Vegas-W: An Enhanced TCP-Vegas for Wireless Ad Hoc Networks

Lianghui Ding, Xinbing Wang, Youyun Xu, Wenjun Zhang, Wen Chen
Department of Electronic Engineering
Shanghai Jiaotong University, China
Email: {lhding, xwang8, xuyouyun,zhangwenjun, Wen Chen}@sjtu.edu.cn

***Abstract*— The performance of TCP-Vegas is not satisfactory in multihop ad hoc networks over IEEE 802.11 MAC protocol. We analyze the problem with a unified network model and simulation results. We observe that the aggregate throughput of all traffics decreases as the load of the network increases. The main reasons lie in Vegas's large minimum congestion window, large reset slow start threshold and aggressive window increase policy. To fix these problems, we propose a modified TCP protocol based on TCP-Vegas for multihop ad hoc networks, called Vegas-W. We extend the congestion window to fraction; change the probing mechanisms of legacy TCP-Vegas in both slow start and congestion avoidance and update slow start threshold tracking the stable window. We evaluate the performance of Vegas-W through *ns*-2. Extensive simulation results under a variety of scenarios show that Vegas-W can improve the throughput up to 87% over legacy TCP-Vegas and up to 27% over FeW, which is another improved algorithm based on TCP-Newreno scenarios.**[1]

I. Introduction

The performance of legacy TCP over multihop ad hoc networks is not satisfactory due to the special features of wireless networks, such as hidden terminal and exposed terminal problems, channel errors, topology variations and routing instability, etc [1]. However, most wireless applications rely on legacy TCP to communicate with the TCP-dominant wired hosts, and it is likely that TCP will remain as the major transport protocol for the clients of 802.11 networks [2]. Hence, the analysis and improvement of TCP performance over multihop ad hoc networks is important and valuable.

Researchers have done much work on improving TCP performance in multihop ad hoc networks in recent yeas [1], [3]–[6]. Most of the previous research mainly considered the problem from the perspective that features of wireless networks and protocols at MAC and routing layers affect the performance of TCP. Packet losses caused by router breakage or transmission errors were distinguished from those due to congestion and treated with different policies. This category of enhancement is suitable for all legacy TCP variations, which consider packet losses as the indication of network congestion.

Since rate control mechanisms of TCP variations are different from each other, their performance in multihop ad hoc networks is not the same. Application of TCP in multihop ad hoc networks requires specific consideration for the particular TCP variation. TCP-NewReno [7] and TCP-Vegas [8] are two widely used transport protocols. For NewReno, K. Nahm etc, recently considered its performance in multihop ad hoc networks and proposed an improved algorithm FeW [2]. As to Vegas, only some experimental results were presented in [9] [11]. Those results showed that Vegas outperforms other TCP protocols in most scenarios for its window adjustment based on round trip time (RTT) variations. To the best of our knowledge, there has been no literature considering the specific features of Vegas in multihop ad hoc networks in depth and proposing related improved algorithms.

In this paper, we analyze the behavior of Vegas in multihop ad hoc networks with a unified network framework and simulation results. It is concluded that large minimum window, large reset slow start threshold W_{th}^r and aggressive window increase policy are main reasons of the low throughput. Based on the analysis, we propose a modified Vegas, called Vegas-W, which improves TCP-Vegas in four aspects for multihop wireless circumstances: (i) We extend the congestion window to fraction with a rate control timer under the TCP sending process; (ii) We change the probing mechanism in slow start phase by keeping congestion window constant until the number of received ACKs is larger than a threshold; (iii) Window increase in congestion avoidance phase is also changed similar to that in slow start phase, but with a larger threshold; (iv) We update slow start threshold W_{th} to keep track of the stable window. The simulation results show that Vegas-W improves throughput of Vegas significantly over a wide variety of scenarios.

The rest of the paper is organized as follows. The background information about TCP-Vegas and related work is described in Section II. Section III presents analytical model and our Vegas-W algorithm. We evaluate the performance of Vegas-W through *ns*-2 in Section IV. This paper is concluded in Section V.

II. Background and Related Work

A. TCP-Vegas

TCP-Vegas [8] adjusts its congestion window according to the phases it performs and the gap between the real and estimated sending rates. The estimation is done once per round trip time RTT. Vegas sets $BaseRTT$ to the minimum of all measured RTTs and computes the $Expected$ rate as $Expected = W/BaseRTT$, where W denotes the window size. Let RTT_a denote the average measured RTT, then Vegas

[1]This work is supported by NSF China (No. 60702046).

calculates the $Actual$ rate as $Actual = W/RTT_a$. The gap between the real and estimated sending rates is defined as $\Delta = (Expected - Actual) * BaseRTT$. Three thresholds α, β and γ are defined in Vegas. TCP senders compare Δ with γ in slow start phase and with α, β in congestion avoidance phase to determine window adjustments.

In slow start phase, congestion window is smaller than slow start threshold W_{th}. When receiving a new ACK, if Δ is less than γ, the TCP sender increases w by one. If not, the sender will decrease window size by a specific percentage p, sets W_{th} to be the reset value W_{th}^r, and switches to congestion avoidance phase.

When TCP sender is in congestion avoidance phase and receives a new ACK, Vegas increases window by $1/W$ if Δ is less than α, decrements it by $1/W$ if Δ is larger than β, and keeps it unchanged when Δ falls between α and β.

More details about Vegas such as fast retransmit and fast recovery etc. refer to [8].

B. Related Work

Previous research on TCP in multihop ad hoc networks was mainly on distinguishing between packet losses caused by router breakage or transmission errors and those due to real network congestion. The retransmit timer was fixed after retransmitting once rather than increasing exponentially in [12]. Feedback at network layer was adopted for notification of routing breakage in [1]. Upon receiving route failure messages, TCP sources snooze all its variables and stop transmitting anymore to shield the effects of multihop ad hoc networks.

In the meanwhile, interaction between TCP layer and link layer has been considered in many documents. MAC layer control frames were utilized to conduct network layer's flow control in [13]. It was observed in [15] that RTT is limited by the number of round trip hops. Fu *et al.* showed that the maximum throughput can be achieved under a specific window limit in [3] and LRED algorithm was adopted for dropping packets. Zhai *et al.* reported that TCP window may be less than one in multihop ad hoc networks and rate based transport control based on the channel busy ratio was proposed to improve the performance [6].

In addition, some research has been done on analyzing and improving the performance of a particular TCP variation in multihop ad hoc networks. Performance of different TCP variations with distinct routing protocols is compared in [9] and [11]. K. Nahm *et al.* observed the interaction between TCP, routing and MAC layers and proposed FeW with a small window increase step based on TCP-NewReno.

However, the performance of TCP-Vegas in multihop ad hoc networks has not been studied in depth. Interaction between Vegas ends and the network, and that between TCP-Vegas and MAC protocols have not been totally addressed. Hence, in this paper, we investigate the behavior of Vegas in multihop ad hoc networks and propose Vegas-W aiming at improving its throughput, and our approach could be applied to the sensor networks as well [10], [14].

III. VEGAS-W

A. Unified Network Framework

Interaction between TCP ends and network can be described with the unified network framework shown in Figure 1, which consists of three components: TCP sources, network and TCP receivers. Network is a container of data and ACK packets with limited resources, which drop packets according to the load with related stochastic probability. All the TCP sources compose the load 'injection' to the network with rate λ_d. All the TCP receivers feedback ACKs to senders through the network with rate λ_a. TCP sources adjust the sending rates, which are controlled by congestion windows, based on the states of ACKs (lost, new or dup) and variations of RTT, and then change the load of the network accordingly.

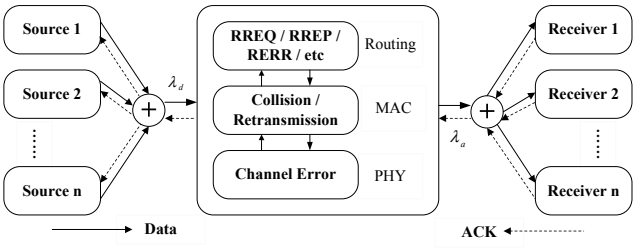


Fig. 1: Model of unified network framework.

Since the rate of ACK packets λ_a is determined by the data rate λ_d , the load of the network is determined by sending rates of all the TCP sources. The capacity of network is limited by physical channel states, wireless spatial reuse ratio, MAC contention resolution algorithms, and routing protocols, etc. According to the results in [3], there should be a load threshold of the network on which maximum throughput can be achieved. Dividing the load threshold by packet size and the number of TCP flows, we define W^* for each flow as the optimal congestion window, on which the aggregate throughput of all flows is maximized.

B. Interaction between TCP ends and network

Follow we analyze the interaction between TCP ends and the network based on simulation results with distinct load on the network, where the network load increases as the number of flows increases. We focus on only one flow, while other flows are considered as background traffics. Throughput of all the flows, that of one flow and the aggregate throughput after some simple modifications are shown in Table I.

TABLE I: Throughput over 7 hops chain topology

flow numbers	1	2	4	8
whole througput(Kbps)	206.8	165.3	102.4	79.4
throughput of one flow (Kbps)	206.8	81.62	25.51	10.16
modified throughput (Kbps)	192.6	197.2	177.8	138.1

Intuitively, the capacity of the network should be shared by all the flows and the whole throughput should keeps almost constant. But the results show that the whole throughput of

all the flows and that of flow 0 decreases as the number of flows increases. This is caused by large minimum congestion window, large reset slow start threshold W_{th}^r and aggressive window increase policy. They induce overload to the network, thus increase MAC collision and overhead of routing re-establishment, which finally reduce the throughput.

We analyze the problem via the window size distribution of one flow as shown in Figure 2. The zero probability of $w = 1$ is due to that the minimum window is 2 here. TCP source with only 1 flow spends about 99% of all the simulation time on $w = 3$ and can stabilize on it. The throughput in the scenario with only 1 flow is close to the maximum throughput analyzed in [16], hence we approximate the optimal window size W_1^* for 1 flow is 3. For other scenarios with multiple flows, the optimal window will be W_1^* divided by the number of flows.

For scenario with 8 flows, W_8^* will be less than 1. However, from Figure 2, the TCP source spends about half of the time on $W = 3$. This is because both the minimum window and reset slow start threshold W_{th}^r are 2, which pushes the window to increase to larger than 2.

For scenarios with 2 and 4 flows, the optimal window is around 1. Assuming large minimum window is the only reason for poor performance, it should be improved when the minimum window is set to be 1. However, the window size distribution is similar (the detailed simulation results are omitted here for space limit), while when we set both minimum window and slow start threshold to be 1, much better throughput is shown in the $4th$ row of Table I. The results show that joint activity of large minimum window and large reset slow start threshold rather than each independent one is the main reason for poor performance.

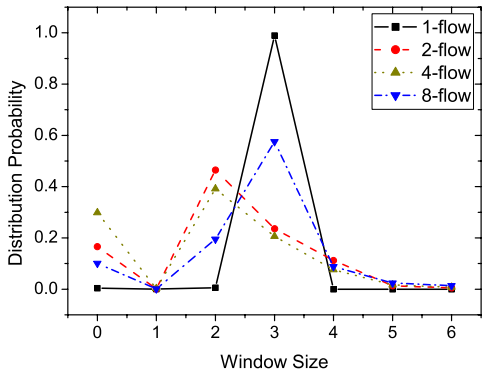


Fig. 2: Window size distribution of flow 0 over a 7-hop chain.

In addition, aggressive window increase policy aggravates the problem. Although minimum window 2 is large, the throughput will be higher than current values, if TCP sources can fix the windows on the minimum ones. But TCP sources spend much time on window 3 and 4, as shown in Figure 2, because congestion window is increased when receiving a new ACK and the value of Δ meets the requirement. But RTT is distributed in a large area as shown in our technical report [**?**]. Thus increasing W based on only one Δ meeting the requirement is too aggressive.

C. Vegas-W

Based on the observation above, we propose our Vegas-W algorithm. Its improvement consists of four aspects: fractional window support, slower start, moderate congestion avoidance, and W_{th} update. With fractional window support, window is extended to less than 1, which aims at reducing the impact of large minimum window problem. Slower start and moderate congestion avoidance increase window based on more than one Δ meeting requirement of thresholds to solve the aggressive window increase problem. Besides, speeds of window increase in slow start phase and congestion avoidance phase are distinguished for different features. W_{th} update avoids too slow window increase when W^* is larger than W_{th}^r.

1) Fractional Window Support: In wireless ad hoc networks, W^* may be fractional, and even less than one, which requires that TCP sources send fractional number of packets in one RTT. We assume that N_{fw} consecutive timeout events occurring with window equal to one is an indication of fractional window requirement. For simplicity, window is set to be 0.5 after the indication. A rate control timer is placed under the TCP sending process to schedule the transmission.

2) Slower Start: Probing the network with more than one RTT and keeps window unchanged until the number of received ACKs is larger than a threshold. Let n_s denote the number of received ACKs satisfying $\Delta < \gamma$ in slow start phase. Let N_s denote the window increase threshold. When n_s is less than N_s and calculated Δ per RTT is less than γ, window keeps constant and n_s increases by 1 . When n_s is larger than N_s, the window increases by 1, and n_s is reset to be 0. Window is decreased by percentage p when Δ is larger than γ. The *Slower Start* process can be described as in (1).

$$W = \begin{cases} W & , \text{ if } \Delta < \gamma \ \& \ n_s <= N_s \\ W + 1 & , \text{ if } \Delta < \gamma \ \& \ n_s > N_s \\ W \times (1 - p) & , \text{ if } \Delta >= \gamma \end{cases} \quad (1)$$

3) Moderate Congestion Avoidance: Let n_{CA} denote the number of received ACKs satisfying $\Delta < \alpha$ in congestion avoidance phase. Let N_{CA} denote the window increase threshold. When n_{CA} is less than N_{CA} and Δ is less than α, window keeps constant and n_{CA} increases by 1. When n_{CA} is larger than N_{CA}, the congestion window increases by $1/W$, and n_{CA} is reset to be 0. When Δ falls between α and β, n_{CA} keeps unchanged. Other operations is same to Vegas. The Moderate Congestion Avoidance can be described as in (2).

$$W = \begin{cases} W + \frac{1}{W} & , \text{ if } \Delta <= \alpha \ \& \ n_{CA} > N_{CA} \\ W & , \text{ if } \alpha < \Delta < \beta \\ & \quad or \ \Delta <= \alpha \ \& \ n_{CA} <= N_{CA} \\ W - \frac{1}{W} & , \text{ if } \Delta >= \beta \end{cases} \quad (2)$$

With *Moderate Congestion Avoidance*, we slow down the window increase speed in congestion avoidance phase and try to make the window stabilize on the optimal window W^* for long time with large window increase threshold N_{CA}.

In the congestion avoidance phase, TCP sources have estimated the bandwidth of the network and try to oscillates

around the stable window, while in the slow start phase, TCP sources probe the network with no prior information. Hence the increase step should be different and distinct window increase thresholds N_s and N_{CA} are used to reflect distinct features of the two phases.

4) W_{th} Update: In legacy Vegas, W_{th} is set to be one half of the window when timeout occurs. This is not suitable in wireless networks, because packets can be dropped for physical transmission errors or collision at the MAC layer rather than congestion. W_{th} should not always be halved strictly when timeout occurs. Besides, the window increase step of Vegas-W is slower than Vegas after application of *Moderate congestion avoidance*, and large gap between W_{th} and W^* will waste bandwidth.

Hence, W_{th} *update* proposed here adjusts W_{th} tracking W^*. For estimating the real optimal window W^* is difficult, stable window is taken as W^* when TCP sources stabilize on it for time longer than a threshold, where the time is normalized by RTT. W_{th} *update* comprises two parts: one is for W_{th} reset when receiving a new ACK, another is for W_{th} decrement when timeout occurs.

Algorithm 1: W_{th} **Update When Receiving a New ACK**
Initialization: Receiving a new ACK with sequence number equal to or larger than beg_seqno
Method:
1. Calculate $Expected$ and $Actual$
2. Calculate the integral number of packets congested: $\Delta = (int)((Expected - Actual) * baseRTT + 0.5)$
3. **IF** $w < W_{th}$
4. **IF** $\Delta > \gamma$
5. $W_{th} \leftarrow$ current window minus 1
6. **END**
7. **ELSE**
8. **IF** $\Delta < \alpha$
9. ns_{CA} keeps unchanged
10. **END**
11. **IF** $\Delta > \beta$
12. $ns_{CA} \leftarrow 0$
13. **END**
14. **IF** $\alpha \leq \Delta \leq \beta$
15. **IF** $Currentwindow > W_s$
16. $ns_{CA} \leftarrow 0$
17. $W_s \leftarrow$ current window
18. **END**
19. $ns_{CA} \leftarrow ns_{CA} + 1$
20. **IF** $ns_{CA} > NS_{CA}$
21. $W_{th} \leftarrow w$
22. $ns_{CA} \leftarrow 0$
23. **END**
24. **END**
25. **END**
26. $beg\_seqno \leftarrow$ sequence number of the ACK

Fig. 3: W_{th} update when receiving a new ACK.

Let ns_{CA} denote the number of stable RTT periods on the window W_s in congestion avoidance phase. When Δ is between α and β, ns_{CA} increases by 1, or it keeps constant. When ns_{CA} is larger than a threshold NS_{CA}, W_{th} is set to be the window size and ns_{CA} is set to be 0. When the current window is larger than W_s, ns_{CA} is set to be 0 and W_s is set to be the current window size. W_{th} update when receiving a new ACK is shown in Figure 3.

Let to_{ss} denote the number of consecutive timeout events with window less than W_{th}. When to_{ss} is larger than a threshold TO_{ss}, W_{th} decreases by 1 and to_{ss} is set to be 0.

IV. Simulation and Results

In this section, we evaluate Vegas-W and compare it with legacy TCP-Vegas and FeW over a wide variety of scenarios. Due to the space limit, we only present the simulation results with DSR routing protocol over chain topology here.

A. Simulation Setup

We implement our algorithm into *ns*-2simulator and evaluate the performance with DSR routing protocol, which is a widely used on demand routing protocols. IEEE 802.11 is set as the MAC layer protocol. Both basic rate and data rate at the 802.11 MAC layer are set to be 2Mbps. Other parameters are kept as default in *ns*-2. Long-lived TCP flows (of 500 seconds) are evaluated over different topologies. The receiving range is 250m and carrier sensing range is 500m. The wireless propagation model is the two ray ground model. The packet size is 1024 bytes. The three thresholds α, β and γ are set to be 1, 3 and 1, and parameter p in slow start phase is set to be $1/8$ as default. Selection of parameters in Vegas-W should consider the tradeoff between algorithm dynamics and network stabilization. Based on simulation results, we choose the value as follows. N_{fw} for fractional window decision is set to be 2. N_s and N_{CA} for moderate window increase are set to be 10 and 100 to distinguish window increase policies in different phases. The two parameters NS_{CA} and TO_{ss} for W_{th} update are set to be 100 and 2, respectively.

B. Simulation Results

Chain is a common topology with limited resources and shared by different TCP flows. We examine Vegas-W algorithm over chain topology with the number of hops from 4 to 16. Scenarios with 1, 2, 4 and 8 TCP flows are evaluated over the chain, which denote different load of the network. All flows are from a same source node to an identical destination node. We plot the throughput in Figures 4(a). Vegas-W obtains significant throughput improvement compared to TCP-Vegas and FeW.

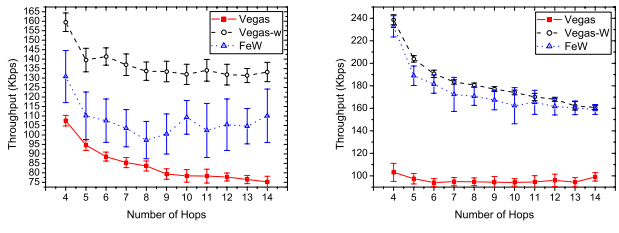
We can see that in Figure 4(a), when the number of flows is 8 and network load is heavy, throughput of Vegas-W is better than that of Vegas by about 62% and that of FeW by about 22% on average. Based on above results, the optimal window W_8^* for 8 flows is less than one. Fractional window support of Vegas-W extends congestion window to less than 1. W_{th} update adjusts W_{th}^r to 1 and slows down the window increase speed. Combination of these two new mechanisms obtains high throughput gain.

When the number of flows is 4 and network load is moderate, the throughput of Vegas-W is better than that of Vegas by about 87% on average, while is similar to FeW.

The optimal window W_4^* is less than and close to 1, while the initial window of legacy Vegas is 2, which induces more packets on flight than the network capacity. The modification of minimum window from 2 to 1, slower start and moderate congestion avoidance make TCP sources inject appropriate load to the network and improve the throughput. This makes TCP-Vegas similar to TCP-NewReno.

When the number of flows is 2 and network load is slightly heavy, throughput of Vegas-W is larger than Vegas about 27% and similar to FeW. The reason is same with that in the scenario with 4 flows.

When there is only one flow over the chain and the load is light, both Vegas-W and Vegas almost obtain same throughput and is larger than FeW by about 7%. For this scenario, W_1^* is larger than $W_{th}^r = 2$ of legacy Vegas, and TCP sources can stabilize on it in most of the time. So loss probability will be small and the system will run fluently. Separation of different window increase thresholds in slow start and congestion avoidance phases and W_{th} update mechanisms of Vegas-W reserve this stable property of Vegas and make Vegas-W obtain close throughput better than FeW.



(a) Eight flows (b) Four flows

(c) Two flows (d) One flow

Fig. 4: Throughput comparison over chain topology with DSR and 95% confidence interval.

V. Conclusion

In this paper, we analyze the poor performance of Vegas over wireless ad hoc networks via simulation together with a unified network framework. We identify the major reasons as large minimum window, large reset slow start threshold W_{th}^r and aggressive window increase policy. We then propose a new algorithm, called Vegas-W, which considers the special features of wireless ad hoc networks and improves legacy Vegas in four aspects: fractional window support, slower start, moderate congestion avoidance and W_{th} update, respectively. We evaluate Vegas-W through *ns*-2, and compare it with legacy Vegas and FeW with DSR over the chain topology in this paper with different network load. The simulation results show that Vegas-W obtains higher throughput than Vegas up to 87% and outperforms FeW up to 27%.

Acknowledgment

This work is supported by NSF China (No. 60702046, 60625103), and Science and Technology Commission of Shanghai Municipality (STCSM) under the grant NO. 05DZ22102; China Ministration of Education (No. 20070248095); Shanghai Jiaotong University Young Faculty Funding; Shanghai Jiaotong University Pre-Research Funding; Qualcom China Research Grant.

References

[1] G. Holland and N. Vaidya, "Analysis of tcp performance over mobile ad hoc networks," in *Proc. IEEE/ACM MOBICOM*, Aug 1999.

[2] K. Nahm, A. Helmy and C. -C. Jay Kuo, "TCP over multihop 802.11 networks: Issues and performance enhancement," in *Proc. ACM MobiHoc*, Urbana-Champaign, IL, USA, May 2005.

[3] 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, USA, Mar 2003.

[4] K. Xu, M. Gerla, L. Qi and Y. Shu, "Enhancing tcp fairness in ad hoc wireless networks using neighborhood red," in *Proc. MobiCom*, San Diego, California, USA, Sep 2003.

[5] Z. Fu, X. Meng and S. Lu, "How bad tcp can perform in mobile ad hoc networks," in *IEEE International Symposium on Computers and Communications (ISCC'02)*, Taormina, Italy, Jul 2002.

[6] H. Zhai, X. Chen and Y. Fang, "Rate-based transport control for mobile ad hoc networks," in *Proc. IEEE WCNC*, New Orleans, LA USA, Mar 2005.

[7] S. Floyd and T. Henderson, "RFC2582: The NewReno modification to TCP's fast recovery algorithm," Apr 1999.

[8] L. S. Brakmo, S. W. O'Malley and L. L. Peterson, "TCP vegas: New techniques for congestion detection and avoidance," in *Proc. ACM SIGCOMM*, Oct 1994.

[9] M. Yahia, J. Bíró, "Behavior of TCP algorithms on ad-hoc networks based on different routing protocols (MANETS) and propagation models," in *Proc. ICWMC*, Bucharest, Romania, Jul 2006.

[10] Mo Li and Yunhao Liu, "Rendered Path: Range-Free Localization in Anisotropic Sensor Networks with Holes", ACM MobiCom 2007, Montreal, Quebec, Canada, September 2007.

[11] M. Berger, S. Lima, A. Manoussakis, J. Pulgarin and B. Sanchez, "A performance comparison of TCP protocols over mobile ad hoc wireless networks," in *Proc. CERMA*, Cuernavaca, Mexico, Dec 2006.

[12] T. D. Dyer and R. V. Boppana, "A comparision of TCP performance over three routing protocols for mobile ad hoc netwoks," in *Proc. ACM MobiHoc*, Oct 2001.

[13] H. Zhai and Y. Fang, "Distributed flow control and medium access in multihop ad hoc networks." *IEEE Trans. Mobile Comput.*, vol. 5, no. 11, pp. 1503–1514, 2006.

[14] Mo Li and Yunhao Liu, "Underground Structure Monitoring with Wireless Sensor Networks", ACM/IEEE IPSN, Cambridge, Massachusetts, USA, April, 2007.

[15] K. Chen, Y. Xue and K. Nahrstedt, "On setting TCP's congestion window limit in mobile ad hoc network," in *Proc. IEEE ICC*, Anchorage,Alaska, May 2003.

[16] J. Li, C. Blake, D. S. J. De Couto, H. I. Lee and R. Morris, "Capacity of ad hoc wireless networks," in *Proc. of MobiCom*, Rome, Italy, Jul 2001.