Dynamics Comparison of TCP Veno and Reno

C. L. Zhang, C. P. Fu, Ma-Tit Yap, C. H. Foh, K. K. Wong, C. T. Lau, M. K. Lai
School of Computer Engineering
Nanyang Technological University, Singapore

{P146199053, ascpfu, astyap, aschfoh, askkwong, asctlau, asmklai}@ntu.edu.sg

Abstract— **TCP Veno was recently proposed to eliminate TCP performance suffering from wireless links. Real network measurements and live Internet results have validated Veno's significant throughput improvement in wireless networks and its harmonious co-existence with TCP Reno connections in wired networks. In this paper, we demonstrate the out-of-phase synchronization of Veno in one-way traffic, as opposed to the in-phase synchronization of Reno. The detailed studies of these behaviors and its interaction with Reno are reported. Moreover, our careful study shows that this out-of-phase synchronization benefits network link utilization, and reduces the occurrence of congestion loss.**

Keywords-TCP Veno; TCP Reno; out-of-phase; in-phase; synchronization

I. INTRODUCTION

TCP is a reliable connection-oriented protocol that implements flow control by means of a sliding window algorithm [3]. TCP Reno, which makes use of slow start and congestion avoidance algorithms to adjust the window size, is widely deployed in the Internet. During the slow start phase, its window is incremented for each *ack* received until packet loss is experienced, at which point the window is halved and then a linear increase algorithm takes over until further packet loss is experienced. This additive increase and multiplicative decrease mechanism leads to periodic oscillations in the congestion window, round trip delay and queue length of the bottleneck buffer in the path.

However, the assumption in TCP Reno that packet loss implies network congestion may not apply to wireless networks, in which packet loss may be induced by noise, link error or reasons other than network congestion. Not making an attempt to distinguish between random and congestion losses, TCP Reno is equally sensitive to both of them. This may lead to significant but unnecessary end-to-end throughput degradation.

Recently, a new TCP variant called TCP Veno [4, 6, 7] was proposed to eliminate the severe suffering from wireless links. It integrates the advantages of two opposing camps - TCP Reno and Vegas [8]. A distinguishing feature of TCP Veno is that its significant improvement over Reno performance is achieved from better unitization of the available bandwidth that is left unused by other existing connections, rather than by aggressively grabbing extra bandwidth from other connections. Besides, Veno only requires modification of the sender-side protocol stack, making it easier to deploy over the current Internet.

Earlier in 1990, Shenker et al. [1-2] have studied the in-phase synchronization phenomenon of Reno in one-way traffic, and observed that the co-existing connections drop packets almost simultaneously when they reach the path capacity. In this paper, we study the dynamics behavior of Veno's congestion control algorithm and its detailed interactions with TCP Reno connections. The experimental results demonstrate one interesting characteristics of Veno – out-of-phase synchronization, which is quite different from Reno's window dynamics observed before. Moreover, our extensive study on co-existence under different situations further verified that Veno indeed works harmoniously with its competing Reno connections over wired networks. Its slight improvement is achieved by this out-of-phase synchronization.

The remainder of this paper is organized as follows. In Section II, we brief TCP Veno congestion control algorithms. Our experimental topology is described in Section III, and the experimental results are presented and detailed analyses are described in Section IV. Some conclusions and future work are given in Section V.

II. THE MECHANISM OF TCP VENO

TCP Veno makes use of the idea of congestion monitoring scheme from TCP Vegas, and integrates it into window evolution scheme of Reno. In Vegas TCP, This monitoring is calculated by the difference between the measured and the expected throughput, namely,

$$DIFF = (Expected - Actual) \qquad (1)$$

with *Expected = cwnd/BaseRTT* and *BaseRTT* is the minimum of all measured RTT (round trip times)[1]. *Actual* is the measured throughput at the sender given by *cwnd/RTT*, where RTT is the actual round-trip time of a tagged packet. Strictly

The author C. P. Fu is adjunct Professor at COIWIN, Chong Qing University of Posts and Telecommunications, P. R. China.

[1]In Vegas, *BaseRTT* is continually updated throughout the live time of the TCP connection with the minimum round-trip time collected so far. In Veno TCP, however, *BaseRTT* is reset whenever packet loss is detected, either due to time-out or duplicate ACKs. *BaseRTT* is then updated as in the original Vegas algorithm until the next fast-recovery or slow-start is triggered. This is done to take into account of the changing traffic from other connections and that the bandwidth acquired by a single connection among the many connections may change from time to time, causing the *BaseRTT* to change also.

speaking, *Expected* as defined is the best possible throughput, since *BaseRTT* is the minimum of all measured RTT.

In Veno, *DIFF*BaseRTT* is used to estimate the number of packets accumulated at the bottleneck buffer. If there is more than an upper threshold (β) of packets queuing for processing, the TCP connection is said to have evolved into a congestive state. Otherwise, it is in the non-congestive state. As in TCP Reno, packet loss in the congestive state (*congestive drop*) will cause the window to be halved. However, packet loss in the non-congestive state (*non-congestive drop*) will only cause the window size to be decreased by a factor of 1/5.

Moreover, Veno refines the additive increase phase of Reno by forcing the TCP connection to stay longer at the operating region. The whole algorithms are described in Figs. 1-2.

```
When packet loss is detected by fast retransmit:
  if (DIFF*BaseRTT < β)      //most likely it is a random loss
    ssthresh = cwnd_loss * (4/5)
  else                       //most likely it is a congestion loss
    ssthresh = cwnd_loss / 2

When packet loss is detected by retransmit timeout timer:
    ssthresh is set to half the current window ;
    slow start is performed; //performs the same action as in Reno
```

Figure 1. TCP-Veno-ssthresh adjusting algorithm

```
During the additive increase period:
  if (DIFF*BaseRTT < β)   //available bandwidth is underutilized
    cwnd=cwnd+1/cwnd when every new ack is received
  else                    //available bandwidth is fully utilized
    cwnd=cwnd+1/cwnd when every other new ack is received
```

Figure 2. TCP-Veno-refined additive increase algorithm

Similar to [1-2], we define an *epoch* of a TCP connection to be the time period during which an entire window's worth of packets have been acknowledged. We will focus on those special epochs in which packet loss occurs, and refer them as congestion epochs. Therefore, Veno's cwnd increases one packet by each epoch when *DIFF*BaseRTT* $< \beta$, or increase one packet only by two epochs when *DIFF*BaseRTT* $\geq \beta$. In contrast, Reno always increase one packet for each epoch. The general incrementing rate of Veno's congestion window is slower than that of Reno.

III. EXPERIMENT NETWORK

In this paper, we use network simulator (ns-2.26) [11] from Lawrence Berkley National Laboratory to study TCP Veno. The experimental network (see Fig. 3) consists of two source and destination pairs. The pair connections share the same bottleneck link using a *droptail* queue. The speed of each link connecting nodes and according routers is 10Mbps with 1ms propagation delay. The bottleneck link connecting Routers A and B have μ packets/s and τ seconds of propagation delay, the queue size along this link is *B* packets for both forward and reverse directions. Typical data packet size is 1kBtyes and typical ack size is 40 Bytes.

As in [2], we define the capacity of the path to be the maximum number of packets (data packets and acknowledgement packets) outstanding along this path, these outstanding packets - data or acks - could be spread along the transmission line, or queued at the buffer of the bottleneck router. Referring to Fig. 3, the pipe size (*C*) between one *Source and* one *Destination* is equal to $B + 2P$, where $P = \mu \cdot (\tau + 1\text{ms} + 1\text{ms}) \cong \mu \cdot \tau$.

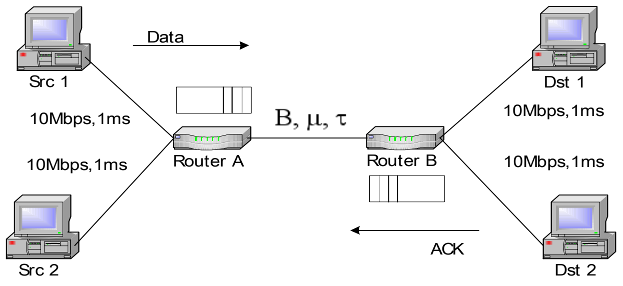


Figure 3. Experiment network topology

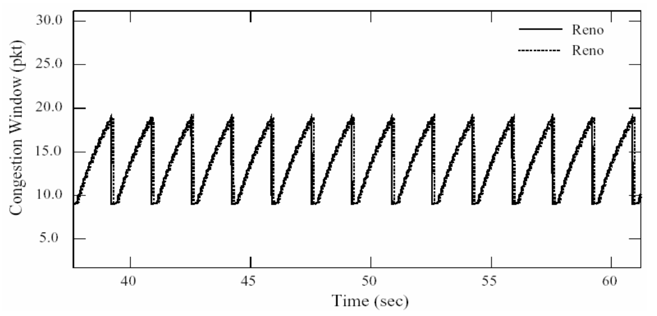
Assuming the source has infinite data to send, and the receiver advertised window has been set large enough, thus, we interchangeably use *wnd* (window) and *cwnd* (congestion window) in the following part.

IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

This section discusses some interesting results on the dynamic behavior of TCP Veno and its co-existence with legacy TCP Reno with highlights of the out-of-phase synchronization paces observed in Veno TCP. We further point out the out-of-phase synchronization benefits Veno's behavior.

A. Dynamics of Veno TCP and Its Interaction with Reno TCP

Two TCP Reno or Veno connections are conducted between the pairs of sources and destinations in Fig. 1 with the configuration of B=15, μ=1.6Mbps, and τ=50ms. As discussed in [1-2], two Reno have in-phase synchronization paces, namely, when one connection drops its window by half, another connection will follow same window-penalty immediately. Fig. 4(a) shows the case of two TCP Reno connections. The phenomenon of the in-phase synchronization is mainly because when the shared link is fully utilized, the two TCP Reno senders continue to increase *cwnd* causing overflow and a packet from each connection will be dropped during this congestion epoch. After then, their window sizes will be halved for both the connections.



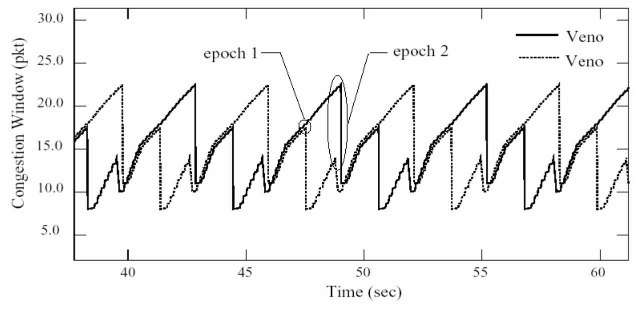


Figure 4. (a) Window evolution of two Reno connections. (b) Window evolution of two Veno connections.

Veno refines the Multiplicative Decrease in Reno to improve the performance in wireless environments. This refinement also produces an interesting phenomenon – the out-of-phase synchronization. It can be understood as follows. In Fig. 4(b), the two Veno connections, when staying in the congestive state, takes more conservative window increment than Reno, namely, each *cwnd* is increased by *one* packet for every two round trip time instead of one round trip time. At "epoch 1", when the link has been fully utilized, one connection increases its window by one packet while another does not. This only causes one packet to be dropped at the router and hence only one connection suffers window-halving penalty. The window-halving penalty has then released much of the bandwidth for another connection to continue in grabbing more bandwidth by conservative window increment.

However, when both Veno connections increase their window sizes during the same round trip time when the link is fully utilized, they will experience packet drops and both connections will reduce their window sizes due to the network congestion. This is depicted in Fig. 4(b) at "epoch 2". It is also notable at "epoch 2" that the connection with a lower window size reduces less than that of the connection with a higher window size. The reason is that the connection with low window size is evolving in non-congestive state (with one packet increasing by one round trip time), while another remains in its congestive state[2]. Noted here that while two connections are competing network resource, the connection with more resource occupying, will suffer severe penalty, and the connection with less resource occupying, will have less severe window reduction. Thus, less aggressive connection will have a chance to grab more resource when loss occurs, meanwhile, more aggressive connection will give up some resource occupied.

[2] In Veno, window only performs half reduction in congestive state when packet loss is detected by three duplicated ACKs, otherwise, it will only take 1/5 window reduction. This packet loss here, while induced by buffer overflowing, can be regarded as transient congestion loss, as discussed in [13].

More interestingly, both connections *interchange* their roles of being a more and a less aggressor in terms of bandwidth usage as *cwnd* evolves, which is unique in TCP Veno. This mechanism will definitely bring about fair competing for limited resource over the Internet. Seeing the congestion epoch 1 and 2, this *intertwined* pattern repeats itself indefinitely, the figure contains only one or a few of these cycle periods to allow the reader to see the details of window evolutions.

The case of one Reno and one Veno sharing a link as shown in Fig. 3 is also studied. The window evolution of both connections is reported in Fig. 5, with evidence of the out-of-phase synchronization phenomenon of Veno. The events at "epoch 3" and "epoch 4" in Fig. 5 are corresponding to that of "epoch 1" and "epoch 2" in Fig. 4(b). In this case, Reno connection is the one always being the more aggressive connection in bandwidth sharing.

The sharing of Reno and Veno also arises an interesting result that shows fairness in bandwidth sharing between Reno and Veno. Usually, a less aggressor in bandwidth sharing, such as TCP Vegas, possesses less bandwidth over a long run when sharing a link with the more aggressive Reno. However, Veno, albeit being a less aggressor, is able to share bandwidth fairly with Reno. Seeing Fig. 5, the solid line is Reno's evolution, dotted line is Veno's connection. Let us look at two typical time that packet loss is occurring - "epoch 3" and "epoch 4".

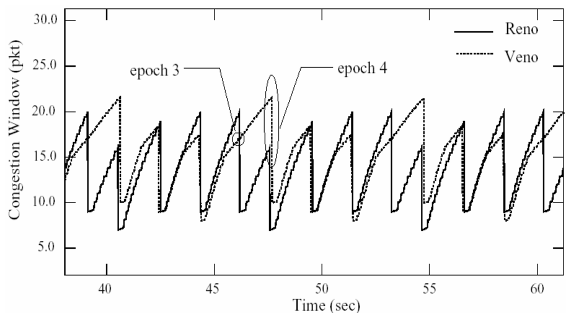
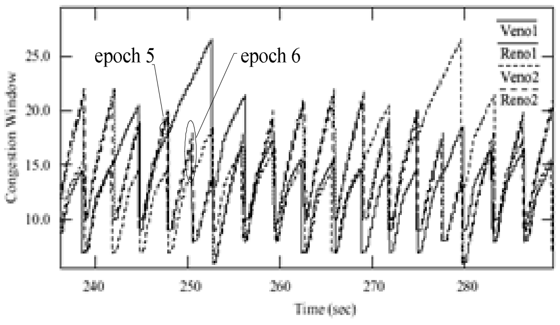


Figure 5. Window evolution of Reno and Veno.

At "epoch 3", Reno connection increase its window by one packet while Veno's connection is not (Veno, after entering the congestive state, does not always increase its *cwnd* after a round trip time, its window is increased by one packet for *every other* round trip time in congestive state). On account of full buffer (already occupied by these two connections) of the bottleneck router, one newly extra packet in Reno's window will lead to a packet loss, while Veno, due to the unchanged of its window, avoid packet loss. Obviously, at this time, Reno's *aggressive* window policy brings about a window-cut (immediacy halved by the detection of this packet loss) while Veno is allowed to grow and receive more bandwidth resource until "epoch 4", where both Reno and Veno suffer a packet loss due to their simultaneous window increase at this time. Thus, both windows are cut into halves. Nonetheless, in the following window evolution phase (from 47.5s to 53s), Reno will gradually exceed Veno's window and grab more bandwidth resource because of its a little bit aggressiveness. At time 53, it will repeat the case of "epoch 3". This *intertwined* pattern repeats its self indefinitely. Thus, over a long run, fairness between Veno and Reno is able to be maintained. Noted that in these two competing connections, Veno has suffered less packet congestion loss (around 9 times in Fig. 5), while Reno has more (around 13 times). This reduced congestion number definitely increases the bandwidth utilization somewhat.

A more interesting scenario of multiple TCP connections with two Veno and two Reno connections' dynamics window evolutions are shown in Fig. 6(a). With experiment settings of B=44, μ=1.8Mbps and τ=50ms with network topology described in Fig. 3



s

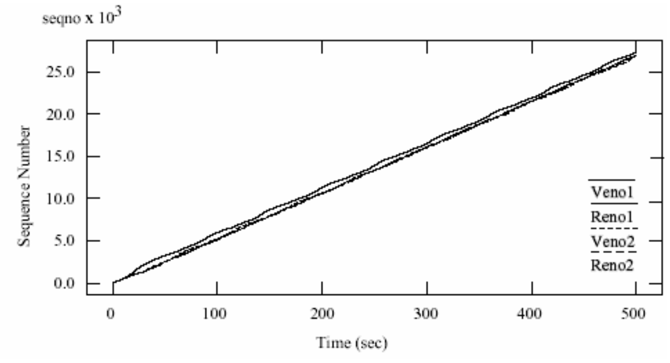


Figure 6. a) Window evoution of two Reno and Veno connections b) Seqnuence number of four connections

The events on "epoch 5" and "epoch 6" are similar to that of "epoch 4" in Fig. 5. At "epoch 5" only Veno 1 does not incur loss, at "epoch 6" Veno 1 and Veno 2 connections do not incur loss. Following a series of synchronization congestions occurrence (all Reno and Veno connections drop one packet exactly at each epoch), the Reno connections show more aggressiveness in these following cycles. After that, the *intertwined* pattern comes up with the Veno 2 connection. We notice that the window evolution becomes more complicated while there are multiple connections, especially when background traffic are introduced, but the out-of-phase phenomenon still exist somehow. In Fig. 6, gives the evolution of sequence numbers vs time and the diagram shows that all four connections share the bottleneck link's bandwidth equally without bias.

B. Fairness of TCP Reno and Veno

Paper [4] has studied Veno TCP can achieve high throughput improvement over Reno in wireless networks. In this section, we investigate the dynamics fairness issues of TCP Reno and Veno over wired networks.

As we observed, two Reno connections always have synchronized in-phase paces. While a Veno connection co-exists with a Reno connection, there are some occasions that Veno does not incur packet loss in specific epoch. To some extent, this behavior will avoid some congestion loss, and may improve the utilization of network link. We design our experiments as follows: either the Reno or the Veno connection starts first to transmit a 20M byte files, the other connection – Reno or Veno – joins in at 20s and begins to transfer a 8M byte files. The setting of the network topology is: transmission rate 2Mbps, a round-trip delay 100ms and the bottleneck link buffer size 15. We plot the congestion window as well as the sequence number against time in Fig. 7, 8 and 9. From the three different scenarios we observe that later coming connection would not be biased by previous running connection, they will share the bandwidth evenly, this can also be seen in each diagram of sequence number *vs* time, where the slope of two connections are almost same.

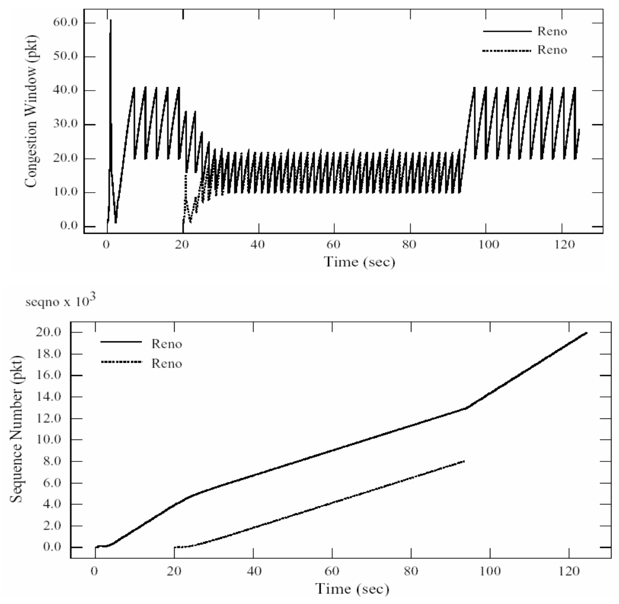
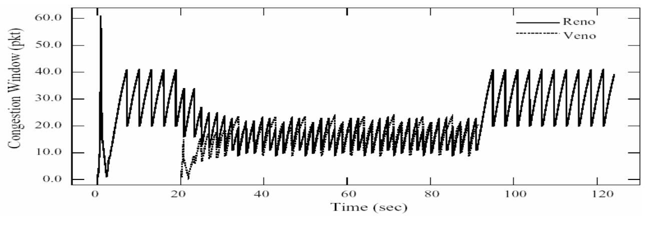


Figure 7. Reno connection is started first, another Reno joins in at 20s



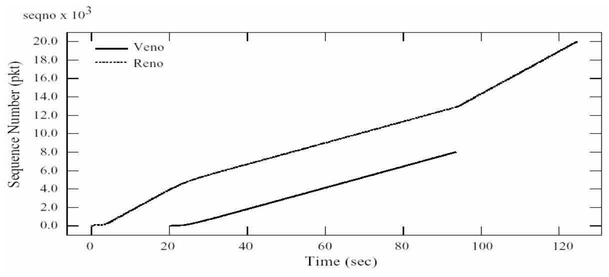


Figure 8. Veno connection is started first, Reno joins in at 20s

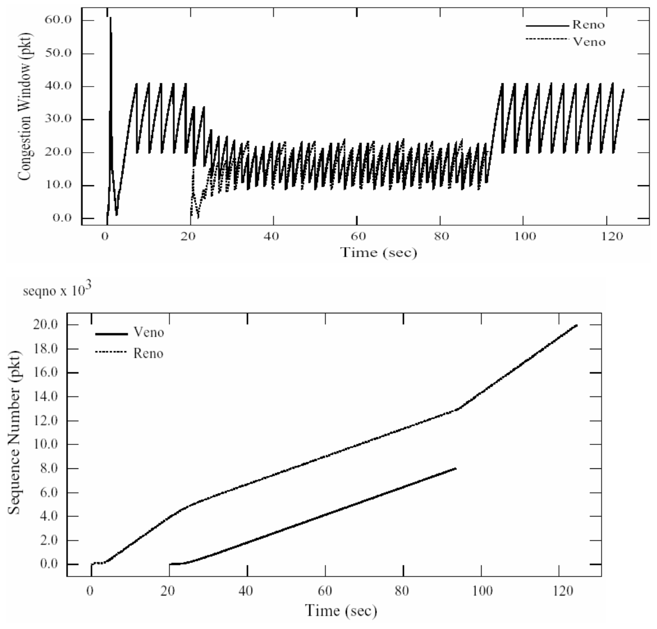


Figure 9. Reno connection is started first, Veno joins in at 20s

In the following Table 1, we calculate the throughput using the file size plus the total retransmitted bytes divided by time in seconds. From the table, we got three results, 1) as for longer transmission over wireline, Veno gests slightly higher throughput regardless of whatever combinations is employed; 2) as for short transmission, Veno also got slightly higher throughput; 3) referring to row 3 of this table, we know this slight improvement is brought about by less retransmitted packets, in other words, congestion losses in Veno are reduced as compared to Reno. These results conform to our above analysis in Section IV.A.

TABLE I. PERFORMANCE EVALUATION OF RENO VS VENO, FIRST CONNECTIONS IS WITH 20M BYTES SENT, SECOND WITH 8M BYTES

	Reno/Reno	Reno/Veno	Veno/Reno
Time Transmitted	124.66/73.52	124.28/71.47	123.89/73.73
Retransmitted packets	105/56	103/45	90/54
Throughput (KByte/s)	161.27/109.51	161.76/112.56	162.4/109.85

V. CONCLUSION

TCP Veno [4, 6, 7] was recently proposed to eliminate TCP performance suffering from wireless links. Real network measurements and live Internet results have validated Veno's significant throughput improvement in wireless networks and its harmonious co-existence with TCP Reno connections in wired networks.

In this paper, we studied the dynamics of TCP Veno and its detailed interaction with TCP Reno over wired networks. We observed out-of-phase synchronization phenomenon appearing in Veno's window evolution, this desired nature leads to Veno's better utilization of wired link. Our further study verifies that either Veno or Reno would not bring about any bias for any connection initiated later, and they can fairly share bottleneck link bandwidth in wired networks. From the practical viewpoint, this advantage will be definitely helpful to deploy sending-side based Veno TCP in current Internet with hybrid wired and wireless links.

REFERENCES

[1] Scott Shenker, Lixia Zhang, David D.Clark, " Some Observations on the Dynamics of a Congestion Control Algorithm" *ACM SIGCOMM Computer Communication Review,* Volume 20 , Issue 5 *Oct. 1990*

[2] Lixia Zhang, Scott Shenker, David D.Clark, " Observations on the Dynamics of a Congestion Control Alogrithm: The Effects of Two-Way Traffic". *Proceedings of the conference on Communications architecture & protocols,* Zurich, Switzerland Pages: 133 - 147, 1991

[3] Cheng Peng Fu, and Soung Chang Liew, "A remedy for performance degradation of TCP Vegas in asymmetric networks," *IEEE Communications Letters*, January 2003.

[4] Cheng Peng Fu, and Soung Chang Liew, "TCP Veno: TCP enhancement for wireless access networks," *IEEE Journal of Selected Areas in Communications*, vol. 21, no. 2, February 2003.

[5] Cheng Peng Fu, and Soung Chang Liew, "A remedy for performance degrateion of TCP Vegas in asymmetric networks," IEEE Comm. Letters, January 2003

[6] Q. X. Pang, S. C. Liew, C. P. Fu, W. Wang, "Performance Study of TCP Veno over wireless LAN and RED router," *IEEE Globecom 2003.*

[7] C. P. Fu, W. Lu, B. S. Lee, "TCP Veno Revisited," *IEEE Globecom 2003.*

[8] L.Brakmo, S.O'Malley, and L.Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In Proceedings of the SIGCOMM' 94 Symposium (Aug. 1994) pages 24-35.

[9] T.V Lakshman, Upamanyu Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss," IEEE/ACM Trans. Networking, Vol5, No. 3, June 1997

[10] J. Mo, R. J. La, V. Anantharam, and J. Walrand. Analysis and Compariation of TCP Reno and TCP Vegas. *IEEE INFOCOM 1999*, March 1999.

[11] Ns. Network Simulator, version 2.26. http://www.isi.edu/nsnam/ns.

[12] A. Veres and M. Boda, "The Chaotic Nature of TCP Congestion Control," *IEEE INFOCOM 2000*, March 2000.

[13] Dhiman Barman, Ibrahim Matta, "Effectiveness of Loss Labeling in Improving TCP Performance in Wired/Wireless Networks," *IEEE International Conference on Network Protocols*, Paris, France, November 12-15, 2002.