A New TCP Congestion Control Mechanism over Wireless Ad Hoc Networks by Router-Assisted Approach

Yao-Nan Lien and Ho-Cheng Hsiao
Department of Computer Science, National Chengchi University
Muzha, Taipei, Taiwan
{lien, g9308}@cs.nccu.edu.tw

Abstract—Due to unawareness of network conditions, regular Transmission Control Protocol (TCP) is not able to fully control the limited resources and distinguish packet loss from congestion loss and random loss. Because of such characteristics, the performance of TCP will be downgraded severely if it runs on wireless networks, such as multi-hop ad hoc networks. In an ad hoc network, nodes play both roles of end system as well as router. We take this advantage to propose a new TCP congestion control mechanism by router-assisted approach. Based on the information feedbacked from routers, a TCP sender is able to adjust its sending speed dynamically in order to avoid overshooting problem. Our proposed protocol has 5~10% higher throughput than TCP NewReno and much less number of retransmissions. The fairness is also achieved while our proposed protocol coexists with other major TCP variants.

Keyword: TCP, congestion control, MANET.

I. INTRODUCTION

The solution presented in this paper belong to Relevance Ring 3 (as defined in the ""Message from the Steering Committee" preceding the SAHNS papers in these proceedings). The reason is that it proposes a solution for general ad hoc networks.

With respect to a path that connects two end points, congestion usually occurs in a bottleneck node. Network elements including routers and end terminals need to work hard to prevent the network from being crashed by congestion. The congestion control within the TCP plays a critical role in adjusting data rate to avoid congestion from happening. Based on the window-adjustment algorithm, a TCP sender not only guarantees the successful packet delivery, but also maintains the appropriate data rate.

TCP protocol is executed at the terminal nodes of a network. It doesn't have real time inside information about the network. The only indicator of network status to the TCP protocol is packet traveling time as well as success or failure of package delivery. Therefore, most current TCP variants count on these indicators to "guess" (estimate) the available bandwidth over the path connecting the sender to the receiver and to adjust data rate accordingly. The accuracy and the promptness of bandwidth estimation are dependent on many complicated factors such as the stability of network traffic and the length of the path. Not surprisingly, most TCP variants are suffering from some performance shortcomings, such as congesting network by sending data too fast as well as decreasing data rate unnecessarily. Unfortunately, MANET (Mobile Ad Hoc Network) is a very unstable network with slow network elements as well as unreliable wireless links such that running TCP over MANET will suffer from even severe performance problems.

A new approach to enhance TCP performance is to have network elements to provide assistance based on the network-inside information such as failure of a link, more accurate available bandwidth, per-hop latency, or simply per-hop saturation level. By utilizing this information, TCP at the terminal nodes will be able to adjust its data rate closer to the network capacity and to improve the performance of both TCP and network accordingly. This approach may not be easy to implement on WAN (Wide Area Network) because upgrading a large number of routers in a WAN is almost a business impossible. However, MANET has no such concern so that it is easy for a MANET to embrace this new approach.

In this paper, we propose a new TCP protocol, called TCP Muzha that uses the assistance provided by routers to achieve better congestion control. To use TCP Muzha, routers are required to provide some information allowing the sender to estimate more accurately the remaining capacity over the bottleneck node with respect to the path from the sender to the receiver. With this information, TCP Muzha will be able to enhance the performance of both TCP and network.

The rest of this paper is organized as follow. In Section II, we review the relative background and research regarding to TCP. We introduce our congestion control mechanism –

TCP Muzha in Section III and evaluate our algorithm against others by simulation in Section IV. Finally, we conclude our main contribution of this paper and highlight some future work in Section V.

II. Related Work

Most variants of TCP use packet loss as the indicator of network congestion although it may not be a reliable signal under some special circumstances. As mentioned in Section 1, TCP doesn't have real time inside information about the network so that it has to "guess" the available bandwidth. However, the accuracy and the promptness of bandwidth estimation may not be high since they depend on many complicated factors. Therefore, most TCP variants are suffering some performance shortcomings. It is not easy to enhance their performance unless routers can provide some assistance.

ECN [9] and RED [10] are designed to enable TCP senders to response faster to congestion in routers. But these two mechanisms provide only explicit (packet marking) or implicit (packet dropping) single-bit congestion-status information (congestion or no congestion) as feedback to the TCP senders. The lack of more delicate information about the router status limits the way in which a TCP sender reacts on the current router condition adequately. Their performance gain is limited since ECN and RED are not able to signal information about the available bandwidth to the TCP senders.

TCP-ELFN [11], TCP-F [12] and ATCP [13] are proposed to overcome the drawbacks of conventional TCP over MANET and they are mainly to deal with the random packet loss caused by unreliable wireless links. They are based on the concept of freezing TCP states and keeping large congestion window without decreasing the transmission rate at the occurrence of routing change.

In addition to dealing with random packet loss, TCP over MANET needs also to deal with the congestion problems caused by the contention nature of MANET. On a wired WAN, overflowed packets will cause the loss of themselves. On the contrast, overflowed packets in a MANET will lead to high chance of collision and MAC contention loss which will result in the loss of all packets that participate in the contention. Many recent studies [14][15][16] have confirmed that TCP with a small upper bound for congestion window size has better performance in 802.11 multihop networks.

In summary, there is a need to take the advantage of MANET to develop a special router-assisted TCP protocol to deal with the congestion control problems caused by not only the random packet loss, but also the contention nature of MANET.

III. TCP Muzha

Most of the TCP are not aware of network condition such that they may not be able to control congestion precisely and promptly resulting in unstable bandwidth utilization. If two end hosts can obtain network information from routers, they will be able to execute congestion control more efficiently.

With respective to a path, congestion usually occurs in the bottleneck point which has the minimum available bandwidth. If the sender can adjust the data rate dynamically according to the status of the bottleneck, TCP induced congestion should be avoid or dissolved efficiently. Therefore, we propose a new congestion control mechanism – TCP Muzha, which takes router-assisted approach to dynamically adjust the size of congestion window (i.e. data rate) according the router information, and to deal with random loss.

Design Objectives

The objectives of TCP Muzha are as follows:

1. Reducing the occurrence of congestion
2. Maximizing throughput
3. Dealing with random loss
4. Providing fairness service

Estimation of Available Bandwidth

The available residual bandwidth in each router depends on many factors such as the length of the queue, queueing time, buffer size and length of the spare queue. We assume each router is able to calculate by itself the available bandwidth and convert it into an index called Data Rate Adjustment Index (DRAI), which will be explained later.

Use of Available Bandwidth

TCP Muzha defines a new IP option named MRAI (Minimum DRAI) in IP packet header. The sender of a TCP Muzha connection sets the MRAI to a maximum value for every packet it sends. Each node compares its own DRAI with this value and replaces it if its value is smaller. The receiver end notices the minimum value of DRAI and sends this information back to the sender by acknowledgement (ACK). The sender can then use this information to adjust its data rate, i.e. the size of its congestion window.

Because of the following reason, each node publishes a DRAI value instead of the original available bandwidth. If there is more than one TCP Muzha connection passing

through a router, which is very likely, most of them may try to adjust their transmitting data rates up to the level they are informed. Disseminating the original residual bandwidth directly to all TCP senders will lead to an immediate traffic burst. Therefore, a router must smartly share its residual bandwidth to all the TCP connections that pass through it. Unfortunately, routers are usually not aware of the types of transport protocols that control the packets which they are forwarding. They cannot simply divide their residual bandwidth by the total number of TCP connections. Although a router may be able to peek into the content of packets to determine their controlling transport protocols, we do not take this approach because it may consume a significant part of node capacity. Furthermore, violating protocol independence principle may induce unexpected reliability problems.

Design of DRAI

Because of the difficulties mentioned above, TCP Muzha takes the following approach: instead of publishing available bandwidth, routers make recommendation to the passing traffic flows to increase or to decrease their data rates. Each router determines a DRAI value, which is a quantified data rate adjustment recommendation, according to its own network status and publishes this information. With respect to each TCP connection, there is a minimum DRAI value called Minimum data Rate Adjustment Index (MRAI). Senders can refer to this value to increase or decrease its data rate (i.e. window size). By this approach, the decision of data rate adjustment is no longer the sole responsibility of senders. Routers which have the knowledge of network status can participate in the decision of data rate adaptation. The sender will be able to adjust its data rate according to the MRAI before actual congestion occurring and doesn't have to rely on the actual occurrence of congestion to trigger the congestion control.

Multi-Level Data Rate Adjustment

The most critical design issue in TCP Muzha is the determination of DRAI. Currently, there doesn't exist any theoretical formula for this. We take empirical approach to design the DRAI formula. Due to a lack of mature knowledge, we choose a coarse grain multi-level quantization formula that defines the data rate adjustment recommendation into levels such as aggressive acceleration (deceleration), moderate acceleration (deceleration), and stabilizing. Further empirical research is needed to find a formula for routers to determine their DRAIs based on their bandwidth utilization.

Dealing with Random Loss

Unlike wired links, wireless links that use the air as a transmission medium suffer from high error rates and the errors occur in bursts. Furthermore, the contention based media access protocols may cause excessive packet loss as well. These will cause the sender to retransmit, timeout, and unnecessary decrease of congestion window which leads to the reduction of throughput. Since our proposal is aimed to solve congestion problem by router-assisted approach, the random loss can also be distinguished by our proposed packet marking scheme. When a marked duplicated ACK with a data rate deceleration index is received, the sender knows that the loss was caused by congestion. Otherwise, the loss can be classified into random loss and sender is able to retransmit the loss packets without unnecessary congestion window reduction.

TCP Muzha Congestion Control Mechanism

Unlike other TCP variants that need to "probe" network bandwidth by increasing their data rates carefully using Slow Start and AIMD, TCP Muzha is able to adjust its data rate based on the recommendation given by routers. Thus, TCP Muzha simplifies the three phases of TCP NewReno into two phases; CA (Congestion Avoidance) phase and FF (Fast Recovery & Fast Retransmit) phase. While TCP session initiated, it directly enters the CA phase. After receiving the new ACK, sender adjusts the CWND size according to the MRAI. After congestion occurs, TCP Muzha inherits most of the congestion control mechanisms from the traditional TCP in order to response to congestion immediately. If three duplicate ACKs are received by the sender, TCP Muzha enters the FF phase and reduces CWND to one half. If transmission timer expires, the sender resets CWND to 1 and returns to the CA phase. The phase (state) transition diagram is shown in Fig. 1 and the congestion control mechanism is summarized in Table 1.

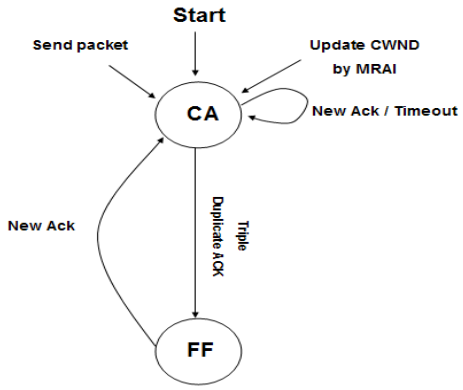

Fig. 1. Phase transition diagram of TCP Muzha

Table 1. Congestion control mechanism of TCP Muzha

Event	Status	Sender Behavior	Note
Receive the ACK of the pervious packet	Congestion Avoidance (CA)	Dynamically adjust CWND according to the returning MRAI	Adjust CWND in every RTT
Receiving *marked* 3 duplicate ACKs	Congestion Avoidance (CA)	(1)CWND = CWND * (1/2) (2) Enter FF phase	Fast respond and half the CWND
Receiving 3 duplicate ACKs	Congestion Avoidance (CA)	(1) Enter FF phase without change of CWND	Retransmit the lost packets
Timeout	Congestion Avoidance (CA)	(1)CWND = 1 (2)Re-enter CA phase	Re-enter the CA phase

Table 2. Simulation Parameters

Parameter	Range
Number of Nodes	4~32
Link Bandwidth	2 Mbps
Transmission Range	250 m
Packet Size	1460 bytes
Queue Management	IFQ

Table 3. DRAI Formula

DRAI	Meaning	Change of CWND
5	Aggressive Acceleration	CWND = CWND *2
4	Moderate Acceleration	CWND = CWND+1
3	Stabilizing	CWND = CWND
2	Moderate Deceleration	CWND = CWND -1
1	Aggressive Deceleration	CWND =CWND *1/2

IV. Performance Evaluation

In this section, we evaluate and compare the performance of proposed congestion control mechanism, TCP Muzha with TCP NewReno, TCP SACK and TCP Vegas in different designed network environments by using the network simulator NS-2[20]. We also observe and show the behavior and the performance while TCP Muzha coexists with TCP NewReno. The general parameters are listed in Table 2 and the DRAI formula used by TCP Muzha is shown in Table 3.

Simulation 1: Change of Congestion Window Size

In this subsection, we investigate the change of CWND for TCP Muzha under a chain topology of 4, 8, 16 hops respectively. An example of the first network topology for the simulation is shown in Fig. 2.

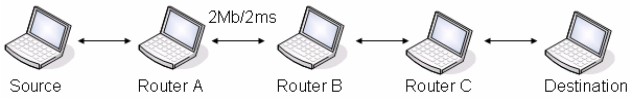

Fig. 2. Chain topology of 4-hop with a single flow

The results are shown in Fig. 3, 4 and 5 respectively. TCP Muzha is capable to adjust its CWND size up to the network bandwidth promptly and to maintain its CWND while facing the event of random loss. TCP Vegas keeps its CWND steadily. But due to its conservative nature in congestion control, the CWND size of TCP Vegas is not able to keep up to the network bandwidth with increase of hop number. TCP NewReno and SACK tend to trigger their congestion control mechanisms more frequently due to periodic packet loss and random loss caused by the instability nature of wireless networks.

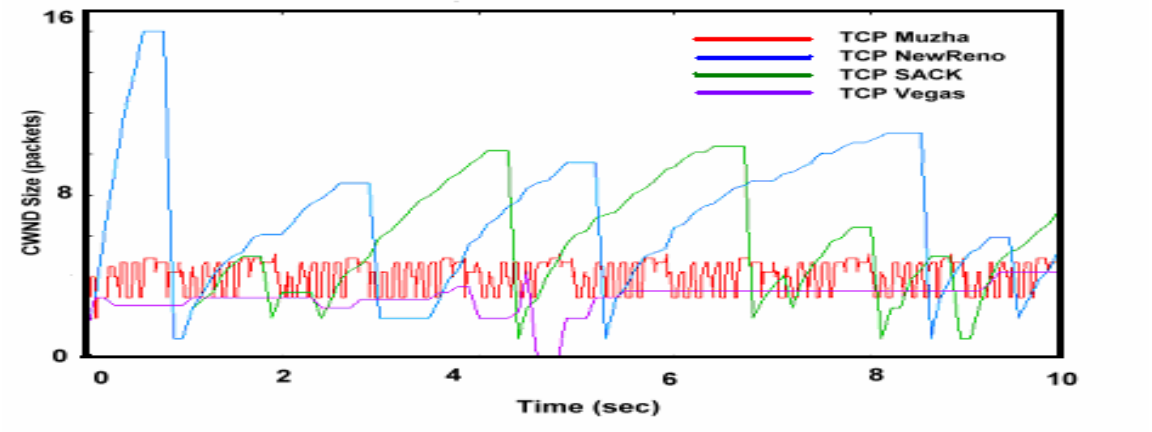

Fig. 3. Change of Congestion Window Size (4-hops)

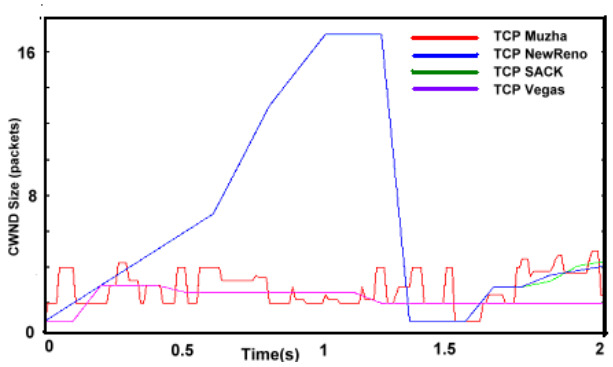

Fig. 4. Change of Congestion Window Size (8-hops)

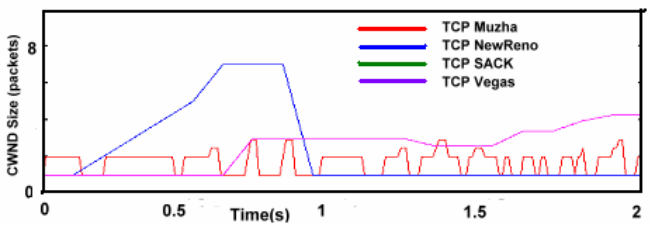

Fig. 5. Change of Congestion Window Size (16-hops)

Simulation 2: Throughput and Retransmission Comparison

In this section, we compare the throughput and retransmission of TCP Muzha with other TCP variants. The network topology and the parameters used in Simulation 2 is the same as those used in Simulation 1. Fig. 6 shows the simulation results from this simulation.

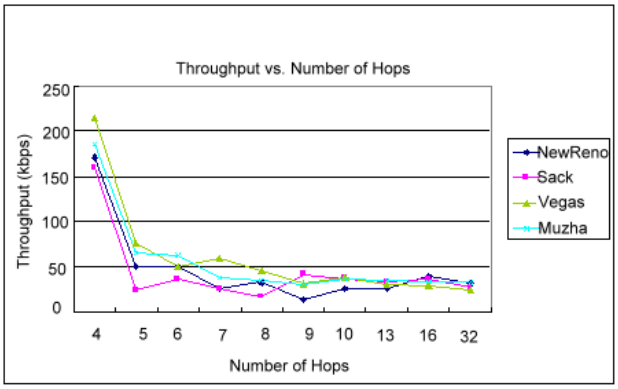

Fig. 6. Comparison of Throughput

From Fig. 6, we observe that TCP Vegas has a higher throughput than other TCP protocols including TCP Muzha when the hop count is less than 8. However TCP Vegas no longer performs well with longer networks (> 8 hops) because TCP Vegas keeps its congestion window size too small. On the other hand, TCP Muzha outperforms TCP NewReno and TCP SACK by 5% ~ 10%. This is because the aggressive window growth of TCP NewReno and TCP SACK cause network overloaded and periodic packet drops which leads to more frequent timeouts in transport layer, more link contentions, and more link failures in MAC layers. TCP Muzha is able to avoid the periodic packet loss due to precise controls of congestion window size according to router feedbacks.

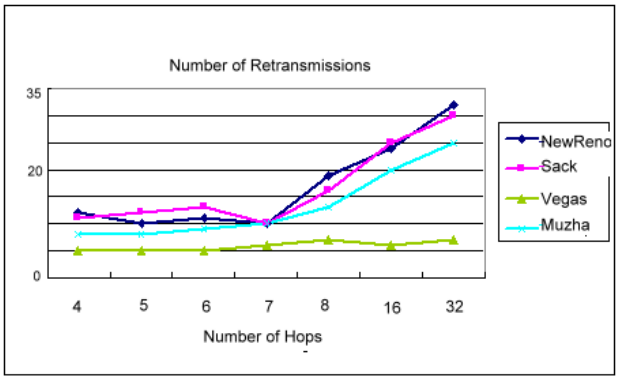

Fig. 7. Comparison of Retransmission

Fig. 7 shows that TCP Vegas causes much less retransmission than other TCP variants. Oppositely, the number of retransmissions in TCP NewReno and TCP SACK is much greater than that in TCP Vegas and TCP Muzha due to their aggressive window mechanisms. With increasing number of hop count, the numbers of retransmissions are all increasing for all three window-based congestion control protocols (NewReno, SACK and Muzha) but TCP Muzha still has the smallest number of retransmissions among three of them.

Simulation 3: Fairness Test

In Simulation 3, we investigate the performance and the fairness issue of TCP Muzha while coexisting with other TCP variants. The topology is shown in Fig. 8 with different setting of hop count. The simulation time is 50 seconds and two sub-simulations were performed: TCP Vegas vs. TCP NewReno and TCP Muzha vs. TCP NewReno. The comparisons of throughput and fairness are showed in Fig. 9 and 10.

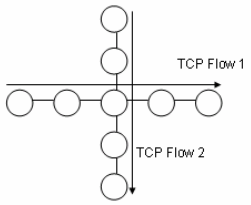

Fig. 8. Cross Topology with 2 TCP flows

Recent research [19] has found that when Reno-style TCP and Vegas perform head-to-head, NewReno generally steals bandwidth from Vegas. As shown in Fig. 9, our simulation shows that TCP NewReno consumes most of the bandwidth and results in the low throughput of TCP Vegas. On the other hand, when our proposed protocol coexists with TCP NewReno, as shown in Fig. 10, TCP Muzha is able to utilize the bandwidth fairly due to precise control of bandwidth usage according to the router feedback. This not only guarantees the fairness requirement but also provides higher throughput.

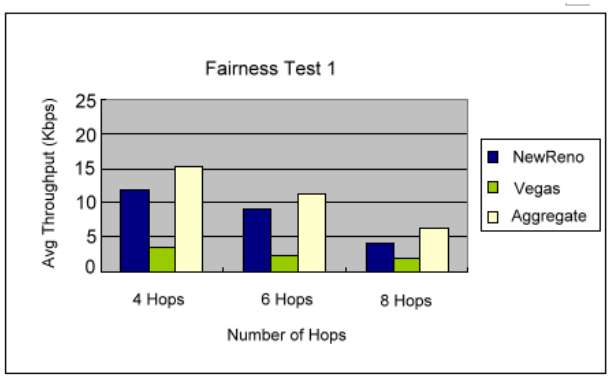

Fig. 9. Fairness Comparison (NewReno vs. Vegas)

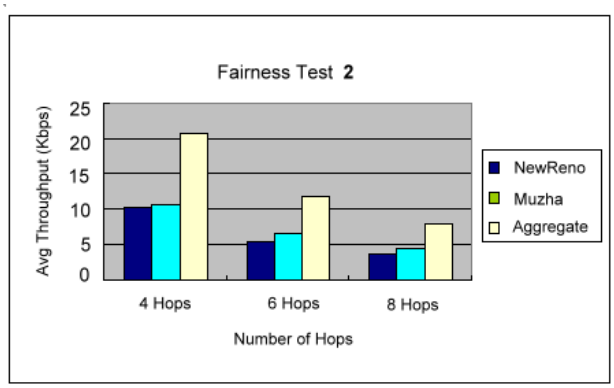

Fig. 10. Fairness Comparison (NewReno vs. Muzha)

V. CONCLUDING REMARKS

From the results obtained from the simulations, we conclude that TCP Muzha can resolve congestion efficiently and has higher average throughput than TCP NewReno. TCP Muzha outperforms TCP NewReno and TCP SACK due to its ability to estimate the available bandwidth more precisely and the ability to deal with random loss. While coexisting with TCP NewReno, TCP Muzha remains a stable performance output and fair utilization of the available bandwidth compared with other major TCP variants. The concept of multi-level data rate adjustment and the details of how to control the size of CWND remain to be investigated. Furthermore, the theoretical formula for DRAI and support of mobility are also essential for future work.

REFERENCES

[1] J. Postel, "Transmission Control Protocol," *IETF RFC 793*, 1981.

[2] V. Jacobson, "Congestion Avoidance and Control," *Proc. of ACM SIGCOMM*, pp. 314-329, Aug. 1988.

[3] V. Jacobson, "Modified TCP Congestion Avoidance Algorithm," Technical report, Apr. 1990.

[4] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," *IETF RFC 2001*, 1997.

[5] D. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, vol.1, pp. 1-14, 1989.

[6] Sally Floyd, T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," *IETF RFC 2582*, 1999.

[7] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," *ACM Computer Communication Review*, vol. 26, no.3, pp. 5-21, 1996.

[8] L. S. Brakmo, S. W. O'Malley, and Larry L. Peterson. "TCP Vegas: New Techniques for Congestion Detection and Avoidance, " *Proc. Of ACM SIGCOMM*, pp. 24-35, Aug. 1994.

[9] Sally Floyd, "TCP and Explicit Congestion Notification," *ACM Computer Communication Review*, 1994.

[10] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transaction on Networking*, vol.1, no.4, pp. 397-413, 1993.

[11] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," *Proc. ACM Mobicom '99*, Seattle, WA, 1999

[12] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "A Feedback based scheme for improving TCP performance in Ad-Hoc wireless networks," in *Proc. of the International Conference on Distributed Computing System (ICDCS 98)*, Amsterdam, Netherlands, May 1998.

[13] J. Liu and S. Singh, "ATCP: TCP for mobile ad hoc networks," *IEEE J-SAC*, vol. 19, no. 7, pp. 1300–1315, 2001.

[14] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, "The impact of multihop wireless channel on TCP throughput and loss," in *Proc. IEEE INFOCOM*, San Francisco, CA, March 2003.

[15] S. Xu and T. Saadawi, "Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks?" *IEEE Communications Magazine*, pp. 130–137, June 2001.

[16] Z. Fu, X. Meng, and S. Lu, "How bad TCP can perform in mobile ad-hoc networks," in *Proc. IEEE ICNP '02*, Paris, France, 2002.

[17] Yi-Cheng Chan, Chia-Tai Chan, and Yaw-Chung Chen, "RoVegas: A Router-based Congestion Avoidance Mechanism for TCP Vegas," *Computer Communications*, vol.27, Issue 16, pp. 1624-1636, Oct. 2004.

[18] R. Carter and M. Crovella, "Measuring Bottleneck Link Speed in Packet-Switched Networks," *International Journal on Performance Evaluation*, pp. 27-28, 1996.

[19] W. Feng and S. Vanichpun, "Enabling compatibility between TCP Reno and TCP Vegas," in Proc. IEEE Symposium on Application and the Internet, pp. 301-308, January 2003.

[20] "The Network Simulator - ns-2", http://www.isi.edu/nsnam/ns/.