ISSN: 0975-833X

Available online at http://www.journalcra.com

International Journal of Current Research
Vol. 16, Issue, 04, pp.28027-28026, April, 2024
DOI: https://doi.org/10.24941/ijcr.47083.04.2024

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

REVIEW ARTICLE

REVIEW: A BEHAVIORAL ANALYSIS OF TCP VARIANTS FOR MANET

***[1]Deepika Ratnawat and [2]Pankaj Raghuvanshi**

[1]Student of Computer Science, M. Tech, Alpine Institute of Technology, Ujjain (M.P.), India;
[2]Lecturer of Computer Science Department, Alpine Institute of Technology, Ujjain (M.P.), India

ARTICLE INFO

Article History:
Received 20th January, 2024
Received in revised form
19th February, 2024
Accepted 15th March, 2024
Published online 30th April, 2024

Key words:
Improved mobile ad hoc network congestion and corruption control can be achieved by implementing the suggestions provided by an analysis of these variances.

***Corresponding author:**
Deepika Ratnawat

ABSTRACT

Wireless networks are inherently distinct from wired networks in numerous ways; TCP congestion control algorithms are not directly applicable to wireless networks due to these differences (e.g., higher error rates, prolonged delays, reduced bandwidth, frequent mobility, etc.). Therefore, several improved techniques for controlling TCP congestion have been introduced. The primary objectives of those methods were to efficiently manage congestion, withstand loss with dependability, and reduce gearbox errors. Unique congestion management and avoidance techniques for the TCP/IP protocols Tahoe, Reno, New-Reno, Lite, TCP Vegas, and SACK are investigated and assessed in this study.

Citation: ***Deepika Ratnawat and Pankaj Raghuvanshi.*** **2024.** "Review: A Behavioral Analysis of TCP Variants for MANET*"*. *International Journal of Current Research*, 16, (04), xxxx-xxxxx.

INTRODUCTION

A dependable, end-to-end, connection-oriented transport layer protocol that offers byte-stream-based services is TCP (Transmission Control Protocol) (1), (2). Many Internet services, such as HTTP (Hypertext Transfer Protocol) and the World Wide Web, as well as FTP (File Transfer Protocol), rely heavily on TCP. Even if the web architecture changes in the future, TCP and its applications will most likely be used continuously. However, in terms of performance and connection fairness, the famous TCP Tahoe and Reno versions (as well as their variations) on the modern Internet should be improved. As a result, a great deal of TCP research has been conducted, and numerous enhancement strategies have been put forth. The TCP Vegas version (3,4) has exceptional performance, making it one of the most promising techniques. TCP Vegas enhances TCP Reno's congestion avoidance mechanism. TCP Vegas dynamically modifies its window size when transmitting packets based on measured RTTs (round trip times). On the other hand, TCP Tahoe/Reno keeps expanding its window size until it detects packet loss. Through modeling and implementation trials, the authors in (3) conclude that TCP Vegas can achieve throughput improvements of up to 40% over TCP Reno.

The assumption that congestion is the only factor contributing to packet loss led to the development of TCP congestion management algorithms in the beginning. However, due to several circumstances, such as inclement weather, obstacles, multipath interference, wireless endpoint mobility, signal fading and attenuation, and packet loss, bit error rates in wireless networks are higher.

TRADITIONAL TCP

Previously, TCP established a connection by permitting the sender to send the maximum number of segments that the recipient had advertised, regardless of whether the router had enough spare capacity to handle the repeated packet injections or if the sender and recipient were connected by slower links.
For two endpoints on the same network that exchange data, the previous version of TCP was appropriate. If either endpoint is on a separate network, issues may arise right away. If the router has to have extra storage in order to hold the packages that are being sent, then another problem may occur. Owing to this constraint, a congestion control method that can manage these issues needs to be developed. RFC (5) states that the slow start approach is the most suitable method for addressing this issue.

SLOW START

To prevent network congestion, the slow start algorithm regulates the datagram flow in the system. The goal is to increase the slow-start congestion window exponentially. This is the slow start algorithm from (5).

Add a congestion window to the per-connection state. Set the congestion window to one packet when starting or restarting after a loss. On each acknowledgement for a new data connection, increase the congestion window by one packet. Send only what is advertised to the recipient and within the given timeframe.

By restricting the number of unacknowledged packets that may be in transit, TCP manages the data flows and attempts to keep a congestion window open.

CONGESTION AVOIDANCE

Congestion avoidance, as described in (5), is a TCP restriction strategy that controls slow start exponential duplication to prevent overflowing the network with segments that may otherwise cause congestion. With this algorithm, TCP reduces the congestion window by half for every loss.

FAST RETRANSMIT

Retransmission and fast recovery (5) rely on the TCP version used in Reno. Segments received out of sequence may cause TCP to produce an instant duplicate acknowledgment. The purpose of this duplication is to alert the sender that the segments they transmitted were received out of order and to let them know what sequence number to expect in the subsequent transmission. TCP waits for the receipt of a duplicate Ack with a small number. If three duplicates are approved, the segment is considered lost. TCP will, in this instance, retransmit the absent segment without waiting for the timer to expire. Below, fast recovery and fast retransmission are put into practice.

FAST RECOVERY: Since a lost packet signifies potential congestion, congestion avoidance is achieved after a rapid retransmit of the absent segment. Fast recovery is the name given to this algorithm.

PERFORMANCE EVOLUTION OF TCP VARIANTS: In this section, we argue the performance of various TCP versions regarding Tahoe, Reno, New Reno, SACK, FACK and Vegas.

TCP Tahoe: Tahoe (6), (7) refers to the TCP congestion control algorithm, which Van Jacobson suggested in his paper, has some enhanced features in TCP implementation, including slow start, congestion avoidance, and fast retransmission. Tahoe recommends that whenever a TCP connection starts or re-starts after a packet loss, it should go through a 'slow start' procedure. This approach is because an initial burst might overwhelm the network, and the connection might never start. A slow start implies that the sender sets the congestion window to 1 and raises the CWD by 1 for each received ACK. Tahoe uses 'Additive Increase Multiplicative Decrease' for congestion avoidance. Tahoe stores half of the window as a threshold value, and a packet loss indicates congestion. After that, it starts slowly and raises CWD to one until it reaches the threshold value. Following that, CWD increases linearly until a packet loss occurs. The essential issue is that Tahoe detects packet losses through timeouts. TCP Tahoe's fast retransmission algorithm outperforms the most when the packets are lost due to congestion. The sender must wait for the retransmission timer to expire before implementing the fast retransmit algorithm. At the same time, fast retransmitting makes Tahoe perform significantly better than a TCP implementation. The issue with TCP Tahoe is that it detects packet loss after the whole timeout period. The speed of TCP Tahoe decreases when a packet loss is found. Transmission flow quickly drops as a result.

TCP Reno: Reno has maintained the fundamental ideas of Tahoe, including slow starts, avoidance, and fast retransmits, except it maintains improvements over Tahoe by adding to the fast recovery phase known as the rapid recovery algorithm (6). TCP Reno stimulates packet losses to estimate the available bandwidth in the network. Although there are no packet losses, TCP Reno continues to increase its window size by one during each round-trip time. When it experiences a packet loss, it reduces its window size to one-half of the current size. Reno suggests an algorithm called 'Fast Retransmit.' Senders can retransmit a segment without waiting for a timeout if the recipient receives three duplicate ACKs, which indicates that the segment was lost. When a single packet is lost from a data window, TCP Reno maintains it by a fast recovery mechanism; in contrast, when multiple packets are lost, Reno's performance is the same here as Tahoe's. This indicates that if multiple packets are lost from the same window, TCP Reno almost immediately drags out of fast recovery and stops until no new packet can be sent. Hence, TCP Reno cannot effectively handle multiple packet losses within a single window. TCP Reno then enters a fast recovery phase (9) if the fast retransmit algorithm finds the packet loss. In this phase, the window size is increased by one packet when a duplicate ACK packet is received. (8)

TCP New Reno: TCP New Reno (9) enhances retransmission in TCP Reno's fast recovery phase. In order to maintain a full transmit window during rapid recovery, a fresh unsent packet from the end of the congestion window is sent for each duplicate ACK returned to TCP New Reno. For every ACK that represents a portion of the the next packet beyond the ACKed sequence number is transmitted, and the sender assumes that the ACK leads to a new hole in the sequence space. Like TCP SACK, New Reno can fill big or many holes in the sequence space since the timeout timer is reset whenever there is progress in the transmit buffer. High throughput is maintained during the hole-filling process even when there are several holes, each carrying multiple packets, because New Reno can deliver new packets close to the end of the congestion window during fast recovery. In fast recovery mode, TCP logs the sequence number of the highest unacknowledged packet. Following acknowledgement of the sequence number, TCP goes back to its congestion avoidance state.

TCP Vegas: The TCP congestion avoidance method known as TCP Vegas uses packet delay as a signal rather than packet loss to help decide how quickly to send packets.(3). TCP Vegas distinguishes itself from other versions like Reno, New Reno, etc., by detecting congestion early on through growing Round-Trip Time (RTT) values of the packets in the

connection. Other versions detect congestion only after it has occurred through packet loss. The Base RTT value must be calculated precisely for the algorithm to work. If the value is too great, it will overwhelm the connection, and if it is too small, the throughput of the connection will be less than the available bandwidth.

TCP-Lite: To minimize the overhead associated with session management, TCP-Lite is a service that offers an alternate transport channel for TCP connections in which no application data is sent or received. During channel setup, teardown, and acknowledgement, TCP-Lite reduces or eliminates pure TCP protocol data units (PDUs) while preserving the order, integrity, reliability, and security of the original TCP transport. Applications that use TCP to communicate between a client and server can use TCP-Lite without modification. In environments where clients require multiple or frequent session establishment, TCP-Lite reliably reduces the amount of data transferred between the client and server (10). To manage the performance choices between an MNC and mobility clients that connect to it, a TCP-Lite transport is applied to a connection profile, which is a collection of configuration properties provided to an MNC. The following features are included in TCP Lite: Slow start, avoidance of congestion, fast retransmission, quick recovery, large window, and protection against wrapped sequence numbers are only a few of the features available.

Comparison of TCP Algorithms

Algorithms/ TCP Variants	TCP Tahoe	TCP Reno	TCP New Reno	TCP Lite	TCP Vegas
Slow Start	Yes	Yes	Yes	Yes	E V
Congestion Avoidance	Yes	Yes	Yes	Yes	E V
Fast Retransmit	Yes	Yes	Yes	Yes	Yes
Fast Recovery	No	Yes	E V	Yes	Yes
Retransmission on mechanism	N	N	N	N	N M
Congestion Control mechanism	N	N	N	N	N M
Selective ACK mechanism	No	No	No	Yes	No

(N = Normal, E V = Enhanced Version, N M = New Mechanism)

Results and Analysis Based on Throughput, Signal Received with error, Packet Loss and Total Bytes Received.

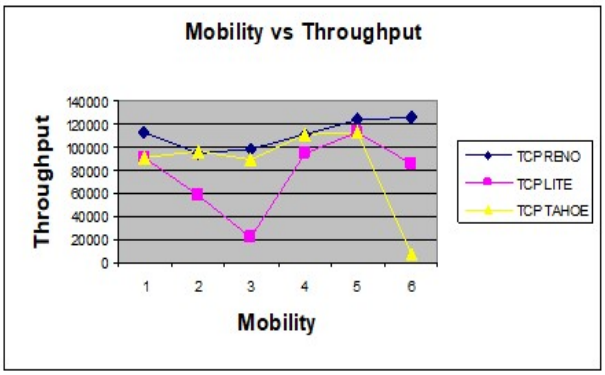

Figure 1. Mobility vs. Throughput

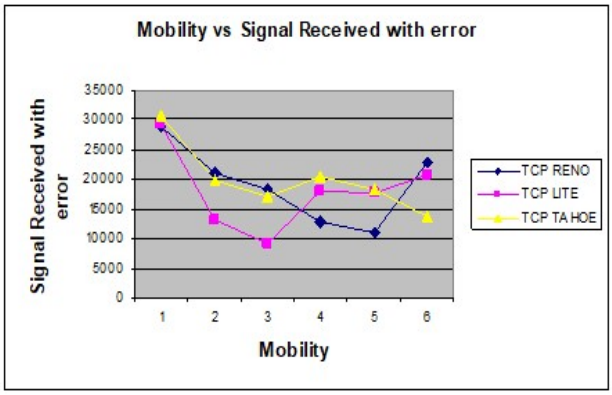

Figure 2. Mobility vs. Signal Received with error

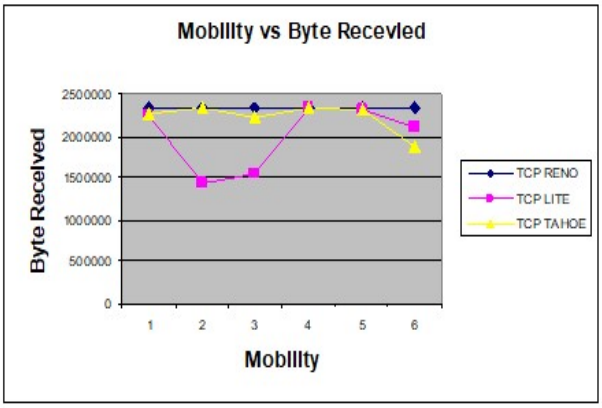

Figure 3. Mobility vs. Byte Received

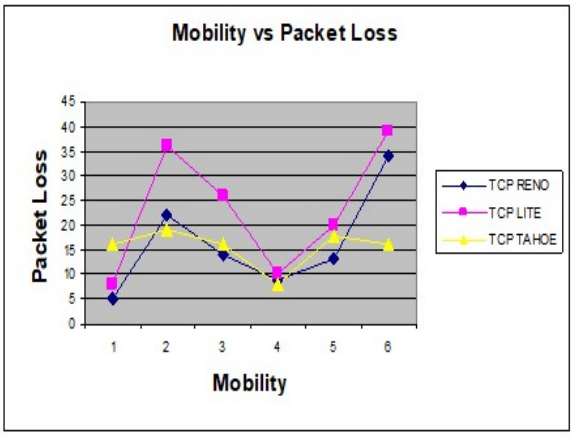

Figure 4. Mobility Vs Packet Loss

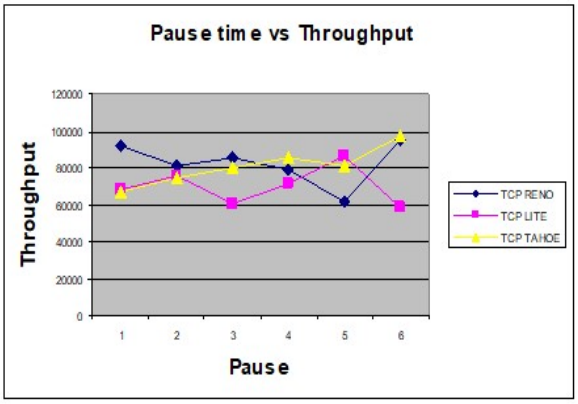

Figure 5 Pause Time vs. Throughput

28022 Deepika Ratnawat and Pankaj Raghuvanshi, Review: A behavioral analysis of tcp variants for manet

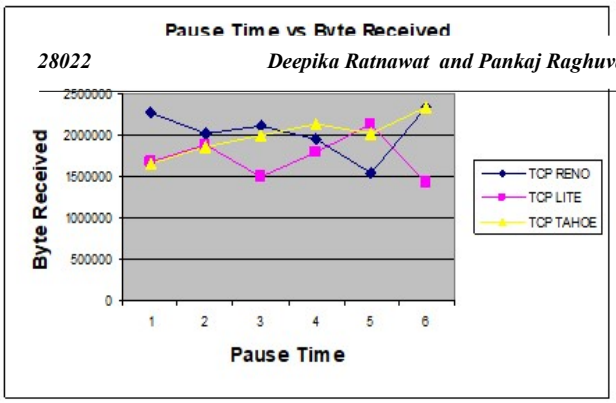

Figure 6. Pause Time vs. Byte Received

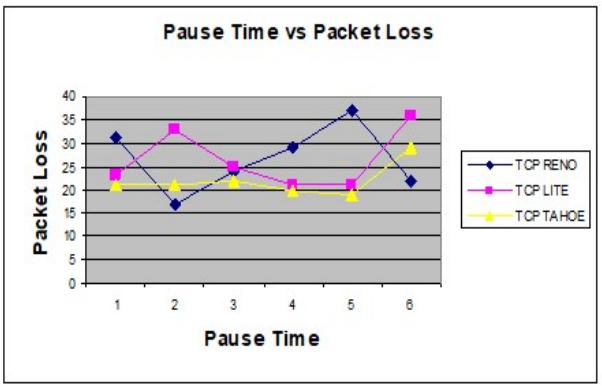

Figure 7. Pause Time vs. Packet Loss

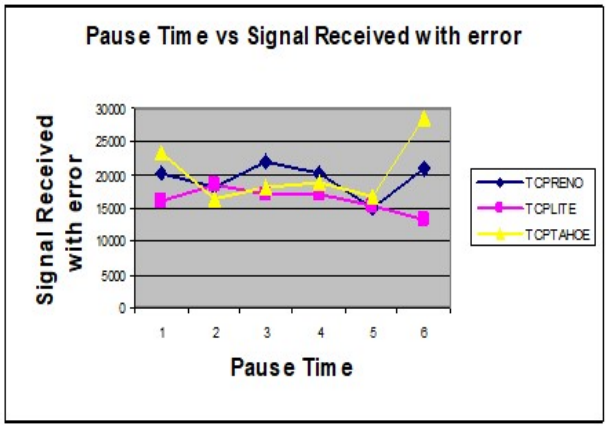

Figure 8. Pause Time vs. Signal Received with error

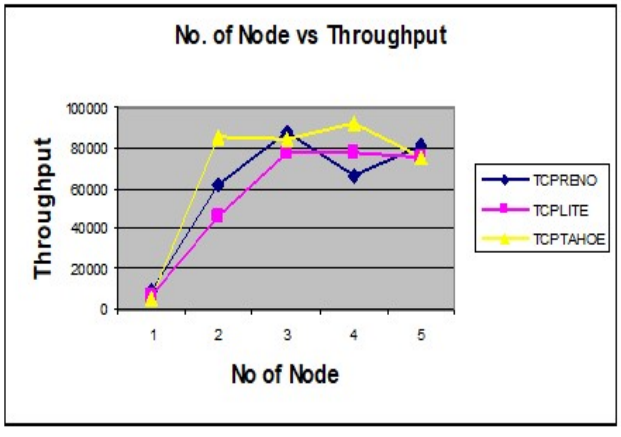

Figure 9. No of Node vs. Throughput

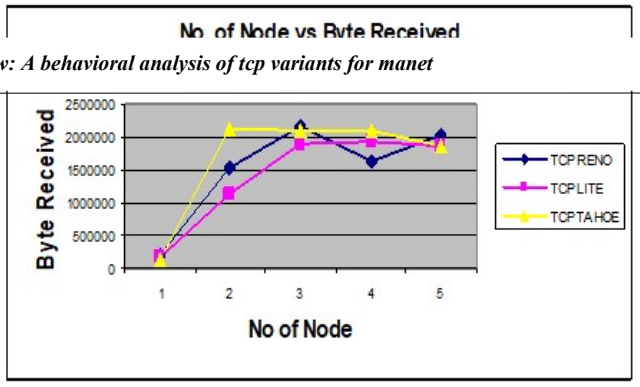

Figure 10. No of Node vs. Byte Received

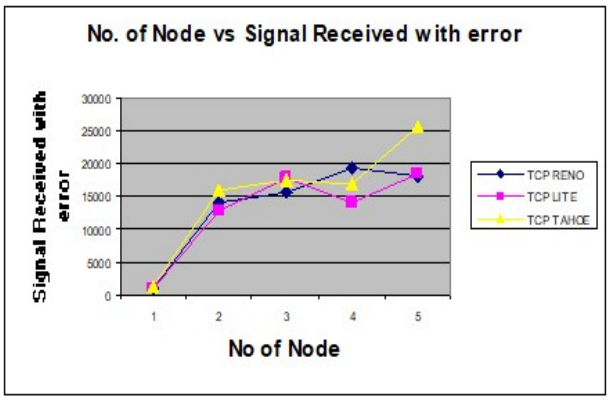

Figure 11. No of Node vs. Signal Received with error

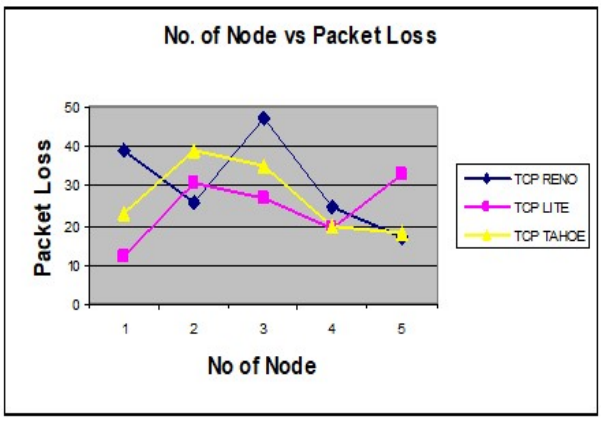

Figure 12. No of Node vs. Packet Loss

GUIDELINES FOR IMPROVING CONGESTION CONTROL ALGORITHM: Wireless environments have many different characteristics, such as higher error rates, longer delays, lower bandwidth, frequent mobility, and so on. TCP congestion control mechanisms are only sometimes directly suitable for wireless networks, and we have seen many improved TCP congestion control mechanisms. Generally, the influence of link corruption on the TCP sender's packet-sending rate is not considered in these enhanced schemes. However, unnecessary packets lost by corruption can be significantly avoided through the decrease of packet sending rate, which may lead to higher reliability, excessive energy consumption of mobile hosts, and less system overheads. For a given TCP connection, it is reasonable to assume that the possibility of packet loss by corruption can be obtained approximately from pe=m/n, where n is the number of total packets, and m is the

sum of packets lost by corruption during the period T. So, by using the corruption loss rate, the congestion control mechanism can be improved. In the following paper, we will propose a new mechanism called refined TCP.

CONCLUSION

In this paper, we have identified the possible causes of congestion over the network. We have also discussed the main intertwined algorithms that help to control congestion over the network. We also saw how TCP implements flow controls by having the receiver advertise the amount of data it is willing to accept. Then, we discussed TCP Reno fast retransmit and fast Recovery, TCP New Reno, and TCP Vegas congestion algorithms. We saw that the introduction of TCP Reno changed the way datagrams are exchanged. TCP Reno has performed remarkably well and has prevented severe congestion on the Internet. Although these algorithms have incredible potency in handling congestion, their limitation abounds.

REFERENCES

1. Postel, J. 1981. "Transmission control protocol," RFC 793, Sept.
2. Holland G. and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," in *Proc. ACM/IEEE Int. Conf. on Mobile Computing*, Seattle, WA, USA, Sept. 1999, pp. 219 –230
3. Brakmo, L.S. L.L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet", IEEE Journal on Selected Areas in Communication, vol. 13(1995), (1465-1490).
4. Ait-Hellal, O. E. Altman "Analysis of TCP Reno and TCP Vegas".
5. Stevens,W. "TCP slow start, congestion avoidance, fast retransmit and fast recovery algorithms," *RFC 2001*, Jan. 1997.
6. Jacobson,V. "Congestion avoidance and control," in *Proc. ACM SIGCOMM, Symp. on Commun. Archit. And Protocols*, Stanford, CA, USA, Aug. 1988, pp. 314–329.
7. Akhtar, M. N. M. A. O. Barry, and H. S. Al-Raweshidy, "Modified Tahoe TCP for wireless networks using OPNET. Simulator," in *Proc of the London Communications Symposium (LCS2003),* London, UK, Sept. 2003.
8. Floyd S. and K. Fall, "Simulation based comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Communication Review*, vol. 26, no. 3, pp. 5–21, July 1996.
9. Floyd S. and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm," *RFC 2582*, Apr. 1999.
10. Alfredsson, S "TCP Lite - A Bit Error Transparent Modification of TCP".
11. Zeng W. G. and Lj. Trajkovic, "TCP packet control for wireless networks," in *Proc. IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Communications* (*WiMob 2005*), Montreal, Canada, Aug. 2005, vol. 2, pp. 196–203.
12. Chiu D. and R. Jain, "Analysis of the increase/decrease algorithms for congestion avoidance in computer networks," *J. of Comput. Netw. ISDN Syst.*, vol. 17, no. 1, pp. 1–14, June 1989.
13. Allman, M. V. Paxson, and W. Stevens, "TCP congestion control," *RFC 2581*, Apr. 1999.
14. Floyd S. and K. Fall, "Simulation based comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Communication Review*, vol. 26, no. 3, pp. 5–21, July 1996.
15. Lee, H. S. Lee, and Y. Choi, "The influence of the large bandwidth-delay product on TCP Reno, NewReno, and SACK," in *Proc. Information Networking Conference*, Oita, Japan, Feb. 2001, pp. 327–334.
16. Anjum F. and L. Tassiulas, "Comparative study of various TCP versions over a wireless link with correlated losses," *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, pp. 370–383, June 2003.
17. Jacobson V. "Modified TCP Congestion Control and Avoidance Algorithms". Technical Report 30, Apr 1990.
18. Fall, K. S. Floyd "Simulation Based comparison of Tahoe, Reno and SACK TCP".

****