A Survey on Different Solutions for Packet Reordering in TCP

K.LakshmiNadh[#1], Y. K. Sundara Krishna[*2], K.Nageswara Rao[#3]

[1] *Department of CSE, Narasaraopeta Engineering College, Narasaraopeta, India*
[2] *Department of CSE, PVP Siddhartha Institute of Technology, Vijayawada, India*
[1] l_nadh@yahoo.co.in
[3] drknrao@ieee.org
[3] *Department of CS, Krishna University, Machilipatnam, Andhra Pradesh, India*
[2] yksk2010@yahoo.com

***Abstract*— Packet reordering is an inevitable phenomenon on the Internet, and it can have an adverse impact on end-to-end performance and network resource utilization. In TCP, reordering leads to unnecessary retransmissions thereby creating a sense of congestion and an effective loss in throughput. TCP performs poorly on paths that reorder packets significantly, where it misinterprets out-of-order delivery as packet loss. The sender responds with a fast retransmit even though actual loss is not occurred. In this paper we have present a survey of solutions for TCP packet reordering.**

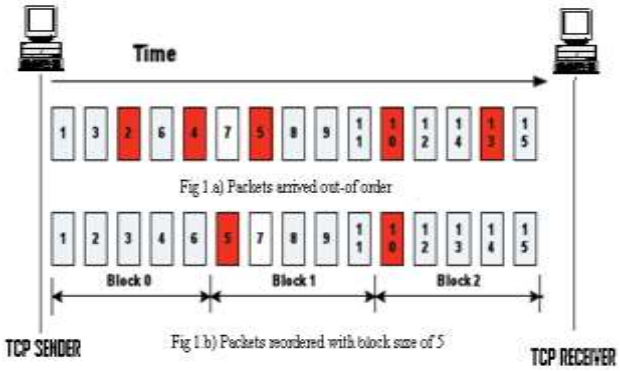
I. INTRODUCTION

The Internet plays a significant role nowadays in our lives. Two main protocols are being implemented in the transport layer of the Internet, namely, UDP (User Datagram Protocol) and TCP (Transmission Control Protocol).TCP is the most important transport layer protocol for point-to-point, connection-oriented, reliable data transfer in the Internet. TCP is the de facto standard for Internet based commercial communication networks. It is a byte-stream protocol, with its flow control and acknowledgement based on byte number rather than packet number [1]. However, the smallest unit of data transmitted in the Internet is a data segment or packet, each identified by a data octet number. When a destination receives a data segment, it acknowledges the receipt of the segment by issuing an ACK with the next expected data octet number. The time elapsed between when a data segment is sent and when an ACK for the segment is received is known as the Round Trip Time (RTT) of the communication between the source and the destination, which is the sum of the propagation, transmission, queueing, and processing delays at each hop of the communication, and the time taken to process a received segment and generate an ACK for the segment at the destination. Due to various reasons, such as multipath routing, route fluttering, and retransmissions, packets belonging to the same flow may arrive out of order at a destination. Such packet reordering violates the design principles of some traffic control mechanisms in TCP and, thus, poses performance problems.

II. TCP PACKET REORDERING

Numerous studies have shown that packet reordering is common [2], especially in high speed networks where there is high degree of parallelism and different link speeds. Reordering of packets decreases the TCP performance of a network, mainly because it leads to overestimation of the congestion of the network. Packet reordering refers to receiving order of a flow of packets differs from its sending order.

Consider that 15 packets are sent from a TCP sender to a TCP receiver, and those packets arrive in the receiver in the order as shown in Fig1.a). According to the packet reordering metrics in [3], there are 5 packets reordered in Fig1.a), yielding a packet reordering rate of 30%, and maximum packet reordering extent of 3.



Therefore, if the network device driver delivers the received packet upward in the same sequence as they were received from the network, the protocol stack will deal with those 5 instances of packet reordering in the TCP layer. In the Figure, the packet reordering instances are highlighted in red. Now assume that the original packet sequence in Fig1.a) can be sorted in blocks. Fig1.a) give the resulting packet sequence with a sorting block size of 5. Before continuing, we give two definitions:

- *Intra-block packet reordering:* packet reordering that occurs within a sorting block.
- *Inter-block packet reordering*: packet reordering that occurs across sorting blocks.

In Fig1.b), intra-block packet reordering includes packet 2, 4, and 13, inter-block packet reordering includes packet 5 and 10.

2.1 Causes of Packet Reordering

As described by [4], Packet Reordering is not mostly caused by pathologies such as broken network equipment, but is a naturally occurring result of parallelism in a device or logical link. An overview of possible causes of Packet Reordering:

• **Switches and routers:** Because link speeds increase faster than CPU processing speeds, routers and switches increasingly have to rely on parallel processors and queues. Those parallel processors and queues can cause Packet Reordering, for instance if an earlier packet is placed in a long queue and a later packet in a shorter queue. In such a situation it is likely that those packets leave the router or switch out-of-order [5].

• **Diffserv scheduling:** If a flow exceeds negotiated constraints, the non-conformant packets are either dropped or given a lower priority. In the latter case, the packets will be placed in different queues resulting in out-of-order delivery.

• **Load splitting:** To balance the load among multiple paths, different packets of the same stream may take different routes leading to different delays causing Packet Reordering [5].

• **Ad hoc routing:** In ad hoc networks, the path between two end hosts can change from time to time. These route changes result in disconnections that are too short-term to result in a TCP timeout. Depending on the underlying routing protocol, the receiver may experience a short burst of out-of-order packets or packet losses.

• **Retransmission**. In this case, a sender infers that a packet has been lost and retransmits the packet. The retransmitted packet will have a sequence number that is smaller than previously observed packets at the measurement point and hence will be deemed "out-of-sequence."

• **Network duplication**. In this case, a non-sender retransmitted copy of a packet is observed. This can occur when the measurement point is within a routing loop (and hence the same packet is observed more than once), or if the network itself creates a duplicate copy of the packet.

• **In network-reordering**. In this case, the network inverts the order of two packets in a connection (for example, because of parallelism within a router or a route change).

This list of causes should not be considered complete. Other causes of Packet Reordering may exist, but these are known best.

2.2 Effects of TCP Packet Reordering

Packet Reordering can have effects on TCP performance. Packet Reordering on TCP are divides the effects in two categories: the effects of *Forward-Path Reordering* and the effects of *Reverse-Path Reordering* on TCP performance.

2.2.1 Forward-Path Reordering

Forward-Path Reordering is the reordering of data, which results in the transmission of duplicate ACKs. When a TCP receiver is receiving in-order data segments, it acknowledges those data segments. However, if the data is out-of-order, the receiving TCP sends a duplicate acknowledgement which repeats the acknowledgement of the last in-order byte received.

According to [4], *Forward-Path Reordering* has five effects:

• **Causes Unnecessary Retransmission:** When the TCP receiver gets packets out of order, it sends duplicate ACKs to trigger fast retransmit algorithm at the sender. These ACKs (3 or more) makes the TCP sender infer a packet has been lost and retransmit it. If the temporary sequence number gap is caused by reordering, then the duplicate ACKs and the fast retransmission are unnecessary and a waste of bandwidth.

• **Limits Transmission Speed:** When fast retransmission is triggered by duplicate ACKs, the TCP sender assumes it is an indication of network congestion. It reduces its congestion window (*cwnd*) to limit the transmission speed, which needs to grow larger from a "slow start" again. If reordering happens frequently, the congestion window is at a small size and can hardly grow larger. It results in a limited speed of packets transmission, and hence a throughput degradation.

• **Obscured Packet Loss:** When along with Packet Reordering, Packet Loss also occurs, TCP often does not "see" the Packet Loss because the Packet Reordering obscures it. The Packet Loss will only be detected by means of a timeout.

• **Poor Round-Trip Time Estimation:** Packet Reordering can, at least in theory, cause TCP round-trip time estimators that do not use the TCP Timestamps Option to stop working accurately due to a lack of round-trip time samples. If the TCP Timestamps Option is not used, most TCP's use an algorithm that samples one round-trip time per window of data. If the byte being sampled is retransmitted, the sample must be discarded. In the event of Packet Reordering, it is very well possible that TCP has to discard most of its samples. This result in an infrequent round-trip time sample and the round-trip time estimator will not be able to keep an accurate estimate of the round trip time. This will, in most cases, increase the impact of obscured Packet Loss because the lost packet will be transmitted later than it needs to be.

• **Reduce Receiver's Efficiency:** Since the TCP receiver has to hand in data to the upper layer in order, when reordering happens, the receiver has to buffer all the out-of-order packets until getting all packets in order. Meanwhile, the upper layer gets data in burst rather than smoothly, which also reduces the system efficiency as a whole.

2.2.2 Reverse-Path Reordering

With *Reverse-Path Reordering*, the ACKs are reordered. One of the major effects of *Reverse-Path Reordering* on TCP performance: TCP loses its self-clocking capability. In TCP, the ACKs roughly reflect the rate at which data can transit the network and be removed from the network by the receiver. This self-clocking property does not work very well when the ACKs are reordered. ACKs are cumulative and whenever *Reverse-Path Reordering* occurs, an ACK for a later packet arrives early thereby acknowledging more than one packet. After the reordered ACK, one or more apparent redundant ACKs arrive. According to, this ACK pattern has at least two harmful effects:

• **Opening TCP's Congestion Window:** A TCP sender opens its congestion window for each ACK that acknowledges unacknowledged data. If the ACKs are reordered, the congestion window will grow too slowly.
• **Bursts of segments:** When an ACK arrives at a TCP sender, it permits the sender to transmit an amount of data equal to the amount of acknowledged data. So, if the ACKs are reordered and one ACK arrives early and acknowledges a large amount of data, the TCP sender is likely to transmit a series of new segments at once. Since such bursts of data typically lead to close spaced ACKs, more *Reverse-Path Reordering* is likely to occur. So, at least for modest periods, this behavior is self-reinforcing.

III. SOLUTIONS FOR PACKET REORDERING

For an in-order network environment, a destination receives segments in the same order as they are sent. The destination realizes the occurrence of a dropped packet when an unexpected segment arrives. It can then embed the information of missing segments into the subsequent ACKs to a source so that the source can retransmit the lost segments to the destination. The solutions for packet reordering include **TCP-Eifel, DSACK TCP, F-RTO, TCP-DOOR, DelAck, TCP-ADA and TCP-DCR.** TCP-DOOR and TCP-Eifel focus on detecting spurious retransmission after activating packet retransmission and congestion response.

TCP-Eifel—Ludwig and Katz proposed the Eifel algorithm [6] to eliminate the retransmission ambiguity and solve the performance problems caused by spurious retransmissions. A source uses the TCP timestamp option to insert the current timestamp into the header of each outgoing segment to a destination. When the destination sends ACKs, it includes the corresponding timestamps into the ACKs. To eliminate the retransmission ambiguity, the source always stores the timestamp of the first retransmitted segment. When the first ACK for the retransmitted segment arrives, the source compares the timestamp of that ACK with the stored timestamp. If the stored timestamp is greater, the retransmission is considered spurious.

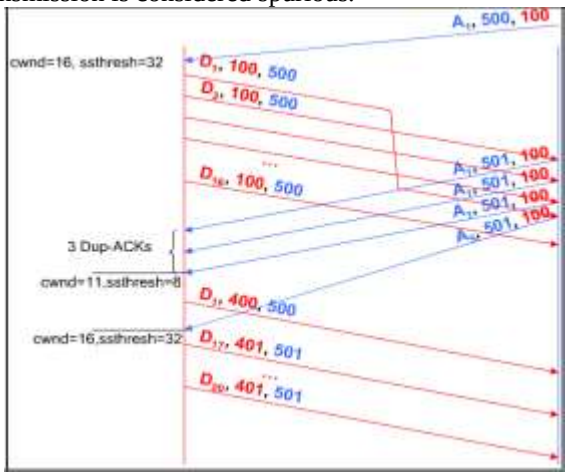


Fig. 2. An illustration of the Eifel algorithm.

Fig. 2 illustrates how the Eifel algorithm works. When the source sends Segment D1 the first time at Time 100, it inserts the current timestamp 100 into the header of the segment. At Time 400, the source initiates a congestion response by retransmitting Segment D1. The original segment differs with the retransmitted one as the latter one contains a timestamp 400 instead of 100. When the destination receives the original Segment D1 first, it sends an ACK with the timestamp of the segment, i.e., 100. When the ACK for the segment arrives, the source finds that the echoed timestamp, 100, is smaller than the stored one, 400. The retransmission is hence identified as spurious.

To solve the problems caused by spurious retransmissions, a source also stores the current values of the ssthresh, and the size of the congestion window, cwnd, when a segment is retransmitted the first time. When a detected spurious retransmission has resulted in a single retransmission of the oldest outstanding segment, the source restores ssthresh and cwnd to the stored values. This technique is simple and effective in improving TCP performance with forwardpath reordering.

DSACK TCP—Floyd et al. discussed the use of duplicate selective acknowledgement (DSACK) [7] to detect segment reordering and retract the associated spurious congestion response. DSACK is an extension of the selective acknowledgement (SACK) option [8] for TCP. It aims to use the SACK option for duplicate segments. The first block of the SACK option field is used to report the sequence numbers3 of a received duplicate segment which has triggered the ACK. When congestion is detected, cwnd is saved before reduction. When a source finds that it has made a spurious congestion response based on the arrival of a DSACK, it performs slow start to increase the current cwnd to the stored cwnd before congestion avoidance. By performing slow start during state restoration, it allows TCP to reacquire ACK-clocking and avoid injecting traffic bursts into the network.

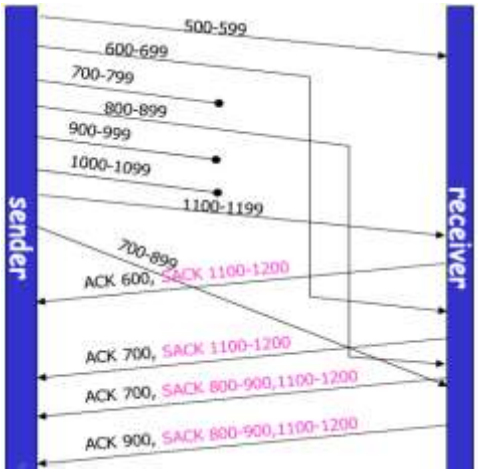


Fig. 3. An example of detecting packet reordering through DSACK

Fig. 3 shows how DSACK is used to detect packet reordering. Suppose Segment 600-699 is reordered such that it arrives after Segment 1100-1199 at the destination. The last acknowledged segment is Segment 500-599. In this case, the destination sends out three duplicate ACKs with the same cumulative ACK for Segment 500-599, although the SACK option fields differ to the source so that Segment 600-699 is retransmitted (assuming dupthresh is three). When the destination receives the retransmitted Segment 600-699, it sends a duplicate ACK 1200 for Segment 1100-1199, but the first block of the SACK option field acknowledges an arrival of a duplicate Segment 600-699. The source then knows that Segment 600-699 has been retransmitted spuriously due to packet reordering. This method can be easily coupled with a scheme using the DSACK information to adjust dupthresh to proactively prevent triggering spurious congestion responses.

F-RTO—An RTO is spurious if there are segments outstanding in the network that would have prevented the RTO, had their acknowledgements arrived earlier at the sender. F-RTO [9] affects the TCP sender behavior only after a retransmission timeout, otherwise the TCP behavior remains unmodified. When RTO expires the F-RTO algorithm monitors incoming acknowledgements and declares an RTO spurious, if the TCP sender gets an acknowledgement for a segment that was not retransmitted due to RTO.

Spurious retransmission timeouts (RTOs) cause suboptimal TCP performance, because they often result in unnecessary retransmission of the last window of data. This document describes the "Forward RTO Recovery" (F-RTO) algorithm for detecting spurious TCP RTOs. F-RTO is a TCP sender only algorithm that does not require any TCP options to operate. After retransmitting the first unacknowledged segment triggered by an RTO, the F-RTO algorithm at a TCP sender monitors the incoming acknowledgements to determine whether the timeout was spurious and to decide whether to send new segments or retransmit unacknowledged segments. Fig.4 represents an example of detecting spurious retransmission timeouts by F-RTO.

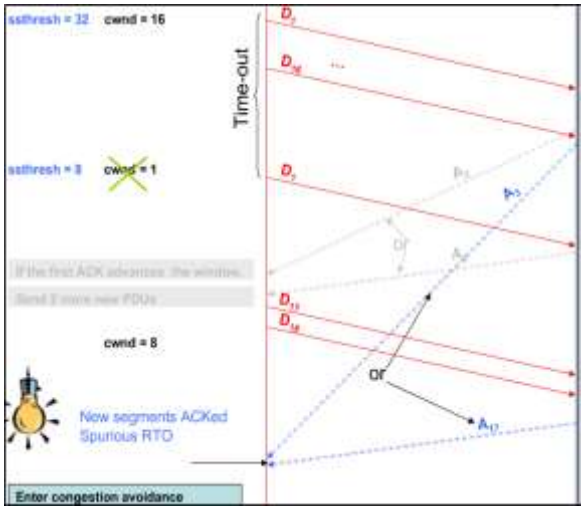


Fig. 4. An example of detecting spurious retransmission timeouts by F-RTO.

TCP-DOOR—Wang and Zhang developed TCP with detection of out-oforder and response (TCP-DOOR) [10], which can be considered as an extension of the Eifel Algorithm. The out-of- order events imply route changes in the networks, which happen frequently in mobile ad hoc networks. The TCP packet sequence number and ACK duplication sequence number, or current timestamps, are inserted into each data and ACK segment, respectively, to detect reordered data and ACK packets. When out-of-order events are detected, a source can either temporarily disable congestion control or perform recovery during congestion avoidance. By temporarily disabling congestion control, the source will maintain its state variable constant for a time period, say t1 seconds, after detecting an out-of-order event. By instant recovery during congestion avoidance, the source recovers immediately to the state before the congestion response, which has been invoked within t2 seconds ago.

However, TCP-DOOR does not distinguish between forward-path reordering or reverse-path reordering. The responses are suitable to alleviate some performance problems caused by forward-path reordering. They do not help reduce bursty traffic, and in fact exaggerate network congestion under reverse-path reordering. Besides, TCPDOOR does not perform well in a congested network environment with substantial persistent packet reordering. It disables congestion control for a time period every time an out-of-order event is detected, which may lead to congestion collapse from undelivered packets.

DelAck— Altman and Jiménez have advocated the use of delayed ACK techniques, known as DelAck [11], to improve the TCP performance in multihop wireless ad hoc networks. DelAck is a receiver-side solution to reduce channel contentions among data segments and ACKs of the same TCP connection. It will also reduce performance degradation due to packet reordering. In a multihop ad hoc network, the forward and reverse traffic between two adjacent hosts on the transmission path for the same connection may share and contend for the same channel. The approach of this proposal is to delay acknowledging the arrivals of data segments and reduce the number of ACKs sent to a source. The connection overhead and hence the channel contentions can be reduced. The idea is to let a receiver generate an ACK for every *d* data segments. An ACK is also generated whenever the first unacknowledged data segment has been received for a certain time period, say 0.1 s. The value of *d* can be configured so that *d* increases with the segment sequence number.

Drawbacks:

- The value of *d* is orthogonal to the segment sequence number in general. Indeed, the value of *d* may depend on the size of the congestion window, which in turn depends on the available bandwidth of a connection.
- The bursts of TCP segments may be injected into the network every time a delayed ACK is received by a source. This can lead to transient network congestion and congestion collapse from undelivered packets.

TCP-ADA — Singh and Kankipati developed TCP with "adaptive delayed acknowledegment" (TCP-ADA) [12]. It is a

receiver-side solution to reduce intra-flow channel contention in mobile ad hoc networks. The key idea of TCP-ADA is similar to that of DelAck [11]. However, DelAck defers acknowledgment until a certain number of data segments are received, while TCP-ADA postpones acknowledgment for a time period. Upon a data segment arrival, TCP-ADA updates Δ, an exponentially weighted moving average of the interarrival time between two successive segment arrivals. A destination will defer sending an ACK of the segment for a time period of $\beta\Delta$. The deferment period is restarted every time a data segment arrives before the deferment timer expires. An ACK is sent to a source if the total deferment period reaches a certain threshold.

Drawbacks:

- When an ACK is sent after receiving a full congestion window of data. This means that a source has to be idle for about one RTT to receive an ACK before it can send new segments to the destination then the loss of ACK-clocking to the network.
- A destination may send just one ACK for a full congestion window of data. The loss of that ACK leads to a long idle period for the connection followed by the expiration of the retransmission timer and the initiation of the slow start phase to reopen the congestion window starting from one segment.

TCP-DCR—Bhandarkar and Reddy devised the delayed congestion response TCP (TCP-DCR) [13] to meliorate the TCP robustness to noncongestion events. It advances the timedelayed fast retransmit algorithm by delaying a congestion response for a time interval after the first duplicate ACK is received.

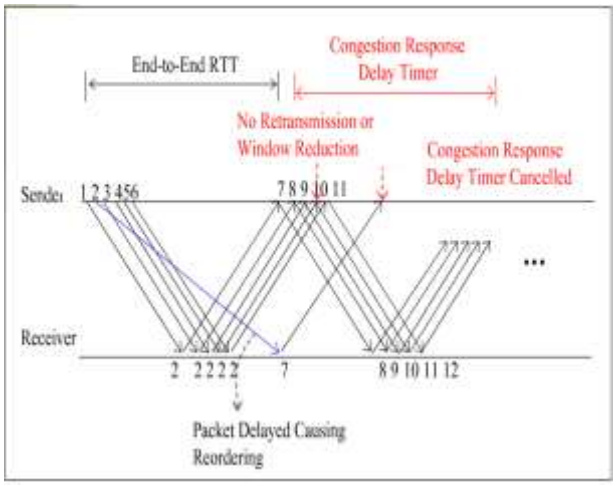


Fig. 5. An illustration of the TCP-DCR

The authors suggested setting this interval to one RTT so as to have sufficient time to deal with forward-path reordering due to link-layer retransmissions for loss recovery. To maintain ACK-clocking, TCPDCR sends one new data segment upon the receipt of each duplicate ACK.

Fig.5 represents an example that delays the congestion response for packet2 by using TCP-DCR. The simulation results in [13] demonstrated that TCPDCR performed significantly better than SACK TCP. TCP-DCR connection throughput hiked 10 times than SACK TCP when more than 5 percent of packets are delayed according to a normal distribution with negligible congestion loss. However, the chosen bottleneck link delay is at least equal to the highest possible reordered delay for their experiments. This implies that a reordering event is unlikely to last longer than the interval for delaying the congestion response.

IV. CONCLUSION

This paper studies the causes and solutions for packet re-ordering of TCP in wireless networks. We have presented solutions that prevent the unnecessary retransmits that occur due to reordering events in networks by allowing the TCP sender to distinguish whether a packet has been lost or reordered in the network. This taxonomy provides a unified terminology and framework for the comparison of different solutions and a theoretical study of the transport protocols will be discussed and developed in the forthcoming article.

REFERENCES

[1] D.D. Clark, "The Design Philosophy of the DARPA Internet Protocols," ACM SIGCOMM Computer Comm. Rev., vol. 18, no. 4,pp. 106-114, Aug. 1988.

[2]V. Paxson, "End-to-End Internet Packet Dynamics," EEE/ACM Trans. Networking, vol. 7, no. 3, pp. 277-292, June 1999.

[3] N. M. Piratla, A. P. Jayasumana, A. A. Bare, "A Comparative Analysis of Packet Reordering Metrics", Proceedings of the 1st International Conference on Communication Systems Software and Middleware (COMSWARE 2006), 2006.

[4] J. C. R. Bennett, C. Partridge, N. Shectman, "Packet Reordering is Not Pathological Network Behavior", IEEE/ACM Transactions on Networking (TON),Volume 7, Jun 1999.

[5] N. M. Piratla, A. P. Jayasumana, A. A. Bare, "A Comparative Analysis of Packet Reordering Metrics", Proceedings of the 1st International Conference on Communication Systems Software and Middleware (COMSWARE 2006), 2006.

[6] R. Ludwig and R.H. Katz, "The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions," ACM SIGCOMM
Computer Comm. Rev., vol. 30, no. 1, pp. 30-36, Jan. 2000.

[7] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, An Extension to the Selective Acknowledgement (SACK) Option for TCP, IETF RFC 2883, July 2000.

[8]M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, TCP Selective Acknowledgment Options, IETF RFC 2018, Oct. 1996.

[9]P. Sarolahti,M. Kojo University of Helsinki" F-RTO: An Algorithm for Detecting Spurious Retransmission Timeouts with TCP and SCTP" February, 2004

[10] F. Wang and Y. Zhang, "Improving TCP Performance over Mobile Ad-Hoc Networks with Out-of-Order Detection and Response,"Proc. ACM MOBIHOC '02, pp. 217-225, June 2002.

[11] E. Altman and T. Jiménez, "Novel Delayed ACK Techniques for Improving TCP Performance in Multihop Wireless Networks," Lecture Notes in Computer Science, vol. 2775, Sept. 2003.

[12] A. K. Singh and K. Kankipati, "TCP-ADA: TCP with Adaptive Delayed Acknowledgement for Mobile Ad Hoc Networks, "Proc. IEEE WCNC 2004, vol. 3, Atlanta, GA, USA, 21–25 Mar. 2004, pp. 1685–90.

[13] S. Bhandarkar et al., "TCP-DCR: A Novel Protocol for Tolerating Wireless Channel Errors," IEEE Trans. Mobile Computing, vol. 4, no. 5, Sept./Oct. 2005, pp. 517–29.

Selected Paper from International Conference on Computing (NECICC-2k15)