senders and *Dst1 ..DstN* are the TCP receivers. The links are labeled with their bandwidth capacity and delay. The parameters are changed when examining different networks. The forward link between the two routers has a capacity of μ_f data packets per second and a propagation delay of τ_f seconds, together with a FIFO buffer of size B_f packets. The reverse link can transmit μ_r acks per second with propagation delay τ_r seconds and a FIFO buffer that can hold B_r acks. To investigate the performance in lossy environment, we assume packets are lost with a probability *LossProb* in the forward link. We use *ns* simulator to test TCP performance in this network model.



(a)



(b)

Figure2 (a)*Dummynet* network configuration and (b) network topology for *ns* simulator

In the following simulations or experiments, we assume the sources have infinite data to send to destination and receiver buffers at the destination are large enough.

3.2. Comparison between SACK TCP_{Veno} and SACK TCP

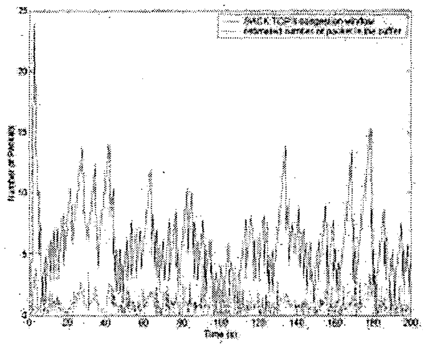
We have tested SACK TCP_{veno} extensively across the public Internet, in the *Dummynet* network and in the *ns* network simulator. These results show SACK TCP_{veno} behaves well across a very wide range of network conditions, Seeing to Section 3.2.1, under less random packet loss conditions, the behavior of SACK TCP_{veno} is generally similar to that of SACK TCP while there is a small improvement; when a SACK TCP_{veno} connection running over network experiences heavy random loss, it shows significant improvement without adversely affecting other TCP connections. Thus, it is compatible and can co-exist harmoniously with the current TCP in different networks, seeing to Section 3.2.2. SACK TCP_{veno}' s better performance is contributable to its more efficient use of the network available bandwidth depending on Veno' s valuable rough estimation of equilibrium point of a TCP connection, not brought about by "stealing" bandwidth aggressively from connections that make use of the current TCP algorithms.

3.3. The Comparison between Single Connection Evolution of SACK TCP and SACK TCP_{veno}

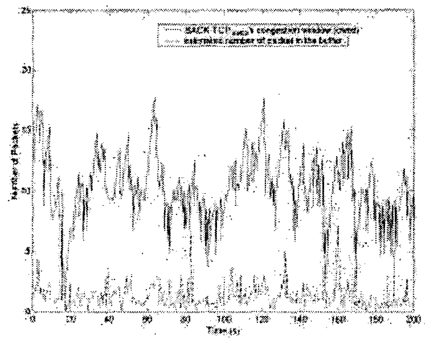
Figure 3 shows the SACK TCP_{Veno}' s and SACK TCP' s congestion window evolution versus time with loss prob. of 1% with the network configuration in Fig. 2(a). The receiver buffer is set to 64kbytes. Packet size is 1460kbytes. *Src* sends bulk data to *Dst* through gateway. We use IPFW command to configure *dummynet* deployed in FreeBSD, seeing to followings:

```
Ipfw flush
Ipfw add pipe 1 ip from Dst to any
Ipfw add pipe 2 ip from any to Dst
Ipfw pipe 1 config bw 800kbit/s queue 8 plr 0.01 delay 50ms
    /*forward bottleneck bandwidth= 800kbit/s, B_f=8 packets, loss rate=1% propagation delay on the pipe 1 is 50ms*/
Ipfw pipe 2 config bw 800kbit/s queue 8 plr 0.01 delay 50ms
    /* reverse bottleneck bandwidth= 800kbit/s, B_r=8 packets, loss rate=1%, propagation delay on the pipe 2 is 50ms */
```

The throughput of SACK TCP$_{Veno}$ (38 pkt/s) is 72% higher than that of SACK (22 pkt/s) at random loss rate of 1%, which indicates that SACK TCP$_{Veno}$ can significantly eliminate performance degradation suffered from the serious random loss.



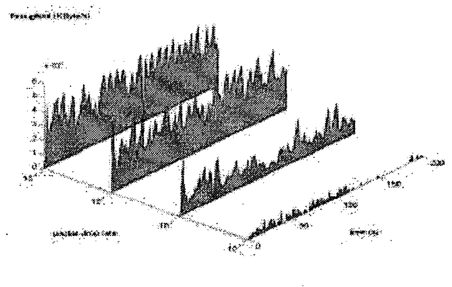
(a)



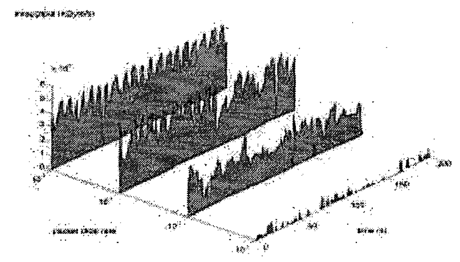
(b)

Figure 3 Congestion window evolution of (a)SACK TCP and (b)SACK TCP$_{Veno}$ over *dummynet* with loss rate 1%

Fig. 4 illustrates the sending rate evolution of a TCP connection running over network model in Figure 2(a) under the different loss probability ranged from 0.01% to 1%, The buffer size was set to 12packets, the link speed to 1.6Mbit/s, and propagation delay to 100ms, maximum segment size is 1460bytes. TCP sending rate fades with increase of loss rate. By comparing Figure 4(a) and 4(b), SACK TCP suffers more serious degradation from random packet loss than SACK TCP$_{veno}$ since TCP Veno can roughly discriminate between congestion loss and random loss to some extent, and leads SACK TCP$_{veno}$ to more carefully adjust congestion window rather than a fixed abrupt reduction by a factor of two when packet loss is detected. However, it is worthwhile to emphasize that under high loss rates, whichever of TCP, SACK TCP or SACK$_{veno}$ is not able to maintain its self-clocking and experience timeout frequently. Seeing to Figure 5(a) and 5(b), the performance of both kinds of TCPs is degraded seriously at loss rate 10^{-1} because both of TCP window are halved induced by frequent timeout actions.



(a)



(b)

Figure 4 the sending rate of (a) SACK TCP and (b) SACK TCP$_{veno}$ in the networks with loss prob. ranged from 10^{-4} to 10^{-1}.

Furthermore, we investigate the utilization under varied loss probability, given the fixed multiple connections of SACK TCP and SACK TCP$_{veno}$, seeing to Fig. 5. In this three-dimension graph, z-axis is the utilization of TCP connections; x-axis is loss probability; y-axis is the number of SACK TCP connections or SACK TCP$_{veno}$ connections running over network model in Fig.2 (b). All the duration of simulation is 200s after TCP connections simultaneously start up.

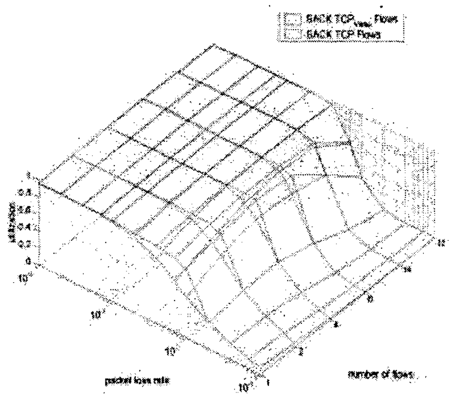


Figure 5 The utilization of SACK TCP and SACK TCP_{veno} under varied loss probability given the fixed multiple connections

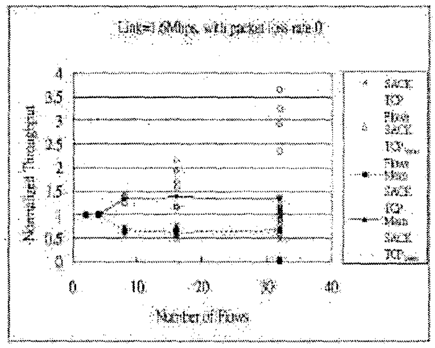
It is observed that the throughput of SACK Veno is always higher than that of SACK under different random loss probability. This graph clearly shows that SACK TCP_{veno} has significant improvement over SACK TCP in heavy loss situation while at low loss rate the throughput of SACK TCP_{veno} is roughly similar to that of SACK TCP. Specifically, SACK TCP_{Veno} is 72% higher than SACK when the random loss is near 1% and SACK TCP_{Veno} is 5% higher than SACK in random loss rate of 0.01% when there is only one TCP connection running in Fig. 2(b). On average, SACK TCP_{Veno} is 44% higher than SACK. When the number of the running connections of SACK TCP or SACK TCP_{veno} increases, the throughput difference is gradually decreased even at the heavy loss range.

The reason is that when there are a lot of concurrent connections competing the same resources (seeing to Figure 5, when there are more than sixteen connections running over networks), network will be heavily congested. Therefore, TCP flows all achieve close to small window (less three packets) per *RTT*, and finally lead timeout action other than fast retransmit to dominate the evolution of TCP connections when encountering packet loss. However, SACK TCP_{veno} or SACK TCP, which is separately derived from TCP Veno and TCP SACK, performs the same severe penalty by halving their window when timeout occurs [1][18].

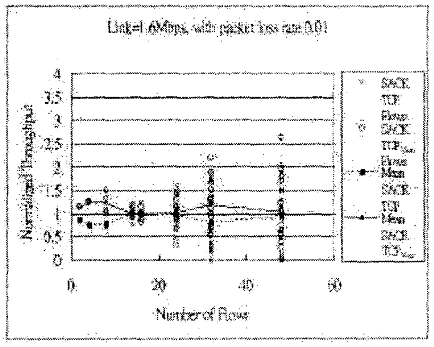
3.4. The Compatibility between SACK TCP and SACK TCP_{veno}

For simulation in this subsection, we used the topology in Figure 2(b) with maximum segment size of 1kbytes, *n* SACK TCP_{veno} and *n* SACK TCP flows share a common bottleneck, where n = 1, 2,4,8,16. We vary the number of flows under the given bottleneck bandwidth 1.6Mbps (Fig. 6) with $B_f=B_r=15$ and 16Mpbs (Fig.7) together with scaled the queue size ($B_f=B_r=150$). Each column represents the results of a single simulation, and each data point is the normalized[5] mean throughput of a single flow. In this case, we take mean throughput of each individual flow over 200 seconds.

Observe that at a low loss rate (16Mbps link, or 1.6Mbps link with less than 6 flows), as seen in Fig. 6(a) and Fig. 7(a), SACK TCP_{veno} receives the fair share. We have seen consistently from all of our experiments that at a high loss rate SACK TCP_{veno} flows receive higher bandwidth than SACK TCP flows. Our explanation is that by employing mechanism of distinguishing between congestion loss and random loss, SACK TCP_{veno} increase more aggressively under higher loss than SACK TCP.



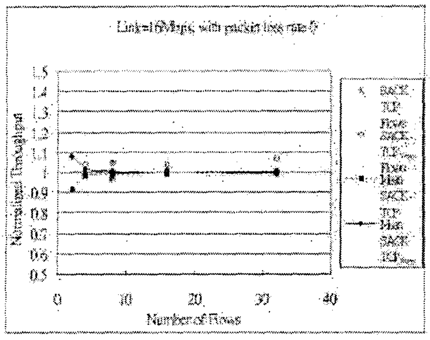
(a)



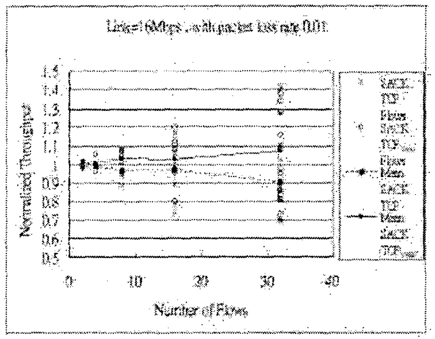
(b)

Figure 6 SACK TCP_{veno} competing with SACK TCP with bottleneck link 1.6Mbps under (a) random loss rate =0 and (b) random loss rate =0.01.

[5] such that a fair share of the link bandwidth is one.



(a)



(b)

Figure 7 SACK TCP$_{veno}$ competing with SACK TCP with bottleneck link 1.6Mbps under (a) random loss rate =0 and (b) random loss rate =0.01.

These figures illustrate that SACK TCP$_{veno}$ and SACK TCP co-exist well. SACK TCP' s throughput is similar to what would be if the competing traffic were SACK TCP instead of SACK TCP$_{veno}$. In other words, SACK TCP$_{veno}$ does not steal bandwidth from SACK TCP when they are competing in same networks. SACK TCP$_{veno}$ is a promising way to improve the performance of TCP.

In order to further demonstrate that SACK TCP$_{veno}$ flows co-exist well when sharing congested bottlenecks with SACK TCP traffic, and perform well over a wide range of network conditions, more experiments are needed. Since there is only space here for summary of our findings, we refer the interested reader to [1] for more detailed results and to the code implemented on NetBSD1.1 and *ns*.

4. Conclusions

In this paper, we have presented SACK Veno algorithm for combining TCP Veno congestion control and SACK TCP retransmission strategy. SACK TCP$_{veno}$ provides performance improvements over SACK TCP in random loss networks, at same time does not cause any adverse impact when they are competing with SACK TCP in low loss networks. All of which are contributable to Veno' s distinguishing on the system equilibrium conditions and its refined AIMD.

Because of the advantage of SACK option, most operating systems[17] start to employ SACK TCP in the implementation of TCP for the Internet transportation. Being different from FACK or RH, which focus on fast retransmit and fast recovery phase of TCP evolution, SACK TCP$_{veno}$ focuses on refining additive increase and mulitiplicative decrease algorithm embedded in Reno. We believe that an effective congestion control algorithm, cooperating with efficient recovery strategy such as SACK option is able to achieve optimal performance.

5. Acknowledgements

This paper was written in support of [1], the proposal for improving the congestion control policy of TCP. We would like to thank Mr. Lui Hung Ngai, Dr. Chan Man Chi and Mr. Yip Chung Sun for their kind support.

References

[1] Fu ChengPeng, Chung Ling-Chi, Liew Soung-chang "TCP Veno: Equilibrium-oriented End-to-end Congestion Control over Heterogeneous Networks", http://www.broadband.ie.cuhk.edu.hk

[2] Dong Lin and Robert Morris, "Dynamics of Random Early Detection", SIGCOMM 97, September 1997. http://www.acm.org/sigcomm/sigcomm97/program.html

[3] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow. "TCP Selective Acknowledgement Options", Request for Comments (Standard Track) RFC 2018, Internet Engineering Task Force, October 1996.

[4] Kevin Fall and Sally Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", Computer Communication Review, July 1996.

[5] S. Floyd, T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm", Request for Comments (Experimental) RFC 2582, April 1999.

[6] Lin, D., Kung, H.T, "TCP fast recovery strategies: analysis and improvements", Proceedings of INFOCOM '98, 29 March-2 April 1998.

[7] Miten N. Mehta and Nitin H. Vaidya Delayed Duplicate-Acknowledgments: A Proposal to Improve Performance of TCP on Wireless Links. Technical Report 98-006, Department of Computer Science, Texas A&M University, February 1998.

[8] Matthew Mathis and Jamshid Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control", Proceedings of SIGCOMM' 96, August 1996.

[9] M. Mathis, J. Mahdavi, "TCP Rate-Halving with Bounding Parameters" Available from http://www.psc.edu/networking/papers/FACKnotes/current.

[10] J. Hoe, "Startup Dynamics of TCP's Congestion Control and Avoidance Schemes", Master's Thesis, MIT, 1995. // J. Hoe, "Improving the Startup Behavior of a Congestion Control Scheme for TCP", In ACM SIGCOMM, August 1996.

[11] L.Rizzo, "Dummynet and Forward Error Correstion", In Proc. Freenix 98, 1998.

[12] S. McCanne, S. Floyd, "ns-LBNL Network Simulator", Obtain via: http://www-nrg.ee.lbl.gov/ns/.

[13] V. Jacobson, R. Braden, "TCP Extensions for Long-Delay Paths", RFC 1072, October 1988.

[14] Allman, M., Paxson, V. and W. Richard Stevens, "TCP Congestion Control", Request for Comments (Standard Track) RFC 2581, April 1999.

[15]Lawence S. Brakmo, Sean W.O' Malley, and Larry L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," Proceedings of ACM SIGCOMM' 94, no.4 pp.24-35, October 1994

[16] FreeBSD handbook

http://www.freebsd.org/handbook/firewalls.html

[17] Thomas Lee, Joseph Davies, "Microsoft Windows 2000 TCP/IP Protocols and Services Technical Reference", Microsoft Press, 1999.

[18] Jacobson, V., "Congestion Avoidance and Control", Computer Communication Review, vol. 18, no. 4, pp. 314-329, Aug. 1988.

[19] http://www.iet.unipi.it/~luigi/ip_dummynet/