

Simulation-based Analysis of TCP Behavior over Hybrid Wireless & Wired Networks

Qiang Ni
Planete Project
INRIA
Sophia Antipolis, FR

Thierry Turletti
Planete Project
INRIA
Sophia Antipolis, FR

Wei Fu
Electronic Engineering
HUST
Wuhan, China

Abstract

This paper describes an ns-based simulation analysis of TCP Tahoe, Reno, NewReno and SACK congestion control over hybrid wireless and wired networks. We compare the throughput performance between the four TCP versions. Since RTT variation statistics are used by some researchers to distinguish the congestion losses from wireless channel losses, we compute RTT variation statistics under different kinds of simulation topologies such as high-speed wireless last hop, slow wireless backbone with both long router queue length and small queue length. The simulation results show that: (1) the average throughput of SACK and NewReno are slightly higher than those of Tahoe and Reno; (2) simple RTT variation schemes can not always predict congestion from wireless link error well. Moreover they do not work well under all kinds of simulation topologies and background load environments.

1. Introduction

TCP was designed for wired IP networks and has been working well in wired IP networks. It uses triple duplicate ACKs (acknowledgments) and TO(timeout) to detect packet losses and treats every packet loss as a congestion indication. The TCP sender drastically reduces its congestion window size under bursty error-prone wireless links, which degrades the TCP throughput heavily. Many schemes have been proposed to improve TCP performance in hybrid wireless and wired networks [1-6]. The schemes can be classified as four main categories: pure end-to-end, split-connection, explicit notification and link-layer schemes. Among them, end-to-end discrimination schemes maintain the original end-to-end semantics of TCP congestion control, which only requires the minimal modifications at the TCP sender and/or receiver, without any changes from the intermediate nodes. Thus many researchers try to design a robust loss

discrimination algorithm to distinguish the congestion losses from wireless random losses. Relative one-way trip times (ROTT)[8], packet inter-arrival times[9], RTT variation statistics[2,5,6] and hybrid schemes[7] have been used for such schemes. Biaz et al.[4,5] claimed that RTT-based congestion avoidance discrimination schemes do not work well some times. However they have only done experiments using one-bottleneck dumbbell network scenario and did not make other tests under different kinds of simulation topologies to analyze the negative reasons. In this paper, we use different ns-2 (network simulator) simulation experiment results of different TCP versions to address such a problem.

The rest of this paper is organized as follows. We first describe the basic TCP congestion control mechanism of Tahoe, Reno, NewReno and SACK in Section 2. Section 3 describes the RTT-based discrimination schemes. Simulation environments and simulation results are discussed in Section 4. Conclusions are presented in Section 5.

2. TCP congestion control

TCP is a reliable window-based flow control and congestion control protocol. The source utilizes ACKs to pace the transmission of segments and interprets triple duplicate ACKs and retransmission timeout as congestion indication. TCP sender reduces the transmission rate by shrinking its window size when it detects the congestion. Tahoe and Reno are the two most widely deployed implementations of TCP[1]. To deal with the Reno's weakness of recovering only one loss per window, NewReno and SACK are proposed accordingly.

TCP Tahoe congestion control includes slow start, congestion avoidance, and fast retransmit[14]. It also implements an RTT-based estimation of the retransmission timeout. In the fast retransmit mechanism, three duplicate ACKs with the same sequence number triggers a retransmission without

waiting for the associated timeout to occur. Then slow start is initiated for window adjustment. However slow start deteriorates the throughput performance and leads to low bandwidth utilization, especially if the error is only transient or random, not persistent. In such a case, the decrease of the congestion window is unnecessary and makes the protocol unable to fully utilize the available bandwidth of the communication channel during the subsequent phase of window re-expansion.

The avoidance of slow start in case of fast retransmit is done since TCP Reno version. Fast recovery is introduced in TCP Reno with fast retransmit. A duplicate ACK is treated as an indication of available bandwidth since a segment has been successfully delivered. In particular, receiving the threshold number of duplicate ACKs triggers fast recovery: the sender retransmits one segment, and sets the congestion threshold to half the current cwnd. Instead of entering slow start as in Tahoe, the sender increases its current cwnd by the duplicate ACK threshold number. Thereafter for as long as the sender remains in fast recovery, cwnd is increased by one for each additional duplicate ACK received. The fast recovery stage is completed when an ACK for new data is received. The sender then sets cwnd to the current congestion threshold value and resets the duplicate ACK counter. In fast recovery, cwnd is thus effectively set to half its previous value in the presence of duplicate ACKs, and grows with additive increase rather than slow start. The problem with the mechanism is that it is not optimized for multiple packet drops from a single window, and this could negatively impact performance. Especially in a hybrid wired and wireless environment, frequent two or more packet losses in the same window will degrade Reno's performance heavily.

The NewReno version of TCP eliminates Reno's wait for a retransmit timer when multiple packets are lost from a window [11]. The change concerns the sender's behavior during fast recovery when a partial ACK is received that acknowledges some but not all of the packets that were out-standing at the start of that fast recovery period. In Reno, partial ACKs take TCP out of fast recovery by deflating the usable window back to the size of the congestion window. In NewReno, partial ACKs do not take TCP out of fast recovery. Instead, partial ACKs received during fast recovery are treated as an indication that the packet immediately following the acknowledged packet in the sequence space has been lost, and should be retransmitted. Thus, when multiple packets are lost from a single window of data, NewReno can recover without a retransmission timeout, retransmitting one lost packet per round-trip time until all of the lost packets from that window have been retransmitted. NewReno remains in fast recovery until

all of the data outstanding when fast recovery was initiated has been acknowledged.

SACK TCP implementation contains a number of SACK blocks, where each SACK block reports a non-contiguous set of data that has been received and queued. SACK TCP are a conservative extension of Reno's congestion control, in that they use the same algorithms increasing and decreasing the congestion window, make minimal changes to the other congestion control algorithms[11]. Adding SACK to TCP preserves the properties Tahoe and Reno TCP of being robust in the presence out-of-order packets, and uses retransmit timeouts recovery method of last resort. The main difference between the SACK TCP implementation and the Reno TCP implementation is in the behavior when multiple packets are dropped from one window of data. As in Reno, the SACK TCP implementation enters recovery when the data sender receives TCP retransmission thresh duplicate acknowledgments. The sender retransmits a packet and cuts the congestion window half. During fast recovery, SACK maintains a variable called pipe that represents the estimated number packets outstanding in the path. The sender sends new or retransmitted data when the estimated number of packets in the path is less than the congestion window. The variable pipe is incremented by one when the sender either sends a new packet or retransmits an old packet. It is decremented by one when the sender receives a duplicate ACK packet with a SACK option reporting that new data has been received at the receiver.

3. RTT-based discrimination for congestion and wireless losses

3.1. Terminologies and notations

In this paper, we denote packet losses caused by network congestion "congestion loss", and packet losses caused by wireless channel transmission error "wireless loss".

Figure 1 illustrates some terms used in this paper. Packets numbered $0, 1, 2, \dots, N$ are respectively the packets $P_0, P_1, P_2, \dots, P_N$ sent. ACK $0, 1, 2, \dots, N$ are respectively the acknowledgements generated for packets $P_0, P_1, P_2, \dots, P_N$. We only concern cumulative acknowledgements in TCP congestion control, we do not address delayed acknowledgements in this paper. Round-trip time(RTT) is collected as the duration from the time when P_i is transmitted, until the time when first acknowledgement for P_i is received by the sender. If a TO occurs while waiting for the acknowledgement of a packet, we think it will be an indication of packet loss,

then the round-trip time for that packet is not calculated in our RTT variation statistics.

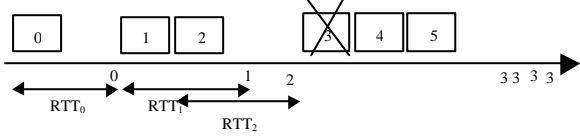


Fig. 1 Terminology and notations

In this paper we denote PER as wireless channel packet error rate and BER as wireless bit error rate. Note that BER is the probability of a single bit error in wireless channel, PL is the length of the packet (header+payload). Since the relationship between BER and PER can be formulated by $PER=1-(1-BER)^{PL}$, we will use the PER metric in this paper.

Average throughput is computed as the average number of bits successfully transmitted to the mobile host in one second.

3.2. RTT-based discrimination scheme

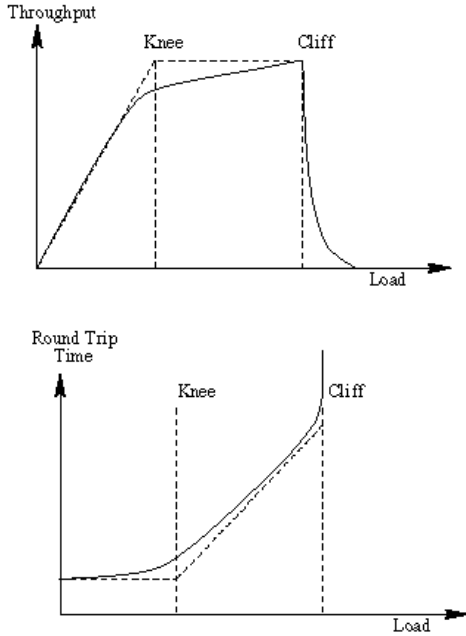


Fig. 2 Throughput and RTT versus network load[17]

Current RTT-based discrimination techniques are based on the following expectation of network behavior. As illustrated in Figure 2, when network load is small, increasing the load should result in a comparable increase in network throughput with only a small increase in round-trip times (RTT). At some point (such as knee point[17]), when the load is large enough, packets start and keep queuing at the bottleneck. Therefore, increasing the load further should result in a small increase in throughput, and a larger increase in round-trip times. After that, if the load

is increased further, the network throughput should drop sharply, round-trip times should become extremely large.

TCP-Vegas [2] requires TCP sender to keep track of the BaseRTT, defined as the minimum of all RTTs measured during the TCP connection. When acknowledgement for the i -th monitored packet is received, the sender calculates the expected throughput as $E_thr=W_i/BaseRTT$. The actual throughput is calculated as W_i/RTT_i . Then the difference D is calculated as $W_i/BaseRTT-W_i/RTT_i$ and f_{Vegas} is defined as $BaseRTT \times D$. f_{Vegas} is compared to two thresholds α and β ($\alpha < \beta$). If $f_{Vegas} > \beta$, TCP-Vegas estimates that network congestion will happen and suggests that the sender's congestion window size be decreased. If $f_{Vegas} < \alpha$, the window size should be increased.

Wang and Crowcroft proposed a congestion avoidance technique based on the Normalized Throughput Gradient (NTG)[18]. This technique evaluates the gain throughput after an increase of the window size. If the increase of throughput is larger than half the throughput observed for the first packet, then the congestion window may be increased.

Jain[17] proposed a congestion avoidance technique based on Normalized Delay Gradient (NDG). This technique looks only at the signs of variations of the round-trip time and the congestion window size. If an increase or decrease of the window results in an increase or decrease of the round-trip time, then the congestion window size is decreased. Otherwise the congestion window size should be increased.

Samaraweera[6] proposed an end-to-end Non-Congestion Packet Loss Detection (NCPLD) algorithm for a TCP connection in a network with a wireless backbone link, such as a low-bandwidth satellite link. NCPLD measures round-trip time at the sender and compares it to the measured delay when there is no congestion to decide whether a loss is a wireless or congestion loss. Samaraweera simulates the algorithm and shows that, when a connection experiences congestion, NCPLD behaves as well as TCP when the wireless error rate is low, and improves throughput over TCP when the wireless error rate is high. However, NCPLD was not evaluated on different wireless topologies.

4. Simulation

4.1. Simulation topologies

In this section, we use ns-2.1b8a to study the TCP performance over hybrid wireline and wireless networks and RTT statistics accordingly. We use four types of

simulation topologies shown in Figures 3, which we call wireless last hop (without congestion loss/or with congestion loss), slow wireless backbone with both long router queue length (all the queue sizes in the routers are set to $20pkts$) and small queue length(the queue sizes of routers of two bottleneck links are set to $3-5pkts$, others are also $20pkts$). Figure 3(a) is a simple no congestion loss wireless last hop. FH denotes the fixed host,BS denotes basestation and MH denotes the mobile host. In Fig 3(b) we introduce a cross traffic to lead to congestion at wireline bottleneck. The Fig 3(c) is a topology of wireless backbone (with both long router queue length and small queue length). In this test we use several Pareto distributed ON/OFF sources to simulate competing short TCP connections. It is reported that such a heavy-tailed Pareto distribution can model current dominant Internet WWW-like traffics [13]. In this experiment, we set the mean ON time to be 1 second, and the mean OFF time to be 2 seconds. During the ON time each source sends at 300 Kbps,160 Kbps and 200Kbps. The shape parameter of the Pareto distribution is set to be 1.5. The number of ON/OFF sources in our experiments is set to be 5. The bottleneck link has a capacity of 0.5Mb and a propagation delay of 30ms. There are three kinds of analytical error models in ns simulation environment such as uniform, two-state Markov (Gilbert) and multi-state Markov channel model. We model the wireless link loss by using both uniform random error model and two state Gilbert model, no fading channel model is used. All experiments start at time of 0 second and stop at 400 second.

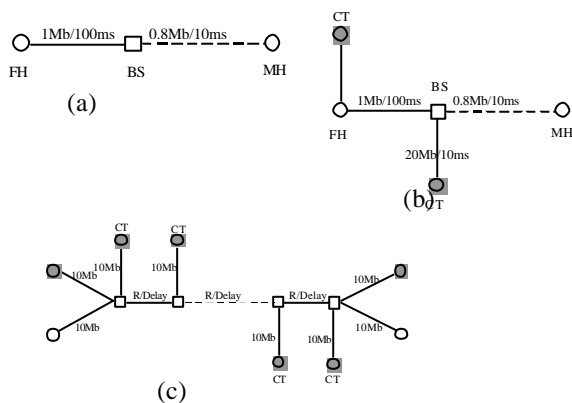
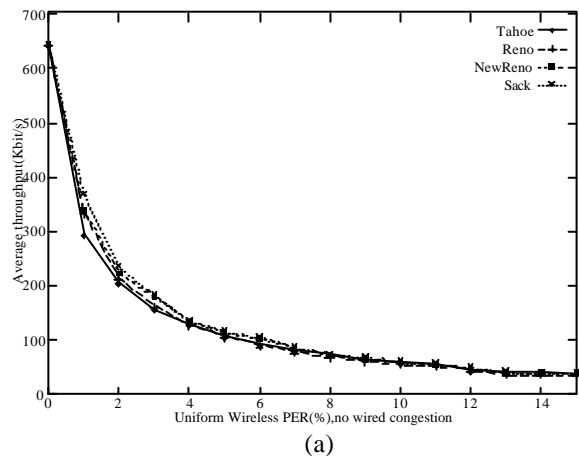


Fig. 3 Simulation Topologies

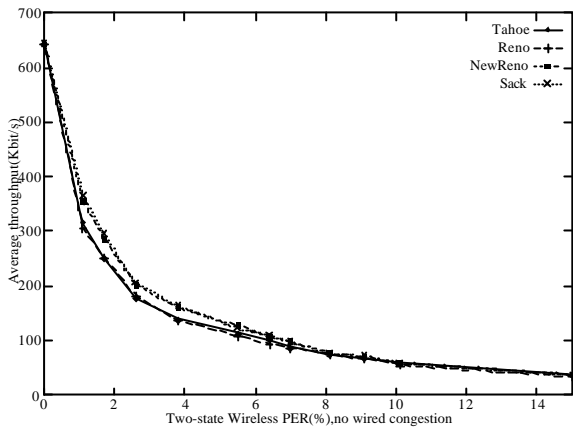
4.2. Simulation results

4.2.1. Throughput comparison. Figure 4-7 show the average throughput comparison between TCP Tahoe, Reno, NewReno and SACK over hybrid wireline and wireless networks. For each scenario, we perform 5 sets of experiments. In each set, PER is randomly generated

based on uniform error model (Fig.4(a)) or two-state gilbert error model (Fig.4(b), Fig.5-Fig.7). We compute and plot the average throughput for each graph. From the figures we notice little differences between the four TCP versions, and show that the performance observed using these two error-models are similar. TCP SACK performs best in most cases than those of Tahoe, Reno and NewReno. Even with random transient errors and no congestion, the four TCP versions unnecessarily reduce the congestion window size and extend the transmission time. TCP Tahoe and Reno perform worst when wireless PER increases to some extent. Tahoe can recover from the packet losses with a slow start. Reno sender does not wait for timeout, but instead recovers by doing a retransmit and fast recovery several times in succession, cutting the congestion window several times. This may slow down the TCP transmission considerably. NewReno's behavior is similar to Reno's until the sender receives the first partial ACK, causes NewReno to retransmit the lost packet immediately and not exit the fast recovery period. The duplicate ACK counter is reset to zero and later increased by the number of duplicate ACKs matching the partial ACK. The congestion window is not affected. SACK TCP's behavior is also similar to Reno, but the receiver can inform the sender about all segments that have arrived successfully by selective acknowledgments. The protocol initializes the $pipe=cwnd-ndup$ and then subtracts one for subsequent duplicate ACKs and adds one for transmitted packets.



(a)



(b)
Fig. 4 Only wireless loss, no Congestion, last-hop (a) uniform error model, (b) two-state model

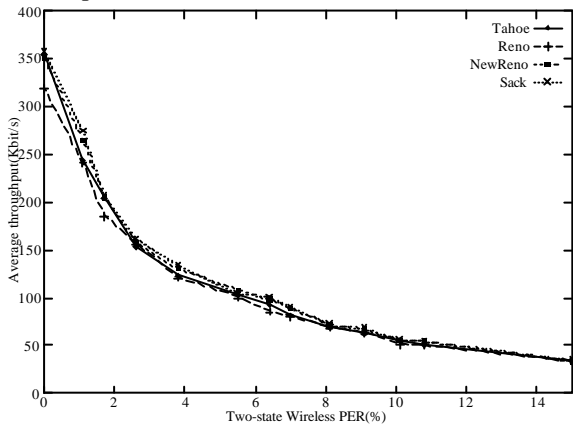


Fig. 5 Congestion & two-state wireless loss, last-hop

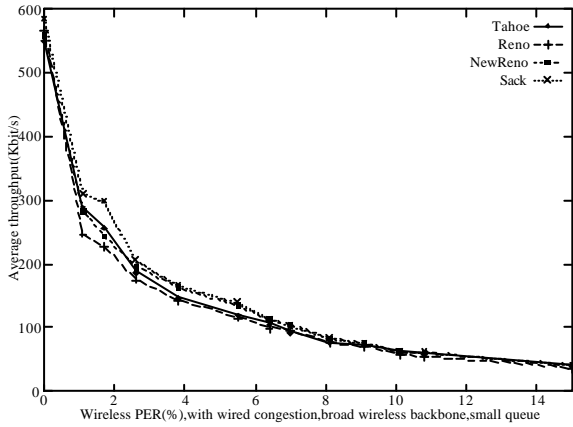


Fig. 6 Congestion & two-state wireless loss, broad wireless backbone, with small queue

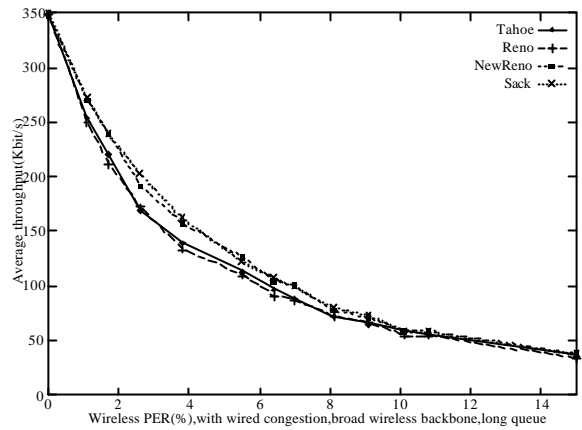


Fig. 7 Congestion & two-state wireless loss, broad wireless backbone, with long queue

4.2.2. RTT statistics. Fig.8-Fig.11 show the traces of sent, dropped and ACK packets of TCP Tahoe, Reno, NewReno and SACK over hybrid wireline and last-hop wireless networks. For each graph, the x-axis shows the packets sent, dropped or acknowledged time in seconds. The y-axis shows the packet sequence number. Packets are numbered starting with packet 0. Each packet sent is marked by a square on the graph. Packets dropped are indicated by an “x” on the graph. We use ns trace tool to record the packet loss. We do not differentiate the marked lost packet due to congestion or wireless. Returned ACK packets and received at the sender are marked by a smaller dot. The size of FTP source file is set to 100Kbytes for the four scenarios. From the figures we show that the transmission time of different TCP versions are different based on their own congestion control schemes. The transmission time of SACK and NewReno is about 6 seconds when PER=2%, which is around 1.8 times shorter than those of Reno and Tahoe versions. So the throughputs of SACK and NewReno are larger than those of Reno and Tahoe.

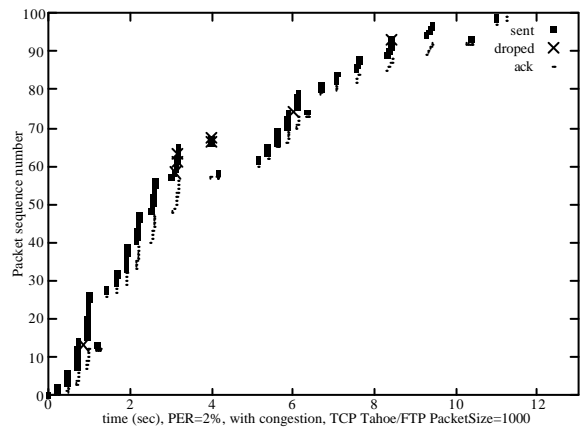


Fig.8 Congestion & wireless loss, last-hop

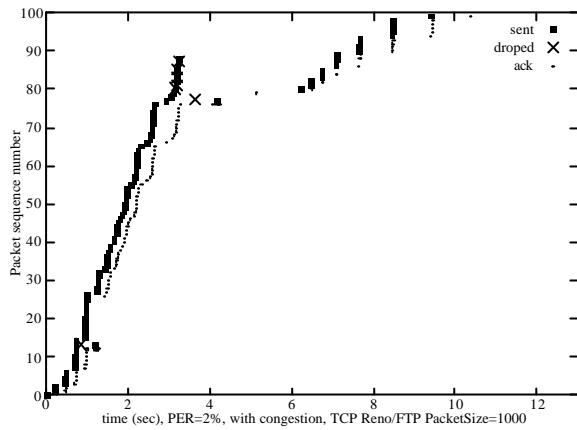


Fig.9 Congestion & wireless loss, last-hop

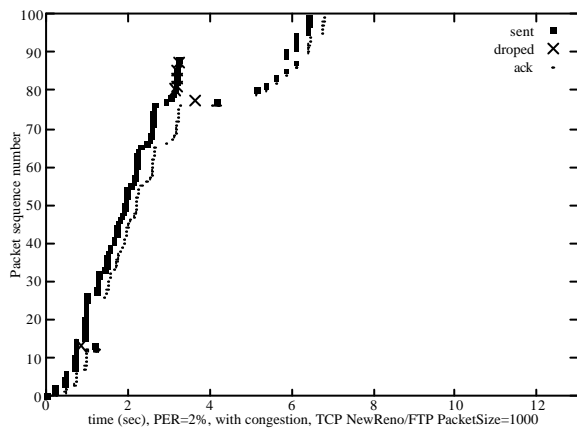


Fig.10 Congestion & wireless loss, last-hop

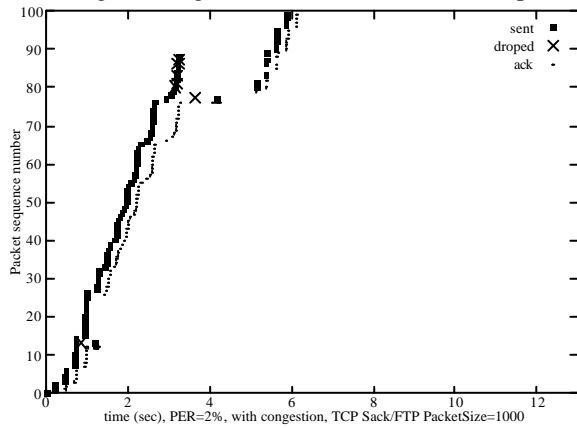


Fig.11 Congestion & wireless loss, last-hop

Fig.12-Fig.13 show RTT statistics of TCP Tahoe, Reno, NewReno and SACK with or without wired congestion. From the figures we can see that in such a simple topology, the RTT will not increase considerably with only wireless loss existing, but will increase a lot due to queue build-up and result in packet loss due to buffer overflow finally. Say, RTT variation computing can be used to distinguish the wireless packet loss from congestion loss in such conditions.

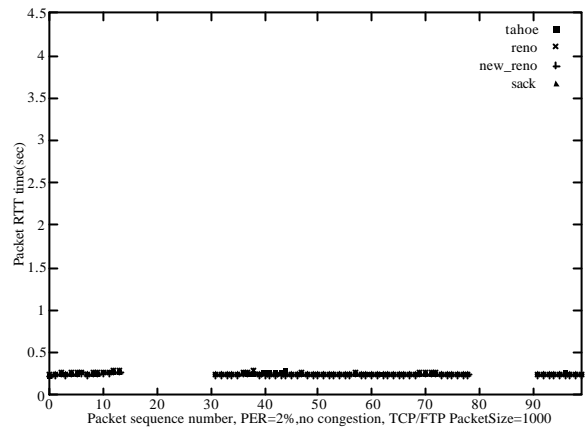


Fig.12 Only wireless loss, no Congestion, last-hop

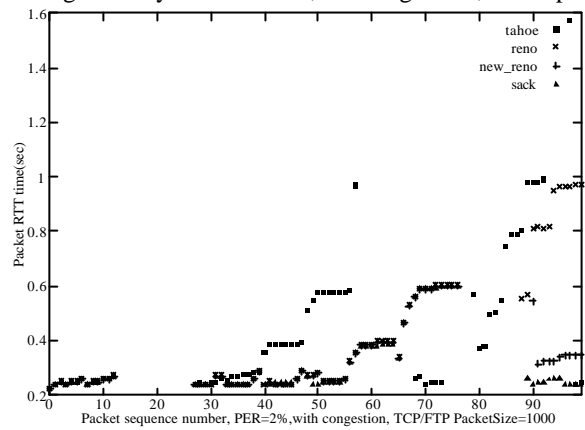


Fig.13 Congestion & wireless loss, last-hop

Fig.14-Fig.19 show the traces of data sent, dropped, acknowledged and the RTT computation of TCP Tahoe over multiple bottleneck wireless backbone. From the figures we can see that in different topologies, RTT variations are different, sometimes random. Indeed, the congestion loss due to a long intermediate queue length will cause a considerable increase in RTT and it is easy to distinguish the wireless packet loss from congestion loss(Fig.17). But a small queue length will not result in considerable increase in RTT(Fig.18) and a narrow wireless backbone will make RTT variation both increase, decrease and constant during congestion (Fig.19). In such a circumstance, it is not possible to distinguish wireless packet loss from congestion loss by RTT variation metrics. [4] has proposed that congestion losses appear to be random and not applicable everytime. Our simulation results also support this observation and give more validation results.

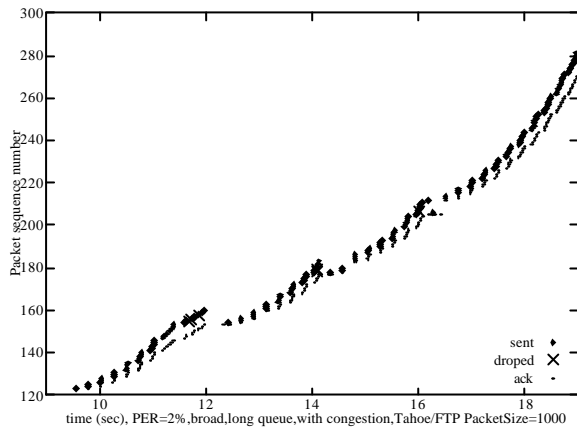


Fig.14 Congestion & wireless loss, broad wireless backbone, with long queue

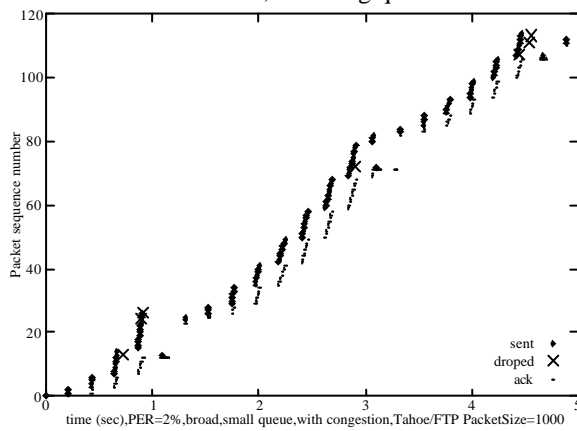


Fig.15 Congestion & wireless loss, broad wireless backbone, with small queue

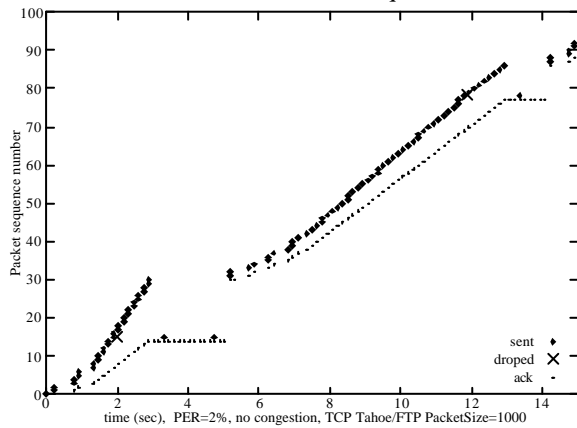


Fig.16 Congestion & wireless loss, narrow wireless backbone

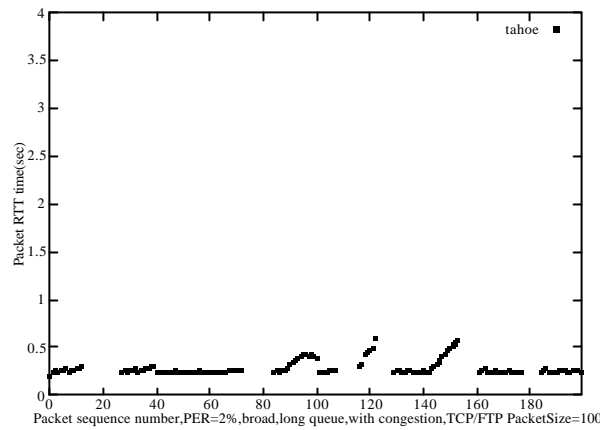


Fig.17 Congestion & wireless loss, broad wireless backbone, with long queue

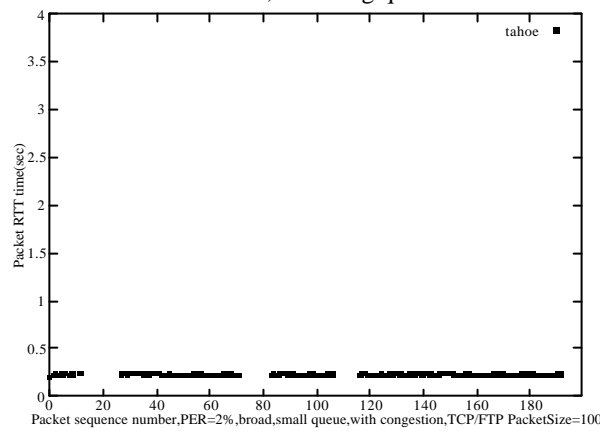


Fig.18 Congestion & wireless loss, broad wireless backbone, with small queue

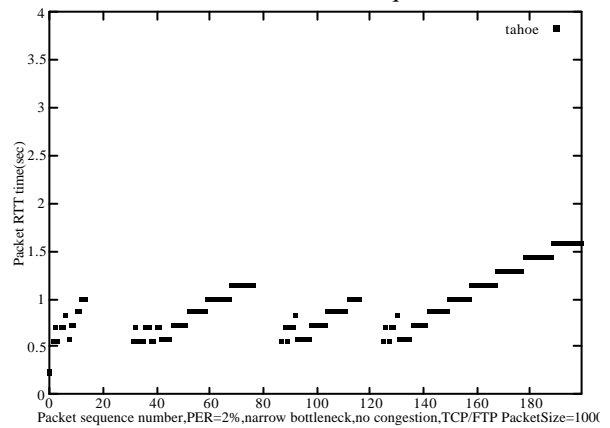


Fig.19 Congestion & wireless loss, narrow wireless backbone

5. Conclusions

Ns-based simulation analysis of TCP Tahoe, Reno, NewReno and SACK over hybrid wireless and wired networks is presented in this paper. Both throughput performance and RTT variation loss discrimination

mechanism are analyzed under different kinds of simulation topologies such as wireless last hop, wireless backbone with both long router queue length and small queue length. The simulation results show that the throughput of SACK and NewReno are slightly higher than those of Tahoe and Reno and only simple RTT variation discrimination scheme can not perform well across all kinds of simulation topologies and background competing environments. Future work will include designing a robust end-to-end discrimination scheme and selective reject ARQ scheme at wireless link layer.

Acknowledgments

The authors would like to thank the anonymous reviewers for their fruitful comments. This work has been supported by the French ministry of industry in the context of the national project RNRT-VTHD++.

References

- [1] V. Tsaoussidis and I. Matta, Open issues on TCP for mobile computing, *The Journal of Wireless Communications and Mobile Computing*, John Wiley & Sons, Issue 1, Vol. 2, February 2002.
- [2] L. Brakmo and S. O Malley, TCP-vegas: new techniques for congestion detection and avoidance. *SIGCOMM 94*, pp. 24-35, Oct. 1994.
- [3] H. Balakrishnan, V.N. Padmanabhan, S. Seshan, and R.H. Katz, A comparison of mechanisms for improving TCP performance over wireless links, *Proceedings of the ACM SIGCOMM 96*, Aug. 1996, pp. 256-269.
- [4] S. Biaz and N. H. Vaidya, Distinguishing Congestion Losses from Wireless Transmission Losses: A Negative Result, *Seventh International Conference on Computer Communications and Networks (IC3N)*, New Orleans, October 1998
- [5] S. Biaz and N. H. Vaidya, Is the round-trip time correlated with the number of packets in flight? *Technical Report 99-006*, March 1999, CS Dept., Texas A&M University
- [6] N.K.G. Samaraweera, Non-congestion packet loss detection for TCP error recovery using wireless links, *IEE proc.-commun.*, Vol.146, No. 4, August 1999.
- [7] S. Cen, P. Cosman, G.M. Voelker, End-to-end differentiation of congestion and wireless Losses, *Proceedings of ACM Multimedia Computing and Networking 2002*, San Jose, CA, Jan. 23-24, 2002. *SPIE* vol. 4673, pp. 1-15.
- [8] Y. Tobe, Y. Tamura, A. Molano, S. Ghosh, and H. Tokuda, Achieving moderate fairness for UDP flows by path-status classification, in *Proc. 25th Annual IEEE Conf. on Local Computer Networks (LCN 2000)*, pp. 252-261, Tampa, FL, Nov. 2000.
- [9] S. Biaz and N. Vaidya, Discriminating congestion losses from wireless losses using inter-arrival times at the receiver, in *Proc. 1999 IEEE Symposium on Application-Specific Systems and Software Engr. and Techn.*, pp. 10-17, Richardson, TX, Mar 1999.
- [10] A. Chockalingam, M. Zorzi, R.R. Rao, Performance of TCP Reno on Wireless Fading Links with Memory, *IEEE ICC 98*, Jun. 1998.
- [11] K. Fall and S. Floyd, Simulation based comparisons of Tahoe, Reno and SACK TCP, *Computer Communications Review*, vol. 26, no. 3, July 1996, pp. 5-21.
- [12] G.T. Nguyen, R.H. Katz, B. Noble, and M. Satyanarayanan, A trace-based approach for modeling wireless channel behavior, *Proceedings of the Winter Simulation Conference*, Dec. 1996, pp. 597-604.
- [13] K. Park, G. Kim, and M. Crovella, On the relationship between file sizes, transport protocols and self-similar network traffic. In *Proceedings of ICNP'96*, 1996
- [14] W. Stevens, TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms, *Internet Request For Comments 2001*, Jan. 1997.
- [15] V. Jacobson, Congestion avoidance and control, *Proceedings of the ACM SIGCOMM 88*, Aug. 1988, pp. 314-329.
- [16] J. Martin, A. Nilsson and I. Rhee, The incremental deployability of RTT-based congestion avoidance for high speed TCP internet connections. *ACM SIGMETRICS*, Jun. 2000.
- [17] R. Jain, A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM CCR*, 19:56-71, 1989.
- [18] Z. Wang, J. Crowcroft, A new congestion control scheme: slow start and search(tri-s), *ACM CCR*, 21:32-43, 1991.