2008 Internatioal Symposium on Telecommunications

Improving TCP Performance Over Wireless Networks

K. W. Lien
School of Electronics and Computer Science
University of Southampton
SO17 1BJ
Email: kwlien@ctu.edu.tw

Jeff S Reeve
School of Electronics and Computer Science
University of Southampton
SO17 1BJ
Email: jsr@ecs.soton.ac.uk

Y. J. Lee
Department of Information Management
Chienkuo Technology University
Changhua City, Taiwan
Email: yjlee@ctu.edu.tw

***Abstract*—This paper proposes a novel end-to-end congestion control mechanism called TCP NewZag. NewZag is simple and effective in dealing with random packet loss, so it can improve TCP performance in the wireless network. TCP NewZag proposes two new mechanisms: (1) a new end-to-end loss differentiation algorithm (LDA); (2) the adjustments of TCP NewReno multiplicative decrease algorithm. These two mechanisms only require the adjustment of TCP sender side, while the receiver side protocol remains the same. Based on experimental measurements of single link, it is shown that NewZag achieves significant throughput improvement when the wireless random packet losses occur. Compared with NewReno in 0.5% random packet loss rate, 20% throughput improvement can be demonstrated.**

I. Introduction

Wireless communication technologies have developed rapidly and wireless Internet access has become common extremely in recent years. In the wired network, TCP governs most Internet traffic, and it is also straightforward to use it for wireless access. TCP was initially designed as an end-to-end protocol for wired networks and suffers from extra challenges in the heterogeneous network. Some solutions have been proposed to improve the performance of TCP in the heterogeneous network. According to [1], [2],these approaches can be classified into three basic groups: link layer proposals [3], [4], split-connection [5], [6], [7], [8] and end-to-end proposals [9], [10], [11], [12], [13], [14], [15], [16]. But, restrictions of these proposals have also been discussed [17]. Thus, NewZag was designed to enhance the TCP performance in wireless Internet access but keep the minimal variation of Internet.

II. TCP NewZag Mechanisms

The development of TCP NewZag refers to two proposed TCP schemes: ZigZag Scheme [18] and TCP Veno [15]. NewZag mechanisms and implementations are described separately in this sections.

A. The New Loss Differentiation Algorithm

The NewZag loss differentiation algorithm (LDA) references [18] and is designed to recognize types of packet losses. The loss type is classified as random packet loss if:

$$\begin{aligned}
&(\text{n} = 1) \ \ \&\& \ \ (RTT_i < RTT_{mean} - RTT_{dev})\\
&\text{OR} \ \ [(\text{n} = 2) \ \ \&\& \ \ (RTT_i < RTT_{mean} - RTT_{dev}/2)]\\
&\text{OR} \ \ [(\text{n} = 3) \ \ \&\& \ \ (RTT_i < RTT_{mean})]\\
&\text{OR} \ \ [(\text{n} > 3) \ \ \&\& \ \ (RTT_i < RTT_{mean} + RTT_{dev}/2)]
\end{aligned}$$

Otherwise, the loss is treated as congestion packet loss. n is the number of lost packets. The RTT_i is the round trip time of packet i. The RTT_{mean} and RTT_{dev} are calculated using the exponential weighted moving average (EWMA) , as follows:

$$\begin{aligned}
RTT_{mean} &= (1-\alpha) * RTT_{mean} + \alpha * RTT_i\\
RTT_{dev} &= (1-2\alpha) * RTT_{dev} + 2\alpha * |RTT_i - RTT_{mean}|
\end{aligned}$$

With reference to [19], we define α as presenting the type 2^Y, where Y might be any integer between -8 and -2. Emulation results show that when $Y = -4$ the experiments got better results.

Detailing the LDA scheme, RTT has a high probability around 84% greater than $RTT_{mean} - RTT_{dev}$ if it is a normalized Gaussian distributed random variable [18]. The congestive loss in the wired links could be classified by the threshold $RTT > RTT_{mean} - RTT_{dev}$ because most of the packet losses in the wired link are caused by high queueing delay and then follows the congestive queue drop. On the other hand, if there is a packet loss and $RTT < RTT_{mean} - RTT_{dev}$, there is a high probability that this packet loss is happening in the wireless links. Thus, the loss type is classified as random packet loss in the wireless links.

The NewZag LDA described above references [18], but has different improvements. The improvement is achieved by using RTT in the NewZag LDA instead of ROTT. RTT is a better solution here because of one main reason: the time synchronization problem. ROTT is a time of packet transmission delay between the sender and receiver. It is measured by taking the difference of receiver's clock and sender's timestamp. If there is a global time synchronization between the receiver and sender, it is easy to estimate ROTT. But, ROTT cannot be measured accurately because the clocks at the terminal hosts are not synchronized with each other. Even though several proposals are suggested [20], [21], the clock synchronization problem is still not suitable for measuring the ROTT [22]. Thus, using ROTT as the reference time could affect the estimation of NewZag LDA and decrease the judgement accuracy of error type. Therefore, RTT is used in NewZag LDA to replace ROTT, so the time clock is always referenced to the sender time. Problems described above are naturally solved.

B. A Modified TCP NewReno

The TCP modified strategies of NewZag are derived from [15]. TCP NewReno algorithm is modified when the loss types are distinguished by the NewZag LDA. If the loss is considered to be congestive, NewZag maintains its congestion control mechanism as standard NewReno. If the loss is classified as a random packet error, NewZag increases the TCP *cwnd* in new strategies. NewZag only slightly modifies NewReno's multiplicative decrease algorithm. Once the packet loss is detected, the throughput can be dramatically reducing. Therefore, since the reason of packet loss reasonable in NewZag, a different method is used to set the new *ssthresh*.

```
If (the error is due to random packet loss)
      set ssthresh = cwnd*(4/5)
else if (the error is due to congestion loss)
      set ssthresh = cwnd / 2
```

The main object of NewZag is to recover *cwnd* at high value when all lost packets are eliminated. If the random packet loss is detected, NewZag reset *ssthresh* to $cwnd{*}4/5$, so *cwnd* can be reset to high value.

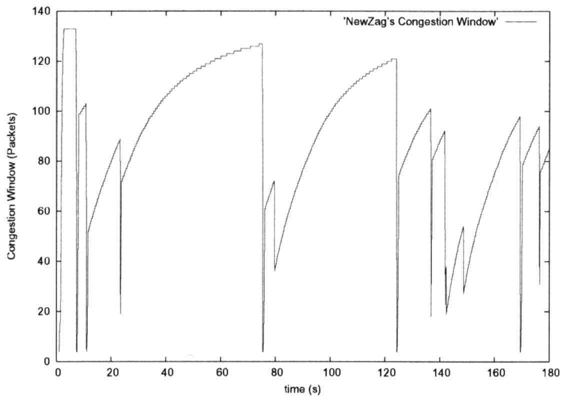

Fig. 1. NewZag's window evolution.

For illustration, Figure 1 and Figure 2 show typical window evolution of NewZag and NewReno. For the observation purpose, the setting of TCP parameters inside the Linux have been increase double. More detailed results will be provided in the next section. In this example, the average of RTTs traced by "ping" (100 packet samples) is 0.2ms in the wired links and 4.25ms in the wireless links. Showing in the NewReno simulation Figure 2, loss events happen 6 times in 14s, 36s,

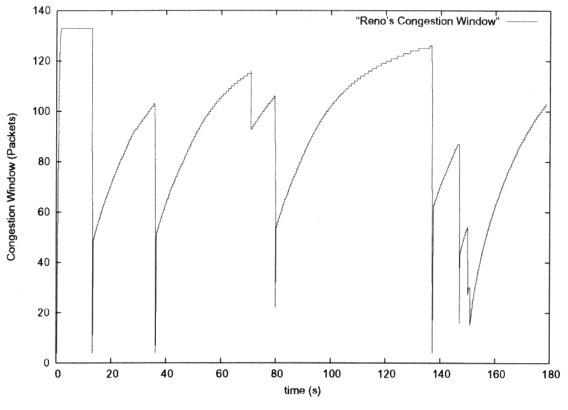

Fig. 2. NewReno's window evolution.

79s, 137s, 147s and 150s. At each loss event, NewReno decreases and resets *ssthresh* to *cwnd/2* and renews *cwnd* to the applicable value. However, in the NewZag simulation , packet losses happen 11 times in the network at around 7s, 10s, 23s, 75s, 79s, 124s, 136s, 141s, 148s, 169s and 176s. In this case, five error events are categorized by NewZag as congestive losses at 10s, 75s, 79s 141s and 148s, so the new *cwnd* has been reduced half, which is similar to NewReno. But, six error events are determined by NewZag as random packet losses at 7s, 23s, 124s, 136s 169s and 176s, so the new *cwnd* has been reduced to 4/5 of current size. Compared to NewReno decreasing *cwnd* to 1/2, NewZag reduces *cwnd* in 1/5. Thus, NewReno should take more time than NewZag to increase *cwnd* to a high value. On the contrary, if the error event comes from the wireless random error, NewZag still remains *cwnd* on the high level and eventually increases the average TCP throughput.

The TCP modified strategies of NewZag are derived from [15], but have one main amendments. The TCP adjustments in NewZag are not the same as Internet Engineering Task Force (IETF) specifications but follows the implementations in the real Linux network stack. In [15], TCP adjustments are proposed based on IETF specifications, such as RFC 2581 [23] and RFC 2988 [19]. However, some mechanisms related to IETF specifications might be simplified for easier implementation in the real network environment. For instance, Linux TCP (kernel 2.4 and 2.6) did implement RFC 2581 (Reno), RFC 2582 [23] (NewReno), RFC 2988 and RFC 3168 [9] as the TCP congestion control mechanisms, but details differ in specification. According to [24], Table I explains the differences between IETF specifications and Linux implementations on mechanisms related to TCP congestion control. Since the NewZag adjustments are based on the real Linux TCP instead of the RFC specifications, demonstration results are closer to the realistic network.

C. Integration of the NewZag LDA and SACK

The NewZag LDA can further cooperate with TCP Selective Acknowledgment (SACK) [25] to improve the evaluation of

TABLE I
TCP CONGESTION CONTROL RELATED IETF SPECIFICATIONS IMPLEMENTED IN LINUX. + = IMPLEMENTED, * = IMPLEMENTED, BUT DETAILS DIFFER FROM SPECIFICATION.

Specification	Status
RFC 1323 (Perf. Extensions)	+
RFC 2018 (SACK)	+
RFC 2581 (Congestion control)	*
RFC 2582 (NewReno)	*
RFC 2861 (Cwnd validation)	+
RFC 2883 (D-SACK)	+
RFC 2988 (RTO)	*
RFC 3168 (ECN)	*

packet loss number, because of three reasons. Firstly, the definition of packet loss numbers (*n*) was not described clearly in [18]. Secondly, the currently used TCP mechanisms (NewReno) on the Linux Internet do not provide explicit loss information to the TCP sender [24]. Thirdly, the LDA must deal with some "fake" loss packets. For instance, packet reordering is often a problem for the TCP sender because it cannot distinguish whether the missing ACKs are caused by a packet loss or by a delayed packet that will arrive later. Thus, the NewZag LDA must speculate which packets are the "real" loss in the network and accurately calculate the packet loss number. If the feedback information on packet loss is not detailed, the accuracy of the proposed LDA might decrease. In Linux, TCP must determine which to cause the packet loss from the four possibilities: 1). The sender receives a triple ACK. 2). A timeout occurs. In this case, *cwnd* is set to 2 and *ssthresh* is set to half of *cwnd* when the packet is lost. 3). TX Queue is full. 4). SACK detects a hole. Thus, NewZag cooperates with TCP SACK in these four situations to detect the packet loss number in every error events. In contrary to NewReno, SACK provides the TCP sender with more accurate feedback on packet loss information during transmission [25]. This is because SACK implements an additional "SACK option" in the "Options field" of the TCP header. It invokes the most recently received packets and the most recently reported information on SACK packets. Thus, NewZag could have extra information to report the packet loss number in the cooperation with SACK. In the Linux operating system, integration of the NewZag and SACK is simple and can easily detect packet loss situations. This is because the SACK option has been implemented in many TCP protocol stacks, including Linux kernel version 2.2 and later. NewZag adjusts only small parts of NewReno and can cooperate with SACK without any extra effort.

D. Nanosecond Implementation into Linux TCP

The system time tick rate of Linux 2.4 kernel uses a 100Hz clock (10ms). It affects the evaluation accuracy since *RTT*, RTT_{mean} and RTT_{dev} are used as important factors inside the NewZag LDA. Because the clock graduation of Linux 2.4 kernel is too large, NewZag LDA might misjudge error types when NewZag classifies the TCP packet loss events in some specific situation. Thus, NewZag modifies time functions of Linux 2.4 kernel and implements 1000Hz clock (1ms) for high accuracy emulation results.

III. Performance Evaluation

The performance evaluation of TCP NewZag is presented in this section. It is demonstrated that NewZag achieves significant throughput improvements over NewReno.

A. Experimental Network Setup

Figure 3 shows the experimental network topology. The server 1 (S1) operates with Linux kernel 2.4 in which NewZag, NewReno are set-up. The server 2 (S2) operates with Linux kernel 2.6 in which TCP NewReno are integrated. The SACK function at S1 and S2 are enabled. The wireless access point (AP) is ASUS-WL500g-Deluxe. The mobile host is a laptop, modelled Dell-D510. MH operates with Linux kernel 2.6 and TCP NewReno is used as default TCP receiver. MH is connected to the router through AP. The queue size of each network interface "txqueuelen" is set to 1000. The IEEE 802.11b standard is implemented for the wireless links. The router is implemented for the use of netem [26] in which delay of wide area networks can be emulated. The DropTail queueing police is used in the intermediate nodes. The wired links RTT between servers and AP is set to 50ms. The wireless link RTT between AP and MH is around 2.85ms. (averaged 50 times traced by "ping"). "ftp" connection is established between the simulated host and the laptop. Thus, the TCP behaviour can be observed through the ftp data transmission.

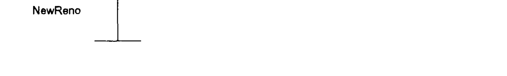

Fig. 3. Experimental Network Topology.

B. Packet Loss from the Wireless Links

In this simulation model, we try to control packet loss only happening in the wireless part and follows the setting packet loss rate. It is a real challenge to control packet loss event in the real network environment, especially in the specific links. Since the topology is constructed on the live network, it is still possible that small amounts of natural lost packet appear during the simulation, so simulation results are averaged in 10 simulations.

Two steps are used to control packet loss appearing only in the wireless links. The single-TCP connection is set-up between MH (receiver) and server 1 (sender). Firstly, the queueing sizes of network nodes are adjusted to fit the current topology bandwidth and the sender and receiver window sizes keep the same setting of Linux kernel 2.4. During transmission, *cwnd* is stable and finally hold at 69 until the end of transmission, so no packet loss at both wired and wireless links is confirmed. Secondly, different packet loss rates: 0%, 0.1%, 0.5%, 1% and 1.5% are used at MH to simulate the operation of NewZag and NewReno. Error model is used and changed the TCP checksum of the packet at the MH if the error generator determines that packet should be dropped at the receiver.

Figure 4 shows throughput difference between NewZag and NewReno under different packet loss rates, results are averaged from 10 times simulations. NewReno performs very similar results at Linux kernel 2.6 and kernel 2.4 of the same experiments with different loss rate. The simulation of NewReno at kernel 2.6 is for the contrast. This is because NewZag only adjusts NewReno mechanisms at Linux kernel 2.4 at this monent, the similar result of NewReno simulation at kernel 2.4 and 2.6 means that NewZag can also perform similar output if NewZag is future implemented into kernel 2.6.

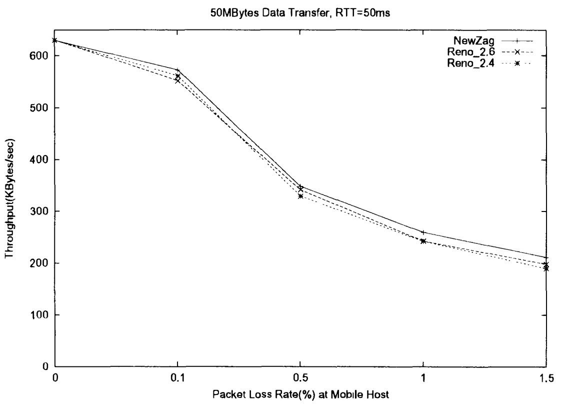

Fig. 4. Simulation of NewZag and NewReno in different packet loss rates on the MH. Reno_2.4 and Reno_2.6 is the NewReno output results from Linux kernel 2.4 and 2.6, separately.

Figure 4 shows that NewZag can perform better performance than NewReno when different loss rates are set at MH. But, the performance improvement of average TCP throughput can only achieve 4% to 7%, because the NewZag LDA cannot accurately determinate the loss type when the loss packets are detected. Table II provides some important data statistics. From this table, two important facts can be found. The first observation is about the success rate of NewZag LDA. When the random loss rate is set as 0.1% at MH, the TCP sender detects 33 loss events. In each loss event, NewZag detects only one lost packet (*n=1*). Thus, if $RTT_i < (RTT_{mean} - RTT_{dev})$, the loss event should be classified as random packet loss, as defined in NewZag LDA. But, NewZag LDA classifies 8 error events as random packet loss. It means that NewZag can only achieve around 24% success rate because the random loss rate are only set at MH in this experiment. The second observation is that there are totally 18 error events happening while $RTT < RTT_{mean}$, around 50%. However, only 8 error events are classified as random packet losses, fitting the condition $RTT_i < (RTT_{mean} - RTT_{dev})$. The rest of 10 (18-8) error events are in the group of $RTT_i > (RTT_{mean} - RTT_{dev})$, defining as congestive errors by NewZag. The same phenomenons happen at error rates 0.5%, 1% and 1.5%. The use of RTT and (RTT_{mean}) can provide current network conditions. If RTT is larger than (RTT_{mean}), it has high probability that the packet loss comes from the congestive network (by queueing drop). On the contrary, if RTT is smaller than (RTT_{mean}), it has high probability that the packet loss is caused by the random packet loss. But, NewZag LDA only considers the relationship of RTT and RTT_{mean} if and only if the packet loss number is equal to 3 (*n=3*) in the earlier section. However, we found that *n=3* rarely appears if there is only the wireless random error, as presented in Table II.

TABLE II
THE OBSERVATION OF NEWZAG LDA IN DIFFERENT LOSS RATES AT MH. THE RESULTS ARE AVERAGED FROM 10 SIMULATIONS.(*n*:NUMBERS OF PACKETS LOSSES)

Loss rate at MH	0.1%	0.5%	1%	1.5%
Total error events	33	180	342	487
n=1	33	175	321	439
n=2	0	4	20	46
n=3	0	1	1	2
NewZag determines error coming from random error	8	31	69	88
NewZag success rate	24%	17%	20%	18%
$RTT < RTT_{mean}$	18	80	140	200
$(RTT < RTT_{mean})/$ Total error events	55%	44%	41%	41%

From two observations above, the NewZag LDA has been slightly adjusted. The loss type is classified as random packet loss if:

$$[(n = 1,2,3) \quad \&\& \quad (RTT_i < RTT_{mean})]$$
$$\text{OR} \quad [(n > 3) \quad \&\& \quad (RTT_i < RTT_{mean} + RTT_{dev}/2)]$$

Figure 5 shows the difference between NewZag and NewReno under different packet loss rates after adjusting the methods. Other parameters are the same as the above experiment setting. It is shown that throughput of NewZag is consistently higher than that of NewReno for a range of random loss probabilities. In particular, at loss rate of 1%, the throughput of NewZag (282.67KB/s) is 17% higher than that of NewReno(234.08KB/s). The remarkable results comes from NewZag's multiplicative decrease algorithm that performs better window adjustment based on the loss differentiation algorithm. Under low loss environments (random loss rate close to 0%), NewZag only performs around 2% higher throughput than NewReno because NewZag operates almost the same as NewReno during the transmission since random packet loss is not a significant element. Thus, the modified

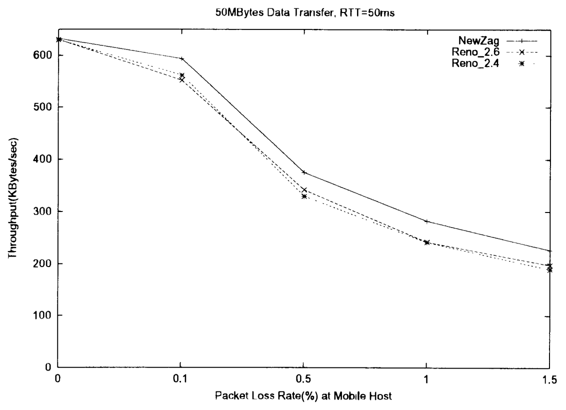

Fig. 5. Simulation of adjusted NewZag LDA and NewReno in different packet loss rates on the MH.

NewZag LDA can perform better performance to determine between congestion loss and random packet loss. We will further discuss modified NewZag with some TCP end-to-end proposals in the future. Even though NewZag might not be able to recognize the random packet loss from all error events, it can still have the same performance as NewReno since NewZag keeps most of NewReno's algorithm.

IV. Conclusion

Random packet loss in the wireless networks could lead to performance degradation in an end-to-end TCP connection. A novel TCP version, called TCP NewZag, is proposed to determine random packet loss from the congestion loss. Numerous demonstrations have been conducted in experimental networks. The experimental results show that TCP NewZag can achieve significant improvement compared with TCP NewReno.

Although the idea of TCP NewZag references several previous proposals with some adjustments, it was not a straightforward task to implement NewZag in the Linux network environment due to the gap between theory and live network. NewZag re-designs clock unit inside the kernel to increase the estimate of three RTT relative variables. From numerous experimental results, it is found that the cooperation of NewZag and SACK can precise feedback the packet loss number to the TCP sender. Even though the NewZag implement focus on the Linux kernel 2.4, we believe that NewZag can still perform well in Linux kernl 2.6 from experimental results.

References

[1] M. Taferner and E. Bonek, *Wireless Internet Access Over GSM and UMTS*. Springer-Verlag Berlin and Heidelberg GmbH & Co. K, Oct. 2001.

[2] H. Elaarag, "Improving TCP Performance over Mobile Networks," *ACM Computing Surveys (CSUR)*, vol. 34, no. 3, pp. 357–374, Sep. 2002.

[3] S. Nanda, R. Ejzak, and B.T.Doshi, "A Retransmission Scheme for Circuit-Mode Data on Wireless Links," *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 8, pp. 1338 - 1352, Oct. 1994.

[4] F. Fitzek, B. Rathke, M. Schlger, and A. Wolisz, "Simultaneous MAC-Packet Transmission in Integrated Broadband Mobile System for TCP," in *ACTS SUMMIT 1998*, Rhodos, Greece, Jun. 1998, pp. 580–586.

[5] R. Yavakar and N. Bhagawat, "Improving end-to-end Performance of TCP over Mobile Internetworks," in *Proc. IEEE MCSA'94*, CA, USA, Dec. 1994, pp. 146–152.

[6] A. Bakre and B. Badrinath, "I-TCP: Indirect TCP for Mobile Hosts," in *Proc. of the 15th Intl. Conference on Distributed Computing Systems*, Vancouver, Canada, May 1995, pp. 136–143.

[7] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP Performance over Wireless Networks," in *Proc. of the First Annual International Conference on Mobile Computing and Networking*, California, USA, Dec. 1995, pp. 2–11.

[8] B. S. Bakshi, P. Krishna, N. H. Vaidya, and D. K. Pradhan, "Improving Performance of TCP over Wireless Networks," in *Proc. of the 17th International Conference on Distributed Computing Systems (ICDCS '97)*, Washington, USA, May 1997, pp. 365–373.

[9] K. Ramakrishnan, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, Sep. 2001.

[10] S. Biaz and N. H. Vaidya, "Discriminating Congestion Losses from Wireless Losses using Inter-Arrival Times at the Receiver," in *IEEE Symposium on Application - Specific Systems and Software Engineering and Technology*, TX, USA, Mar. 1999, pp. 10–17.

[11] Y. T. Tobe, Y. Molano, and H. A. Ghosh, S. Tokuda, "Achieving Moderate Fairness for UDP Flows by Path-statusclassification," in *Local Computer Networks, 2000. LCN 2000. Proceedings. 25th Annual IEEE Conference on*, Tampa, FL, USA, Nov. 2000, pp. 252–261.

[12] F. Peng, S. Cheng, and J. Ma, "An Effective Way to Improve TCP Performance in Wireless/mobile Networks," in *EUROCOMM 2000. Information Systems for Enhanced Public Safety and Security. IEEE/AFCEA*, Munich, Germany, May 2000, pp. 250–255.

[13] V. Tsaoussidis and H. Badr, "TCP-probing: Towards an Error Control Schema with Energy and Throughput Performance Gains," in *Proceedings of the 2000 International Conference on Network Protocols*, Osaka, Japan, Nov. 2000, pp. 12–21.

[14] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi, and R. Wang, "TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links," in *Proceedings of ACM Mobicom 2001*, Rome, Italy, 2001, pp. 16–21.

[15] C. P. Fu and S. Liew, "TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 2, pp. 216–228, Feb. 2003.

[16] K. Xu, Y. Tian, and N. Ansari, "TCP-Jersey for Wireless IP Communications," *IEEE JSAC*, vol. 22, no. 4, pp. 747–756, May 2004.

[17] G. Montenegro *et al.*, "Long Thin Networks," RFC 2757, Jan. 2000.

[18] S. Cen, P. Cosman, and G. Voelker, "End-to-end Differentiation of Congestion and Wireless Losses," *IEEE/ACM Transactions on Networking*, vol. 11, no. 5, pp. 703–717, Oct. 2003.

[19] V. Paxson and M. Allman, "Computing TCP's Retransmission Timer," RFC 2988, Nov. 2000.

[20] V. Paxson, "On Calibrating Measurements of Packet Transit Times," in *Proc. of SIGMETRICS*, Wisconsin, USA, 1998, pp. 11–21.

[21] K. Anagnostakis, M. Greenwald, and R. Ryger, "cing: Measuring Network Internal Delays Using Only Existing Infrastructure," in *Proc. IEEE INFOCOM'03*, vol. 3, San Francisco, USA, Mar. 2003, pp. 2112–2121.

[22] D. Mills, "Improved Algorithms for Synchronizing Computer Network Clocks," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 245–254, 1995.

[23] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581, Apr. 1999.

[24] P. Sarolahti and A. Kuznetsov, "Congestion Control in Linux TCP," in *Proc. of the FREENIX Track: 2002 USENIX Annual Technical Conference*, California, USA, Jun. 2002, pp. 49–62.

[25] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, Oct. 1996.

[26] "The Netem Website." [Online]. Available: http://linux-net.osdl.org/index.php/Netem