Wireless Netw (2010) 16:183–198
DOI 10.1007/s11276-008-0123-4

TCP CERL: congestion control enhancement over wireless networks

Hosam El-Ocla

Published online: 24 June 2008

Abstract In this paper, we propose and verify a modified version of TCP Reno that we call TCP Congestion Control Enhancement for Random Loss (CERL). We compare the performance of TCP CERL, using simulations conducted in ns-2, to the following other TCP variants: TCP Reno, TCP NewReno, TCP Vegas, TCP WestwoodNR and TCP Veno. TCP CERL is a sender-side modification of TCP Reno. It improves the performance of TCP in wireless networks subject to random losses. It utilizes the *RTT* measurements made throughout the duration of the connection to estimate the queue length of the link, and then estimates the congestion status. By distinguishing random losses from congestion losses based on a dynamically set threshold value, TCP CERL successfully attacks the well-known performance degradation issue of TCP over channels subject to random losses. Unlike other TCP variants, TCP CERL doesn't reduce the congestion window and slow start threshold when random loss is detected. It is very simple to implement, yet provides a significant throughput gain over the other TCP variants mentioned above. In single connection tests, TCP CERL achieved an 175, 153, 85, 64 and 88% throughput gain over TCP Reno, TCP NewReno, TCP Vegas, TCP WestwoodNR and TCP Veno, respectively. In tests with multiple coexisting connections, TCP CERL achieved an 211, 226, 123, 70 and 199% throughput improvement over TCP Reno, TCP NewReno, TCP Vegas, TCP WestwoodNR and TCP Veno, respectively.

Keywords Wireless networks · TCP Reno · TCP NewReno · TCP Vegas · TCP Westwood · TCP Veno · Transmission Control Protocol (TCP) congestion control · Random loss · Congestive loss · Lossy networks

H. El-Ocla (✉)
Department of Computer Science, Lakehead University, 955 Oliver Road, Thunder Bay, ON, Canada P7B 5E1
e-mail: hosam@lakeheadu.ca

1 Introduction

TCP Reno (referred to as Reno in this paper) is the most widely deployed reliable transport protocol used in computer networks. The congestion control mechanism of Reno plays an important role in enabling Reno to run efficiently in a variety networks from high-bandwidth, low-delay local area networks to low-bandwidth, long-delay wide area networks. Reno congestion control comprises four intertwined algorithms: slow start, congestion avoidance, fast retransmit and fast recovery [1]. These algorithms are highly dependent upon the assumption that network congestion is the prime cause for segment losses. During transmission, the sender identifies the loss of a segment either by the arrival of several duplicate acknowledgments or the absence of an acknowledgment within a timeout interval. The sender reacts to segment loss by decreasing its congestion window size, resulting in a reduction of the load on the intermediate links, thereby controlling the congestion in the network.

A well-known problem with Reno is its performance degradation in networks subject to random losses [2–4]. Reno does not distinguish and isolate congestive losses from random losses. The Reno sender assumes all segment losses are signals of network congestion, and reacts to random segment loss by reducing the congestion window. As a consequence, Reno suffers drastic performance degradation in lossy networks.

Many schemes have been proposed to alleviate the effects of random losses on TCP performance [5–12].

As shown in [7], techniques that deal with random loss in an end-to-end manner are promising since significant performance gains can be achieved without extensive support from intermediate nodes in the network. This lack of support ensures that end-to-end techniques can be deployed easily [13–16]. Another point worth mentioning is that local techniques make their most significant performance gains when the random loss is high. However, segment loss rates of 1–2% are typical in wireless channel with IS-95 CDMA-based data service [17] as well as other wireless networks. At typical segment loss rates, end-to-end techniques can achieve throughput substantially close to that achieved by the various local techniques, as shown in Fig. 12 of [7].

In this paper we propose and verify a sender-side modification of Reno that we call TCP Congestion Control Enhancement for Random Loss (referred to as CERL in this paper). CERL is an end-to-end mechanism that achieves immediate throughput improvement over wireless networks once it is implemented. We demonstrate it's throughput improvement under a wide range of network settings, something that was never emphasized in the study of Veno [9–12]. The key idea behind TCP CERL is to use the *RTT* estimates made throughout the duration of a connection to estimate the queue length of the bottleneck router. This queue length estimate, combined with a dynamically set threshold value, allows CERL to effectively distinguish between random losses and congestive losses. The sender reacts to the two kinds of segment loss with different treatment. If a segment loss is detected via duplicate acknowledgments when the network load is estimated to be low, the segment loss is treated as a random loss. In this case, the sender only retransmits the lost segment, and doesn't reduce the congestion window and slow start threshold. We implemented this idea as a TCP module in the network simulation tool *ns-2* [18]. Simulation shows that CERL gains significant throughput improvement over Reno, NewReno, Vegas, WestwoodNR and Veno.

The rest of the paper is organized as follows. In Sect. 2, we give a brief overview of the previously mentioned TCPs. In Sect. 3, the concepts and mechanisms of CERL are introduced and discussed. In Sect. 3, we test the effectiveness of CERL by comparing CERL with Reno, NewReno, Vegas, WestwoodNR and Veno using simulations conducted with *ns-2*. Finally, we conclude this paper in Sect. 5.

2 Overview of other TCPs

In this section we briefly describe the other TCPs that CERL is compared to in this paper.

In Reno, a single packet loss will be indicated via the reception of three duplicate ACKs causing the missing segment to be retransmitted (Fast Retransmit) and Fast Recovery to be entered. If multiple segments were dropped from this window of data, the acknowledgment for the retransmitted segment will acknowledge some but not all of the segments transmitted before the Fast Retransmit. This is called a partial acknowledgment. Upon reception of a partial ACK, Reno will exit Fast Recovery. After three duplicates of this partial ACK arrive, Reno will Fast Retransmit and enter Fast Recovery again [1].

Similar to Reno, NewReno will Fast Retransmit and enter Fast Recovery after receiving three duplicate ACKs. However, NewReno will not exit Fast Recovery upon reception of a partial ACK. Instead, it retransmits the missing segment indicated by the partial ACK immediately without waiting for three duplicate ACKs. Fast recovery will be exited when an ACK which acknowledges all of the segments transmitted before Fast Retransmit was entered arrives [5].

The main goal of Vegas is to keep the amount of extra bytes in the network between two threshold values. It uses the estimated number of bytes in the network to decide whether or not the congestion window should increase linearly, not change, or decrease linearly during the next RTT, as discussed in [6].

Westwood monitors ACKs to estimate the bandwidth currently used by and thus, available to the connection. On the event of a timeout or the reception of three duplicate ACKs, rather than cutting the slow start threshold *ssthresh* in half as is done by Reno, Westwood sets *ssthresh* equal to $BWE \times RTTmin$, where BWE is the current bandwidth estimate and $RTTmin$ is the minimum RTT observed. Thus, *ssthresh* is set to the bandwidth-delay product, which is equal to the ideal window by definition in [19].

Veno uses a mechanism similar to that used by Vegas to estimate the buffer backlog of an end-to-end connection at the bottleneck router. Similar to CERL, when congestion is indicated via three duplicate ACKs, Veno attempts to distinguish between congestive loss and random loss by comparing the previously mentioned estimate to a statically set threshold value [9–12].

When there is no packet loss and Veno[1] measures the backlog $l \geq N = 3$ (where N is an arbitrary threshold that is assumed to be 3), Veno increases the window size by one for every two round trips. By increasing the window size slower than Reno in this critical region of $l \geq 3$ Veno attempts to defer the onset of self induced congestion loss, thus forcing the TCP connection to stay longer in the "optimal" transmission rate region. When packet loss

[1] In our explanation of Veno we have used symbols consistent with those used in our treatment of CERL.

occurs and $l < 3$, Veno considers the loss to be due to random loss and cuts the window size by 1/5 instead of 1/2 as in Reno. All other parts of Reno, including initial slow start, fast retransmit, fast recovery, and the backoff algorithm remain intact with Veno.

3 TCP CERL

CERL and Veno are similar in concept. Specifically, they both attempt to distinguish between random loss and congestive loss to allow for the independent treatment of each case. However, the mechanisms utilized by CERL differ greatly from those used in Veno, as should become clear in the remaining parts of this section.

3.1 Measure bottleneck link queue length

We assume that there is only one bottleneck link in the path of a TCP connection that performs first-in-first-out drop-tail queueing. So *RTT* consists of two parts: the queueing delay at the bottleneck link and the sum of round-trip propagation delay and service delay. The queueing delay is equal to $\frac{l}{B}$ where l is the queue length and B is the bandwidth of the bottleneck link [20]. We denote the sum of round-trip propagation and service delay with T and it can be considered to be constant for the given TCP. Using *RTT*, B and T, the bottleneck queue length l can be calculated using the following equation:

$$l = (RTT - T)B \tag{1}$$

We set T to be the smallest *RTT* observed by the TCP sender and l is updated with the most recent *RTT* measurement every time a new *RTT* measurement is received. Note that CERL attempts to measure the queue length of the bottleneck router, rather tham the bottleneck router backlog of an end-to-end connection, as is done in Veno.

3.2 Distinguish random losses from congestive losses

In CERL, the queue length l measured in (1) is used to estimate the congestion status of the link. Specifically, we set a dynamic queue length threshold N:

$$N = A * l_{max} \tag{2}$$

where l_{max} is the largest value of l observed by the sender and A is a constant between 0 and 1. For now we will remark that A has been set to 0.55 although an in-depth discussion of the value of the constant A is presented in Sect. 4.7. If $l < N$ when a segment loss is detected via three duplicate acknowledgments, CERL will assume the loss to be random rather than congestive; the lost segment will be retransmitted but the congestion window and slow start threshold will not be reduced. Otherwise, CERL will assume the loss is caused by congestion, and the congestion window and slow start threshold will be reduced as in Reno. However, multiple segment losses in one window of data will only reduce the congestion window once, as will be explained in detail in Sect. 3.4.

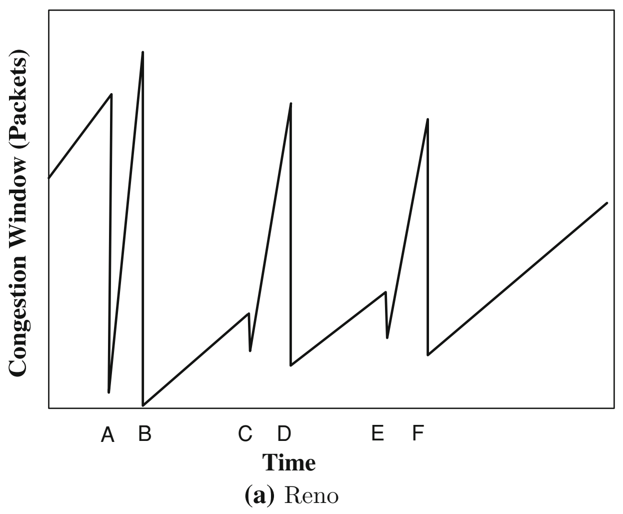
As previously stated, CERL makes the comparison $l < N$ to decide the congestive status of the network. Since l and N both contain a multiple of the constant B, we can divide both sides of the inequality $l < N$ by B and the resulting inequality will still be true. Therefore, actually measuring B is not necessary and we set B equal to one in our CERL implementation. In short, we are primarily concerned with comparing the currently observed *RTT* with the maximum *RTT* that has been observed.

3.3 Congestion window inflation

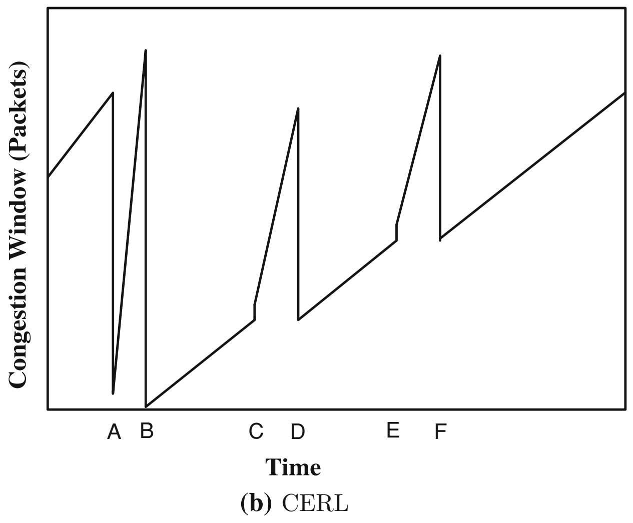
In Reno, a TCP receiver is required to generate an immediate acknowledgment (a duplicate ACK) when an out-of-order segment is received. Therefore, each duplicate acknowledgment received by the sender indicates that one segment has left the network. Since duplicate acknowledgments do not acknowledge any new data, the fast recovery algorithm must inflate the congestion window in order to keep the network pipe full [19]. Specifically, the congestion window is incremented by one segment for each duplicate acknowledgment received. After the first ACK that acknowledges new data is received, the window is then deflated by setting congestion window equal to the slow start threshold.

In CERL, the Reno code handling window inflation and deflation is modified slightly. In the case of the first congestive loss in the current window of data, the window inflation is the same as it is in Reno. Otherwise, the current value of the congestion window is saved. To deflate the window, the congestion window is then set to this value when the first ACK that acknowledges new data arrives.

Figure 1 illustrates the congestion window inflation/deflation mechanisms of both Reno and CERL. Note that we ignore the receiver's advertised window in this example. At times A, C, and E in Fig. 1(a), Reno detects congestion via three duplicate ACKs and therefore, *ssthresh* is set to *cwnd*/2 and *cwnd* to *ssthresh* + 3**SegmentSize*. Note that the latter setting of *cwnd* causes the congestion window to be immediately inflated by three segments. Between times A and B, C and D, and E and F, more duplicate ACKs are received causing the congestion window to be inflated. At times B, D, and F, an ACK that acknowledges new data is received, causing the congestion window to be deflated by setting *cwnd* equal to *ssthresh*.



(**a**) Reno



(**b**) CERL

Fig. 1 Congestion window evolution

At time A in Fig. 1(b), CERL detects the first congestive loss in the current window of data via three duplicate ACKs. The behavior of CERL is identical to that of Reno in this case. At time C, another congestive loss in the current window of data is detected via three duplicate acknowledgments. Since this is not the first congestive loss, *oldcwnd* is set equal to *cwnd* and *cwnd* is set equal to *cwnd* + 3**SegmentSize*. The value of ssthresh is not changed in this case. Between times C and D, more duplicates acknowledgments arrive causing the congestion window to be inflated. At time D, an ACK that acknowledges new data is received, causing the congestion window to be deflated by setting *cwnd* equal to *oldcwnd*. At time E, CERL detects another segment loss via 3 duplicate ACKs and decides that this loss was random. Between time E and up to and including time F, the behavior of CERL is identical to the behavior that occurred in the case of the second congestive loss in the current window of data, as described above.

3.4 Implementation

In summary, CERL makes the following changes to Reno:

1. Each time a new *RTT* estimate is received, T, l, l_{max} and N are updated.
2. The fast recovery algorithm is modified so that when three duplicate acknowledgments are received, the congestion window is only decreased by half when $l \geq N$ and only if this is the first time congestion has been detected during this window of data. Otherwise, the current value of the congestion window is saved in the variable *oldcwnd* to support the CERL window deflation mechanism previously discussed. The pseudocode implementing all of these changes is shown below.

```
IF ($l \geq N$&highestAck–1>lastDecMaxSentSeqno)
THEN
   ssthresh=min(cwnd,awnd)/2
   IF (ssthresh<2*segsize)
   THEN
          ssthresh=2*segsize
   END IF
   cwnd==ssthresh+3*segsize
   lastDecMaxSentSeqno=MaxSentSeqno
ELSE
   oldcwnd=cwnd
   cwnd=cwnd+3*segsize
END IF
```

Figure 2 illustrates how the above code prevents multiple congestive window decreases in one window of data. In this example, we assume that the TCP sender initially sends out segments 1–10 and segments 2 and 7 are lost. Eventually, three duplicates of ACK 2 are received by the sender causing the congestion window to be decreased, the variable *lastDecMaxSentSeqno* to be set to 10 (the highest sequence number sent by the sender so far), and segment 2 to be retransmitted. Since segment 7 was lost as well, instead of receiving a cumulative ACK acknowledging segment 2 as well as all other segments received, the sender receives three duplicates of ACK 7. Since 7 is less than 10, the current value of *lastDecMaxSentSeqno*, the congestion window is not decreased. Segment 7 is still retransmitted regardless of this fact.

3. We also modified fast recovery algorithm code that handles window deflation. The pseudocode is shown below.

```
IF (oldcwnd > 0)
THEN
  cwnd=oldcwnd
  oldcwnd=−1.0
ELSE
  cwnd=ssthresh
END IF
```

In conclusion, CERL is superior to Veno in many aspects as follows. (1) The threshold N is calculated dynamically to be more sensitive to the status of the buffer, and is not just a constant value as in Veno. (2) When $l < N$ and packet loss occurs, the window size is not reduced unlike Veno where it is reduced by 1/5. (3) Multiple segment losses in one window of data will only reduce the congestion window once. This is achieved by the novel congestion window inflation and deflation mechanism as described earlier. We will show in Sect. 4 that these differences will have a tremendous impact on the throughput regardless of network conditions.

4 Evaluation

4.1 Network configuration

We simulate a typical mixed wired/wireless network in which wireless nodes connect to a wired network via a base station. Since typical Internet users will download data significantly more than they will upload data, we focus on a situation in which the wireless nodes are the receivers of a bulk data transfer. All of the protocols we tested are supported by ns-2 by default except for Westwood and Veno. Support for the remaining two protocols required the installation of custom-built ns-2 modules. We installed the WestwoodNR module available from [22]. As well, the authors of Veno provided us with a Veno ns-2 module.

Figure 3 shows the network topology used in the simulations. In Fig. 3, S_1 to S_n are TCP senders and R_1 to R_n are TCP receivers. We use the notation S_iG_0 to denote the link between the ith sender and router G_0 and G_1R_i to denote the link between the router G_1 and the ith receiver. The routers G_0 and G_1 perform first-in-first-out drop-tail queueing. The router buffers are all set to a constant value of 90 packets unless otherwise noted.

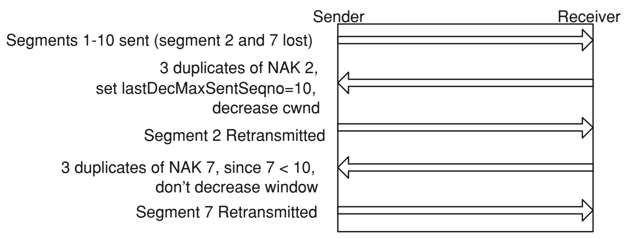

Sender
Receiver
Segments 1-10 sent (segment 2 and 7 lost)
3 duplicates of NAK 2, set lastDecMaxSentSeqno=10, decrease cwnd
Segment 2 Retransmitted
3 duplicates of NAK 7, since 7 < 10, don't decrease window
Segment 7 Retransmitted

Fig. 2 In CERL, multiple segment losses in one window of data only reduce the congestion window once

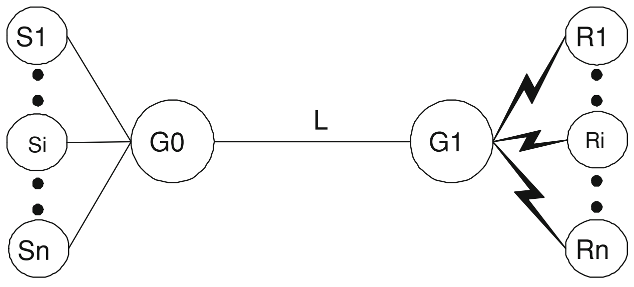


Fig. 3 Network topology for simulation

As discussed in [23], in order to simulate an ideal network, the advertised window size could be set to the bandwidth delay product. This would reflect the minimum window size that would allow data to flow over the network without the loss of any packets. Unfortunately this does not reflect a real world situation, as neither the bandwidth nor the delay of a network are precisely known quantities in the real world. In a simulation, the window size should be set large enough to allow the transmission to overwhelm the network. In this way the transmission protocol will be allowed to eventually settle on a suitable window size. If the advertised window size is set too low, the connection will be artificially throttled, and the simulation will not yield satisfactory results. We therefore set The TCP receiver's advertised window to a constant value of three times the router buffer size in all experiments. Setting the threshold window to a value that is sufficiently higher than the buffer size in the routers allows the connection to overwhelm the network as previously discussed.

The maximum segment size of TCP is set to 1460 bytes. Random losses are imposed on the links between G_1 and the TCP receivers using an exponentially distributed error model, as was done in [7]. We denote the bandwidth, propagation delay of link L, and the random segment loss rate between G_1 and the TCP receivers by B, T_p, and q, respectively. In the following sections, if the values of these variables are not mentioned, assume that $B = 2.0$ Mbps, $T_p = 50$ ms, and $q = 1\%$. We measure the throughput received by the TCP receiver in all experiments where throughput measurements are made.

4.2 Congestion window dynamics

Figure 4 shows the CERL and Reno congestion window dynamics when the router buffers are set to 27 packets. In this experiment, we assume $n = 1$ connection and ftp transfers data over this connection for a duration of 120 s. As well, the bandwidth and propagation delay of links S_1G_0 and G_1R_1 are set to 10 Mbps and 10 ms, respectively. Note that the congestion window inflation and deflation is not shown in Fig. 4 to make the graphs easier to interpret.

In Fig. 4(b), we see that the congestion window drops to 1 segment at 2.5 and 64.5 s. This is due to the fact that CERL falls back to slow start when a timeout occurs, as does Reno [1]. Although the random loss does have some impact on the congestion window growth of CERL, its congestion window growth is definitely superior to that of Reno. In this situation, the average throughput of CERL was 1.69 Mbps and the average throughput of Reno was 0.74 Mbps. Therefore, CERL gains an 128% throughput improvement over Reno in this experiment. In more simulations, under q up to 10%, the congestion window of CERL was still superior to that of Reno.

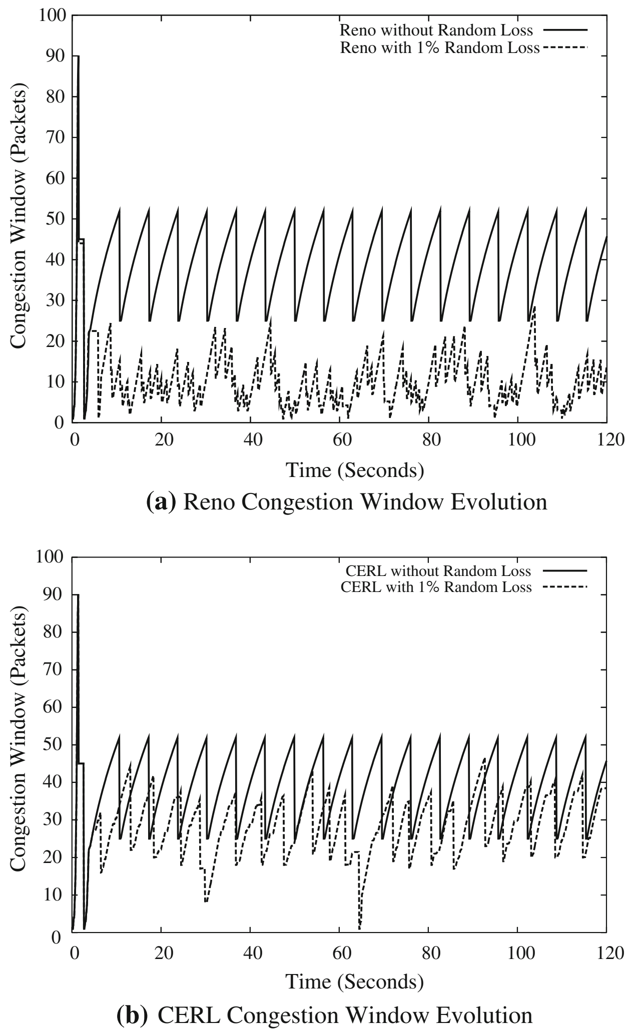


Fig. 4 Congestion window evolution

4.3 CERL behavior in absence of random loss

In this section, we demonstrate that the CERL random loss distinguishing mechanism does not affect the throughput of CERL when random loss is not present. To facilitate the demonstration, we define CERL2 as a modified version of CERL in which the code preventing multiple segment losses in one window of data from reducing the congestion window more than once is removed.[2]

In the multiple connection tests used in this section, the performance of a TCP when coexisting with 5 Reno connections is evaluated ($n = 6$ connections). We set $B = 5.0$ Mbps and $T_p = 80$ ms. S_1 is a TCP sender running either CERL, CERL2, or Reno. Five other TCP senders, S_2 to S_6, run Reno. Each sender initiates an ftp transfer to one of the receivers, R_1 to R_6. The bandwidth between all senders and G_0 and all receivers and G_1 is set to 10 Mbps. The propagation delay of links S_1G_0 and G_1R_1 is set to a fixed value of 10 ms. The ftp transfer between S_1 and R_1 always lasts for a duration of 480 s and starts at $t = 0$ s.

In order to test the TCPs under a wide range of traffic conditions, we randomly set the remaining simulation parameters as described below. The propagation delay of links S_iG_0 and G_1R_i, where i is an integer value ranging from 2 to 6, is set to a randomly generated number between 0.1 and 10.0 ms. The ftp transfer start time of each connection between senders S_2 to S_6 and receivers R_2 to R_6 is set to a randomly generated number between 1 and 200 s. As well, the ftp transfer end time is set to 480 s minus a randomly generated number between 1 and 200 s.

The random values are generated using a uniformly distributed random variable and a combined multiple recursive generator [24]. We chose 20 different seeds from among the 64 recommended seeds listed in the file rng.cc of the ns-2 source code [18] and we number the seeds from 0 to 19. Figure 5 illustrates the ftp transfer start and end times that resulted when the seed was set to 0.

In Fig. 6, we measure the throughput of CERL when it coexists with 5 Reno connections, CERL2 when it coexists with 5 Reno connections, and so on. These measurements are made as the seed ranges from 0 to 19 and $q = 0\%$. The performance of CERL2 is identical to that of Reno at every seed value. This illustrates that the CERL random loss distinguishing mechanism is not wrongly deciding a segment loss was random when random loss is not present, indicating that the threshold constant A has been set to an appropriate value. In Fig. 6, we also see that the performance of our regular CERL implementation often performs better than Reno due to the prevention of multiple window decreases strategy implemented by CERL.

[2] We commented out the line of code which sets lastDecMaxSent Seqno=MaxSentSeqno.

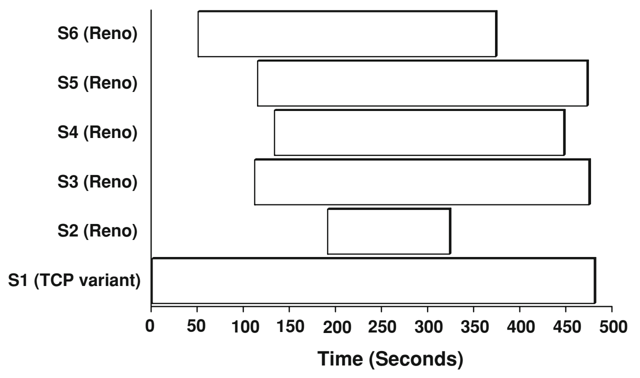


Fig. 5 The ftp transfer start and end times when the seed is set to 0

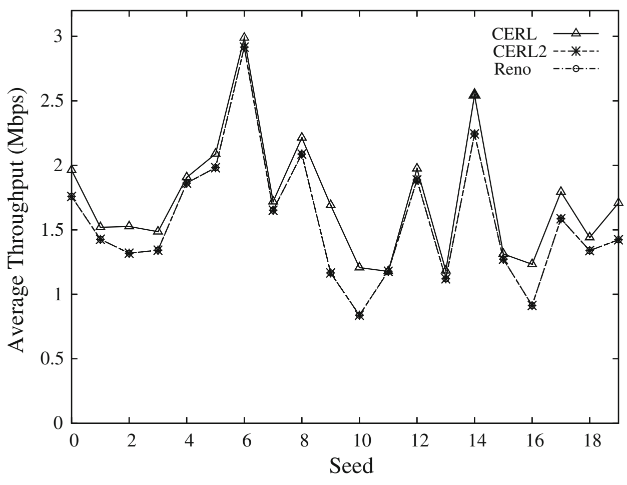


Fig. 6 Average throughput versus seed when $B = 2.0$ Mbps, $T_p = 80$ ms, and $q = 0\%$

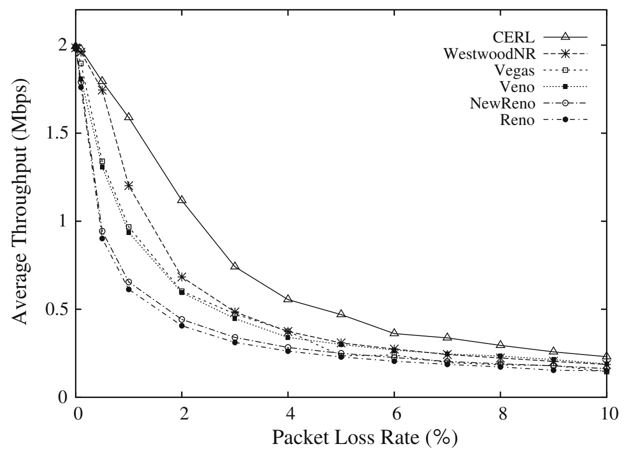
4.4 Throughput evaluation

In this section we demonstrate the throughput improvement that CERL is capable of achieving over Reno, NewReno, Vegas, WestwoodNR, and Veno. In total, we perform six different experiments in which a specific network parameter is varied. We show both a single connection and multiple connection version of each experiment. Some of the experiments performed are similar to those performed in [8].

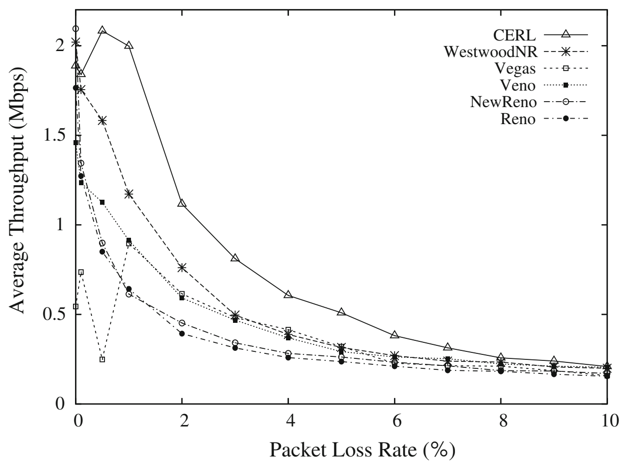
In the single connection tests, there is $n = 1$ connection and ftp transfers data over this connection for a duration of 480 s. As well, the bandwidth and propagation delay of links S_1G_0 and G_1R_1 are set to 10 Mbps and 10 ms, respectively. We run each TCP variant by itself in order to focus on the effectiveness of the mechanisms handling random losses, as was done in [7].

The multiple connection tests used in this section were previously described in Sect. 4.3. The only difference is that S_1 is now a TCP sender running either CERL, WestwoodNR, Vegas, Veno, NewReno, or Reno. It should also be noted that all multiple connection tests were performed using a constant seed value of 0. We will not show the results at other seed values because the results were similar.

In Fig. 7(a), we compare the various TCPs when q ranges from 0 to 10% and each TCP is run alone. The range of q assumed here is similar to that used in [7, 8]. As shown in Fig. 7(a), the throughput of all TCPs is close under very light ($q < 0.1\%$) or very heavy ($q > 10\%$) random loss. When the loss rate is very low, its effect on the congestion window evolution of each TCP is minimal and therefore, they all perform well. When the random loss is very heavy, timeouts occur frequently. All six TCPs operate similarly by falling back to slow start when a timeout occurs [1], and they all perform poorly under this condition. When $0.1\% < q < 10\%$, the TCPs are often informed of segment losses via duplicate acknowledgments rather than timeouts. By distinguishing and discriminating the segment losses detected



(a) Single connection, $B = 2.0$ Mbps.



(b) Multiple coexisting connections, $B = 5.0$ Mbps.

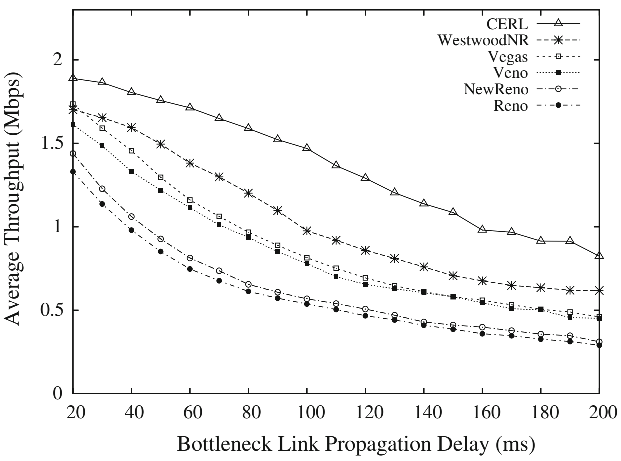
Fig. 7 Average throughput versus packet loss rate when $T_p = 80$ ms

via duplicate acknowledgments, CERL can effectively improve the congestion window evolution, thus gaining higher throughput. As shown in Fig. 7(a), CERL gains an 159, 142, 64, 32 and 70% throughput improvement over Reno, NewReno, Vegas, WestwoodNR, and Veno respectively, when $q = 1\%$. Also shown in Fig. 7(a), CERL gains an 175, 153, 85, 64 and 88% throughput improvement over Reno, NewReno, Vegas, WestwoodNR and Veno, respectively, when $q = 2\%$. As previously mentioned, packet loss rates of 1–2% typically occur in real wireless networks.

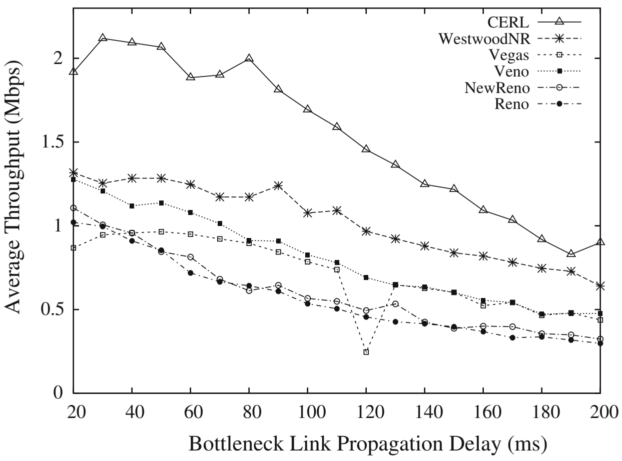
In Fig. 7(b), we compare the various TCPs when q ranges from 0 to 10% and each TCP coexists with 5 Reno connections. The results are fairly similar to those presented in the single connection version of this experiment shown in Fig. 7(a). However, there are some differences. Firstly, the throughput of CERL is in some cases higher than the throughput that occurs when random loss is not present. This is due to the fact that CERL is able to utilized some of the extra bandwidth left by the degradation of the Reno connections. Secondly, the throughput of Vegas is highly degraded as q ranges from 0 to 3%. This is because of the well-known issue regarding the degradation of Vegas when coexisting with Reno connections, as discussed in [25, 26]. As shown in Fig. 7(b), CERL gains an 211, 226, 123, 70 and 119% throughput improvement over Reno, NewReno, Vegas, WestwoodNR and Veno, respectively, when $q = 1\%$. Also shown in Fig. 7(b), CERL gains an 184, 147, 82, 47 and 89% throughput improvement over Reno, NewReno, Vegas, WestwoodNR, and Veno respectively, when $q = 2\%$.

Figure 8(a) compares the throughput of the various TCPs when T_p ranges from 20 to 200 ms and each TCP is run alone. As T_p increases, a larger congestion window is needed to utilize the full bandwidth of the link. Therefore, the random loss has a much higher impact on the throughput of each TCP as the propagation delay of the link increases. Moreover, the TCPs which rely on feedback information to make various estimates, such as CERL, Veno and WestwoodNR, do not receive this information fast enough for them to remain unaffected. However, note that the throughput of CERL still remains much higher than that of the other TCPs. For example, the throughput of CERL is significantly decreased when $T_p = 200$ ms, but it still gains an 184, 165, 56, 33 and 82% throughput improvement over Reno, NewReno, Vegas, WestwoodNR and Veno, respectively.

Figure 8(b) compares the throughput of the various TCPs when T_p ranges from 20 to 200 ms and each TCP coexists with 5 Reno connections. The results are fairly similar to those obtained in Fig. 8(a). However, once again CERL is able to use some of the extra bandwidth left unused and the performance of Vegas degrades due to the coexisting Reno connections. Note that when $T_p = 200$ ms, CERL gains an 202, 178, 106, 41 and 89%



(a) Single connection, $B = 2.0$ Mbps.



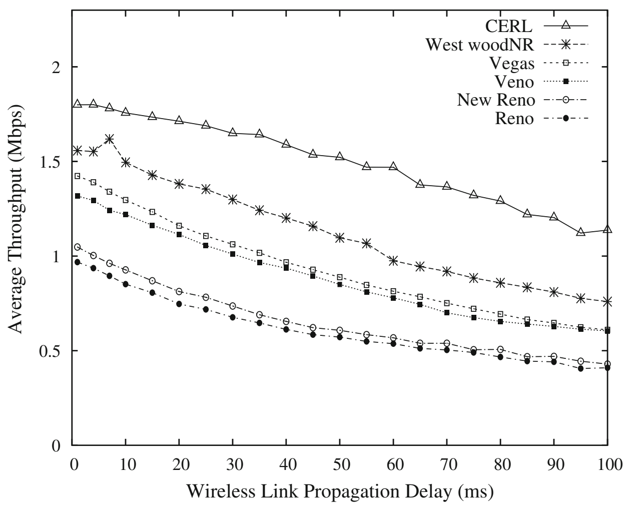
(b) Multiple coexisting connections, $B = 5.0$ Mbps.

Fig. 8 Average throughput versus bottleneck link propagation delay when $q = 1\%$

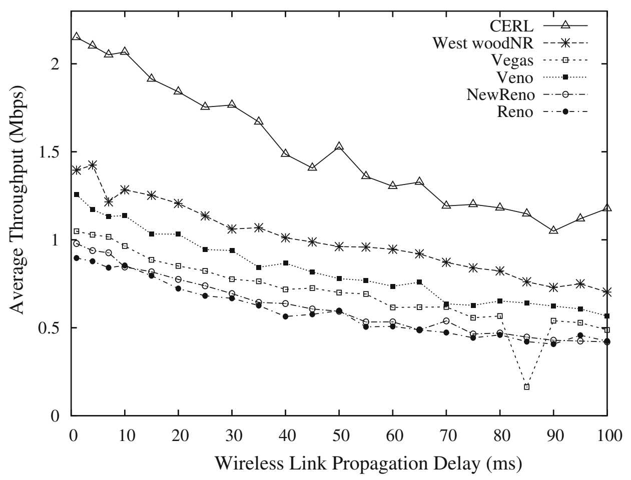
throughput improvement over Reno, NewReno, Vegas, WestwoodNR and Veno, respectively.

Figure 9(a) compares the throughput of the various TCPs as the propagation delay between G_1 and the TCP receiver ranges from 1 to 100 ms and each TCP is run alone. Each TCP follows a trend similar to those of Fig. 8(a). Figure 9(b) shows the multiple connection version of this experiment. In comparison to the single connection version of this experiment shown in Fig. 9(a), the performance of each TCP variant is much worse. However, CERL is still able to out perform the other TCPs.

Figure 10(a) shows the throughput of the various TCPs under different values of B ranging from 1 to 8 Mbps in the case where each TCP is running alone in the network. All of the TCPs eventually reach a point where the throughput does not increase as the bottleneck bandwidth increases. This observation can be well explained using the square root formula [27–29]:



(a) Single connection, $B = 2.0$ Mbps.

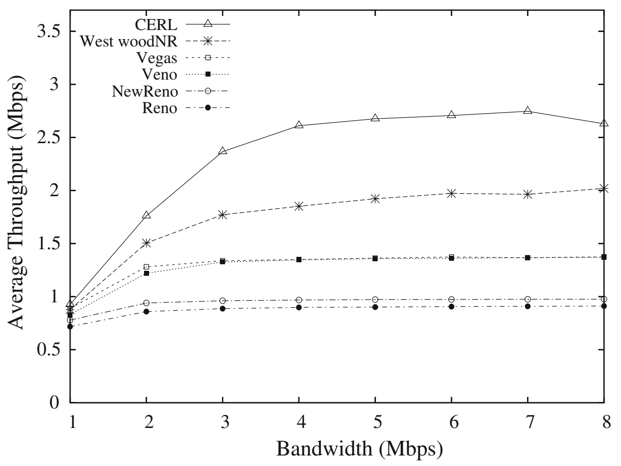


(b) Multiple coexistingconnections, $B = 5.0$ Mbps.

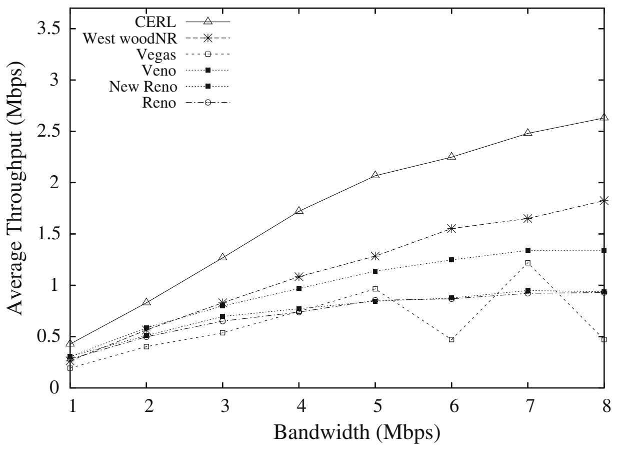
Fig. 9 Average throughput versus wireless link propagation delay when $T_p = 50$ ms, and $q = 1\%$

$$E_q = \frac{MSS}{RTT}\sqrt{\frac{C}{q}} \tag{3}$$

where E_q is the TCP throughput under random loss, *MSS* is the TCP maximum segment size, *RTT* is the average round-trip time, *C* is a constant close to 1 that differs according to different networks environments. This formula shows that random losses impose an upper limit on the throughput of a TCP connection. Therefore, all of the TCPs in Fig. 10(a) can use only a fraction of the available bandwidth. However, by distinguishing and discriminating the random losses, CERL's upper throughput limit is much higher than that of the other TCPs. This shows that CERL will be much more efficient than other TCPs in high-speed wireless networks such as 802.11b WLAN. Note that when $B = 8$ Mbps, CERL gains an 188, 169, 91, 30 and 92% throughput improvement over Reno, NewReno, Vegas, WestwoodNR and Veno, respectively.



(a) Single connection.



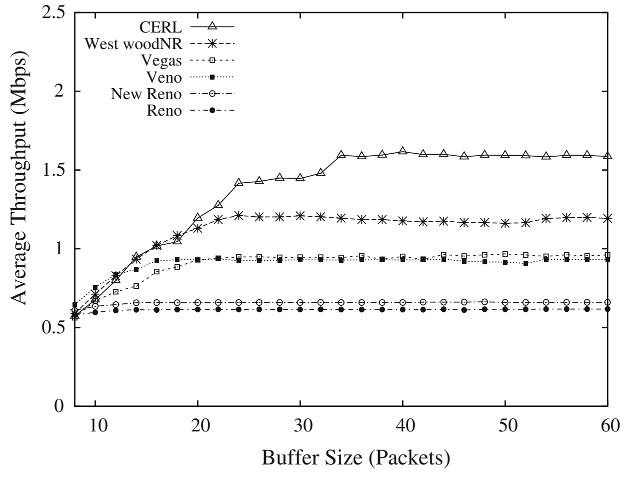
(b) Multiple coexisting connections.

Fig. 10 Average throughput versus bottleneck bandwidth when $T_p = 50$ ms and $q = 1\%$

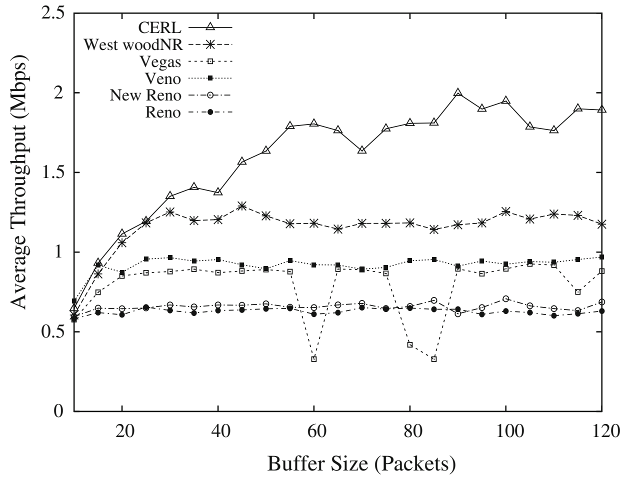
Figure 10(b) shows the throughput of the various TCPs under different values of B ranging from 1 to 8 Mbps in the case where each TCP coexists with 5 Reno connections. The results are similar to those shown in Fig. 10(a). However, once again we witness the Vegas performance degradation issue. Note that when $B = 8$ Mbps, CERL gains an 183, 181, 458, 44 and 96% throughput improvement over Reno, NewReno, Vegas, WestwoodNR and Veno, respectively.

Figure 11(a) compares the throughput of the various TCPs, each running alone in the network, as the buffer size of routers G_0 and G_1 ranges from 8 packets to 60 packets and $T_p = 80$ ms. The bandwidth-delay product as defined in [19] equals $B \times RTT$ and is approximately 35 packets[3] in this experiment. Therefore, when the buffer size is set to any

[3] Since the advertised window and router buffers are specified in units of packets in ns-2, we must divide the bandwidth-delay product by the packet size of 1460 bytes.



(a) Single connection, $B = 2.0$ Mbps.



(b) Multiple coexisting connections, $B = 5.0$ Mbps.

Fig. 11 Average throughput versus buffer size when $T_p = 80$ ms, and $q = 1\%$

value less than 35 packets, all TCPs will be unable to utilized the full bandwidth of the bottleneck link. However, regardless of the fact that we would expect every TCP to utilize the full bandwidth of the link when the buffer size is greater than or equal to 35 packets, the throughput of every TCP in Fig. 11(a) eventually stays fairly constant. This is due to the fact that random losses impose an upper limit on the throughput of a TCP connection, as previously discussed above. Figure 11(b) shows the multiple connection version of this experiment. The results are similar to those presented in Fig. 11(a) and can be interpreted in a similar way.

4.5 Throughput evaluation using an exclusively wireless topology

We also conducted similar tests to those mentioned above using a different network topology which reflects a more

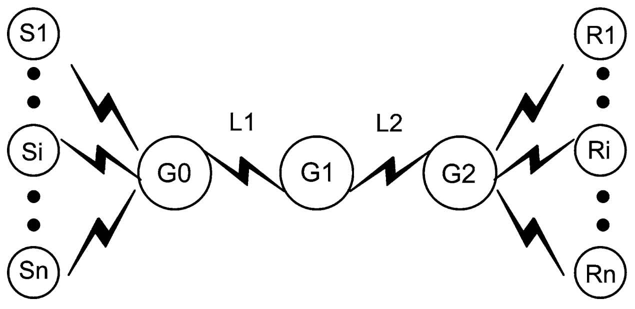


Fig. 12 Network topology for fully wireless simulations

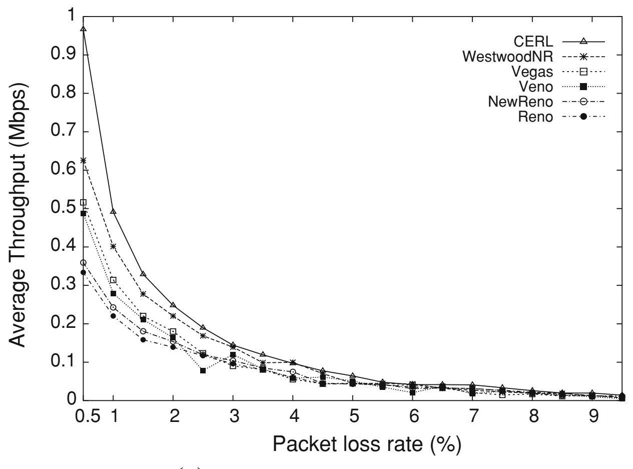
random wireless environment. In these tests we introduced an additional bottleneck router and resultant bottleneck link between routers G_0, G_1, and G_2 and made every single connection wireless. All of the simulations and results presented in this section used this new wireless topology.

Figure 12 shows the network topology used in these simulations. In Fig. 12, S_1 to S_n are TCP senders and R_1 to R_n are TCP receivers. We use the notation S_iG_0 to denote the wireless link between the ith sender and router G_0 and G_1R_i to denote the wireless link between the router G_1 and the ith receiver. The routers G_0, G_1 and G_2 perform first-in-first-out drop-tail queueing. The router buffers are all set to a constant value of 90 packets unless otherwise noted. We denote the bandwidth, propagation delay of links $L1$ and $L2$, and the random segment loss rate of all connections by B, T_p, and q, respectively. In the following sections, if the values of these variables are not mentioned, assume that $B = 2.0$ Mbps, $T_p = 50$ ms, and $q = 1\%$. We measure the throughput received by the TCP receiver in all experiments where throughput measurements are made.

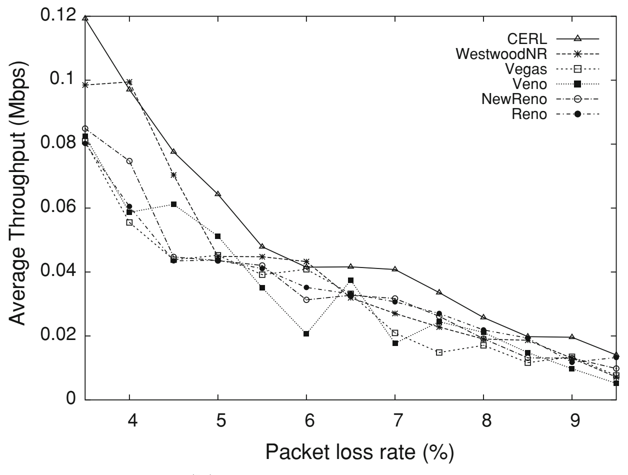
Figure 13(a) shows the results of throughput tests using this topology with random loss values ranging from 0.5 to 10%. Even in this wireless only topology with multiple bottleneck links, CERL continues to outperform all other protocols. Figure 13(b) shows the results of this same test with the graph zoomed in on the area from 3 to 10% loss. It is more apparent from this zoomed in view of the graph that the throughput of CERL stays higher than other protocols for almost the entire range of loss values. It's also obvious that with a high amount of loss between each connection, the difference in throughput between the various protocols decreases compared to the results we obtained using the topology in the previous section.

The remaining simulations conducted using this topology all use a constant loss rate of 1% as this amount of loss is typical in modern wireless networks as previously referenced.

Figure 14 is a graph comparing the various protocols with varying buffer sizes in the bottleneck links. On this graph, CERL clearly outperforms all of the other protocols.



(a) Single connection, $B = 2.0$ Mbps.



(b) Zoomed in view of graph (a)

Fig. 13 Average throughput versus packet loss rate when $T_p = 50$ ms

Figure 15 shows the throughput of the various TCPs with varying bandwidth amounts in the bottleneck links. Under these conditions CERL is superior for the range of values of bandwidths.

Figure 16 compares the throughput of the protocols using varying delay times between the bottleneck routers. With lower delays, CERL is the clear winner, although the distinction decreases as the delay increases. From this experiment we can see that network communications would effectively be crippled on any network with large amounts of loss and excessive delays, regardless of the transfer protocol used.

4.6 Fairness

In this section, we verify the fairness of CERL when it competes with Reno. In all experiments, we assume $n = 2$ connections. Two ftp transfers, which begin at the same time, occur over these connections for a duration of

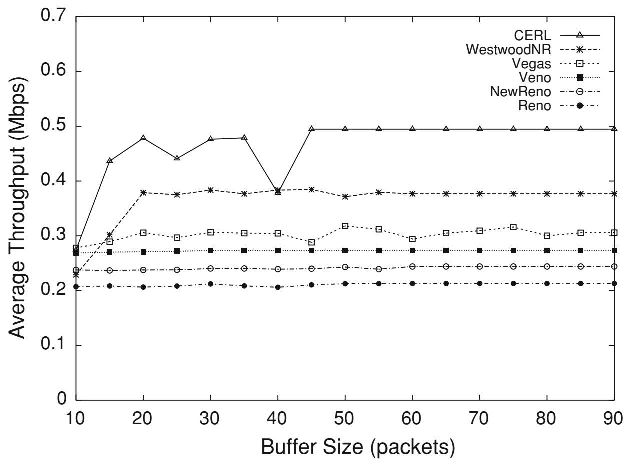


Fig. 14 Average throughput versus buffer size when $T_p = 50$ ms, and $q = 1\%$

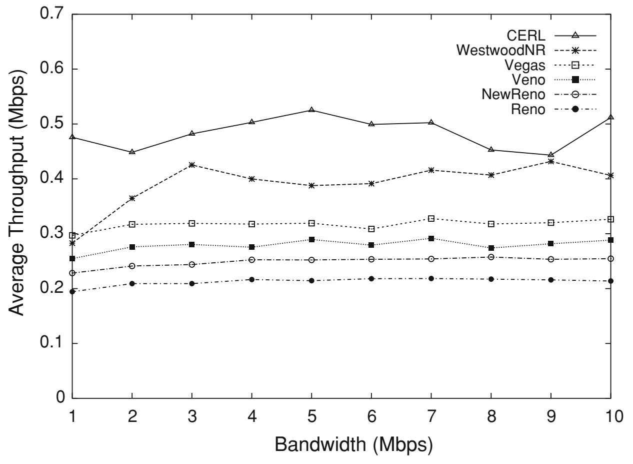


Fig. 15 Average throughput versus bottleneck bandwidth when $T_p = 50$ ms, and $q = 1\%$

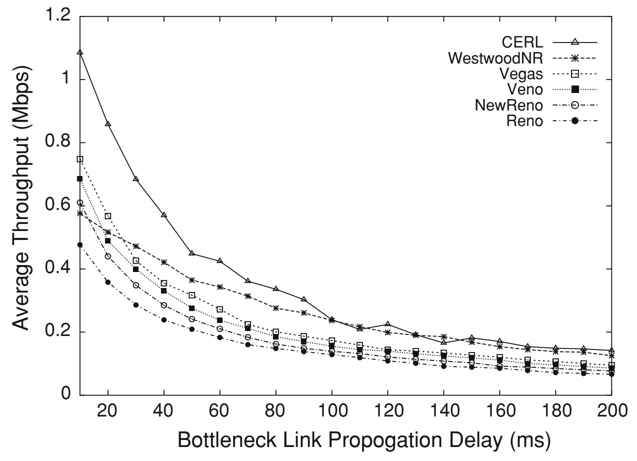


Fig. 16 Average throughput versus bottleneck link propagation delay when $q = 1\%$

120 seconds. As well, the bandwidth and propagation delay of links S_1G_0, S_2G_0, G_1R_1 and G_1R_2 are set to 10 Mbps and 10 ms, respectively.

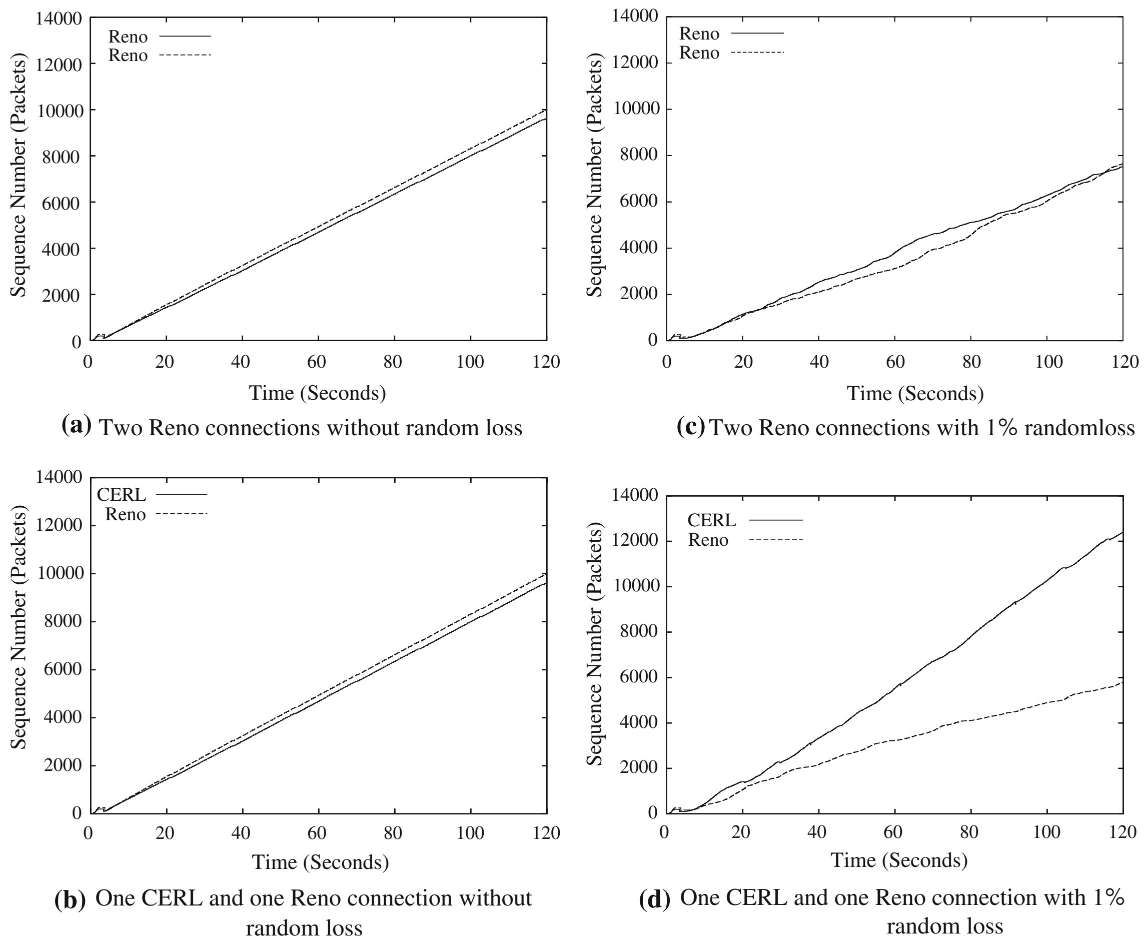
Figure 17(a) shows the segment sequence number P evolution of two Reno connections without random loss, and Fig. 17(b) shows the P evolution of one Reno connection and one CERL connection, both without random loss. By comparing these two figures, we see that the behavior of CERL is the same as that of Reno. Without random loss, both CERL and Reno treat segment losses as congestion signals and react to them in a similar way.

Figure 17(c) shows the P evolution of two Reno connections when $q = 1\%$, and Fig. 17(d) shows P evolution of one Reno connection and one CERL connection, both with $q = 1\%$. Figure 17(d) shows that CERL utilizes not only its fair share, i.e., half of the link bandwidth, but also the bandwidth left by the coexisting Reno. At the same time, the throughput of the coexisting Reno is slightly decreased, but it is still close to that of Reno in Fig. 17(c), where CERL is absent. This result suggests that Reno doesn't suffer much from the presence of CERL. Also note that the segment sequence number progression as compared to the graphs without random loss are not as straight. This is due to the fact that the random loss is frequently causing segments to be retransmitted.

We also performed further fairness experiments with $n = 3$ connections. The parameters used are the same as the first fairness tests, except for the bottleneck bandwidth which is increased to 3 Mbps to accommodate the extra connection.

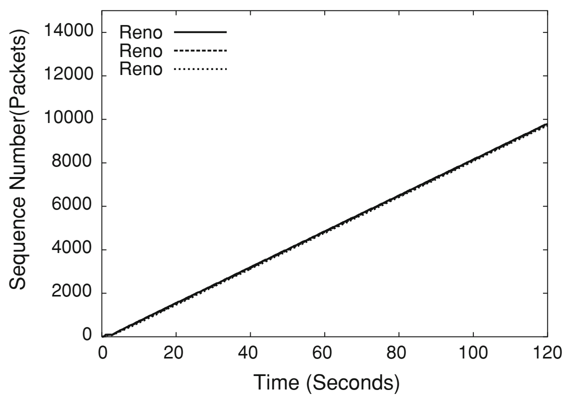
Figure 18(a) shows the P evolution of three Reno connections with no random loss, Fig. 18(b) shows the results obtained with one CERL and two Reno connections when there is no random loss, and Fig. 18(c) shows the P evolution with two CERL and one Reno connections when there is no random loss.

Figure 18(d) shows the results obtained from three Reno connections with 1% random loss, Fig. 18(e) shows the results obtained with one CERL and two Reno connections also with 1% random loss, and Fig. 18(f) shows the results obtained with two CERL and one Reno connections also with 1% random loss.

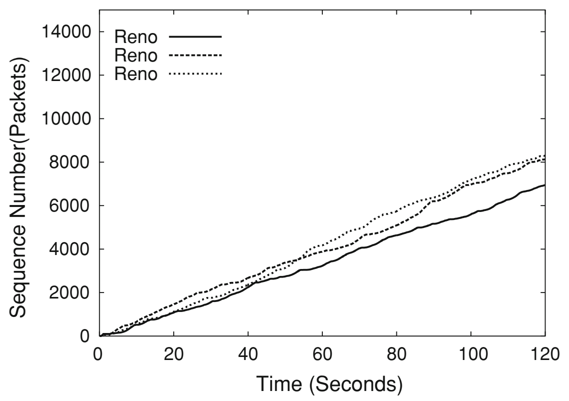


(a) Two Reno connections without random loss
(c) Two Reno connections with 1% randomloss
(b) One CERL and one Reno connection without random loss
(d) One CERL and one Reno connection with 1% random loss

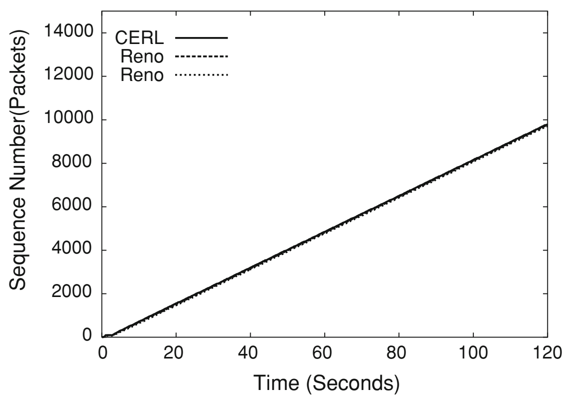
Fig. 17 Packet sequence number dynamics of two concurrent TCP flows sharing one bottleneck link



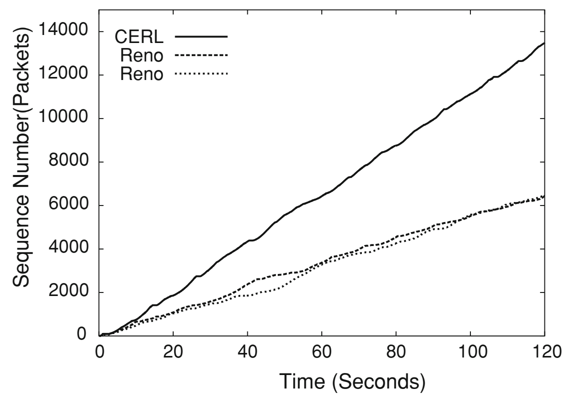
(a) Three Reno connections without random loss



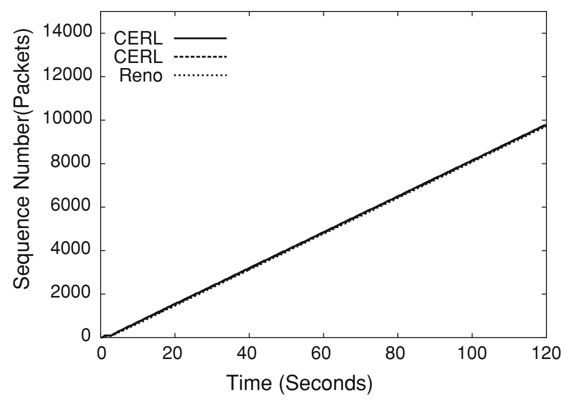
(d) Three Reno connections with 1% random loss



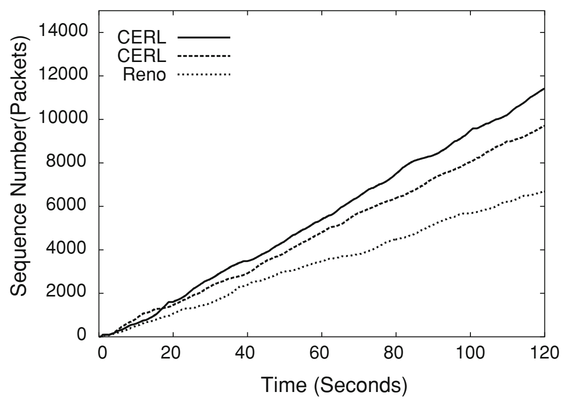
(b) One CERL and Two Reno connections without random loss



(e) One CERL and Two Reno connections with 1% random loss



(c) Two CERL and One Reno connections without random loss



(f) Two CERL and One Reno connections with 1% random loss

Fig. 18 Packet sequence number dynamics of three concurrent TCP flows sharing one bottleneck link

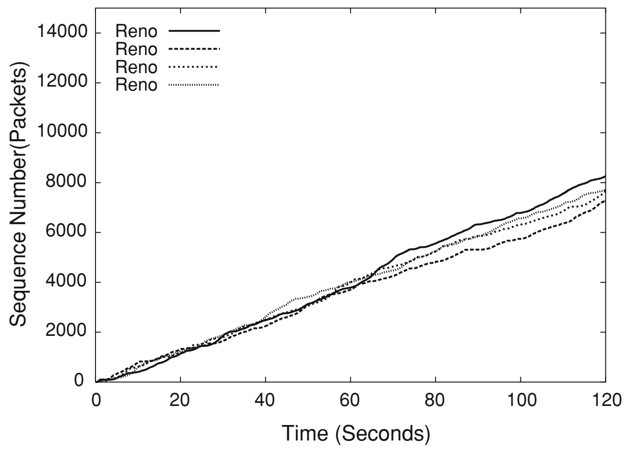
In all cases, the behaviour of CERL was very similar to when $n = 2$ connections. When no loss was present the behaviour of CERL was virtually identical to that of Reno. With 1% loss, the CERL connections used some of the available bandwidth left by the Reno connections while the Reno connections themselves degraded only slightly.

When all of the connections are using Reno, there is a small amount of competition between the connections, but the P evolution is virtually the same for each connection. Similarly, when there are two CERL connections and one Reno connection, the CERL connections compete slightly as do the two Reno connections, but the two CERL connections display a superior segment sequence evolution to the Reno connection.

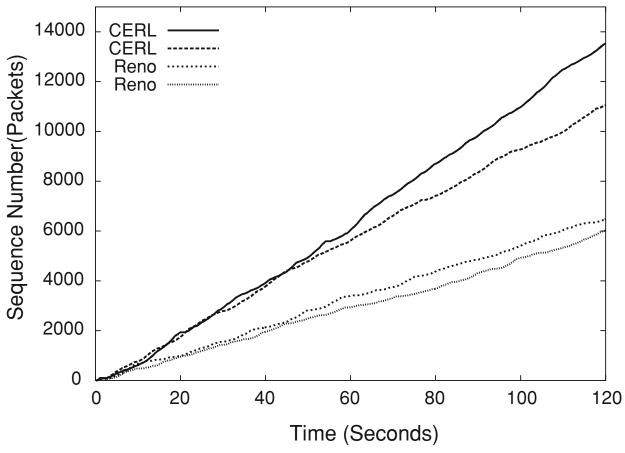
We conducted even further experiments in fairness comparing the P evolution of $n = 4$ connections. All of the network parameters stayed identical to the previous experiments except for the bottleneck bandwidth which we again increased to 4 Mbps to accommodate the extra connection.

Figure 19(a) shows the results with four Reno connections and 1% random loss. Figure 19(b) shows the results with two CERL connections and two Reno connections using 1% random loss. Here again, there was a small amount of competition between the connections, but the two CERL connections still display a superior segment sequence evolution to either of the two Reno connections.

We conclude that in networks where random loss is not present, CERL is completely compatible with Reno. In networks where random loss is present, CERL tends to utilize the available bandwidth that is left by Reno due to performance degradation, and the throughput of Reno will be slightly decreased. However, note that this occurrence is not unique to CERL. WestwoodNR, Veno, and Reno all exhibit decreased fairness when random loss is present, as shown in [8], [9], and [30], respectively.



(a) Four Reno connections with 1% random loss



(b) Two CERL and Two Reno connections with 1% random loss

Fig. 19 Packet sequence number dynamics of four concurrent TCP flows sharing one bottleneck link

4.7 Selecting a suitable value of A

Ideally, if the value of l were always accurate, we could set A equal to 1. However, CERL can only measure the RTT of one segment at a time and this RTT measurement may not always accurately represent the queue length due to the amount time it takes for an RTT estimate to arrive. Therefore, N should be set to some value less than l_{max} to decrease the chance of the congestion window not being reduced when it should have been. Even though this strategy may not prevent every unnecessary window drop, the behavior of CERL will simply be identical to that of Reno in this situation. Moreover, as long as this value is low enough, CERL will not have an unfair advantage over Reno in the event that the congestive status of the network is incorrectly estimated to be random. In fact, wrongly estimating the congestive status to be random could actually decrease the throughput of a CERL TCP.

In order to maximize protocol robustness, a set of design guidelines are proposed in [21]. The main principle behind all guidelines proposed is that a protocol should be “designed defensively”. Hence, even though we could set A higher to increase the performance of CERL in the presence of random loss, it is much safer to set it to a lower value that is known to be safe.

With this in mind, the value of A should be set such that the protocol’s throughput will be maximized, while its interference with other connections will simultaneously be minimized. We chose the value of A based on numerical simulation as has been done with Veno’s mechanism (specifically the experimental assignment of $N = 3$).

Figure 20 illustrates the effect of incremental values of A on the throughput of a CERL connection with 1% random loss. The throughput expands quickly from $A = 0.0$ to $A = 0.20$ and then levels off at $A = 0.50$. For maximum throughput a suitable value of A should therefore be between 0.50 and 1.00.

Figure 21 illustrates the effect of incremental values of A on the fairness of a CERL connection transmitting data simultaneously with a Reno connection. The graph of CERL has been removed for clarity. As the value of A increases, the throughput of the simultaneous Reno connection decreases, although certainly not in a linear or predictable way. In fact, at the value of $A = 0.55$ the throughput of the Reno connection is highest compared to other values of A in this range. We therefore set $A = 0.55$ in order to maximize the throughput of CERL while at the same time maximizing the fairness of CERL when co-existing with a Reno connection. Certainly there are other values that could be chosen, for example $A = 0.20$ which

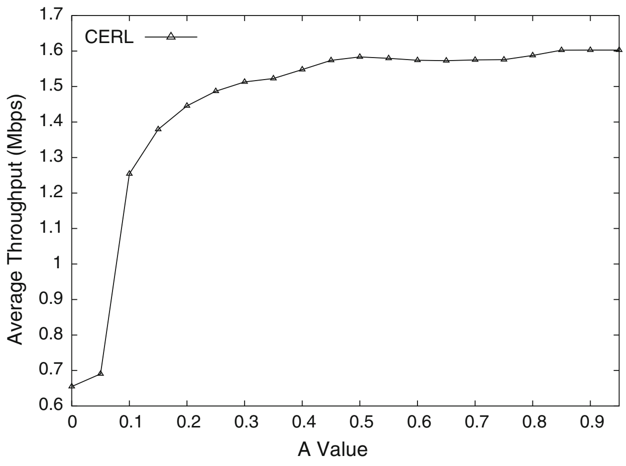


Fig. 20 Average throughput versus value of *A*

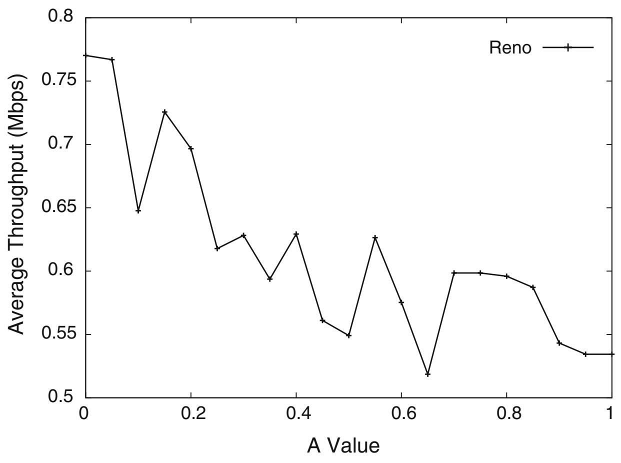


Fig. 21 Throughput of shared Reno connection versus value of *A*

from analyzing the two graphs we can see would increase the fairness of CERL but also decrease the throughput. Keep in mind that our choice of $A = 0.55$ reflected our desire to achieve maximum throughput however the design of CERL is such that with one simple change, one can choose less throughput in order to gain more fairness.

5 Conclusion

In this paper we proposed a new version of TCP named TCP CERL, aimed at improving TCP performance under wireless networks subject to random losses. When designing CERL, we wanted to ensure that it would never perform worse than Reno. By setting our threshold function conservatively, we are very confident that this goal has been accomplished, as our simulation results illustrate. Further simulations demonstrate that CERL is capable of achieving excellent throughput gains in a wide range of network configurations. More importantly, these gains are not achieved by aggressively stealing bandwidth from coexisting Reno connections. Rather, our results indicate that CERL competes fairly with Reno when no random loss is present and has a minimal impact on coexisting Reno connections when random loss is present. Besides exhibiting fairness and substantial throughput gains, CERL can be deployed in real-world networks easily since it only requires the sender-side code of Reno to be modified. The modifications required are very simple and yet allow CERL to obtain a significant throughput improvement over Reno, NewReno, Vegas, WestwoodNR, and Veno in both single connection and more realistic, multiple connection tests. In single connection tests, CERL achieved an 175, 153, 85, 64 and 88% throughput gain over Reno, NewReno, Vegas, WestwoodNR, and Veno, respectively. In tests with multiple coexisting connections, CERL achieved an 211, 226, 123, 70 and 199% throughput improvement over Reno, NewReno, Vegas, WestwoodNR and Veno, respectively. Of course, we have not verified whether or not the throughput gains achieved by CERL will be as substantial in a non-simulated, live network. This will be the subject of future research.

Acknowledgements The author is grateful to Mr. Marshall Hahn, Mr. Jiang Wu, and Mr. Kevin Smeltzer for their support of this work. This work was supported in part by National Science and Engineering Research Council of Canada (NSERC) under Grant 250299-02.

References

1. Allman, M., Paxson, V., & Stevens, W. (1999). *TCP congestion control*, RFC 2581, April 1999.
2. Kumar, A. (1998). Comparative performance analysis of versions of TCP in a local network with a lossy link. *IEEE/ACM Transactions on Networking, 6*(4), 485–498.
3. Lakshman, T. V., & Madhow, U. (1997). The performance of TCP/IP for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transaction on Networking, 5*(3), 336–350.
4. Pentikousis, K. (2000). TCP in wired-cum-wireless environments. *IEEE Communications Surveys & Tutorials, 3*(4), 4th Quarter 2000.
5. Floyd, S., Henderson, T., & Gurtov, A. (2004). *The NewReno modification to TCP's fast recovery algorithm*, RFC 3782, April 2004.
6. Brakmo, L. S., O'Malley, S. W., & Patterson, L. L. (1994). Vegas: New techniques for congestion detection and avoidance. In *Proceedings of ACM SIGCOMM'4*, pp. 24–25, October 1994.
7. Balakrishnan, H., Padmanabhan, V. N., Seshan, S., & Katz, R. H. (1997). A comparison of mechanisms for improving TCP performance over wireless Links. *IEEE/ACM Transactions on Networking*, December 1997.
8. Casetti, C., Gerla, M., Mascolo, S., Sanadidi, M. Y., & Wang, R. (2002). TCP Westwood: End-to-end congestion control for wired/wireless networks. *Wireless Networks, 8*(5), 467–479.
9. Chi, C. L., Chengpeng, F., & Chang, L. S. (2001). Improvements achieved by SACK employing TCP Veno equilibrium-oriented mechanism over lossy networks. *EUROCON*.

10. Fu, C. P., & Liew, S. C. (2003). TCP Veno: TCP enhancement for transmission over wireless access networks. *IEEE Journal on Selected Areas in Communications, 21*(2), 216–228.
11. Pang, Q., Liew, S. C., Fu, C. P., Wang, W., & Li, V. O. K. (2003). Performance study of TCP Veno over WLAN and RED router. In *Proceedings of Globecom*.
12. Zhang, C. L., Fu, C. P., Yap, M.-T., Foh, C. H., Wong, K. K., Lau, C. T., & Lai, M. K. (2004). Dynamics comparison of TCP Veno and Reno. In *Proceedings of Globecom*.
13. Clark, D. (1988). The design philosophy of the DARPA Internet protocols. In *Proceedings of SIGCOMM'88 ACM Computer Communication Review*, pp. 106–114.
14. Reed, D. (2000, April). *The end of the end-to-end argument?* [online]. Available: http://www.reed.com/dprframeweb/dprframe.asp?section=paper&fn=endofendtoend.html
15. Kempf, J., & Austein, R. (2004). *The rise of the middle and the future of end-to-end: Reflections on the evolution of the Internet architecture*, RFC 3724, March 2004.
16. Patel, P., Wetherall, D., Lepreau, J., & Whitaker, A. (2003). TCP meets mobile code. In *Proceedings of the Ninth Workshop on Hot Topics in Operating Systems (HotOS IX)*, May 2003.
17. Mitzel, D. (2000). Overview of 2000 IAB wireless Internet working workshop. Internet Engineering Task Force (IETF), ser. Rep. RFC3002.
18. ns-2 network simulator. LBL, URL: http://www.isi.edu/nsnam/ns
19. Stevens, W. R. (1994). *TCP/IP illustrated*, vol. 1. Addison Wesley.
20. Wu, J., & El-Ocla, H. (2004). TCP congestion avoidance model with congestive loss. *IEEE International Conference on Networks*, November 2004.
21. Anderson, T., Shenker, S., Stoica, I., & Wetherall, D. (2002). Design guidelines for robust Internet protocols. *HotNets-I*, October 2002.
22. UCLA Computer Science Department (2002, March) *TCP WESTWOOD – Modules for NS-2* [online]. Available: http://www.cs.ucla.edu/NRL/hpi/tcpw/tcpw_ns2/tcp-westwood-ns2.html
23. Allman M., & Falk, A. (1999). On the effective evaluation of TCP. *ACM Computer Communications Review, 29*(5), 59–70.
24. L'Ecuyer, P. (1999). Good parameters and implementations for combined multiple recursive random number generators. *Operations Research, 47*(1), 159–164.
25. Mo, J., La, R. J., Anantharam, V., & Warland, J. J. Analysis and comparison of TCP Reno Vegas. In *Proc. INFOCOM 99, 3*, pp. 1556–1563.
26. Hengartner, U., Bolliger, J., & Gross, T. (2000). TCP Vegas revisited. In *Proc. INFOCOM 2000*, pp. 1546–1555.
27. Mathis, M., Semke, J., & Mahdavi, J. (1997). The macroscopic behavior of TCP congestion avoidance algorithm. *Computer Communications Review, 27*(3), 67–82.
28. Padhye, J., Firoiu, V., Towsley, D. F., & Kurose, J. (2000). Modeling TCP Reno performance: A simple model and its empirical validation. *IEEE/ACM Transactions on Networking, 8*(2), 133–145.
29. Altman, E., Avrachenkov, K., & Barakat, C. (2000). A stochastic model of TCP/IP with stationary random losses. *Communication Review, 30*, 231–242.
30. Vardalis, D., & Tsaoussidis, V. (2002). Efficiency/Fairness tradeoffs in networks with wireless components and transient congestion. *The Journal of Supercomputing*, 281–296.

Author Biography

Hosam El-Ocla has received the M.Sc. degree in the Electrical Engineering Department of Cairo University, Cairo, Egypt in 1996. He joined the Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan in 1997, and he has been awarded the Doctor of Engineering degree in 2001 in the same University. Dr. El-Ocla is currently an Associate Professor of Computer Science Department, Lakehead University, Canada. Some of his research interests are on computer communications, networks, and protocols. He is a member in several technical societies such as: senior member of IEEE, member of Institute of Physics (IOP), and technical committee member on telecommunications of IASTED. In addition, he is a Professional Engineer of Ontario (P.Eng.). Dr. El-Ocla is a reviewer for several journals and was invited for talks and to chair sessions of different conferences.