



TCP: Recognition of Broken Order over Wireless Multipath Ad-hoc Network

Monika Nath

Dept. of Computer Science and Engg.
Swami Devi Dyal Group Of Professional Institutions
Barwala, Haryana, India

Nidhi Sharma

Dept. of Computer Science and Engg.,
Swami Devi Dyal Group Of Professional Institutions
Barwala, Haryana, India

Abstract— *The TCP/IP protocol suite consists of the Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) as the transport protocols. TCP is a complex transport layer protocol that provides applications with reliable, end-to-end, connection-oriented, and in-sequence data delivery. It performs both flow control, and congestion control on behalf of the applications, recovers from packet losses in the network, and handles resequencing of data at the receiver. Of the traffic carried by the Internet, TCP accounts for about 90% of the bytes, with UDP accounting for the most of the remaining traffic. In this paper, we focus on the effect of out of order over the performance of routing protocols under TCP-SACK for ad-hoc networks. The unique characteristics of ad-hoc network environments clearly will impact the requirements imposed on a transport layer protocol. Thus, it is interesting to both investigate from a top-down standpoint how a protocol as well established as TCP would work over such environments, and to study from a bottom-up standpoint what kind of transport layer behavior ad-hoc networks necessitate.*

Keywords— *MANET, TCP, SACK.*

I. INTRODUCTION

Advances in wireless communications have enabled access to the Internet for mobile users, but the Transport Control Protocol/Internet Protocol (TCP/IP) protocol stack of the Internet, designed to be independent of the link layer technology, involves mechanisms that lead to poor performance when used for mobile networks. The availability of wireless communication devices with increased processing capabilities allows the connectivity of mobile users to the global Internet. This will certainly change the way we communicate, but in mobile computing and communication many challenges are yet to be met. The major hindrances are related to mobility management and the poor performance of legacy protocols over wireless networks. These problems restrict large-scale deployment of such technologies. Various changes have been proposed to overcome the difficulties of using it for mobile networks.

A. Transmission Control Protocol

TCP is an effective connection-oriented transport control protocol that provides the essential flow control and congestion control required to ensure reliable packet delivery. Numerous enhancements and optimizations have been proposed over the past few years to improve TCP performance for infrastructure-based WLANs and cellular networking environments. Infrastructure-based wireless networks are one hop wireless networks where a mobile device uses the wireless medium to access the fixed infrastructure. The mobile multihop ad hoc environment brings fresh challenges to TCP protocol.

B. TCP Congestion Control

TCP uses window based transmissions. The number of unacknowledged packets transmitted on the channel is determined by the size of the congestion window. Hence, the progression of the congestion window can be directly related to the throughput enjoyed by the connection. Further, the arrival of ACKs from the TCP receiver drives the progression of the sender's congestion window. Initially, when a connection is initiated, the TCP sender enters the slow-start phase. In this phase, the congestion window is increased by one for every ACK that is received. Hence, there is an exponential increase of the congestion window, with the window doubling every round-trip time. Once the window size exceeds an ssthresh threshold, the window increases by one for every round-trip time (rtt). This phase is referred to as the congestion avoidance phase where the progression of window is linear. The sender continues to perform linear increase, probing for more available network bandwidth. The increase continues till a loss is perceived. On experiencing a loss, the sender infers congestion (loss-based congestion detection) and reduces the congestion window. The nature of reduction depends on the nature of loss. If the loss is notified by the arrival of triple duplicate ACKs, then a multiplicative decrease of the window is performed, wherein the window is decreased to half its current value, and the connection enters the congestion avoidance phase. On the other hand, if the loss is detected through a retransmission timeout, then the window is reduced to one and the connection enters the slow-start phase again.

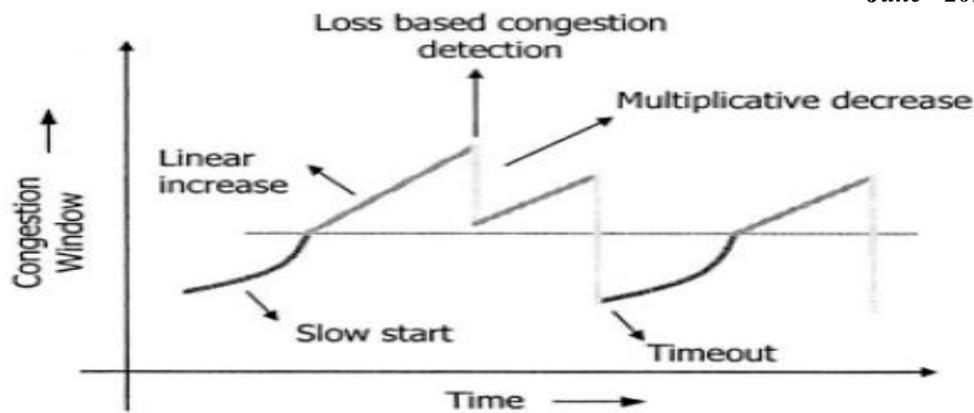


Figure 1: An example of the Internet congestion algorithm.

C. Window-based Transmissions

One of the motivating factors for TCP being window based is the avoidance of the maintenance of any fine-grained transmission timers on a per-flow basis. Instead, TCP uses the principle of self-clocking (ACKs triggering further data transmissions) for connection progression. For wire line environments, where per-flow bandwidths can scale up to several megabits per second, such a design choice is clearly essential. However, the use of a window based transmission mechanism in ad-hoc networks may result in the critical problem of burstiness in packet transmissions. Thus, if several ACKs arrive back-to-back at the sender, a burst of data packets will be transmitted by the sender even if it were in the congestion avoidance phase (where one packet will be transmitted for every incoming ACK). Unfortunately, ACK bunching or several ACKs arriving at the same time is a norm in ad-hoc networks because of the short-term unfairness of the CSMA/CA MAC protocol typically used in such networks. Such short-term unfairness results in the data stream of a TCP connection assuming control of the channel for a short period, followed by the ACK stream assuming control of the channel for a short period. Interestingly, such a phenomenon will occur even when the ACK stream does not traverse the exact same path as the data stream. This is because even if the paths were completely disjoint, the vicinity of the TCP sender and the vicinity of the TCP receiver still are common contention areas for the data and ACK streams. The impact of such burstiness of traffic has two undesirable effects:

- Varying round-trip time estimates: TCP relies on an accurate round-trip time (rtt) calculation to appropriately set the timer for its retransmission timeout (RTO). Coupled with the low bandwidths available to flows, the burstiness results in artificially inflating the round-trip time estimates for packets later in a burst. Essentially, the round-trip time of a packet is impacted by the transmission delay of the previous packets in the burst due to the typically small available rates. TCP sets its RTO value to where is the exponential average of rtt samples observed, and is the standard deviation of the rtt samples. Hence, when rtt samples vary widely due to the burstiness, the RTO values are highly inflated, potentially resulting in significantly delayed loss recovery.
- Higher induced load: Spatial re-use in an ad-hoc network is the capability of the network to support multiple spatially disjoint transmissions. Unfortunately, due to the burstiness and the short term capture of channel by either the data stream or the ACK stream, the load on the underlying channel can be higher than the average offered load.

D. TCP Timer Management

TCP uses multiple timers to do its work. The most important of these is the retransmission timer. When a segment is sent, a retransmission timer is started. If the segment is acknowledged before the timer expires, the timer is stopped. If, on the other hand, the timer goes off before the acknowledgement comes in, the segment is retransmitted. In the latter case, the expected delay is highly predictable, so the timer can be set to go off just slightly after the acknowledgement is expected, as shown in Fig. 2(a). Since acknowledgements are rarely delayed in the data link layer, the absence of an acknowledgement at the expected time generally means either the frame or the acknowledgement has been lost.

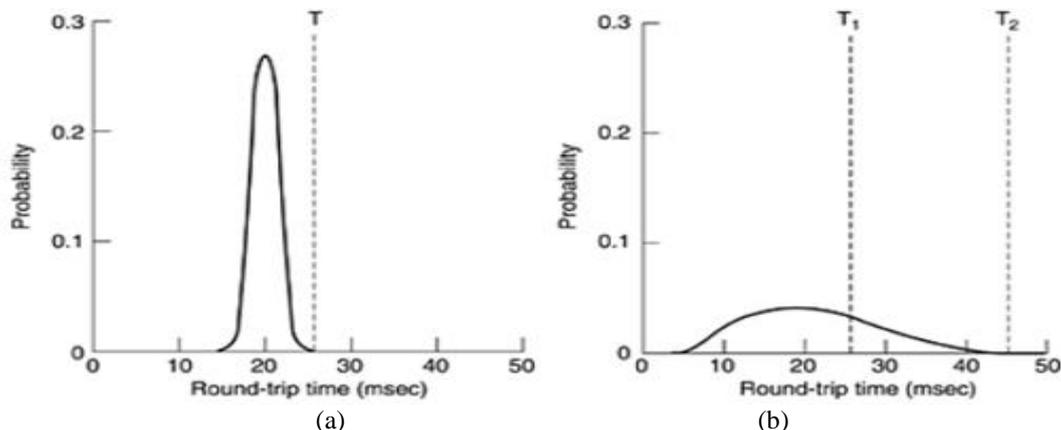


Figure 2: Probability versus RTT.

TCP is faced with a radically different environment. The probability density function for the time it takes for a TCP acknowledgement to come back looks more like Fig. 2(b) than Fig. 2(a). Determining the round-trip time to the destination is tricky. Even when it is known, deciding on the timeout interval is also difficult. If the timeout is set too short, say, T_1 in Fig. 2(b), unnecessary retransmissions will occur, clogging the Internet with useless packets. If it is set too long, (e.g., T_2), performance will suffer due to the long retransmission delay whenever a packet is lost. Furthermore, the mean and variance of the acknowledgement arrival distribution can change rapidly within a few seconds as congestion builds up or is resolved.

E. TCP Selective Acknowledgment (TCP-SACK)

TCP-SACK (Selective Acknowledgment) [8] conserves the basic ideology of the TCP functionalities. The TCP-SACK works best when various packets got dropped from one window of data. The receiver uses the 'option' fields of TCP header (SACK option) for notifying the sender of three blocks of non-contiguous set of data received and enqueued by the receiver. The first starting block represents the most recent packet received, and the next blocks represent the most recently reported SACK blocks. The sender keeps a scoreboard in order to provide information about SACK blocks received so far. In this way, the sender can conclude that whether there are missing packets at the receiver. If so, and its congestion window permits, the sender retransmits the next packet from its list of missing packets. In case there are no such packets at the receiver and the congestion window allows, the sender simply transmits a new packet.

As per the previous discussion, fast retransmission and fast recovery can only handle one packet loss from one window of data within one transmission time out period; TCP may experience poor performance when multiple packets are lost in one window. To overcome this limitation, recently the Selective Acknowledgment option (SACK) is suggested as an addition to the standard TCP implementation. In the event of multiple losses within a window, the sender can conclude that 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.

The SACK implementation also enters fast recovery upon the receipt of generally three duplicate acknowledgments. Then, its sender retransmits a packet and reduces the congestion window by half. During fast recovery, SACK controls the estimated number of packets outstanding in the path (transmitted but not yet acknowledged) by maintaining a variable called *pipe*. This variable determines if the sender may send a new packet or retransmit an old one, in that the sender may only transmit if pipe is smaller than the congestion window. At every transmission or retransmission, pipe is incremented by one, and it is decremented by one when the sender receives a duplicate ACK packet containing a SACK option informing it that the receiver has received a new data packet.

The fast recovery is over when the sender receives an ACK acknowledging all data that were outstanding when fast recovery was entered. If the sender receives a partial ACK, i.e., an ACK that acknowledges some but not all outstanding data, it does not exit fast recovery. For partial ACKs, the sender reduces pipe by two packets instead of one, which guarantees that a SACK sender never recovers more slowly than it would do if a slow start had been invoked [1].

Generally speaking, Selective Acknowledgment (SACK) is a strategy that corrects the above TCP behaviour in the face of multiple dropped segments. With selective acknowledgments, the data receiver can inform the sender about all segments that have arrived successfully, so the sender needs to retransmit only the segments that have actually been lost.

Even though the TCP selective acknowledgment mechanism can allow a SACK-enabled sender to retransmit in one RTT multiple lost packets in one transmission window and hence avoid continuous timeouts, this mechanism does not distinguish the reasons for packet losses and still assumes that all losses are caused by congestion. As a result, TCP congestion control procedures are inappropriately called for, which affects the sender's transmission rate.

II. RELATED WORK

There are many simulation-based investigations of the problematic aspects of TCP in wireless network environments. For example, [2] shows that long sudden delays, mostly attributed to handovers, are common in the GPRS wireless WAN. It explores the influence of these delays on TCP performance and concludes that the spurious timeouts they trigger may lead to unnecessary retransmissions. Another experimental work [3] suggests that carefully designed probing mechanisms can cancel incompetent TCP behavior over wireless networks. The authors propose a TCP-Probing modification that prevents a significant portion of the timeouts and unnecessary retransmissions caused by handovers. Reference [4] presents a simulation-based investigation for measuring the effect of handovers on several TCP versions. The authors suggest that forwarding might improve TCP performance at handover, pointing out that duplicate packets should be filtered at the access point.

In [5], C. Chen, H. Wang et al. proposed a receiver-aided mechanism in which the TCP receiver monitors the contention state of the connection and accordingly informs the TCP sender about it via ACK mechanism. TCP receiver uses end-to-end delay as contention criteria.

In [6], TCP New-Reno tries to overcome the issues present in TCP-Reno by modifying the fast recovery mechanism. During fast recovery, when a fresh ACK is received TCP New-Reno handles them as below :

- If it ACKs all the segments which were outstanding when TCP New-Reno entered Fast Recovery, then it exits Fast Recovery and sets *cwnd* and *ssthresh* and continues congestion avoidance.
- If the ACK is partial then it deduces that the next segment in line was lost and retransmits that segment and sets *dupacks* to 0. It exits Fast Recovery when all the data segments in the window are acknowledged, until then every progress in sequence number generates a redundant packet retransmission which is instantly acknowledged.

TCP New Reno suffers from the fact that it takes one RTT to detect each packet loss. When the ACK for the first retransmitted segment is received, only then can we deduce which other segment was lost. To achieve TCP-LBA mechanism a few changes are performed on TCP header. Firstly, Sequence number field in the TCP header is changed to total number of pieces. Secondly, the Acknowledgement number field in the TCP header is changed to the number of the current packet. Thirdly, out of 6 bit reserved field in TCP header 2 bits are made use of. Of the four values that can be generated using the 2 bits packet type, two are made use of and the remaining two are reserved. When TCP sender is sending packet it fills the field packet type with value 0x00.

J. Huang et al. [7], presented a new method to improve TCP SACK'S performance over wireless links. The proposed method attempts to differentiate congestion and corruption loss by the use of tagged segments. In this method, a sender is able to discern congestion from corruption in large likelihood. In contrast to most other existing approaches employing explicit notification, proposed method is attractive in that it requires no modification at the receiver. Through simulations authors have compared the performance of proposed method with TCP SACK over both low and long delay noisy links. The results show that our method significantly improves TCP SACK over a wide range of channel conditions in both environments.

L. Subedi et al. [8], compare the performance of various TCP algorithms in wireless and wired networks. Simulation results indicated that in wireless networks with signal attenuation, fading, and multipath, TCP Reno outperforms other congestion control algorithms in terms of congestion window size and file download response time. TCP Reno has larger congestion window size, shorter file download response time, and higher throughput and goodput than the remaining three TCP algorithms. In case of wired networks, TCP Reno with SACK has the best overall performance.

Georgi Kirov [9], focuses on the different congestion control mechanisms implemented by the Transmission Control Protocol (TCP). The author presents an experimental estimation of the TCP control algorithms: Slow-Start and Congestion Avoidance without Fast Retransmit, Tahoe that includes Fast Retransmit and Fast Recovery, and Reno using a modified version of the Fast Recovery. The TCP performance analysis is based on different scenarios of the network simulation with low percentages of the packet loss. The results for Reno are slightly better than Tahoe. The advantage of the Reno algorithms in comparison with Tahoe one is when packet loss is detected, the window size is reduced to one half of the current window size and the congestion avoidance, but not slow start is performed.

III. PROPOSED METHODOLOGY

Due to mobility of nodes route failure may occur, hence packets coming from TCP sender may not arrive in order at the TCP receiver, which means routes in MANETs, are short-lived due to frequent link breakages. To reduce delay due to route re-computation and node mobility, some routing protocols such as Temporally Ordered Routing Algorithm (TORA) and Optimized Link State Routing (OLSR) maintain multiple routes between a sender-receiver pair. In such a case, packets coming from different paths may not arrive at the receiver in order. Being unaware of multi-path routing, TCP receiver would consider such out-of-order packet arrivals as a sign of congestion. The receiver will thus generate duplicate ACKs that cause the sender to invoke congestion control algorithms like fast retransmission (upon reception of three duplicate ACKs), which degrades TCP's performance a lot.

A. Problem Statement

TCP/IP protocol was designed for wired networks which provides end to end reliable communication between nodes and assures ordered delivery of packets. It also provides flow control and error control mechanisms. As it is still a successful protocol in wired networks, losses are mainly due to congestion. But in case of ad hoc networks packet losses are due to congestion in the network and due to frequent link failures so when we adapt TCP to ad hoc networks it misinterprets the packet losses due to link failure as packet losses due to congestion and in the instance of a timeout, backing-off its retransmission timeout (RTO). This results in unnecessary reduction of transmission rate because of which throughput of the whole network degrades. Therefore, route changes due to host mobility can have a detrimental impact on TCP performance. The following steps are the major objectives for this paper:

- Study of TCP-SACK congestion control mechanism.
- Implementing the exiting TORA and OLSR routing protocols in ns2
- Simulate and compare two types of TCP sender side congestion control mechanisms TCP-SACK under TORA and OLSR routing protocol over simulation time using ns2.
- TCP-Recognition of Broken Order.

B. Research Methodology

Network Simulator (Version 2), widely known as NS2, is simply an event driven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviours. Due to its flexibility and modular nature, NS2 has gained constant popularity in the networking research community since its birth in 1989. Ever since, several revolutions and revisions have marked the growing maturity of the tool. Among these are the University of California and Cornell University who developed the REAL network simulator, the foundation which NS is based on. Since 1995 the Defence Advanced Research Projects agency (DARPA) supported development of NS through the Virtual Inter Network Test-bed (VINT) project. Currently the National Science Foundation (NSF) has joined the ride in development. Last but not the least, the group of researchers and developers in the community are constantly working to keep NS2 strong and versatile. NS-2 provides a split-programming model; the simulation kernel is implemented using

C++, while the Tcl scripting language is used to express the definition, configuration and the control of the simulation. This split-programming approach has proven benefits over conventional programming methods. Also, NS-2 can produce a detailed trace file and an animation file for each ad hoc network simulation that is very convenient for analyzing the routing behavior.

Table I *Salient Simulation Parameters*

Parameter	Value
Simulation time	150 Sec
Simulation area	1000m x 1000m
Routing Protocol	TORA, OLSR
No. of nodes	50
Packet size	512 Bytes
Max queue length	50
TCP-Variant	TCP-SACK
MAC type	IEEE-802.11

Figure 4, tell about the congestion window size [CWND] with the 150 simulated times, CWND means the maximum number of window size that can be sent by the TCP sender at a time. Whenever the CWND reduces drastically, it means packets are reached out-of-order in which the TCP receiver produces either three duplicate acknowledgment or the expected acknowledgment is not come with in the retransmission time out period, If time out is occurred, slow start threshold is reduced to half of the current congestion window size, the CWND is reduced below SSTRHESHOLD that is the TCP sender enters in to slow start and start sending packet from one and continuous sending exponentially until it reaches to slow start threshold (SSTRHESHOLD). If the CWND is reduced but not below SSTRHESHOLD, then the TCP sender has received three DUPACKs, so it will start from congestion avoidance phase. In figure 4, between 0 and 100 simulation seconds the sender sent a maximum of CWND 32 for TORA routing protocol which is much more than OLSR routing protocol.

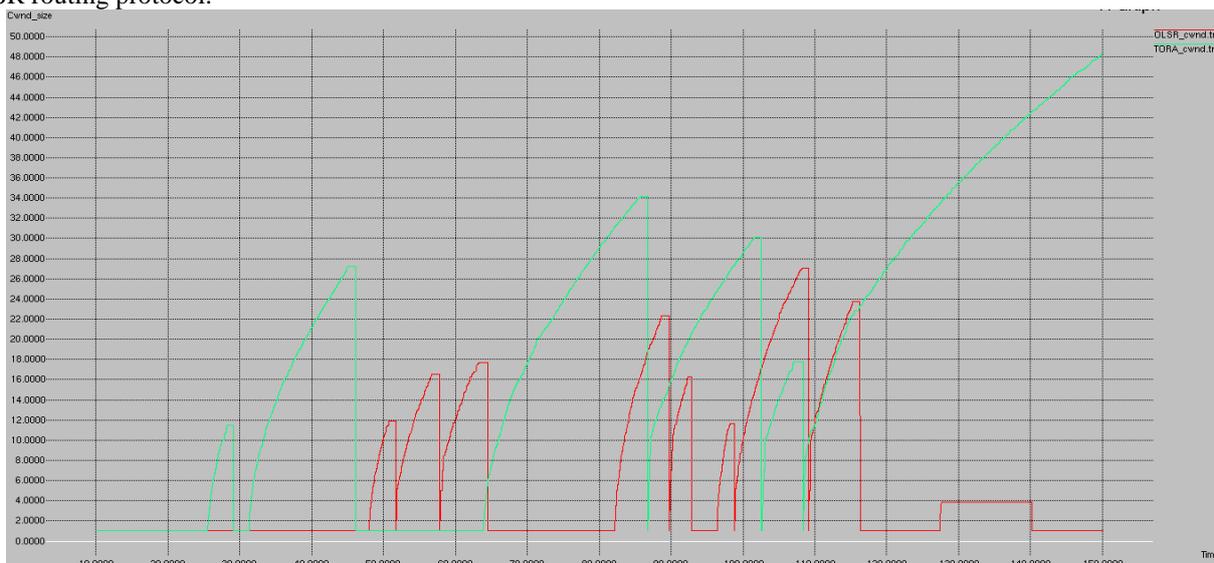


Figure 3: Congestion window Vs simulated time with TCP-SACK.

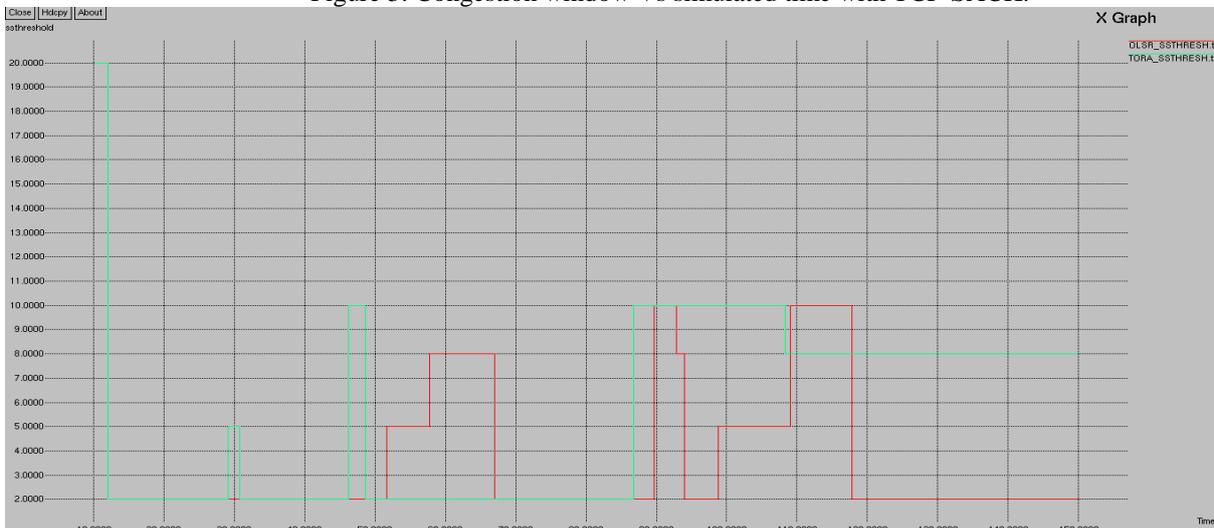


Figure 4: Slow start threshold over 150 simulated times with TCP-SACK.



Figure 5: Sequence number over 150 simulated times with TCP-SACK.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have evaluated the effect of broken order on different routing protocols namely TORA and OLSR routing protocol under TCP-SACK over wireless multihop adhoc environment. From the simulation results, it is observed that routing protocol namely TORA performs better than OLSR routing protocol by maintaining higher CWND size and good packet sequence number flow. From Simulation results, it is confirmed that out-of-order packet is really delivered due to multi-path route between TCP sender and receiver, which has significant performance degradation effect. Further in this direction our aim is to implement TCP Detection of Out-of-Order and Response with Time Stamp (TCP-BO) algorithm in TCP-SACK, when multi-path routing protocol (TORA), is used in mobile ad hoc networks has been evaluated.

References

- [1] G. Holland and N.H. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks," Proc. ACM MOBICOM Conf., pp. 219-230, Aug. 1999.
- [2] A. Gurtov, "Effect of delays on TCP performance," in *Working Conference on Emerging Personal Wireless Communications*, 2001, pp. 87-108.
- [3] A. Lahanas and V. Tsaoussidis, "Experimental evaluation of TCP-Probing in mobile networks," *Journal of Supercomputing*, vol. 23, pp. 261-279, 2002.
- [4] J. Schuler and T. Schwabe, "A comparison of the performance of four TCP versions during mobile handoff," *IEEE MWCN*, Sept. 2002.
- [5] C. Chen, H. Wang, XinWang, M. Li, A.O. Lim, "A Novel Receiver-aided Scheme for Improving TCP Performance in Multihop Wireless Networks", in Proc. of Int. Conference on Communications and Mobile Computing, pp. 272-277, 2009.
- [6] W. Long, W. Zhenkai, "Performance Analysis of Improved TCP over Wireless Networks", in Proc. of 2nd Int. Conference on Computer Modeling and Simulation, pp. 239-242, 2010.
- [7] Jeng-Ji Huang, Jin-Fu Chang, "A New Method to Improve the Performance of TCP SACK over Wireless Links", IEEE, 2003.
- [8] L. Subedi, M. Najiminaini, and L. Trajković, "Performance Evaluation of TCP Tahoe, Reno, Reno with SACK, and NewReno Using OPNET Modeler", Proceeding of International Conference on Methods and Models in Computer Science, IEEE, 2009.
- [9] Georgi Kirov, "A Simulation Analysis of the TCP Control Algorithms", International Conference on Computer Systems and Technologies, 2005.