TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks

Cheng Peng Fu, *Associate Member, IEEE,* and Soung C. Liew, *Senior Member, IEEE*

***Abstract*—Wireless access networks in the form of wireless local area networks, home networks, and cellular networks are becoming an integral part of the Internet. Unlike wired networks, random packet loss due to bit errors is not negligible in wireless networks, and this causes significant performance degradation of transmission control protocol (TCP). We propose and study a novel end-to-end congestion control mechanism called TCP Veno that is simple and effective for dealing with random packet loss. A key ingredient of Veno is that it monitors the network congestion level and uses that information to decide whether packet losses are likely to be due to congestion or random bit errors. Specifically: 1) it refines the multiplicative decrease algorithm of TCP Reno—the most widely deployed TCP version in practice—by adjusting the slow-start threshold according to the perceived network congestion level rather than a fixed drop factor and 2) it refines the linear increase algorithm so that the connection can stay longer in an operating region in which the network bandwidth is fully utilized. Based on extensive network testbed experiments and live Internet measurements, we show that Veno can achieve significant throughput improvements *without* adversely affecting other concurrent TCP connections, including other concurrent Reno connections. In typical wireless access networks with 1% random packet loss rate, throughput improvement of up to 80% can be demonstrated. A salient feature of Veno is that it modifies only the sender-side protocol of Reno without changing the receiver-side protocol stack.**

***Index Terms*—Congestion control, congestion loss, random loss, transmission control protocol (TCP) Reno, TCP Vegas, TCP Veno, wireless access networks.**

I. Introduction

Wireless communication technology has been making significant progress in the recent past and will be playing a more and more important role in access networks, as evidenced by the widespread adoption of wireless local area networks (WLANs), wireless home networks, and cellular networks. These wireless access networks are usually interconnected using wired backbone networks, and many applications on the networks run on top of the transmission control protocol/Internet protocol (TCP/IP).

Manuscript received May 1, 2002, revised September 30, 2002. This work was supported in part by the Area of Excellence in Information Technology of Hong Kong (AoE/E-01/99), and in part by the Hong Kong Research Grant Council under the Competitive Earmarked Grant CUHK 4229/00E. The work of C. P. Fu was done while with the Information Engineering Department, The Chinese University of Hong Kong, Shatin, Hong Kong.

C. P. Fu was with The Chinese University of Hong Kong Shatin, Hong Kong. He is now with Nanyang Technological University, Singapore (e-mail: Franklin.Fu@ieee.org).

S. C. Liew is with The Chinese University of Hong Kong, Shatin, Hong Kong (e-mail: soung@ie.cuhk.edu.hk).

Digital Object Identifier 10.1109/JSAC.2002.807336

TCP is a reliable connection-oriented protocol that implements flow control by means of a sliding window algorithm. TCP Tahoe and Reno [22], [24], which make use of the slow start (SS) and congestion avoidance (CA) algorithms to adjust the window size, have enjoyed much success to date. In particular, Reno is currently the most widely deployed TCP stack, enabling all sorts of Internet applications.

The Reno algorithm, however, is increasingly being stretched by the emergence of heterogeneity on the Internet in which different segments of the network may have widely different characteristics. Optical-fiber links are highly reliable while wireless links are subject to high-bit-error rates; asymmetric digital subscriber line (ADSL) access lines are asymmetric and have different capacities in the two directions; satellite networks can have much higher propagation delay than terrestrial networks; and wireless access networks may suffer high packet loss rates.

Reno treats the occurrence of packet loss as a manifestation of network congestion. This assumption may not apply to networks with wireless channels, in which packet loss is often induced by noise, link error, or reasons other than network congestion. As defined in [26], we refer to such packet loss as random packet loss. Misinterpretation of random loss as an indication of network congestion causes Reno to back down the sending data rate unnecessarily, resulting in significant performance degradation [11], [12], [26].

To tackle this problem, we can break it down to two parts: 1) How to distinguish between random loss and congestion loss and 2) How to make use of that information to refine the congestion-window adjustment process in Reno.

In 1994, TCP Vegas [10], which employs proactive congestion detection, was proposed with the claim of being able to achieve throughput improvement ranging from 37% to 71% compared with Reno. Reference [4] reproduced the claims made in [10] and showed that by reducing packet loss and subsequent retransmissions, Vegas could indeed offer higher throughput than Reno. Recent work [20], [28], [30], however, showed that the performance of Vegas connections degrades significantly when they coexist with other concurrent Reno connections. The bias against Vegas is particularly severe in asymmetric networks [3].

Despite its limitations, an innovative idea in Vegas is that it uses a very simple mechanism to gauge the network condition. Instead of using Vegas' estimate on the network condition to prevent packet loss proactively, we could use it to determine whether packet losses are likely to be due to network congestion or random phenomena. Specifically, if a packet loss is detected while the estimate indicates that the network is not congested, we could declare that the loss is random.

Reno could be modified so that it reduces the sending rate *less* aggressively when random loss is detected, thus preventing unnecessary throughput degradation. We refer to the algorithm that combines these ideas from *Vegas* and *Reno* as *Veno*.[1]

Over the past few decades, TCP has been extensively studied [5], [7], [8], [11], [12], [16]–[18], [32], [35]. Numerous proposals [9], [10], [12]–[17], [23], [25], [31], [33], [37], [38] were presented to improve TCP performance. However, if practical deployment issues and overall evaluation over the heterogeneous networks were to be considered [6], most proposals do not compare well with Reno. Reno remains the dominant version used in practice. This paper is an attempt to demonstrate that Veno is a viable enhancement to Reno.

Based on extensive network testbed experiments and live Internet measurements, we show that Veno can achieve significant throughput improvements *without* adversely affecting other concurrent TCP connections in the same network, including concurrent connections that make use of the legacy Reno implementation. We present evidence, showing that Veno derives its improvement from its efficiency in tapping unused bandwidth, *not* from hogging network bandwidth at the expense of other TCP connections. In wireless networks with random loss of 1%, throughput improvement of up to 80% can be demonstrated. Veno involves only modification of Reno on the sender side without requiring any changes of the receiver protocol stack or intervention of the intermediate network nodes. It can therefore be deployed immediately in server applications over the current Internet, coexisting compatibly with the large installed base of Reno.

The rest of this paper is organized as follows. Section II describes the Veno's mechanism. It presents the observations that led us to the refined multiplicative decrease and additive increase algorithms. In Section III experimental results are firstly presented to show the ineffectiveness of the refined algorithms, and then a comprehensive evaluation of the Veno's performance over real network and live Internet are demonstrated. Section IV contains the conclusions and suggestions for future work.

II. TCP Veno

Distinguishing between congestion loss and random loss, and providing different measures to deal with them, is a fundamental problem that remains unsolved for TCP. Veno makes use of a mechanism similar to that in Vegas to estimate the state of the connection, nonetheless, such a scheme is used to deduce what kind of packet loss–congestion loss or random loss—is most likely to have occurred, rather than to pursue preventing packet loss as in Vega. If packet loss is detected while the connection is in the congestive state, Veno assumes the loss is due to congestion; otherwise, it assumes the loss is random.

Veno makes use of a mechanism similar to that in Vegas to estimate the state of the connection, nonetheless, such a scheme is used to deduce what kind of packet loss ÿ congestion loss or random loss -- is most likely to have occurred, rather than to pursue preventing packet loss as in Vegas.

[1]TCP Veno has been implemented in NetBSD1.1, FreeBSD4.2 and Linux 2.4. Its enhanced version with selective acknowledgement (SACK) option has also been implemented, its source codes are available by email request, for more detailed work, please see the reference [1] and [41] documented in July 2001, we also noticed that at same time another modified TCP called Westwood [40]proposed an alternative approach to deal with TCP suffering in wireless networks.

A. Distinguishing Between Congestive and Noncongestive States

In Vegas [10], the sender measures the so-called *Expected* and *Actual* rates

$$Expected = cwnd/BaseRTT$$
$$Actual = cwnd/RTT$$

where *cwnd* is the current TCP window size, *BaseRTT* is the minimum of measured round-trip times, and *RTT* is the smoothed round-trip time measured. The difference of the rate is

$$Diff = Expected - Actual.$$

When $RTT > BaseRTT$, there is a bottleneck link where the packets of the connection accumulate. Let the backlog at the queue be denoted by N. We have

$$RTT = BaseRTT + N/Actual.$$

That is, we attribute the extra delay to the bottleneck link in the second term of the right side above. Rearranging, we have

$$N = Actual^* \, (RTT - BaseRTT) = Diff^* \, BaseRTT.$$

Vegas attempts to keep N to a small range by adjusting the TCP window size proactively, thus avoiding packet loss due to buffer overflow altogether. Unfortunately, this proactive window adjustment also puts Vegas at a disadvantage when there are coexisting Reno connections in its path [4], [20], [30] since it is less aggressive than Reno's policy, which continues to increase window size until packet loss occurs.

A second problem with Vegas is that the backlog it measures is not necessary the data backlog [3]. In asymmetric networks where the bottleneck link is on the reverse path, the backlog N as indicated by the above equation may be due to the backlog of TCP acknowledgment on the bottleneck link. Because of this backlog, Vegas would not continue to increase the window size to allow a higher data-sending rate on the forward path even though its forward path is still congestion free. Reno, in contrast, probes for additional forward bandwidth by increasing window size continually.

The main idea of our algorithm, Veno, is that we will use the measurement of N not as a way to adjust the window size proactively, but rather as an indication of whether the connection is in a congestive state. The essential idea of Reno, in which the window size is increased progressively when there is no packet loss, remains intact in Veno.

Specifically, if $N < \beta$ when a packet loss is detected, Veno will assume the loss to be random rather than congestive, and a different window-adjustment scheme from that in Reno will be used; otherwise, Veno will assume the loss to be congestive, and the Reno window adjustment scheme will be adopted. Our experiments indicate that $\beta = 3$ is a good setting.

It is worth pointing out that the original Vegas algorithm, *BaseRTT* is continually updated throughout the live time of the TCP connection with the minimum round-trip time collected so far. In our work, however, *BaseRTT* is reset whenever packet loss is detected, either due to time-out or duplicate acknowledgements (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.

B. Congestion Control Based on Connection State

Reno employs several fundamental algorithms. In the following sections, we review these algorithms and describe how Veno modifies them.

1) Slow Start Algorithm: At the start-up phase of a connection, Reno sets the window size, *cwnd*, to one and sends out the first packet. Thereafter, each time a packet is acknowledged, the window size increases by one. Thus, *cwnd* increases from one to two after the first round-trip time, from two to four after the second round-trip time (because of the reception of two ACKs), and so on. This results in an exponential increase of the sending rate over time until packet loss is experienced. Veno employs the same initial slow-start algorithm as Reno's without modifications.

2) Additive Increase Algorithm: In addition to *cwnd*, there is a window threshold parameter *ssthresh* (slow-start threshold) in Reno. When *cwnd* is below *ssthresh*, the slow-start algorithm will be used to adjust *cwnd*. However, when $cwnd > ssthresh$, the window increase rate is reduced to avoid congestion. Specifically, *cwnd* will be increased by one after each round-trip time rather than after the reception of each ACK. Essentially, *cwnd* is set to $cwnd + 1/cwnd$ after each ACK to achieve a linear increase effect. In Reno, the initial value of *ssthresh* is 64 kB. The value of *ssthresh* is changed dynamically throughout the duration of the connection, as will be detailed later.

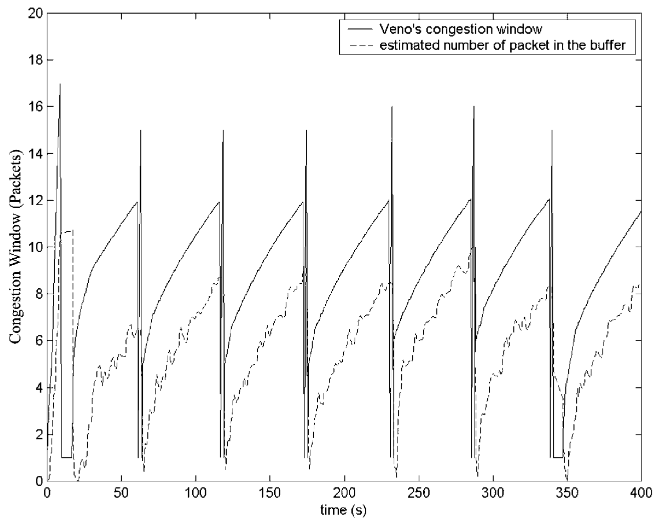
In Veno, the Additive Increase Algorithm of Reno is modified as follows:

```
If $(N < \beta)$ // available bandwidth not
  fully utilized
  set $cwnd = cwnd + 1/cwnd$ when each new ACK
  is received
else if $(N \geq \beta)$ // available bandwidth
  fully utilized
  set$cwnd = cwnd + 1/cwnd$ when every other
  new ACK is received
```

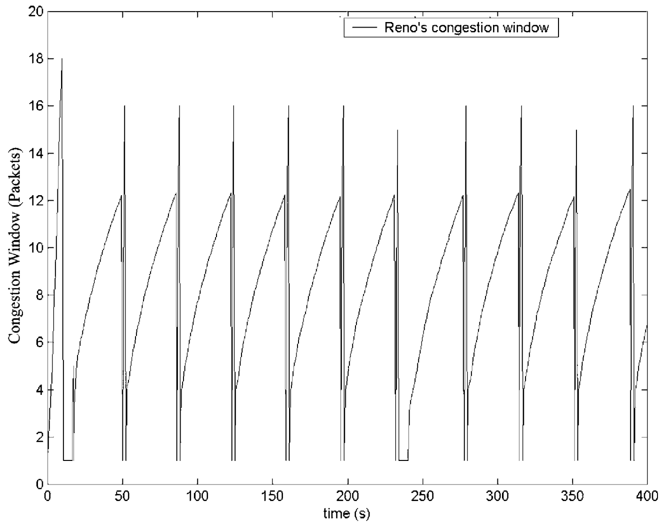
The first part is the same as the algorithm in Reno. In the second part, the backlog packets (N) in the buffer exceeds β, and we increase the window size by one every two round-trip times so that the connection can stay in this operating region longer.

For illustration, Fig. 1(a) and (b) shows the difference between the window evolutions of a Veno and a Reno connection in an experiment over our campus network.

As can be seen in the figures, Veno experiences fewer oscillations than a Reno does. In addition, Veno stays at the large-window region longer because it does not increase its window size as quickly as Reno does when the critical region $N \geq \beta = 3$ is reached. This aspect of Veno is also drastically different from the algorithm of Vegas, in which a window-decreasing scheme is adopted when the critical region is reached.



(a)



(b)

Fig. 1. (a) TCP Veno evolution. (b) TCP Reno evolution, when there is no random loss.

Generally, the TCP throughput is given by $cwnd/RTT$. When there is no backlog, increasing *cwnd* does not increase *RTT*, and therefore the throughput goes up accordingly. However, when there is backlog, increasing the window size will not increase the TCP throughput anymore because there is a corresponding increase of *RTT* due to the increasing backlog at the bottleneck link. TCP throughput flattens out with increasing *cwnd* when there is backlog. In Veno, increasing the window more slowly (compared with Reno) in the critical region of backlog $N \geq 3$ not only does not cause any throughput reduction, but also has the effect of deferring the onset of self-induced congestion loss and, thus forcing TCP to stay longer in the "optimal" transmission-rate region. In Fig. 1 of 400s experimental trace, we see Veno experiences almost 40% less congestion loss (if excluding packet losses resulted by initial slow start) than Reno does.

Additionally, from the practical standpoint, as Reno has been widely deployed in the current Internet, any modified TCP version must conform to specifications as defined in RFC793, RFC1122, and RFC2914. Veno continues to increase window when imminent congestion is detected but with a slower speed than Reno does. Veno complies with the rule as set out in the RFCs.

3) Multiplicative Decrease Algorithm: In Reno, there are two ways to detect packet loss. A packet can be declared to be lost if it has not been acknowledged by the receiver when a timer expires. If that is the case, the slow-start algorithm is initiated again with *ssthresh* set to $cwnd/2$ and *cwnd* reset to one. This has the drastic effect of suddenly reducing the sending rate by a large amount. That is, expiration of timer is interpreted as an indication of *severe* congestion. Veno does not modify this part of the algorithm.

Reno also employs another algorithm, fast retransmit, to detect packet loss. Each time the receiver in a Reno connection receives an out-of-order packet, it retransmits the last ACK to the sender. For example, if the receiver has received packet 3, but not packet 4 when packet 5 arrives, it will send out an ACK with ACK number $= 4$. This tells the sender that the receiver is still expecting packet 4. To the sender, this is a duplicate ACK because the receiver has also previously issued an ACK 4 when it receives packet 3. If packet 6 arrives while packet 4 is still missing, the receiver will issue yet another ACK 4. In Reno, when the sender receives three duplicate ACKs, it declares the corresponding packet to be lost even if the timer has not expired. A complementary algorithm to fast retransmit, called fast recovery, is then employed to alleviate the congestion as follows.

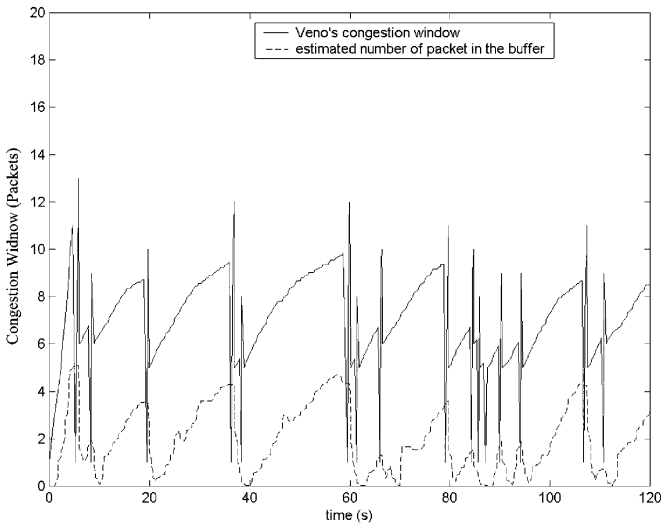
1) Retransmit the missing packet
   set $ssthresh = cwnd/2$
   set $cwnd = ssthresh + 3$.
2) Each time another dup ACK arrives, increment *cwnd* by one packet.
3) When the next ACK acknowledging new data arrives, set *cwnd* to *ssthresh* (value in step 1).

Veno modifies only the part in step 1 where *ssthresh* is set. Specifically

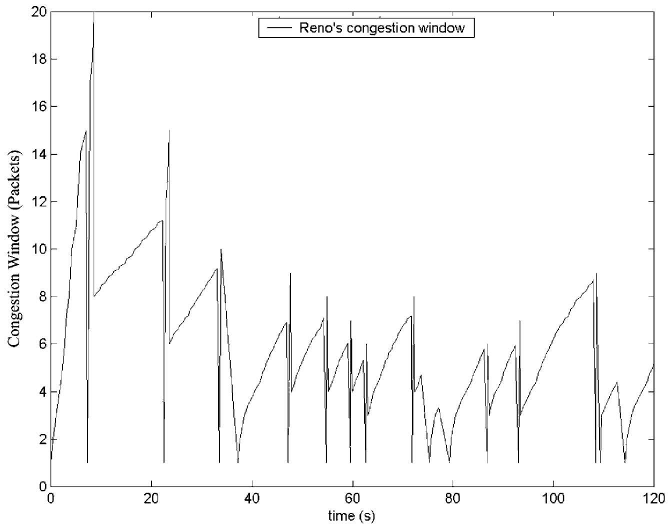
```
if $(N < \beta)$    // random loss due to bit errors is most likely to have occurred
  $ssthresh = cwnd * (4/5)$;
else $ssthresh = cwnd/2$; // congestive loss is most likely to have occurred
```

In other words, if the connection is not in the congestive state (i.e., if $N < \beta$), Veno assumes that the loss is random. Accordingly, it reduces the *ssthresh*, hence the *cwnd*, by a smaller amount. In the above example and the experimental results presented in this paper, a factor of $4/5$ is used. In general, we could use any factor larger than $1/2$, but smaller than one so that the cutback in window size is less drastic than the case when loss is due to congestion. Our experimentation shows that a factor of larger than $3/4$ is desirable.

For illustration, Fig. 2 shows typical window evolutions of Veno and Reno obtained experimentally. More detailed results will be shown in the next section. In this example, the round-trip time as traced by *traceroute* is 180 ms and the random loss rate is 1%. For Veno, packets losses are artificially induced in the network at 9s, 19s, 37s, 39s, 61s, 65s, 79s, 84s, 85s 86s, 89s, 95s, 109s, and 112s. As compared to Reno's blindly reducing window by $1/2$, Veno decreases the window by a factor of $1/5$ at the losses of 9s, 39s, 61s, 65s, 84s, 85s, 86s, 89s, 95s, and 112s. But for the losses at 19s, 37s, 79s and 109s, Veno reduces the window by half due to the false alarms (Veno misinterprets random loss as congestion loss). We see, in these cases, Veno' behavior degenerates to that of Reno's, but the consequence should be no worse than that experienced by Reno. In some sense, as addressed in Section III-A.2, such severe penalty, corresponding to these kinds of random loss, may be the right action since the connection has been trapped into congestive state $(N \geq \beta)$ even thought the buffer overflowing has not happened. Intuitively, seeing Fig. 2(a) and (b), Veno's window evolution seems more robust and has sort of immune capability to random loss.



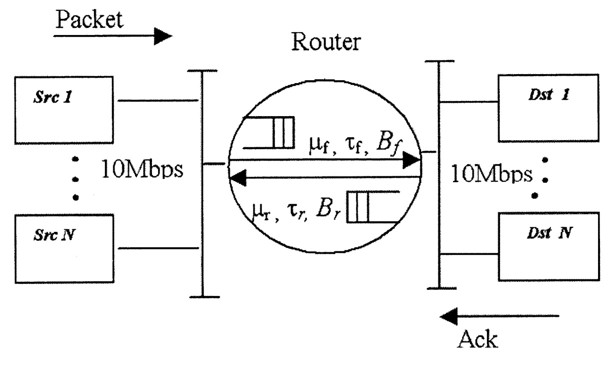
(a)



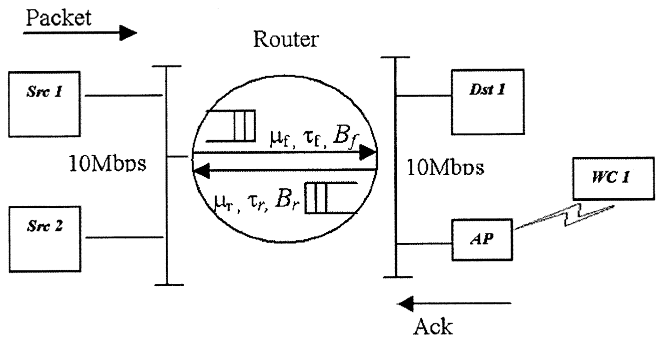
(b)

Fig. 2. (a) Veno's window evolution. (b) Reno's window evolution.

In summary, Veno only refines the additive increase, multiplicative decrease (AIMD) mechanism of Reno. All other parts of Reno, including initial slow start, fast retransmit, fast recovery, computation of the retransmission timeout, and the backoff algorithm remain intact. Only the sender stack is modified.



(a)



(b)

Fig. 3. Experimental networks. (a) Random loss is artificially induced in router. (b) Random loss actually occurs in the wireless link; no random loss is artificially induced in router.

A final point to be noted is that the clock resolution in the conventional TCP is 500 ms, which is too coarse grained for the measurement of *RTT* for calculation of the *Actual* and *Expected* rates. In Veno, a millisecond resolution timestamp is recorded for each transmitted segment once the connection leaves the initial slow start stage. The detailed implementation is discussed in [1].

III. Performance Evaluation of TCP Veno

We describe the experimental set-ups in Section III-A and study the effectiveness of packet loss distinguishing scheme in Veno. In Sections III-B and III-C, we present and compare the performance results of Reno and Veno TCP in lossy and nonlossy experimental networks (i.e., networks with random loss and without random loss). It is demonstrated that Veno not only achieves significant throughput improvements over Reno, but can also coexist harmoniously with the Reno in that it does not degrade the performance of Reno. Live Internet measurements in Section III-D further back-up these observations.

A. Verification of Packet Loss Distinguishing Scheme in Veno

1) Experimental Network Setup: We performed two experiments. One with packet loss artificially induced, the other on real wireless network where random packet loss actually occurs.

Fig. 3(a) shows our first experimental set-up. $Src1, \ldots, SrcN$ are TCP senders that run TCP Reno or Veno over NetBSD1.1. $Dst1, \ldots, DstN$ are TCP receivers with Red Hat Linux installed. The links are labeled with their bandwidth capacities.

A drop-tail *router*[2] with first-in–first-out (FIFO) queues is set up using FreeBSD4.2. It has embedded IPFW [19], [34] commands to configure the forward buffer size B_f, backward buffer size B_r, forward bandwidth μ_{f}, reverse bandwidth μ_{r}, forward propagation delay τ_{f}, and reverse propagation delay τ_{r}. Different random packet loss rates can also be artificially induced using the embedded system command in FreeBSD4.2.

In Fig. 3(b), a wireless client (*WC1*) is connected to the router through a wireless access point (AP), TCP connection is set up between a Veno source (*Src2*) and the wireless client (*WC1*). In addition, a UDP connection can be set up between *Src1* and *Des1* to provide the background traffic. By logging packets from Veno source (*Src2*) to the router, and from the router to the wireless client (*WC1*), and together with *TCPsuite* developed by us, we can infer which packets are lost due to buffer overflow in the router and which packets are in the wireless link. That is, congestion loss and random loss can be successfully separated. In Section III-A-2, we use this setup to verify the effectiveness of Veno's packet loss distinguishing scheme.

2) The Verification of Packet Loss Distinguishing Scheme: In this section, we report our test on the effectiveness of the proposed packet loss distinguishing scheme in Veno. For this purpose, we first consider a single TCP Veno connection sharing bottleneck link without background traffic, and then study the case in which background traffic is provided by a UDP connection. The configuration experimented here features a $\mu_{\mathrm{f}} = \mu_{\mathrm{r}} = 1.6$ Mb/s bottleneck link with FCFS discipline and buffer size of $B_f = B_r = 12$. TCP and UDP packets are assigned the same priority. The round-trip time (traced by *tracerouter*) is about 60 ms between *WC1* and *Src2*. Table I shows the results of three runs of the experiment within one day. A 12 MB file was transferred between *WC1* and *Src2* in each Veno's test.

In the first run of the experiment, Veno experienced 240 fast retransmits when the UDP sending rate is 500 kb/s. Out of the 240 fast retransmits, 84 were triggered when Veno estimated the connection to be in noncongestive state, and 156 were triggered when Veno estimated the connection to be in congestive state.

For the 84 fast retransmits, there was only one wrong estimation, when the packet loss was due to buffer overflow in the router rather than noncongestive loss in the wireless link. The accuracy of the estimation is close to $83/84 = 99\%$. Data with zero and 1 Mb/s UDP background traffic, as well as those in the other two runs of the experiment show consistently high estimation accuracy.

For the other 156 fast retransmits triggered with Veno assuming the connection to be in congestive state, the accuracy is rather low. Only ten out of the 156 fast retransmit were due to correct state estimations. However, misinterpreting random loss as congestion loss does not cause any adverse effects since the congestion window is reduced by half, the same as in Reno. At most, the misdiagnosis does not bring any throughput improvement to Veno. As a matter of fact, most of these 146 random losses occur during the period in which the connection is being trapped into congestive states (*in which imply there are at least β extra packets being accumulated in the buffer of the bottleneck router along the TCP connection path, and buffer overflowing still has not occurred*). In some sense, window-halved

[2]A drop-tail router drops the newly arriving packets when the buffer is full.

TABLE I
EFFECTIVENESS OF VENO'S PACKET LOSS DISTINGUISHING SCHEME UNDER DIFFERENT UDP SENDING RATES (#FF4/5, #FF1/2: FAST RETRANSMIT TRIGGERED WHEN VENO ESTIMATED THE CONNECTION TO BE IN NON-CONGESTIVE AND CONGESTIVE STATES, RESPECTIVELY. 4/5 AND 1/2 REFER TO THE REDUCTION FACTOR OF THE CONGESTION WINDOW)

Veno TCP		No background traffic		Background traffic (UDP) with sending rate of 500kb/s		Background traffic (UDP) with sending rate of 1Mb/s	
Actual type of packet loss		Congestion	Random	Congestion	Random	Congestion	Random
First	#FF4/5	0	105	1	83	2	90
Run	#FF1/2	6	120	10	146	12	136
Second	#FF4/5	0	64	1	78	2	83
Run	#FF1/2	3	110	8	102	16	127
Third	#FF4/5	0	46	1	67	1	53
Run	#FF1/2	3	123	9	139	19	132

reductions corresponding to these kinds of random loss may be right actions because the available bandwidth has been fully utilized in these situations. These kinds of random loss could be regarded as congestive drops because system has been congested to some extent.

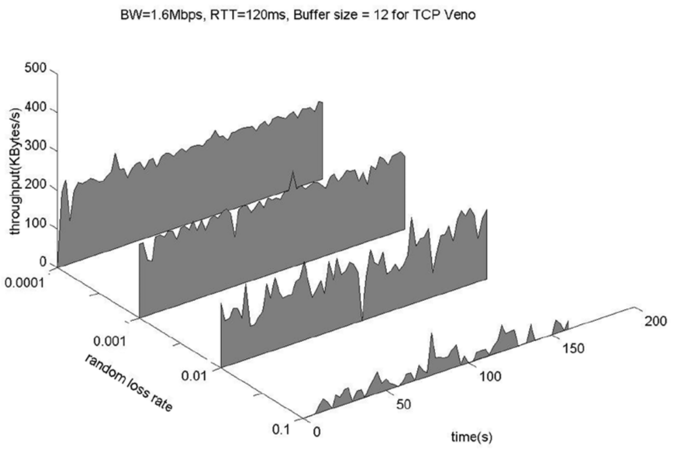
We may also examine the congestive losses (columns under "Congestion" in Table I) themselves and see how many of them caused the wrong window reduction in Veno. For example, when the UDP background traffic in 500 kb/s, in the first round, one out of ten congestive losses were misdiagnosed. Looking at all the results, the rate of misdiagnosing congestive losses as random losses is no more than 17% (two out of twelve in the first run when UDP traffic is 1 Mb/s), and usually much lower than that (zero when there is no background traffic and the Veno connection is the only traffic, and no more than 12% when the background traffic is 500 kb/s).

B. Single Connection Experiments

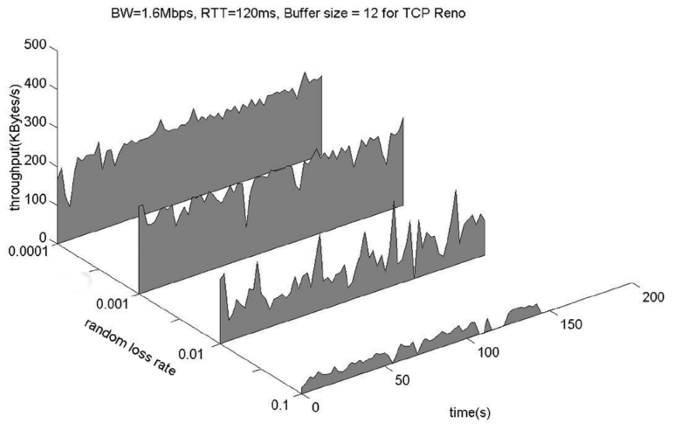
In this section, we present single-TCP connection results—i.e., only one of the source-destination pairs in Fig. 3(a) is turned on. Fig. 4 shows the evolution of the average sending rate for loss probabilities ranging from 10^{-4} to 10^{-1}. The buffer size at the router is set to be 12, the link speed $\mu_\text{f} = \mu_\text{r} = 1.6$ Mb/s, round-trip propagation delay is 120 ms, and the maximum segment size is 1460 B. In Fig. 4, the z axis is the sending rate of a TCP connection, x axis is time, and y axis is segment loss probability. Each data point of the sending rate corresponds to the amount of data sent over an interval of 160 ms divided by 160 ms.

The durations of all experiments are 150s. When the packet loss rate is relatively smaller $(< 10^{-3})$, the evolutions of Veno and Reno are similar. But when the loss rate is close to 10^{-2} packets/s, which corresponds to bit error rate of 8.56×10^{-7} b/s for segment size of 1460 B or to 2.44×10^{-6} b/s for segment size of 512 B, Veno shows large improvements over Reno.

To better observe the throughput difference between Reno and Veno under different packet loss rates, throughput versus loss rate is plotted in Fig. 5. One 32-MB file is transferred for each run. Other parameters are the same as the above experiment setting. Throughput is derived by dividing 32 MB by the time taken to transfer the file. It is shown that the throughput of TCP Veno is consistently higher than that of TCP Reno for a range of random loss probabilities. In particular, at loss rate of 10^{-2} packets/s, the throughput of TCP Veno (146.5 kB/s) is 80% higher than that of TCP Reno (81.9 kB/s). These desirable results are mainly attributed to Veno's refined multiplicative decrease algorithm that performs intelligent window adjustment based on the estimated connection state.



(a)



(b)

Fig. 4. Average sending rate for loss probability ranging from 10^{-4} to 10^{-1}.

Under low-loss environments (random loss rate is less than or equal to 10^{-3} packets/s), however, Veno and Reno have roughly the same throughput. At these low rates, random loss is not a significant factor any more. The likelihood of having a packet loss within a congestion window, *cwnd*, a fundamental parameter determining the TCP behavior, is small in this case.

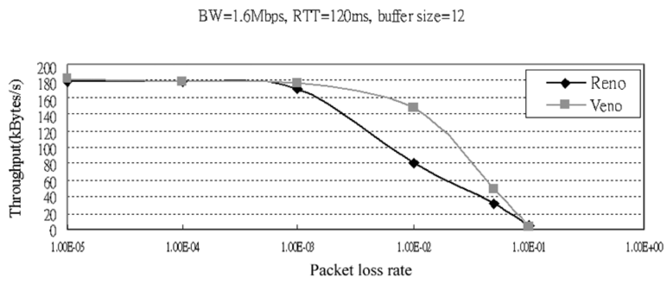


Fig. 5. Throughput versus packet loss rate for Reno and Veno.

Also, under very heavy random loss situation (close to 10^{-1}), neither Reno nor Veno can maintain the TCP self-clocking mechanism and they both experience frequent timeouts. When timeouts occur, Veno and Reno operate similarly by falling back to the slow start algorithm. With reference to Fig. 4(a) and (b), both TCPs degrade significantly at loss rate 10^{-1}. However, heavy packet loss rate of 10^{-1} seldom occurs in real wireless networks. Random packet loss rate of 10^{-2} is more typical. RFC3002 [29], for example, shows that wireless channel with IS-95 CDMA-based data service has an error rate of 1%–2% with little correlation among losses.

C. Experiments Involving Multiple Co-existing Connections

For experiments in this section, we again used the network depicted in Fig. 3(a). The maximum segment size is 1460 Bytes. To verify the compatibility between Veno and Reno, we set up four connections consisting of a mixture of Veno and Reno to share a common bottleneck link of 4 Mb/s with $B_f = B_r = 28$ and round-trip propagation time of 120 ms. Each connection transferred a 32-MB file. Fig. 6(a) shows the sequence number evolution of four Reno connections, and Fig. 6(b) shows the sequence number evolution of two Reno connections (solid curve) and two Veno connections (dashed curve), with no random loss introduced. As shown in the two graphs, Veno competes with Reno in a fair manner, since throughputs in all connections are roughly the same.

Fig. 6(c) and (d) shows the results of the same experiments with random loss rate of 10^{-2} introduced. As indicated in Fig. 6(d), Veno connections exhibit substantially higher throughput than coexisting Reno connections. Note, however, that the evolution of two Reno connections in Fig. 6(d) is close to that of Fig. 6(c), where Veno is absent. We can therefore infer that Veno does not achieve its superior throughput by aggressively grabbing bandwidth away from Reno; rather it is better able to use the available bandwidth that will be otherwise left unused. Since Veno does not adversely affect coexisting Reno, we can conclude that Veno is compatible with Reno.

It is interesting to observe that the slopes of the Veno connections in Fig. 6(d) are roughly the same as those in Fig. 6(b). This means that the Veno connections are not adversely affected by the random loss rate of 0.01.

Comparison of the two Veno connections within Fig. 6(b) or (d), also shows that the two Veno connections share bandwidth with each other in a fair manner. The fairness among Veno connections and their compatibility with Reno connections are further verified through additional experiments, the results of which are depicted in Fig. 7.

We measured the throughput for a total of five connections consisting of a mixture of Reno and Veno connections. Fig. 7 shows the average throughput per connection for Reno and Veno. The horizontal axis is the number of Veno connections. For example, the experiment corresponding to the point marked 3 on the axis consisted of two Reno and three Veno connections. Besides the average throughput per connection, the standard deviations are also shown. From these two graphs, three important observations can be made.

The first observation is about fairness. All the data points exhibit small standard deviations. Therefore, fairness is observed within the group of Reno or Veno connections.

The second observation is that the curves are nearly horizontal regardless of the combination of Reno and Veno connections. The significance of this observation is that coexisting Veno will not starve the bandwidth used by Reno connections, or vice versa. They can coexist harmoniously.

The third observation is that Veno has somewhat higher throughput than Reno in nonlossy environments, but in lossy networks, significant improvements over Reno are obtained. Obviously, these improvements can be attributed to their efficient utilization of the available bandwidth.

D. Live Internet Measurements

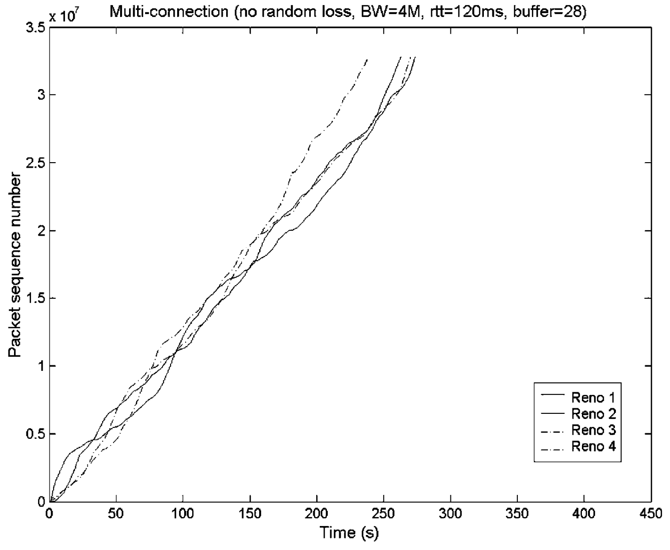
In the preceding section, a lossy network model was artificially introduced at the router in order to simulate the behaviors of TCP in different situations. The results of live Internet measurements are presented in this section. The performance over WLAN and wide area network (WAN) is studied.

1) Measurement of TCP Veno and TCP Reno in WLAN: A wireless LAN as shown in Fig. 8(a) was set up for single-connection experiments. The wireless base station is Proxim RangeLAN2-7510 and the wireless network adapter of the laptop is RangeLAN2-7400.

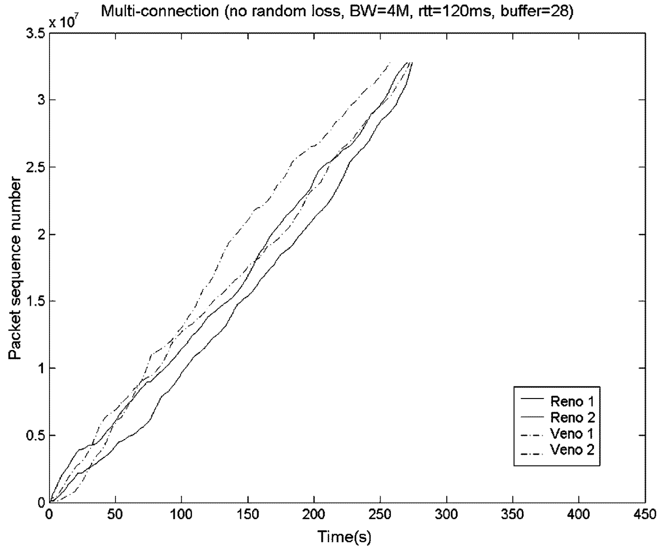
The base station is located in one room, the laptop in another room. The distance between them is about 8 m. We use the laptop as a client to download an 8-MB file from the data server, which is connected to the $10/100$ Ethernet switch. The round-trip time between laptop and server is about 20 ms (traced by traceroute). Fig. 8(b) shows ten throughput measurements over one day duration. In total, we conducted our tests over a period of five days. All the results are similar. Generally, the throughput of a Veno connection is about 336 kb/s, 21% improvement over that of a Reno connection (about 255 kb/s). In our latest experiments on 802.11, Veno gets throughput improvement of up to 35%.

Although the throughput measurement is a straightforward metric to gauge the performance of TCP, it does not capture the detailed behavior of TCP connections. In practice, its aggressiveness [6] can be evaluated by measuring the number of retransmitted packets, which depends on how many timeouts are experienced, and how many fast retransmits are triggered. In this section and the following section, these metrics are used to evaluate whether the behavior of Veno and Reno TCP is aggressive or not.

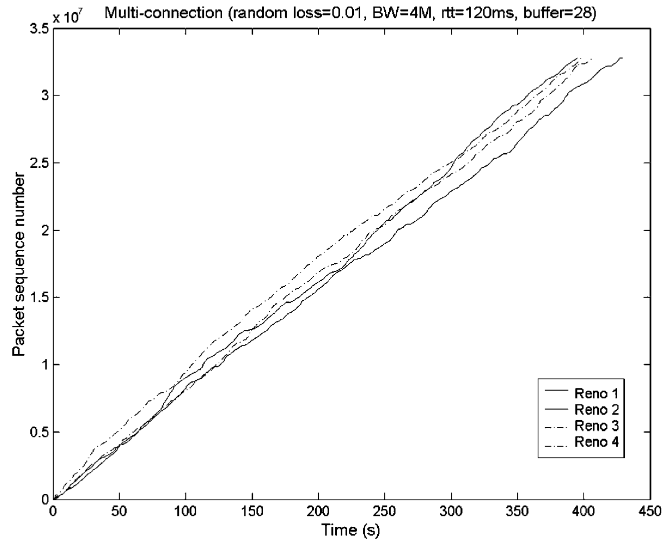
Using *tcptrace*, the average number of retransmitted packets for the Veno connection was found to be 42.9 packets, a 36% reduction as compared with 67 retransmitted packets of the Reno connection. The average number of timeouts experienced by



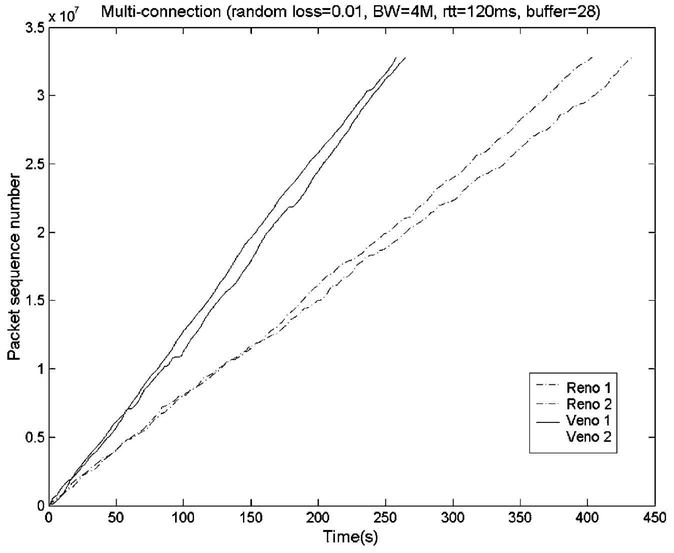
(a)



(b)



(c)



(d)

Fig. 6. (a) Four Reno connections with no random loss. (b) Two Reno and two Veno connections with no random loss. (c) Four Reno with random loss rate 10^{-2}. (d) Two Reno and two Veno with random loss rate 10^{-2}.

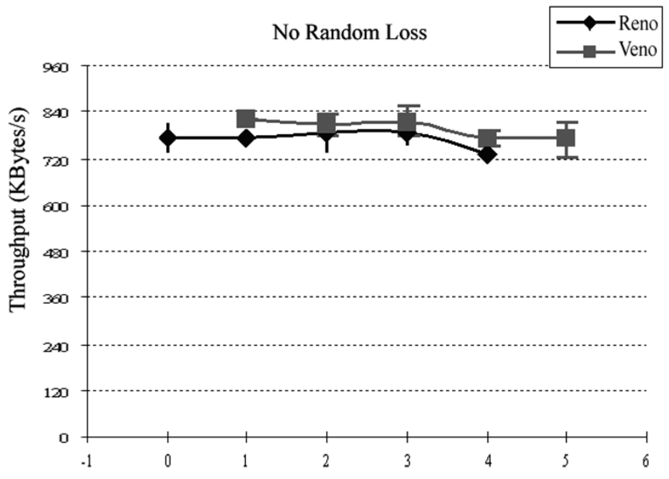
Veno is 4.1, a 53% reduction with respect to the 8.7 timeouts of Reno. Moreover, using the *TCPsuite* tool developed by us, the average number of fast retransmits triggered by Veno was found to be 13.9 versus Reno's 23.8, a reduction of 42%.

The gathered data suggest that Veno's improvement over Reno is attributed to its efficient utilization of the available bandwidth. As a matter of fact, with respect to the frequencies of retransmits and timeouts, a Veno connection with the refined AIMD employed is more conservative than a Reno connection, particular in nonlossy or lower loss situations when only the modified additive increase (and not the modified multiplicative decrease) comes into effect. This is further confirmed by the experiments for metropolitan and cross-country WAN presented in the following part.

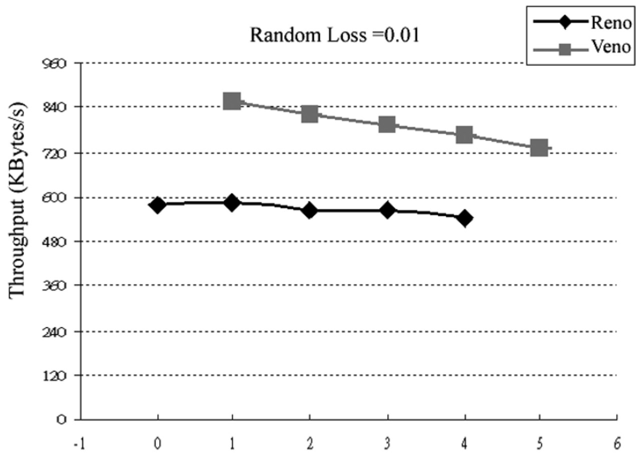
2) Measurement of Veno and Reno in WAN: We conducted live measurement over the wide-area Internet. As shown in Fig. 9, we set up a server at the Chinese University of Hong Kong (CUHK) with Veno and Reno installed.[3] Clients were separately located at the Tsing Hua University (TSU) in China, and the University of Hong Kong (HKU). CUHK and HKU are connected via metropolitan WANs, while CUHK and TSU are connected via cross-country WANs. Additionally, a client at the University of California, Berkeley, is being planned. This paper, however, only presents results related to CUHK, TSU, and HKU.

We first conducted an experiment without introducing any wireless links along the tested path. Therefore, the random packet loss probability should be very small, and one would expect the improvement of Veno over Reno to be small or has similar behavior as seen in Fig. 6(b). The purpose of this experiment is to verify that Veno indeed works over a live network environment.

[3]Our implementation lets users choose between different TCP versions using the socket function *setsockpot()*

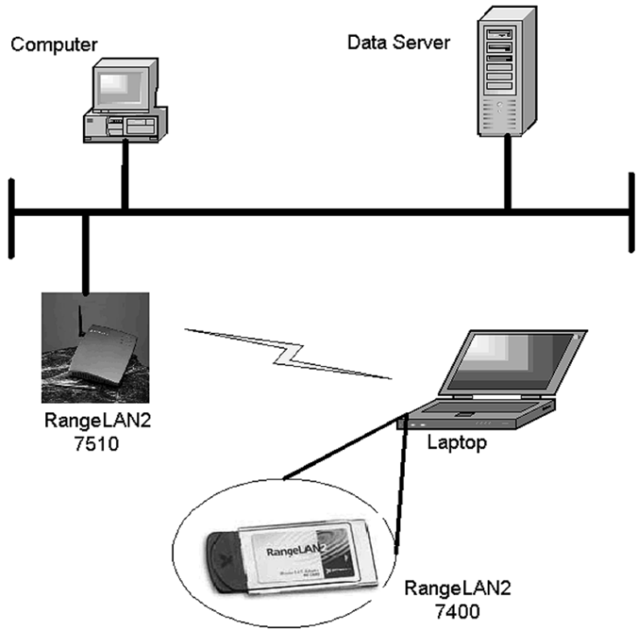


(a)

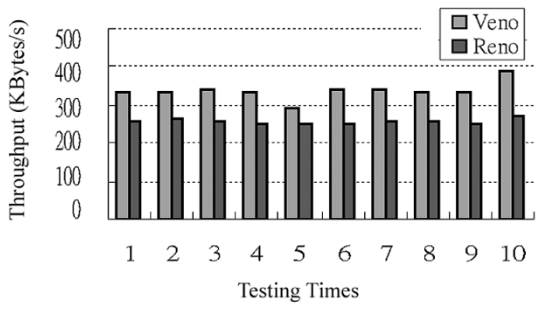


(b)

Fig. 7. Average throughput versus number of Veno connections given the sum of Veno and Reno connections is fixed at five. (a) Nonlossy network. (b) Lossy network.



(a)



(b)

Fig. 8 (a) Wireless LAN topology. (b) Wireless LAN measurement.

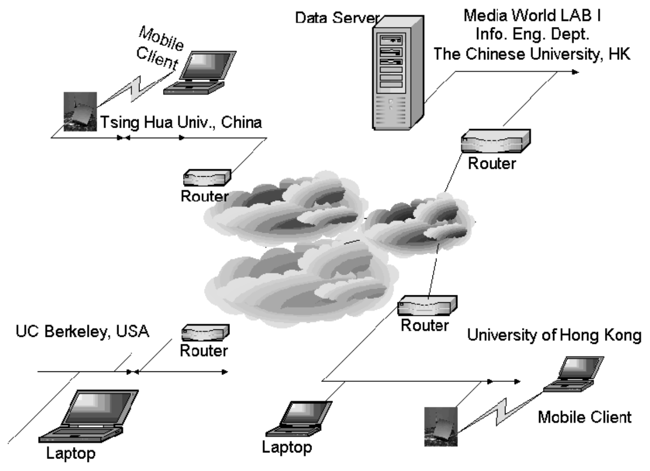


Fig. 9. WAN measurement topology.

The measured results in different time slots within one day are shown in Table II. There are six time slots within the day. For each time slot, we conducted five tests with one-minute breaks among the consecutive tests. In each test, two TCP connections, one Veno and one Reno, were set up simultaneously for transfer of a 16-Mb data file between CUHK and HKU. The data shown in each column are the averaged values over these five runs. The round-trip time as traced by *traceroute* between HKU and CUHK is about 45 ms with 10 ms variation over different time slots.

Totally, we conducted the tests for the six time slots over five days (from Monday to Friday). The general trends of the results among the five days are similar and, therefore, we show only the results of one day in Table II. With reference to Table II, Veno only obtains slightly higher throughput than Reno. Based on the reduced amount of timeouts, fast retransmits, and retransmitted packets, Veno is shown to be less aggressive than Reno. Veno does not "steal" bandwidth from Reno and can coexist harmoniously with a Reno connection. Its slight higher performance achievement is attributed to its better utilization of the available bandwidth.

In the next set of experiments, we set up WLAN as access networks in HKU. As can be seen in Fig. 9, the mobile client (laptop) is connected to the outside Internet via this wireless access network. The models adopted here are the same as that in Section III-D.1. The base station and the client are located in separated rooms.

A 16-MB file was transferred between the server and the mobile client in each test. Table III shows the results of our tests between CUHK and HKU during different periods of the day for

TABLE II
MEASUREMENTS FOR MULTIPLE CONNECTIONS (ONE RENO AND ONE VENO) BETWEEN CUHK AND HKU
(TO: TIMEOUT, FF: FAST RETRANSMIT TRIGGERED, RETRAN. PKS: RETRANSMITTED PACKETS)

	Reno/Veno					
Time slots	10:00~11:00	11:00~12:00	13:00~14:00	14:00~15:00	15:00~16:00	16:30~17:30
Th(KB/s)	212.9/223.6	183.0/184.5	258.8/278.6	214.9/232.4	226.8/244.8	191.9/198.2
# TO	19.2/14.8	22.5/15.5	16.2/7.4	16.6/7.8	14.6/7.2	16.5/13.0
# FF	51.8/36.4	69.3/50.3	38.6/22.6	39.4/24.4	38/22.6	42.7/29.7
# Retran. Pkts	485/220	602.7/276.7	561.4/238	532.6/180.8	412.4/158.6	400.5/248.5

TABLE III
MEASUREMENTS OF SINGLE CONNECTION OVER METROPOLITAN WANs BETWEEN CUHK AND HKU
(TO: TIMEOUT, FF: FAST RETRANSMIT TRIGGERED, RETRAN. PKS: RETRANSMITTED PACKETS)

	Reno/Veno					
Time slots	10:30	11:30	12:30	15:00	16:00	21:00
Th(KB/s)	234.3/307.8	243.6/342.3	248.0/ 340.2	260.6/328.5	230.6/302.5	251.1/338.2
# TO	26/18.6	33.2/16	26/12.6	12/12.8	33.8/20.7	26.2/18
# FF	58.6/41.0	60.6/38.2	56.5/36.5	41.7/36.6	75.6/40.2	56.8/34.3
# Retran. Pks	895.4/560.6	1067/719	1012/598	427/397	925/587	997/580

TABLE IV
MEASUREMENTS FOR SINGLE CONNECTION OVER CROSS-COUNTRY WANs BETWEEN CUHK AND TSU
(TO: TIMEOUT, FF: FAST RETRANSMIT TRIGGERED, RETRAN. PKS: RETRANSMITTED PACKETS)

	Reno/Veno				
Time slots	10:00~11:00	11:00~12:00	12:00~13:00	14:00~15:00	15:00~16:00
Th(KB/s)	65.5/64.4	51.3/60.3	45.1/54.5	51.5/58.7	70.2/73.9
# TO	8.1/6.5	7.6/5.8	8.9/5.3	9.5/5.8	6.5/6.4
# FF	19.9/11.5	21.6/10.5	24.6/18.5	28.3/18.7	14.5/14.0
# Retran. Pkts	185.8/127.5	166.7/103.9	170.1/137.3	156.6/137.2	98.5/97.1

Reno and Veno. There are six time slots within the day. For each time slot, we conducted ten tests, five for Reno and five for Veno in the following sequence: A Reno connection was first tested and then after it was completed, a test for Veno was started one minute later. Thereafter, there was an eight-minute break, after which Reno and Veno was tested again with one minute between the tests. This was repeated until all ten tests were completed. The values in Table III for each time slot are the average values over the five tests. The round-trip time as traced by *traceroute* between HKU and CUHK is about 45 ms with 10 ms variation over different time slots.

We conducted the tests for the six time slots over five days (from Monday to Friday). The general trends of the results among the five days are similar, and therefore we show only the results of one day in Table III. With reference to Table III, Veno obtains throughput improvement of up to 41% over Reno in different time slots. The numbers of timeouts, fast retransmits triggered, and retransmitted packets are much lower in Veno.

The above measurements are obtained in the metropolitan WANs. We next look at test results over cross-country WANs between TSU host and CUHK to Table IV. We also set up one wireless LAN (same model as above) as access network in Tsing Hua University. The round-trip time as measured by *traceroute* between TSU and CUHK was about 290 ms with 30-ms variation among different tests. Again, the general trends of the results among the five days are similar. Table IV only shows measurement results of one day.

With reference to the above two tables, we see that generally Veno achieves a much higher throughput than Reno, particularly in metropolitan WAN. Nonetheless, the improvement is not consistent throughout the day, perhaps due to variability of other factors in the live Internet itself, e.g., how much interfering traffic there is over the cross-country path when the experiments were conducted. Again, Veno is less aggressive than Reno in terms of the numbers of timeouts, fast retransmits triggered, and retransmitted packets.

IV. CONCLUDING REMARKS AND FUTURE WORK

Random packet loss in the wireless access networks can cause performance degradation in an end-to-end TCP connection. We have proposed and demonstrated that a novel TCP version called TCP Veno can deal with random packet loss effectively on an end-to-end basis. Numerous experiments [1] have been conducted in experimental networks and live Internet straddling across different areas in Hong Kong, and Mainland China. The detailed investigations clearly indicate that TCP Veno can achieve significant improvement without *adversely* affecting other concurrent TCP connections in the same network.

Veno is desirable from the following three standpoints.

1) Deployability: For any improved TCP to be worthwhile, it must be easy to deploy over the existing Internet. Therefore, before modifying the legacy TCP design, we must ask the question: "*Is this modification or design amenable to easy deployment in real networks?*" To realize this goal, ideally, there should little or preferably no changes required at intermediate routers. For the two ends of the communicating terminals, it is preferred that only one side requires changes, so that the party interested in the enhanced performance can incorporate the new algorithm without requiring or forcing all its peers to adopt the changes. Generally, it is preferred that the sender that sends large volumes of data to many receivers to modify its algorithm (e.g., the Web server) rather than the receivers. The simple modification at the TCP sender stack makes Veno easily deployable in real networks. Any party who is interested in the enhanced performance can single-handedly do that by installing the Veno stack.

2) Compatibility: Compatibility refers to whether a newly introduced TCP is compatible with the legacy TCP in the sense that it does not cause any detrimental effects to the legacy TCP, and vice versa, when they are running concurrently in the same network. Veno coexists harmoniously with Reno without "stealing" bandwidth from Reno. Its improvement is attributed to its efficient utilization of the available bandwidth.

3) Flexibility: Flexibility refers to whether the TCP can deal with a range of different environments effectively. It is difficult to categorically declare one TCP version to be flexible or not flexible. We can say, however, Veno is more flexible than Reno in that it can deal with random loss in wireless networks better, alleviate the suffering in asymmetric networks [1], and has comparable performance in wired networks.

Although the idea of Veno is straightforward on hindsight, it was not that obvious in the beginning when this work was first started. It combines elements from two opposing TCP camps: 1) Reno which uses reactive congestion control and 2) Vegas which uses proactive congestion control. Veno can still be regarded as a reactive algorithm because it certainly does not aim to eliminate packet loss entirely: it makes use of the idea of state estimation in Vegas to formulate better strategies to deal with packet loss and to stay in the "optimal" operating region longer.

This paper has not addressed the issue of bursty packet loss in wireless networks. SACK [14] option has been proposed and shown to be effective in dealing with multiple packet losses within a congestion window. Veno by itself does not solve the problem of multiple packet losses. However, Veno and SACK can be combined easily to yield an enhanced TCP implementation, SACK Veno.

Reference [2] has investigated the performance of SACK Veno under different network conditions. The results show that SACK Veno can increase the throughput of pure SACK by up to 60% at packet loss rate of 0.01. As with comparison between Veno and Reno, experiments show that the improvement of SACK Veno can be attributed to its better efficiency rather than aggressiveness in grabbing bandwidth from other connections. Many TCP protocol stacks are now providing the SACK option. If SACK is already available on the receiver side, SACK Veno requires only the modification of the sender stack.

Generally speaking, we can see TCP Veno borrows the idea of congestion detection scheme in Vegas and intelligently integrates it into Reno's additive increase phase. Its state estimation is used as supplementary reference information to decide how to refine additive increase at the next step and how much the window is to be reduced once fast retransmit is triggered. Of course, principally, we could also use other predictive congestion detections (e.g., PBM' [7], Packet Pair [25]) or/and the other better predictive congestion detection schemes, or combinations of these schemes to refine Reno evolution.

What TCP Veno proposes is to refine Reno's AIMD evolution over heterogeneous networks by using the complete judgment of network state estimation – congestive state or non-congestive state, rather than merely depending on packet loss occurrence.

To date, much work has been explored to center on accurately making out which of packet losses are due to congestion and which are due to bit-errors or other noncongestion reasons, but, limited to too many uncertain factors (i.e., background traffic changing along the connection path) in real networks, progress in this kind of judging looks very slow. Perhaps, it may be more meaningful and more practical for a TCP connection to differentiate between congestive drops (*occurring in congestive state*) and noncongestive drops (*occurring in noncongestive state*). Veno adopts such differentiation to circumvent the packet-loss-type-distinguishing. Further study on this issue will be conducted in future work.

Acknowledgment

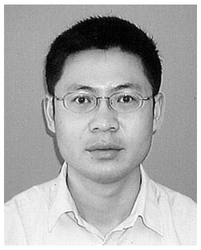
The train of thoughts leading to Veno started from reading Floyd's RFC2914, . . ., the list of TCP-compatible congestion control procedures is not limited to AIMD with the same increase/ decrease parameters as TCP. Other TCP-compatible congestion control procedures include rate-based variants of AIMD; AIMD with different sets of increase/decrease parameters that give the same steady-state behavior . . ..

The authors would like to thank Dr. V. Paxson for his invaluable constructive suggestions and original tool provided. Without his guidance, this work will be impossible to finish.

The authors would like to thank Prof. X. Li, Dr. J. L. Wang, Tsinghua University, Beijing, China, Prof. O. K. Li, Dr. K. L. Yeung, University of Hong Kong, and Prof. T. S. Yum, Chinese University of Hong Kong, for their great help, and in sincerely acknowledging Miss L. C. Chung, Mr. H. N. Lui, Mr. C. H. Nam, and Mr. W. Wang for their experimental work, and Dr. K. L. E. Law, University of Toronto, for his constructive discussion, and many thanks for anonymous reviewers of IEEE JSAC for their valuable comments and suggestions. The authors are grateful to Dr. Q. B. Xie – founding member of SCTP, at Motorola for his encouragement, B. Bing, at the research faculty at Georgia Institute of Technology, for his comments, and Dr. D. M. Chiu, Research Lab, SUN Microsystem Inc., for his careful guidance, in particular, his paper [39] that has deeply influenced the design of Veno's algorithm.

References

[1] C. P. Fu, "TCP Veno: End-to-End Congestion Control Over Heterogeneous Networks," Ph.D. dissertation, The Chinese Univ. Hong Kong,, Hong Kong,, 2001.
[2] L.-C. Chung, "SACK TCP$_{\mathrm{Veno}}$: An Enhanced Version of SACK TCP," M.Phil thesis, The Chinese Univ. Hong Kong,, Hong Kong,, 2001.
[3] C. P. Fu, L.-C. Chung, and S. C. Liew, "Performance Degradation of TCP Vegas in Asymmetric Networks and its Remedies," in *Proc. ICC'2001*, vol. 10, Helsinki, Finland, 2001, pp. 3229–3236.
[4] J. S. Ahn, P. B. Danzig, Z. Liu, and E. Yan, "Evaluation with TCP Vegas: Emulation and Experiment," *ACM Comput. Commun. Rev.*, vol. 25, no. 4, pp. 185–195, Aug. 1995.
[5] M. Allman, S. Dawkins, and D. Glover, "Ongoing TCP Research Related to Satellites," Internet Engineering Task Force (IETF), ser. RFC2760, 2000.
[6] M. Allman and A. Falk, "On the Effective Evaluation of TCP," *ACM Comput. Commun. Rev*, vol. 29, no. 5, Oct. 1999.
[7] M. Allman and V. Paxson, "On Estimating End-to-End Network Path Properties," in *Proc. SIGCOMM'99*, Cambridge, Aug. 1999, pp. 263–274.
[8] M. Allman, V. Paxson, and W. R. Stevens, "TCP Congestion Control," Internet Engineering Task Force (IETF), ser. RFC 2581, Apr. 1999.
[9] R. K. Balan, B. P. Lee, and K. R. R. Kumar, "TCP HACK: TCP Header Checksum Option to Improve Performance Over Lossy Links," in *Proc. INFOCOM'2001*, Anchorage, AK, pp. 309–318.
[10] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance,," in *Proc. SIGCOMM'94*, London, U.K., Oct. 1994, pp. 24–35.
[11] H. Balakrishnan, V. Padmanabhan, S. Sechan, and R. Katz, "A Comparison of Mechanisms for Improving TCP Performance Over Wireless Links," in *Proc. SIGCOMM'1996*, vol. 26, Stanford, CA.
[12] H.Hari Balakrishnan, "Challenges to Reliable Data Transport Over Heterogeneous Wireless Networks," Ph.D. dissertation, Univ. California, Berkeley, CA, 1998.
[13] A. Chockalingam, M. Zorzi, and V. Tralli, "Wireless TCP Performance with Link Layer FEC/ARQ," in *Proc. ICC'99*, 1999, pp. 1212–1216.
[14] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno, and SACK TCP," *Comput. Commun. Rev.*, vol. 26, no. 3, pp. 5–21, July 1996.
[15] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, and A. Romanow. (1999, Nov.). An Extension to the Selective Acknowledgment (SACK) Option for TCP. [Online]. Available: http://www.ietf.org/internet-drafts/draft-floyd-sack-00.txt
[16] S. Floyd, "Congestion Control Principle," Internet Engineering Task Force (IETF), ser. RFC2914, Sept. 2000.
[17] ——, "TCP and Explicit Congestion Notification," *ACM Comput. Commun. Rev.*, vol. 24, no. 5, pp. 8–23, Oct. 1994.
[18] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Trans. Networking*, vol. 7, pp. 458–472, Aug. 1999.
[19] FreeBSD 4.2 Handbook. [Online]. Available: http://freebsd.org/handbook/index.html
[20] U. Hengartner, J. Bolliger, and T. Gross, "TCP Vegas Revisited," in *Proc. INFOCOM'2000*, pp. 1546–1555.
[21] J. Hoe, "Improving the Start-Up Behavior of a Congestion Control Scheme for TCP," in *Proc. SIGCOMM'1996*, Palto Alto, CA, pp. 270–280.
[22] V. Jacobson, "Congestion Avoidance and Control," in *Proc. SIGCOMM'88*, Stanford, CA, pp. 314–329.
[23] R. Jain, "A delayed-based approach for congestion avoidance in interconnected heterogeneous computer networks," *ACM Comput. Commun. Rev.*, vol. 19, no. 5, pp. 56–71, Oct. 1989.
[24] V. Jacobson, Modified TCP Congestion Avoidance Algorithm, Apr. 30, 1990. end2end-interest mailing list.
[25] S. Keshav, "A Control-Theoretic Approach to Flow Control," in *Proc. SIGCOMM'91*, Zurich, Switzerland, Sept. 1991, pp. 3–15.
[26] T. V. 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.
[27] T. V. Lakshman, U. Madhow, and B. Suter, "Window-Based Error Recovery and Flow Control with a Slow Acknowledgment Channel: A Study of TCP/IP Performance," in *Proc. INFOCOM '97*, pp. 1199–1209.
[28] R. J. La, J. Walrand, and V. Anantharam. (1998, July) *Issues in TCP Vegas* [Online]. Available: http://www.path.berkeley.edu/~hyongla
[29] D. Mitzel, "Overview of 2000 IAB Wireless Internet Working Workshop," Internet Engineering Task Force (IETF), ser. Rep. RFC3002.
[30] J. Mo, R. J. La, V. Anantharam, and J.Jean Walrand, "Analysis and Comparison of TCP Reno and Vegas," in *Proc. INFOCOM'99*, vol. 3, pp. 1556–1563.
[31] C. Parsa and J. J. Garcia-Luna-Aceves, "Improving TCP performance over wireless networks at the link layer," *ACM Mobile Networks and Applications J.*, vol. 5, no. 1, pp. 57–71, 2000.
[32] V. Paxson, "Measurements and Analysis of End-to-End Internet Dynamics," Ph. D. dissertation, Univ. California, CA, 1997.
[33] C. Parsa and J. J. Garcia-Luna-Aceves, *Improving TCP Congestion Control Over Internets with Heterogeneous Transmission Media*. Berkeley, CA: Comput. Eng. Dept., Univ. California, 1999.
[34] L. Rizzo. (1997) Dummynet: a simple approach to the evaluation of network protocols. *ACM Comput. Commun. Rev.* [Online], pp. 31–41
[35] S. Raman and S. McCanne, "A Model, Analysis, and Protocol Framework for Soft State-Based Communication," in *Proc. SIGCOMM'1999*, Cambridge, MA, pp. 15–25.
[36] I. T. Ming-Chit, D. Jinsong, and W. Wang, "Improving TCP performance over asymmetric networks," *ACM Comput. Commun. Rev.*, vol. 30, no. 3, pp. 45–54, 2000.
[37] Z. Wang and J. Crowcroft, "A new congestion control scheme: slow start and search (Tri-S)," *ACM Comput. Commun. Rev.*, pp. 32–43, Jan. 1991.
[38] S. Y. Wang and H. T. Kung, "Use of TCP decoupling in improving TCP performance over wireless networks," *Wireless Network*, vol. 7, pp. 221–236, 2001.
[39] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *J. Comput. Networks and ISDN*, vol. 17, no. 1, pp. 1–14, June 1989.
[40] C. Casetti, M. Gerla, S. Mascolo, M.Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proc. ACM Mobicom 2001*, Rome, Italy, July 16–21, 2001, pp. 287–297.
[41] C. Chung, C. P. Fu, and S.C. Liew, "Improvements achieved by SACK employing TCP Veno equilibrium-oriented mechanism over lossy networks," in *Proc. EUROCON'2001, Int. Conf. Trends in Communications*, Bratislava, Slovakia, July 2001, pp. 202–209.



Cheng Peng Fu (S'01–A'02) received the B.Eng. and M.Phil. degrees in electromagnetic theory and microwave technology, and the Ph.D. degree in information engineering from the Shanghai University of Science and Technology, Shanghai, China, and the Chinese University of Hong Kong, Hong Kong, in 1990, 1995, and 2001, respectively.

He is one of founding members of CERNET (China Education and Research Network), which was launched in 1995 to network 1000 universities in China by TCP/IP technology while he served Shanghai Jiao Tong University as a Faculty Member at Networking Center from 1995 to 1997. Meanwhile, he also designed and deployed Shanghai Jiao Tong Campus ATM Network, Shanghai Education and Research Network, and Shanghai Telemedicine Center. Then, he joined SUN Microsystem Inc., Shanghai, China, as a Senior Member of the Technical Staff. He is currently an Assistant Professor at Nanyang Technological University, Singapore.

He has worked at Datun Meikuang TV as a news reporter for two years after graduation in 1990. He has conducted research and published actively in areas of network protocol and architecture design, distributed object design of J2EE, and microwave analysis and measurement.

Soung C. Liew (S'84-M'87-SM'92) received the S.B., S.M., E.E., and Ph.D. degrees from the Massachusetts Institute of Technology (MIT), Cambridge.

From 1984 to 1988, he was at the MIT Laboratory for Information and Decision Systems, where he investigated fiber-optic communications networks. From March 1988 to July 1993, he was at Bellcore (now Telcordia), NJ, where he engaged in broadband network research. He is currently a Professor at the Chinese University of Hong Kong, Hong Kong.

He is currently Co-Director of the Area of Excellence in Information Technology (AoE-IT), Hong Kong, a joint project with participations from the Chinese University, University of Science and Technology, and the University of Hong Kong. Besides academic activities, he is also active in the industry. He cofounded two technology start-ups in Internet Software and is currently a Consultant for the Hong Kong Applied Science and Technology Research Institute. He is the holder of three U.S. patents. His research interests include Internet protocols, multimedia communications, optical networks, and broadband packet switch design, and he initiated and coordinated the first inter-university ATM network testbed in Hong Kong, in 1993.

Dr. Liew is a Member of the Research Grant Council Engineering Panel of Hong Kong, and Fellow of IEE and HKIE. He is listed in Marquis *Who's Who in Science and Engineering*.