Performance Evaluation of TCP Variants over Routing Protocols in Multi-hop Wireless Networks

Mohit P. Tahiliani
Department of Computer Engineering
N.I.T.K., Surathkal,
India
tahiliani.nitk@gmail.com

K. C. Shet
Department of Computer Engineering
N.I.T.K., Surathkal,
India
kcshet@nitk.ac.in

T. G. Basavaraju
Department of Computer Engineering
A.I.T., Bangalore,
India
tg.braju@gmail.com

***Abstract*—Wireless internet has become popular in recent years due to the tremendous growth in the number of mobile computing devices and high demand for continuous network connectivity regardless of physical locations. In this paper, we investigate the effects of routing protocols on the performance of Transmission Control Protocol (TCP) variants in multi-hop wireless networks. Through simulations we study the effects of Destination Sequenced Distance Vector (DSDV), Optimized Link State Routing (OLSR), Ad hoc On demand Distance Vector (AODV) and Dynamic Source Routing (DSR) routing protocols on TCP Tahoe, TCP Reno, TCP Newreno, TCP with Selective Acknowledgment (SACK) option and TCP Vegas. The simulations are carried out for static as well as mobile nodes. The performance metric used is throughput. Another metric, *expected throughput* is used for the comparison of throughput when nodes are mobile.**

Keywords: Multi-hop wireless networks, TCP, Tahoe, Reno, Newreno, SACK, Vegas, DSDV, OLSR, AODV, DSR.

I. INTRODUCTION

Tremendous growth in the number of mobile computing devices and high demand for continuous network connectivity has resulted in widespread deployment of multi-hop wireless networks. Applications of multi-hop wireless networks range from broadband home networking, community networking and enterprise networking to medical systems, security surveillance systems, transportation systems, defense and building automation [14].

The key requirement of any network is reliable delivery of data. Transmission Control Protocol (TCP) which has been designed and fine tuned for wired networks is largely affected by the dynamic nature of multi-hop wireless networks. TCP assumes packet loss as a sign of congestion and hence, reduces its data sending rate. However, in wireless networks, congestion is not the only reason for a packet loss in the network. Link failures, collisions, hand-offs, etc are other possible reasons for a packet loss in wireless networks. TCP is unable to differentiate packet losses due to congestion and packet losses due to reasons such as link failures, collisions, etc and thus, results in overall performance degradation.

A lot of research on multi-hop wireless networks has focused on the performance issues of TCP [2, 3, 4, 5, 6, 9] in wireless networks. In this paper we study the effects of routing protocols on the performance of TCP Tahoe, TCP Reno, TCP Newreno, SACK TCP and TCP Vegas in static as well as mobile multi-hop wireless networks.

TCP implements congestion control mechanisms such as Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery [1]. Tahoe, Reno and Newreno are based on these four congestion control mechanisms (Tahoe does not implement Fast Recovery though) whereas Vegas implements a modified slow start algorithm and a different retransmission policy [8]. Selective acknowledgment (SACK) [7] mechanism modifies Fast Recovery mechanism to efficiently recover from the state of congestion.

We choose two proactive routing protocols: Destination Sequenced Distance Vector (DSDV) and Optimized Link State Routing (OLSR) as well as two reactive routing protocols: Ad hoc On demand Distance Vector (AODV) and Dynamic Source Routing (DSR) for our study since they are accepted as the standard routing protocols for multi-hop wireless networks [12]. The performance of these routing protocols varies depending on the network topology [12], which has a major impact on the performance of TCP.

The remainder of the paper is organized as follows. In Section II we discuss the related work done in the analysis of TCP variants. In Section III we brief about the congestion control mechanisms in Tahoe, Reno, Newreno, SACK and Vegas. Section IV presents the simulation environment designed for multi-hop wireless networks using network simulator - 2 (ns-2) and performance metrics used for the study. Section V discusses the simulation results in detail. Section VI gives conclusions and possible future directions.

II. RELATED WORK

G. Holland et al [3] studied the effects of mobility on TCP Reno's performance in mobile ad hoc networks. Only one routing protocol, DSR is used for the simulations. Authors introduce a new metric called *expected throughput* to compare the performance by measuring the differences in throughput when the number the hops vary. We have used this metric for our work.

The new metric *expected throughput* is also used by A. Jain et al [6] to compare the performance of Reno, SACK and Vegas. Relative performance of two reactive routing protocols - Ad hoc On demand Distance Vector (AODV) and Dynamic Source Routing (DSR) with respect to TCP variants is also studied in static as well as mobile topologies. However, DSDV routing protocol is not considered which in fact, leads to better TCP performance in static topologies. We discuss this in more detail in Section V.

978-1-4244-9034-/10/$26.00©2010 IEEE

DSDV routing protocol is used by M. Gerla et al [2] to study the performance of TCP in wireless multi-hop networks by investigating the interaction between TCP and MAC layer. The TCP used by authors is TCP Tahoe. Three different types of static topologies: a string topology, a ring topology and a grid topology are used for performance evaluation. We have designed a similar string topology for our study.

Performance of TCP Tahoe in mobile ad hoc networks is analyzed with respect to the underlying routing protocols such as AODV, DSR, DSDV and Signal Stability Adaptive (SSA) in [9]. OLSR protocol, however, is not considered in [9].

In [5], K. Chandran et al study the effect of route failures on TCP's performance in ad hoc wireless networks. The paper describes the drawbacks of TCP mechanism which lead to packet loss in the network and degrade the overall performance. Also a feedback based scheme is proposed to improve the performance of TCP in ad hoc wireless networks.

The performance study of TCP variants has also been carried out on different types of networks such as traditional wired networks, mobile ad hoc networks, WiMAX Mesh networks, Satellite environments, etc.

III. TCP VARIANTS

A series of congestion collapses in internet is observed for the first time in October 1986 [1]. In 1988, Van Jacobson proposed three algorithms for congestion avoidance and control: Slow Start, Congestion Avoidance and Fast Retransmit. Later, a data recovery algorithm called Fast Recovery [10] was also proposed by Jacobson. These four algorithms are fundamental congestion control algorithms and are included in modern TCP implementations. However, above mentioned congestion control algorithms have also undergone several modifications to improve the performance of TCP on wired as well as wireless networks.

A. TCP Tahoe

TCP Tahoe implementation is based on Slow Start, Congestion Avoidance and Fast Retransmit algorithms. Slow Start algorithm is designed to initiate the "Self-Clocking" [1] mechanism of TCP. A variable called *congestion window (cwnd)* is used to provide an upper limit to the amount of data that can be sent to the receiver. The sender can send only minimum of *cwnd* and the receiver advertised window (which is used for flow control) worth of packets. Slow Start tries to reach the equilibrium quickly (a TCP connection is said to be in equilibrium if it is running stably with a full window of data in transit) by exponentially increasing the *cwnd*.

Congestion Avoidance algorithm closely obeys "Conservation of Packets" principle [1] once the connection reaches equilibrium. A variable called *ssthresh* (slow start threshold) is maintained to ensure that *cwnd* does not increase exponentially once the connection reaches equilibrium. After reaching *ssthresh*, *cwnd* is increased linearly rather than exponentially.

In Fast Retransmit algorithm, after receiving a small number (generally 3) of duplicate acknowledgements [7] for the same TCP packet, the sender infers that a packet has been lost. Hence it reduces *cwnd* to half and retransmits the lost packet without waiting for a retransmission timer to expire. Each time a packet loss is encountered; Slow Start algorithm is triggered till *cwnd* reaches *ssthresh*.

The drawback of TCP Tahoe is that if *ssthresh* value is large, it takes more time to reach the equilibrium, resulting in low throughput in the network. However, if *ssthresh* value is small, exponentially increasing *cwnd* has an advantage that the time required to reach equilibrium is less. Thus, in the latter case, resources of a network are efficiently utilized, improving the overall performance of the network.

B. TCP Reno

TCP Reno retains all the algorithms implemented in TCP Tahoe. However, a new algorithm called the Fast Recovery algorithm is also implemented in TCP Reno. Fast Recovery algorithm considers a duplicate acknowledgement as an indication that a packet has left the network. Hence, when a sender receives three duplicate acknowledgements, it retransmits the lost packet, updates *ssthresh*, and reduces *cwnd* by half as in Fast Retransmit.

Fast Recovery algorithm, however, tries to estimate the amount of outstanding data in the network and increases *cwnd* by one packet for each duplicate acknowledgement received. Thus it maintains the flow of traffic rather than restarting the flow using Slow Start as in Tahoe. However, if multiple packets are lost from one window of data, TCP Reno waits for retransmission timer to expire, retransmits the packet and goes into Slow Start mode. This happens because for each packet loss, Reno enters Fast Recovery phase, reduces *cwnd* and exits Fast Recovery phase on receiving a partial acknowledgement (acknowledges some but not all of the outstanding packets) [11]. After multiple such reductions, *cwnd* becomes so small that there will not be enough duplicate acknowledgements to trigger Fast Recovery algorithm. Hence the retransmission timer expires.

The major drawbacks of TCP Reno are: (i) overall performance of the network degrades significantly due to frequent retransmission timeouts when multiple packets are lost from one window of data and (ii) Recovery from congestion is slow since a maximum of one lost packet can be retransmitted per Round Trip Time (RTT) (i.e. each time a duplicate acknowledgment is received) when the connection is in Fast Recovery phase.

C. TCP Newreno

TCP Newreno implementation overcomes the drawback of TCP Reno by modifying the Fast Recovery algorithm [7] to eliminate Reno's wait for a retransmit timer when multiple packets are lost from one window of data.

In TCP Newreno, when a sender receives a partial acknowledgement, it does not come out of Fast Recovery phase as in Reno. Instead, it retransmits the lost packet and continues to be in Fast Recovery phase. Thus, in a multiple packet loss scenario Newreno does not reduce *cwnd* multiple times by entering and exiting Fast Recovery phase multiple times. It stays in Fast Recovery phase till all the packets of the same window are acknowledged. Thus, Newreno overcomes one

drawback of Reno by avoiding many of the retransmit timeouts when multiple packets are lost from one window of data. However, both Reno and Newreno can retransmit at most one lost packet per RTT when the connection is in Fast Recovery phase.

The major drawback of Newreno is that the constraint of retransmitting at most one lost packet per RTT results in substantial delay in retransmitting the later dropped packets in the window [7]. Thus the available bandwidth is not effectively utilized.

D. TCP with Selective Acknowledgment (SACK) option

Congestion control algorithms implemented in SACK TCP [7] are extension of TCP Reno's congestion control algorithms. SACK TCP uses TCP Reno's congestion control algorithms to increase and decrease the size of *cwnd*. However, it makes minimal changes to other congestion control algorithms like Fast Recovery and Fast Retransmit.

The main difference between SACK TCP and TCP Reno is in the behavior when multiple packets are lost from one window of data. SACK overcomes the drawback of Reno and Newreno by estimating which packets have been successfully delivered. Thus it improves overall throughput of the network by avoiding unnecessary delays in retransmitting the lost packets.

When congestion is detected by the loss of a data packet, SACK TCP enters Fast Recovery phase as in TCP Reno. It retransmits the lost packet and reduces the *cwnd* by half. To estimate the number of outstanding packets in the network, SACK TCP uses a variable called *pipe* [7]. A new packet is transmitted by SACK TCP only if the value of *pipe* is less than the value of *cwnd*.

The value of *pipe* is incremented by one if a lost packet is retransmitted or if a new packet is transmitted and decremented by one if a duplicate acknowledgment is received with a SACK option. SACK TCP source maintains a data structure called *scoreboard* [7] to keep track of the acknowledgments from previous SACK options. When partial acknowledgment is received, *pipe* is decremented by two packets instead of one packet. A detailed description of mechanisms of Tahoe, Reno, Newreno and SACK is provided in [7].

E. TCP Vegas

TCP Vegas implementation tries to detect the incipient stages of congestion before packet losses occur. It uses proactive mechanisms to increase and decrease the size of *cwnd*. Other TCP variants assume packet loss as a sign of congestion in the network whereas TCP Vegas uses the difference in the expected RTT [8] and the actual RTT [8] to adjust the *cwnd* size [8]. Thus, the performance of TCP Vegas largely depends on the accuracy of RTT estimation.

A modified Slow Start algorithm is implemented in TCP Vegas to start the "Self-Clocking" mechanisms of TCP. It also has a new retransmission policy which retransmits the lost packet after receiving one (rather than three) duplicate acknowledgement if the estimated RTT is greater than the retransmission timeout value. Brakmo et al [8] provides a detailed description about the mechanisms of TCP Vegas.

The major drawback of TCP Vegas is that it lacks mechanisms that handle rerouting of connection [13]. Rerouting a path may change the RTT of the connection and may affect the accuracy of RTT estimation. If the new route has shorter RTT, *cwnd* size will be increased but it does not degrade the performance of TCP Vegas [13]. However, if the new route has longer RTT, TCP Vegas incorrectly assumes that the increase in RTT is due to congestion and thus reduces its *cwnd* size, resulting in substantial decrease in throughput [13].

IV. SIMULATION SETUP AND METHODOLOGY

The results in this paper are based on the simulations done on ns-2 [16], a discrete event simulator. We have chosen static as well as mobile topologies for the study.

A. Static topologies

We have designed a linear string topology of 8 nodes, similar to that in [2]. We consider a single TCP connection that covers a variable number of hops, from 1 to 7. The nodes are configured to use 802.11 MAC protocol. The distance between two nodes is equal to the transmission range which is by default set to 250 meters. Two-ray ground reflection model is used as a radio propagation model. The channel data rate is 2 Mbps. Keeping all the above mentioned parameters fixed, we switch the TCP protocol and the routing protocol. TCP packet size is fixed to 1500 bytes and the maximum window size is fixed to 32. Simulation results are discussed in section V.

B. Mobile topologies

In mobile topologies we designed a network model consisting of 30 nodes in a 1500 x 300 meter flat, rectangular area. Our network model is analogous to the one in [3]. The mobility patterns are generated using the mobility pattern generator provided in ns-2 (ns-2.xx/indep-utils/cmu-scen-gen/setdest/ where xx represents the ns-2 version). This generator is designed based on random waypoint mobility model. The mean speed with which nodes move is 10 m/s. We generate 25 such mobility patterns and our simulation results are based on the average throughput of 25 mobility patterns. Other parameters are same as mentioned above for static topologies. Simulation results are discussed in section V.

C. Performance Metric

The performance metric used in our study is throughput. In static topologies we measure the throughput of TCP connection and compare the changes observed on increasing the number of hops from 1 to 7.

But in mobile topologies the distance between the source and destination keeps varying. The number of hops on the path from source to destination may increase or decrease. Hence, we use another performance metric called *expected throughput* as defined in [3]. It is calculated as follows:

Let T_i denote the throughput obtained for the string topology, where i denotes the number of hops and $1 \leq i \leq \infty$. When $i = \infty$ it means that the network is partitioned and hence throughput $T_\infty = 0$. Let t_i be the duration for which the shortest

distance between source and destination in mobile topology is i hops ($1 \leq i \leq \infty$). The expected throughput is then calculated as:

$$\frac{\sum_{i=1}^{\infty}(t_i * T_i)}{\sum_{i=1}^{\infty}(t_i)} \quad (1)$$

The throughput measure obtained by simulations is called actual throughput. This actual throughput is then compared with the expected throughput.

V. Results and Analysis

A. Static topologies

Table I through IV show the throughput (in Kbps) obtained for each variant of TCP with DSDV, OLSR, AODV and DSR routing protocols respectively.

TABLE I. Throughput (in Kbps) using DSDV

No. of Hops, H	Tahoe	Reno	Newreno	SACK	Vegas
1	750.64	750.64	750.64	751.21	509.47
2	376.10	376.10	376.10	376.22	255.16
3	253.84	252.90	252.90	195.26	163.77
4	179.73	140.70	157.95	161.34	107.89
5	149.55	136.42	124.66	154.38	102.21
6	140.39	125.33	126.09	143.40	72.90
7	133.76	123.63	135.80	76.19	83.03

TABLE II. Throughput (in Kbps) using OLSR

No. of Hops, H	Tahoe	Reno	Newreno	SACK	Vegas
1	749.49	749.49	749.49	749.44	509.02
2	375.37	375.25	375.25	375.28	254.61
3	182.57	185.30	185.30	191.16	154.23
4	149.66	129.07	149.34	159.11	85.58
5	138.46	109.65	93.59	128.89	73.44
6	115.29	106.38	90.82	114.01	63.20
7	105.60	77.73	84.72	86.73	64.95

TABLE III. Throughput (in Kbps) using AODV

No. of Hops, H	Tahoe	Reno	Newreno	SACK	Vegas
1	750.59	750.59	750.59	751.04	509.68
2	376.22	376.22	376.22	376.22	255.08
3	211.43	200.55	211.96	203.65	156.39
4	147.48	130.67	152.39	149.05	109.31
5	124.26	112.65	126.66	129.97	95.59
6	110.94	97.25	106.32	111.98	95.31
7	51.05	54.15	55.74	32.40	51.33

Our studies show that all variants of TCP consistently perform better with DSDV. One of the reasons is that DSDV is a proactive routing protocol which maintains a routing table. In reactive routing protocols such as AODV and DSR, routes are calculated only when there is demand. Hence, reactive routing protocols incur delay before sending the data packets. Another reason is that since there are fewer routing packets (control packets) in DSDV, there are fewer collisions and hence less number of packets are dropped, resulting in high throughput.

TABLE IV. Throughput (in Kbps) using DSR

No. of Hops, H	Tahoe	Reno	Newreno	SACK	Vegas
1	750.66	750.66	750.66	750.67	509.58
2	375.82	375.82	375.82	376.22	255.05
3	119.92	109.96	109.96	114.73	160.73
4	70.40	58.89	60.06	64.72	105.25
5	62.06	67.40	67.60	52.55	90.45
6	63.17	61.79	66.72	61.07	91.96
7	67.17	58.22	62.17	59.76	71.62

However, though OLSR is a proactive routing protocol like DSDV, TCP variants achieve lesser throughput with OLSR as compared to throughput obtained with AODV when $H < 6$. The major drawback of OLSR is that it incurs high control overhead by frequently transmitting "Hello" packets and other control packets to update the routing table. This introduces substantial delay in transmitting actual data packets, resulting in overall performance degradation.

TCP variants, however, perform better with OLSR as compared to DSR. Comparing throughput values obtained with AODV and throughput values obtained with DSR, it is observed that better throughput is achieved by all TCP variants when AODV is used. Thus, TCP variants achieve least throughput with DSR routing protocol. The major drawback with source routing mechanism used in DSR is that entire route information (from source to destination) is stored in packet header. This leads to severe degradation of throughput as the number of hops increases. It can observed from Table IV, when $H > 2$, the throughput values for DSR decrease drastically.

When DSDV is used, Vegas achieves least throughput among all TCP variants since it implements proactive mechanisms (modified Slow Start, etc) to increase and decrease *cwnd*. All other variants achieve far better throughput since Slow Start mechanism tends to reach equilibrium much quicker by exponentially increasing *cwnd*. Tahoe achieves highest throughput till $H \leq 3$ because there are only fewer packet drops. When $H > 3$ there are more packet drops and hence Newreno and SACK outperform Tahoe and Reno since Newreno and SACK implement mechanisms to efficiently handle multiple packet losses.

When OLSR is used, SACK TCP performs better than all other TCP variants. However, when $H > 4$, Tahoe performs slightly better than SACK TCP. Frequent collisions due to transmission of control packets in OLSR often lead to reduction of *cwnd*. TCP Tahoe efficiently utilizes the available bandwidth by exponentially increasing *cwnd* from 1 to *ssthresh* each time a packet loss is detected.

TCP Newreno and SACK TCP outperform all TCP variants when AODV is used. The reason is, since AODV is a reactive routing protocol, it continuously sends routing packets (control packets) which lead to collisions in the network. Collisions increase the probability of multiple packets being dropped from

one window of data. Thus, TCP Tahoe and TCP Reno cannot handle multiple packet losses from one window of data whereas TCP Newreno and SACK TCP implementations mainly focus on efficiently recovering from multiple packet losses from one window of data. TCP Tahoe, however, performs better than TCP Reno since Reno enters and exits Fast Recovery for each packet loss and finally enters Slow Start phase *only when the retransmission timer expires* whereas Tahoe enters Slow Start phase immediately after a packet loss is detected.

TCP Vegas achieves lowest throughput among all TCP variants since it adopts a proactive mechanism based on RTT estimation to increase or decrease *cwnd* size. This mechanism tries to avoid congestion by reduces its sending rate if RTT of a connection increases beyond certain threshold, thus resulting in lowest throughput. However, the performance of TCP Vegas is least affected by the source routing mechanism of DSR because the performance of Vegas mainly depends on the accuracy of RTT estimation. Other TCP variants are independent of RTT estimation and thus are affected largely by DSR's source routing mechanism.

B. *Mobile topologies*

Table V through VIII show the expected throughput, actual throughput (in Kbps) obtained and the percentage of expected throughput achieved for each variant of TCP with DSDV, OLSR, AODV and DSR respectively.

TABLE V. THROUGHPUT (IN KBPS) USING DSDV

TCP Variant	Expected Throughput	Actual Throughput	Percentage Achieved
Tahoe	354.708	159.691	45.02
Reno	345.816	160.873	46.51
Newreno	346.603	160.725	46.37
SACK	343.595	156.304	45.49
Vegas	235.265	87.607	37.23

TABLE VI. THROUGHPUT (IN KBPS) USING OLSR

TCP Variant	Expected Throughput	Actual Throughput	Percentage Achieved
Tahoe	334.507	204.002	60.98
Reno	326.288	208.154	63.79
Newreno	325.637	203.380	62.46
SACK	334.891	205.850	61.47
Vegas	224.998	103.678	46.07

We scale the expected throughput values to 100 and actual throughput values accordingly for all TCP variants as shown in figure 1, 2, 3, 4 and 5.

We observe that highest throughput is achieved by almost all variants of TCP with DSR and the least throughput is achieved by all the variants of TCP with DSDV. It is also observed that reactive routing protocols give better throughput than proactive routing protocols.

The reason is that link failures and route changes are frequent when nodes are mobile. DSDV and OLSR do not adapt well to frequent route changes [12]. Each time a link failure occurs the entire routing information has to be gathered before sending the data packet. The formation of the routing table each time a link breaks, incurs large delays in sending the data, thus resulting in poor throughput in the network.

TABLE VII. THROUGHPUT (IN KBPS) USING AODV

TCP Variant	Expected Throughput	Actual Throughput	Percentage Achieved
Tahoe	334.608	269.008	80.39
Reno	327.920	260.419	79.41
Newreno	335.345	272.697	81.31
SACK	333.778	274.340	82.19
Vegas	234.373	188.790	80.55

TABLE VIII. THROUGHPUT (IN KBPS) USING DSR

TCP Variant	Expected Throughput	Actual Throughput	Percentage Achieved
Tahoe	297.910	251.247	84.33
Reno	295.059	234.950	79.62
Newreno	295.884	252.036	85.18
SACK	294.611	254.093	86.25
Vegas	234.322	173.720	74.13

TCP Newreno and SACK TCP perform better than other TCP variants with most of the routing protocols. The reason is that when nodes are mobile, link failures are frequent and hence more packets drop. The performance of TCP Tahoe and TCP Reno degrades significantly when multiple packets are lost from one window of data whereas Newreno and SACK efficiently handle multiple packet losses, thus improving overall throughput of the network. However Newreno and SACK perform best with DSR.

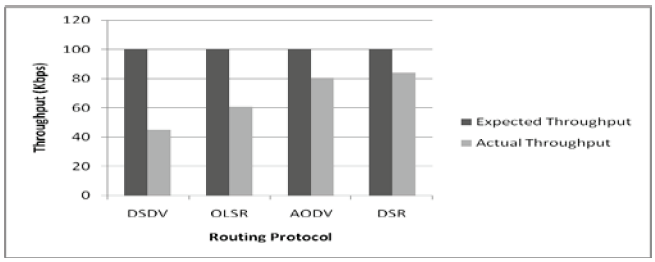

Figure 1. Throughput (in Kbps) using TCP Tahoe

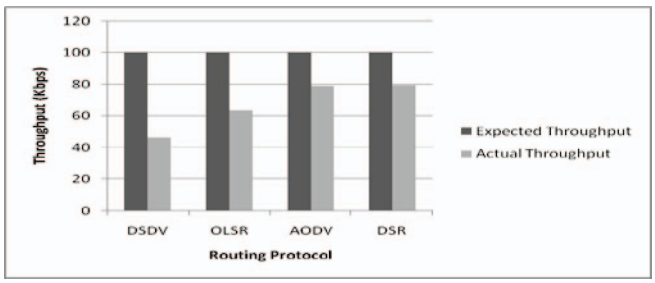

Figure 2. Throughput (in Kbps) using TCP Reno

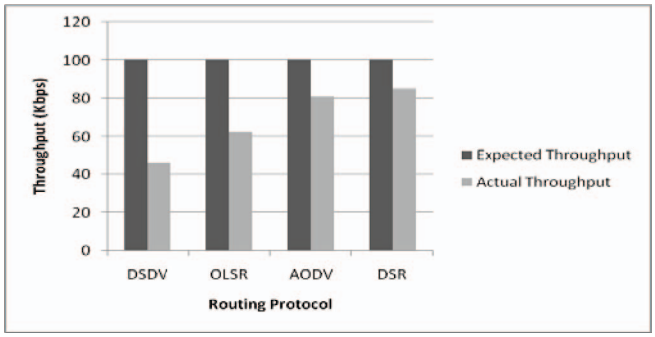

Figure 3. Throughput (in Kbps) using TCP Newreno

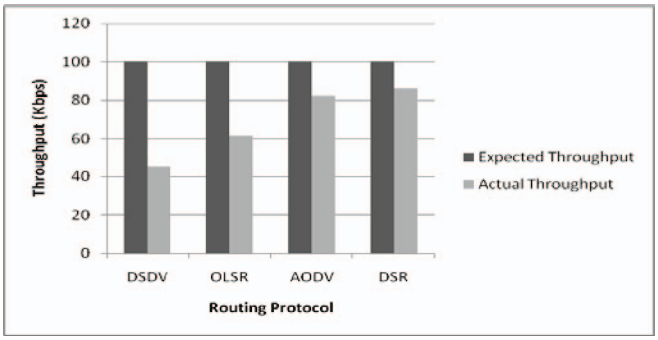

Figure 4. Throughput (in Kbps) using SACK TCP

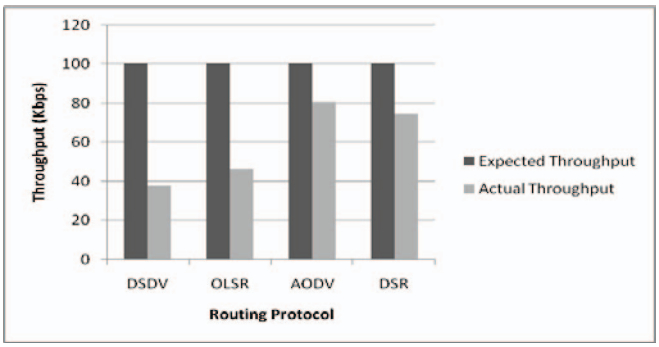

Figure 5. Throughput (in Kbps) using TCP Vegas

Reno performs slightly better than Newreno and SACK with DSDV and OLSR. TCP Vegas achieves least throughput because in mobile topologies, frequent route failures occur, leading to rerouting of packets. Rerouting of packets affects the mechanism of RTT estimation largely and thus affects the performance of TCP Vegas as explained in section III.

VI. Conclusions and Future Work

Through simulations we have studied the effects of Destination Sequenced Distance Vector (DSDV), Optimized Link State Routing (OLSR), Ad hoc On demand Distance Vector (AODV) and Dynamic Source Routing (DSR) routing protocols on TCP Tahoe, TCP Reno, TCP Newreno, SACK TCP and TCP Vegas in static as well as mobile multi-hop wireless networks.

TCP fails to distinguish between the packet losses due to congestion and the packet losses due to link failures. Hence the performance of TCP largely depends on the routing protocols. Each routing protocol varies in the way it reacts to link failures. Routing protocols also differ in the way they form the routes. More routing overhead reduces the overall throughput of the network. More number of collisions due to increased routing overload makes the situation worse for TCP connections.

Currently we are also investigating the effects of routing protocols on the performance of TCP Westwood and high-speed TCP variants such as Highspeed TCP, Scalable TCP, TCP CUBIC and Compound TCP. In future, we intend to study the performance of TCP variants on a real time test-bed and compare real time experimental results with simulation results.

References

[1] V. Jacobson, "Congestion Avoidance and Control," in Proceedings of ACM SIGCOMM '88, (Stanford, CA, Aug. 1988).

[2] M. Gerla, K. Tang, R. Bagrodia, "TCP Performance in Wireless Multi-hop Networks," in Proceedings of IEEE WMCSA '99, (New Orleans, LA, Feb. 1999).

[3] G. Holland, N. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks," in Proceedings of ACM/IEEE MOBICOM '99, (Seattle, Washington, Aug. 1999).

[4] G. Holland, N. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks – Part II: Simulation Details and Results," tech. rep. (Texas A&M University, 1999).

[5] K. Chandran, S. Raghunathan, S. Venkatesan, R. Prakash, "A Feedback Based Scheme For Improving TCP Performance in Ad Hoc Wireless Networks," in Proceedings of ICDCS '98, (Amsterdam May 1998).

[6] A. Jain, A. Pruthi, R. C. Thakur, M. P. S. Bhatia, "TCP Analysis over Wireless Mobile Ad Hoc Networks," in Proceedings of IEEE ICPWC 2002, (New Delhi, India, Dec. 2002).

[7] K. Fall, S. Floyd, "Simulation-based comparisons of Tahoe, Reno, and SACK TCP," ACM Computer Communication Review, 26(3), 5-21 (1996).

[8] L. Brakmo and L. Peterson, "TCP Vegas: End-to-end congestion avoidance on a global internet," IEEE Journal on Selected Areas in Communication, vol. 13, 1465 – 1480, (Oct. 1995).

[9] R. Shorey, A. Ahuja, S. Agrawal, J. P. Singh, " Performance of TCP over Different Routing Protocols in Mobile Ad-Hoc Networks," IEEE Vehicular Technology Conference 2000, (Tokyo, Japan 2000).

[10] S. Floyd, T. Henderson, "The Newreno Modification to TCP's Fast Recovery Algorithm," Request For Comments 2582, Experimental, (April 1999).

[11] S. Fahmy and T. P. Karwa, "TCP Congestion Control: Overview and Ongoing Research" Purdue University, 2001.

[12] J. Broch, D. A. Maltz, D. B. Johnson, Y. Hu and J. Jetcheva, "A Performance comparison of multi-hop wireless ad hoc network routing protocols," in Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking, pp. 85-97, (Oct. 1998).

[13] J. Mo, Richard J. La, V. Anantharam, J. Warland, "Analysis and Comparison of TCP Reno and Vegas," in Proceedings of IEEE INFOCOM '99, (March 1999).

[14] I. F. Akylidiz, X. Wang, W. Wang, "Wireless mesh networks: a survey," *Computer Networks*, Elsevier, 445-487 (January 2005).

[15] TCP Evolution and Comparison, "Which TCP will Scale to Meet the Demands of Today's Internet?," Whitepaper, FastSoft, (Pasadena 2008).

[16] K. Fall, K. Vardhan, "The *ns* Manual," The VINT Project, (January 2009).