IMPLEMENTATION OF FREEZE TCP AND COMPARISON OF RESULT ANALYSIS WITH OTHER TCP VARIANTS

[1] Ms.Chetna Kapuriya, [2] Prof. Kaushal Doshi

[1]M.E.[Elec. & Comm.] Student, Faculty of Electronics & Communication Dept. Marwadi Education Foundation's Group of Institution.
[2] Asst.Professor, Faculty of Electronics & Communication Dept. Marwadi Education Foundation's Group of Institution.
kapuriyachetna@gmail.com, kaushal.doshi@gmail.com

***ABSTRACT*:** *Generally, in wired networks at transport layer flow control is required ad also monitoring of TCP traffic is done. For these purpose, in majority of the cases, intermediaries are used to enhance the performance. But still it cannot be considered as a fully end-to-end signaling. So to overcome this requirement of other intermediaries we will use Freeze TCP protocol. "Freeze-TCP" mechanism is a true end-to-end scheme and does not require the involvement of any intermediaries (such as base stations) for flow control. Furthermore, this scheme does not require any changes on the "sender side" or intermediate routers. Before implementing TCP over wireless network one must has the knowledge about the TCP working on wireless environment and some TCP variants and their working principles. For this purpose we implemented the TCP for wired link and their variants like TCP SACK, RENO, TAHOE etc. and also the results has been obtained.*

INTRODUCTION

Transmission control protocol (TCP) is a transport layer protocol which provides reliable end to end data delivery between end hosts in traditional wired network environment. In TCP, reliability is achieved by retransmitting lost packets. Thus, each TCP sender maintains a running average of the estimated round trip delay and the average deviation derived from it. Packets will be retransmitted if the sender receives no acknowledgment within a certain timeout interval (e.g., the sum of smoothed round trip delay and four times the average deviation) or receives duplicate acknowledgments[7]. Due to the inherent reliability of wired networks, there is an implicit assumption made by TCP that any packet loss is due to congestion. To reduce congestion, TCP will invoke its congestion control mechanisms whenever any packet loss is detected. Since TCP is well tuned, it has become the de facto transport protocol in the Internet that supports many applications such as web access, file transfer and email. Due to its wide use in the Internet, it is desirable that TCP remains in use to provide reliable data transfer services for communications within wireless networks and for those across wireless networks and the wired Internet. It is thus crucial that TCP performs well over all kinds of wireless networks in order for the wired Internet to extend to the wireless world[16].

The implication of the difference is that packet losses are no longer mainly due to network congestion; they may well be due to some wireless specific reasons. As a matter of fact, in wireless LANs or cellular networks, most packet losses are due to high bit error rate in wireless channels and handoffs between two cells, while in mobile ad hoc networks, most packet losses are due to medium contention and route breakages, as well as radio channel errors. Therefore, although TCP performs well in wired networks, it will suffer from serious performance degradation in wireless networks if it misinterprets such non-congestion related losses as a sign of congestion and consequently invoke congestion control and avoidance procedures, as confirmed through analysis and extensive simulations carried out[21].

As TCP performance deteriorates more seriously in ad hoc networks compared to WLANs or cellular networks, we divide wireless networks into two large groups: one is called one-hop wireless networks that include WLANs and cellular networks and the other is called multi-hop wireless networks that include MANETs. To understand TCP behavior and improve TCP performance over wireless networks, given these wireless specific challenges, considerable research has been carried out and many schemes have been proposed. As the research in this area is still active and many problems are still wide open, this chapter serves to pinpoint the primary causes for TCP performance degradation over wireless networks, and cover the state of the art in the solution spectrum, in hopes that readers can better understand the problems and hence propose better solutions based on the current ones [21].

TCP PERFORMANCE IN MANET

Even though TCP ensures reliable end-to-end message transmission over wired networks, a number of existing researches have showed that TCP performance can be substantially degraded in MANET. This section continues with a description of different types of constraints influencing TCP performance in MANET[21][25].

1. Route Failure

In MANET, the mobility of the node is considered as the major reason for the route failure

and the route reestablishment is instantly needed in case of route failure. However, it is likely that a new route establishment may experience longer duration than the RTO of the sender. In consequence of that, the TCP sender will unnecessary invoke congestion control mechanism.

2. Network Partitioning

A network partition takes place when a node departs from the network, resulting in an isolation of some parts of a mobile ad-hoc network. These fragmented portions are defined as partitions. In MANET, TCP considers network partitioning as one of the most imperative challenges which is mainly caused due to the mobility or energy-constrained (limited battery power) operation of nodes. When the source and the destination of a TCP connection lie in different parts of the network, all transmitting packets are found to be dropped by the network. As a result, the congestion control algorithm will be invoked instantly by the TCP sender

3. Hidden and Exposed Node Impact

Figure presents a typical hidden node condition where packet transmission starts from node A to node E. Since, node B cannot sense node D, node B assumes the channel as an idle channel and therefore initiates its transmission by dispatching a Request to Send (RTS) to node C. However, transmitting RTS unexpectedly introduces collisions because node C is found in the interference range of node D. This problem is termed as "Hidden Node" impact where node D is called the hidden node with respect to node B.

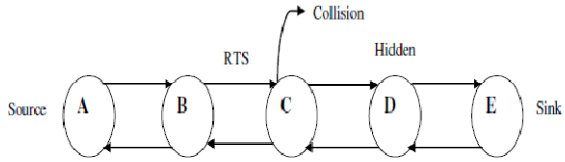


Fig. Hidden node Impact

Figure depicts a condition through which the exposed node problem can be realized. When node D intends to transmitting data toward node E, node C will not be able to send any data frame to node B. Node C must wait until node D finishes its current transmission to node E. This is because node D is within the sensing range of node C. This problem is known as "Exposed Node" impact where node D is called the exposed node with respect to node C.

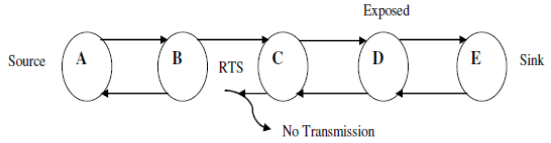


Fig. Exposed Node Impact

Simple TCP:

It is well known that TCP is a connection-oriented transport protocol that is aimed at guaranteeing end-to-end reliable ordered delivery of data packets over wired networks. For this purpose, basic functionalities such as flow control, error control, and congestion control are indispensable. While these functions have a clean-cut definition of their own, in practice they are closely coupled with one another in TCP implementation [1].

In TCP, a sliding window protocol is used to implement flow control, in which three windows are used, namely, Congestion Window, advertised window, and Transmission Window. Congestion window indicates the maximum number of segments (Without causing confusion, the term segment and packet are used interchangeably henceforth) that the sender can transmit without congesting the network. As shown next in details on congestion control, this number is determined by the sender based on the feedback from the network. advertised window, however, is specified by the receiver in the acknowledgements it. Advertised window indicates to the sender the amount of data the receiver is ready to receive in the future. Normally, it equals to the available buffer size at the receiver in order to prevent buffer overflow. Transmission window means the maximum number of segments that the sender can transmit at one time without receiving any ACKs from the receiver. Its lower edge indicates the highest numbered segment acknowledged by the receiver. Obviously, to avoid network congestion and receiver buffer overflow, the size of transmission window is determined as the minimum of the congestion window and the receiver's advertised window [7].

TCP variants:

Most of the progress made in TCP is centered on error recovery and congestion control. Representative innovations include fast transmissions and fast recovery, selective acknowledgements, random early detection (RED) [2] in routers, and explicit congestion notification (ECN). Notice that depending on what features are included, there are several TCP flavors, including TCP Tahoe, TCP Reno, TCP New Reno, etc. Among them, TCP Reno is by far most widely deployed [2].

1) Fast retransmission and fast recover

As noted earlier, a packet can be assumed lost if three duplicate ACKs are received. In this case, TCP performs a fast retransmission of the packet. This mechanism allows TCP to avoid a lengthy timeout during which no data is transferred. At the same time, *ssthresh* is set to one half of the current congestion window, i.e., *cwnd,* and *cwnd* is set to *ssthresh* plus three segments. If the ACK is received approximately one round trip after the missing segment is retransmitted, fast recovery is entered. That is, instead of setting *cwnd* to one segment and starting with slow start, TCP sets *cwnd* to *ssthresh*, and then steps into congestion avoidance phase. However, only one packet loss can be recovered during fast retransmission and fast recovery. Additional packet

losses in the same window may require that the RTO expire before retransmission [25].

2) Selective acknowledgment

Owing to the fact that fast retransmission and fast recovery can only handle one packet loss from one window of data, TCP may experience poor performance when multiple packets are lost in one window. To overcome this limitation, recently the selective acknowledgement option (SACK) is suggested as an addition to the standard TCP implementation.

The SACK extension adopts two TCP options. One is an enabling option, which may be sent to indicate that the SACK option can be used upon connection establishment. The other is the ACK option itself, which may be sent by TCP receiver over an established connection if SACK option is enabled through sending the first option. The SACK option contains up to four (or three, if SACK is used in conjunction with the Timestamp option used for RTTM [24]) SACK blocks, which specifies contiguous blocks of the received data. Each SACK block consists of two sequence numbers which delimit the range of data the receiver has received and queued. A receiver can add the SACK option to ACKs it sends back to a SACK-enabled sender. In the event of multiple losses within a window, the sender can infer which packets have been lost and should be retransmitted using the information provided in the SACK blocks. A SACK-enabled sender can retransmit multiple lost packets in one RTT instead of detecting only one lost packet in each RTT [23].

3) Random Early Detection (RED

Random Early Detection (RED) is a router-based congestion control mechanism that seeks to detect incipient congestion and notify some TCP senders of congestion by controlling the average queue size at the router. To notify the TCP senders of congestion, the router may mark or drop packets, depending on whether the senders are cooperative. As a response, the senders should reduce their transmission rate. This is done in two algorithms. The first algorithm is to compute the average queue size by using exponential weighted moving average. If we denote by avg and q the average queue size and the current queue size, respectively, then avg = (1-wq)*avg + wq*q, where wq is the queue weight. The other algorithm is to compute the packet-marking or packet-dropping probability pa. If avg falls in between minth and maxth, the packet marking probability pb = maxp (avg - minth) / (maxth - minth) and the final marking probability pa = pb/(1 – count*pb), where maxp and count are design parameters, respectively, denoting the maximum value for pb and the number of packets having arrived since last packet marking or dropping. If avg exceeds maxth, pa = 1, which means that the router marks or drops each packet that arrives. Through control over the average queue size prior to queue overflow, RED succeeds in preventing heavy network congestion and global synchronization as well as improving fairness. Notice that numerous variants of RED have been proposed to improve various performance of the original RED [22] [24].

4) Explicit Congestion Notification (ECN)

Most of current Internet routers employ traditional “drop-tail” queue management. In other words, the routers drop packets only when the queue overflows, which could lead to the undesirable global synchronization problem as well as heavy network congestion. Recently, active queue management (AQM) mechanisms have been proposed since they can detect congestion before the queue overflows at the routers and inform TCP senders of the congestion, thereby avoiding some of these problems caused by the “drop-tail” policy. In the absence of Explicit Congestion Notification (ECN), however, the only choice that is available to AQM for indicating congestion to end systems is to drop packets at the routers. With ECN, AQM mechanisms have an alternative to allow routers to notify end systems of congestion in the network [23].

SLIDING WINDOW METHOD

In many cases, it is possible to limit the number of acknowledgements, in order to relieve traffic on the network, by fixing a sequence number at the end of which an acknowledgement is required. This number is in fact stored in the *window* field of the TCP/IP header.

This method is effectively called the "*sliding window method*" because to some extent a range of sequences is defined that does not need acknowledgements and which moves as acknowledgements are received [14].

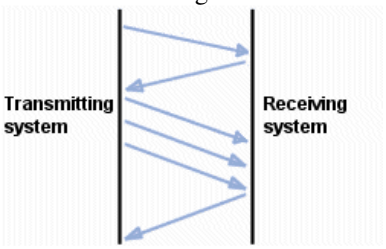
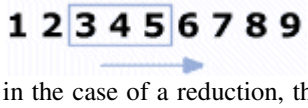


Fig. Sliding window technique in TCP

In addition, the size of this window is not fixed. In fact, the server can include the size of the window which seems most suitable in its acknowledgements by storing it in the window field. So, when the acknowledgement indicates a request to increase the window, the client will move the right border of the window.



Conversely, in the case of a reduction, the client will not move the right border of the window towards the left but wait for the left border to advance (with the arrival of the acknowledgements).

Ending a connection

The client can request to end a connection in the same way as the server. Ending a connection is done in the following way [16]:

- One of the machines sends a segment with the *FIN* flag set to 1, and the application puts itself in a waiting state, i.e. it finishes receiving the current segment and ignores the following ones.
- After receipt of this segment, the other machine sends an acknowledgement with the *FIN* flag set to 1 and continues to send the segments in progress. Following this, the machine informs the application that a *FIN* segment has been received, then sends a *FIN* segment to the other machine, which closes the connection.

TCP Tahoe:

Tahoe refers to the TCP congestion control algorithm which was suggested by Van Jacobson in his paper[1]. TCP is based on a principle of 'conservation of packets', i.e. if the connection is running at the available bandwidth capacity then a packet is not injected into the network unless a packet is taken out as well. TCP implements this principle by using the acknowledgements to clock outgoing packets because an acknowledgement means that a packet was taken off the wire by the receiver. It also maintains a congestion window CWD to reflect the network capacity. However there are certain issues, which need to be resolved to ensure this equilibrium [1].

1) Determination of the available bandwidth.
2) Ensuring that equilibrium is maintained.
3) How to react to congestion

Slow Start:

TCP packet transmissions are clocked by the incoming acknowledgements. However there is a problem when a connection first starts up cause to have acknowledgements you need to have data in the network and to put data in the network you need acknowledgements. To get around this circularity Tahoe suggests that whenever a TCP connection starts or re-starts after a packet loss it should go through a procedure called 'slow-start'. The reason for this procedure is that an initial burst might overwhelm the network and the connection might never get started. A slow start suggests that the sender set the congestion window to 1 and then for each ACK received it increase the CWD by 1. so in the first round trip time(RTT) we send 1 packet, in the second we send 2 and in the third we send 4. Thus we increase exponentially until we lose a packet which is a sign of congestion. When we encounter congestion we decreases our sending rate and we reduce congestion window to one. And start over again. The important thing is that Tahoe detects packet losses by timeouts. In usual implementations, repeated interrupts are expensive so we have coarse grain time-outs which occasionally checks for time outs. Thus it might be some time before we notice a packet loss and then re-transmit that packet.

Congestion Avoidance:

For congestion avoidance Tahoe uses 'Additive Increase Multiplicative Decrease'. A packet loss is taken as a sign of congestion and Tahoe saves the half of the current window as a threshold value. It then set CWD to one and starts slow start until it reaches the threshold value. After that it increments linearly until it encounters a packet loss. Thus it increase it window slowly as it approaches the bandwidth capacity.

Problems:

The problem with Tahoe is that it take a complete timeout interval to detect a packet loss and in fact, in most implementations it takes even longer because of the coarse grain timeout. Also since it doesn't send immediate ACK's, it sends cumulative acknowledgements, therefore it follows a 'go back n' approach. Thus every time a packet is lost it waits for a timeout and the pipeline is emptied. This offers a major cost in high band-width delay product links.

TCP RENO:

This Reno retains the basic principle of Tahoe, such as slow starts and the coarse grain re-transmit timer. However it adds some intelligence over it so that lost packets are detected earlier and the pipeline is not emptied every time a packet is lost. Reno requires that we receive immediate acknowledgement whenever a segment is received. The logic behind this is that whenever we receive a duplicate acknowledgment, then his duplicate acknowledgment could have been received if the next segment in sequence expected, has been delayed in the network and the segments reached there out of order or else that the packet is lost. If we receive a number of duplicate acknowledgements then that means that sufficient time has passed and even if the segment had taken a longer path, it should have gotten to the receiver by now. There is a very high probability that it was lost. So Reno suggest an algorithm called **'Fast ReTransmit'.** Whenever we receive 3duplicate ACK's, we take it as a sign that the segment was lost, so we retransmit the segment without waiting for timeout. Thus we manage to re-transmit the segment with the pipe almost full [4].

Another modification that RENO makes is in that after a packet loss, it does not reduce the congestion window to 1. Since this empties the pipe. It enters into a algorithm which we call **'Fast-ReTransmit'.** The basic algorithm is presented as under:

1) Each time we receive 3 duplicate ACK's we take that to mean that the segment was lost and we re-transmit the segment immediately and enter 'Fast-Recovery'
2) Set SSthresh to half the current window size and also set CWD to the same value.
3) For each duplicate ACK receive increase CWD by one. If the increase CWD is greater than the amount

of data in the pipe then transmit a new segment else wait. If there are 'w' segments in the window and one is lost, then we will receive (w-1) duplicate ACK's. Since CWD is reduced to W/2, therefore half a window of data is acknowledged before we can send a new segment. Once we retransmit a segment, we would have to wait for at least one RTT before we would receive a fresh acknowledgement. Whenever we receive a fresh ACK we reduce the CWND to SSthresh. If we had previously received (w-1) duplicate ACK's then at this point we should have exactly w/2 segments in the pipe which is equal to what we set the CWND to be at the end of fast recovery. Thus we don't empty the pipe, we just reduce the flow. We continue with congestion avoidance phase of Tahoe after that [5].

Problems:

Reno perform very well over TCP when the packet losses are small. But when we have multiple packet losses in one window then RENO doesn't perform too well and its performance is almost the same as Tahoe under conditions of high packet loss. The reason is that it can only detect a single packet loss. If there is multiple packet drop then the first info about the packet loss comes when we receive the duplicate ACK's. But the information about the second packet which was lost will come only after the ACK for the retransmitted first segment reaches the sender after one RTT.

roblem is that if the widow is very small when the loss occurs then we would never receive enough duplicate acknowledgements for a fast retransmit and we would have to wait for a coarse grained timeout. Thus is cannot effectively detect multiple packet losses.

TCP SACK:

TCP with 'Selective Acknowledgments' is an extension of TCP Reno and it works around the problems face by TCP RENO and TCP New-Reno, namely detection of multiple lost packets, and re-transmission of more than one lost packet per RTT.

SACK retains the slow-start and fast retransmits parts of RENO. It also has the coarse grained timeout of Tahoe to fall back on, in case a packet loss is not detected by the modified algorithm.

SACK TCP requires that segments not be acknowledged cumulatively but should be acknowledged selectively. Thus each ACK has a block which describes which segments are being acknowledged. Thus the sender has a picture of which segments have been acknowledged and which is still outstanding. Whenever the sender enters fast recovery, it initializes a variable pipe which is an estimate of how much data is outstanding in the network, and it also set CWND to half the current size. Every time it receives an ACK it reduces the pipe by 1 and every time it retransmits a segment it increments it by 1. Whenever the pipe goes smaller than the CWD window it checks which segments are un received and send them. If there are no such segments outstanding then it sends a new packet. Thus more than one lost segment can be sent in one RTT [5].

SIMULATION AND RESULTS:

SIMULATION PARAMETERS:

TCP PARAMETERS	VALUE
SLOW START INITIAL COUNT	1
RECEIVE BUFFER SIZE (BYTES)	8,760
MAXIMUM ACK SEGMENT	2
DUPLICATE ACK THRESHOLD	3
INITIAL RTO (SECONDS)	1.0
MINIMUM RTO (SECONDS)	0.5
MAXIMUM RTO (SECONDS)	64
RTT GAIN	0.125
DEVIATION GAIN	0.25

SIMULATION RESULTS:

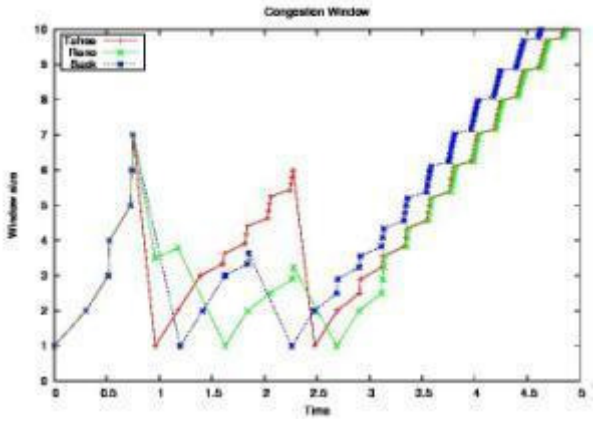


Fig Plot of time Vs window size

Figure shows the plot of time vs. window size. We can see that the window size variation in TAHOE is maximum among all three TCP variants TCP TAHOE, TCP RENO and TCP SACK. Also the observation can be done that the RANO and TAHOE behaves same as the time of the network increases.

Figure shows the plot of congestion window size vs. time. It shows the the progression of the congestion window with time and the number of packets transmitted for all the three TCP variants namely TAHOE, RENO and SACK.

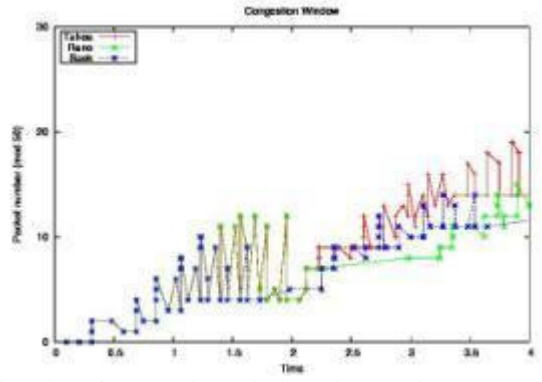


Fig. plot of congestion window size vs. time

Observation:

From the plots it can be observed that the time taken by the SACK version of TCP to overcome the congestion is less compared to others. This difference is seen clearly in the plot of window size. In the plot of packet numbers, the SACK overpowers the RENO but is not able to cope with the TAHOE version.

CONCLUSION

The TAHOE version of TCP follows the concept of go-back-n method to cope up with the congestion. The RENO TCP is better than the TAHOE because it follows the congestion control mechanism to cope up with the congestion. But in SACK a provision is made available such that when congestion occurs, only the selected numbers of packets are being retransmitted and not all of the data is retransmitted. Because of this characteristic, SACK is able to cope up with the congestion much faster than the other TCP variants. This difference is seen in the congestion window progression plot clearly.

REFRENCES

[1] V.Jacobson "Congestion Avoidance and Control" SIGCOMM Symposium no Cummunication Architecture and protocols.

[2] V.Jacobson "Modified TCP Congestion Control and Avoidance Alogrithms". Technical Report 30, Apr 1990.

[3] S.Floyd, T.Henderson "The New Reno Modification to TCP's Fast Recovery Algorithm" RFC 2582, Apr 1999.

[4] O. Ait-Hellal, E.Altman "Analysis of TCP Reno and TCP Vegas".

[5] K.Fall, S.Floyd "Simulation Based Comparison of Tahoe, Reno and SACK TCP".

[6] L.S.Brakmo, 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).

[7] A. Ahuja, S. Agarwal, J. P. Singh and R. Shorey, "Performance of TCP over different Routing Protocols in Mobile Ad-hoc Networks," IEEE Vehicular Technology Conference 2000, vol. 3, pp. 2315 - 2319, Tokyo.

[8] I. Ali, R. Gupta, S. Bansal, A. Misra, A. Razdan and R. Shorey, "Energy Efficiency and Throughput for TCP Traffic in Multi-Hop Wireless Networks," IEEE INFOCOM'02, New York, 2002.

[9] V. Anantharaman, S.-J. Park, K. Sundaresan, and R. Sivakumar, "TCP Performance over Mobile Ad-hoc Networks: A Quantitative Study," 'To appear in Wireless Communications and Mobile Computing Journal (WCMC), Special Issue on Performance Evaluation of Wireless Networks, 2003.

[10] E. Ayanoglu, S. Paul, T. F. LaPorta, K. K. Sabnani and R. D. Gitlin, "AIRMAIL: a link-layer protocol for wireless networks," ACM Wireless Networks, Feb. 1995.

[11] H. Balakrishnan, V. Padmanabhan, S. Seshan and R. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," Proceedings of ACM SIGCOMM'96, Aug. 1996.

[12] H. Balakrishnan, S. Seshan and R. H. Katz, "Improving reliable transport and handoff performance in cellular wireless networks," ACM Wireless Networks, Dec. 1995.

[13] A. Bakre and B. R. Badrinath, "I-TCP: indirect TCP for mobile hosts," Proc. 15th International Conf. On Distributed Computing systems (ICDCS), May 1995.

[14] P. Bhagwat, P. Bhattacharya, A. Krishna and S. K. Tripathi, "Enhancing throughput over wireless LANs using channel state dependent packet scheduling," IEEE INFOCOM'96, San Francisco, March 1996

[15] K. Brown and S. Singh, "M-TCP: TCP for mobile cellular networks," ACM computer communication review, vol. 27, no. 5, Oct. 1997

[16] R. Caceres and L. Iftode, "Improving the performance of reliable transport protocols in mobile computing environments," IEEE JSAC. Vol.19 No.7, July 2001.

[17] M. C. Chan, R. Ramjee, "TCP/IP performance over 3G wireless links with rate and delay variation," MobiCom'02, Sep. 2002.

[18] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "A Feedback-based Scheme for Improving TCP Performance in Ad Hoc Wireless Networks," IEEE Personal communications, 8 (1):34-39, February 2001.

[19] K. Chen, Y. Xue, K. Nahrstedt, "On Setting TCP's Congestion Window Limit in Mobile Ad Hoc Networks," IEEE ICC'03, Anchorage, Alaska, May, 2003.

[20] A. DeSimone, M. C. Chuah and O. C. Yue, "Throughput Performance of Transport-Layer Protocols over Wireless LANs," Proc. Globecom '93, Dec. 1993.

[21] T. D. Dyer and R. V. Boppana, "A Comparison of TCP Performance over Three Routing Protocols for Mobile Ad Hoc Networks," ACM Mobihoc, October 2001.

[22] S. Floyd, K. Fall, ``Router mechanisms to Support end-to-end congestion control", LBL Technical report, February 1997.

[23] S. Floyd, V. Jacobson, "Random early detection gateways for congestion avoidance," IEEE/ACM Transaction on Networking, vol. 1, no. 4, Aug. 1993.

[24] Z. Fu, P.Zerfos, H. Luo, S. Lu, L. Zhang, M. Gerla, "The Impact of Multihop Wireless Channel on TCP Throughput and Loss", IEEE INFOCOM'03, San Francisco , March 2003.

[25] Z. Fu, X. Meng, and S. Lu, "How Bad TCP can Perform in Mobile Ad-Hoc Networks," IEEE Symposium on Computers and Communications, Italy, July 200233rd Hawaii International Conference on System Sciences (HICSS '00), January 2000.