ISSN:0975-9646

Implementation of TCP Recognition of Broken Order (TCP-BO) Algorithm

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 transmission control protocol (TCP) is one of the most popular and widely used end-to-end protocols for the Internet today. Unlike routing, where packets are relayed hop-by-hop toward their destination, TCP actually provides reliable end to-end transmission of transport-level segments from source to receiver. As TCP was designed for wired networks it considers that all packet loss in the network is due to congestion. Wireless medium is more exposed to transmission errors and sudden topological changes. So in this paper, we have analyzed the performance of TORA ,which was one of important hierarchal routing protocol under TCP-SACK. In this paper, we also proposed TCP Recognition of Broken Order and Answer with Time Stamp (TCP-BO) algorithm, the negativity of TCP has been conquering, and it is confirmed that broken order packet is delivered due to multi-path routing protocol. The simulation shows that, TCP-BO algorithm has higher performance than TCP-SACK. The TCP with selective acknowledgement scheme (TCP SACK) improves TCP performance by allowing the TCP sender to retransmit packets based on selective ACKs provided by the receiver.**

Keywords— ***MANET, TCP, SACK.***

I INTRODUCTION

Transmission Control Protocol (TCP) is a connection oriented point-to-point protocol. It is a means for building a reliable communications stream on the top of the unreliable Internet Protocol (IP). TCP is the protocol that supports nearly all Internet applications.

TCP in the Internet protocol stack

Figure 1, Shows the structure of the Internet protocol stack, in which the TCP/IP is composed of the Network (IP) layer and Transport (TCP) layer. Each layer is responsible for a particular purpose which is to make various hosts to communicate with each other; hosts may be computers, or processes with in a computer. (IPS) Internet protocol stack is redesigned from the formerly used OSI reference model.

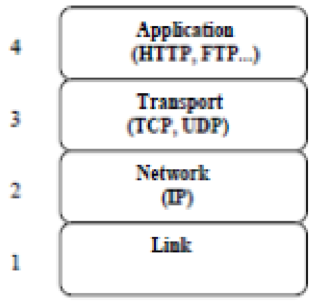


Figure 1: Internet protocol stack

The application layer is liable for the production and consumption of the user's data which passes through each layer of the stack and is transferred transversely the network.

Various Application programs such as, Internet Explorer, File Transfer, E-mail, and World Wide Web (WWW) run under this layer. Each application programs has a fragment in it that resides in the application layer; this fragment is called Application Layer Entity, e.g.

• *Internet Explorer:* It uses Hyper Text Transfer Protocol (HTTP) protocol of the application layer for the exchange of messages.

• *File Transfer:* It uses File Transport Protocol (FTP).

• *E-mail:* It uses Simple Mail Transfer Protocol (SMTP) to send messages.

• *World Wide Web (WWW):* It uses the application protocols with some additional components. The application layer and transport layer communicate with each other by using ports and sockets.

The transport layer is responsible for the end-to-end transmission of the data formed by the application layer. Mostly used transport protocols are the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP). UDP provides a unreliable data delivery over IP, as it is a connectionless transport protocol. While TCP is connection oriented and it guarantees delivery of the packets routed by the network layer. Every application is coupled with an exacting port number in the transport protocol.

The network layer is mainly work with routing of packets between sender and receiver hosts. For this purpose, every node in the global Internet is having a unique IP address, which is helpful for uniquely identifying a specific host. The Network layer supports different routing protocol like Dynamic Source Routing Protocol (DSR), Temporally Ordered Routing Algorithm (TORA).

The link layer or data-link layer or network interface layer specifies the mechanism of how the packets of the network layer are transported over the physical medium between two nodes. The data link layer uses ARP Address Resolution Protocol in order to determine the IP address of the particular host in to hardware. It deals with physical transmission details such as frame size, synchronization, etc.

A. Connection setup

In TCP, both the hosts (sender and receiver) that want to communicate with each other for a certain amount of time, they handshake with each other. Handshaking consists of three phases. Connections are established in TCP by means of the three-way handshake. To establish a connection, one side, say, the server passively waits for an incoming connection by executing the LISTEN and ACCEPTS primitives, either specifying a specific source or nobody in particular. The other side, say, the client, executes a CONNECT primitive, specifying the IP address and port to which it wants to connect, the maximum TCP segment size it is willing to accept, and optionally some user data (e.g., a password). The CONNECT primitive sends a TCP segment with the SYN bit on and ACK bit off and waits for a response.

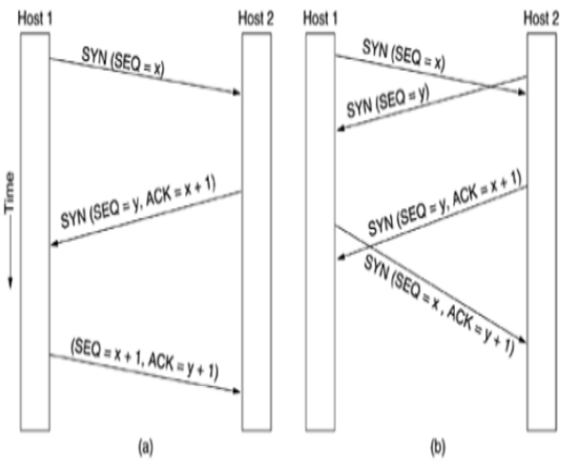

Host 1
Host 2
Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y, ACK = x + 1)
(SEQ = x + 1, ACK = y + 1)
Time
SYN (SEQ = x)
SYN (SEQ = y)
SYN (SEQ = y, ACK = x + 1)
SYN (SEQ = x, ACK = y + 1)
(a)
(b)

Figure 2: Concept of Hand Shaking

B. 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 with in 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 Acknowledgement 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 [4].

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.

B. Temporally Ordered Routing Algorithm (TORA)

The Temporally Ordered Routing Algorithm (TORA) is a highly adaptive and competent disseminated multi-path routing algorithm based on the concept of link reversal. TORA is projected for highly vibrant mobile, multihop wireless networks. It is a source-initiated on-demand routing protocol. This means, the protocol is called whenever and where ever there is a need to route packets between sender-receiver pair.

It finds multiple routes from a source node to a destination node. The main feature of TORA is that the control messages are localized to a very small set of nodes near the occurrence of a topological change. All the nodes maintain routing information about adjacent nodes. This protocol has three basic functions:

Route creation

Route maintenance

Route erasure [9].

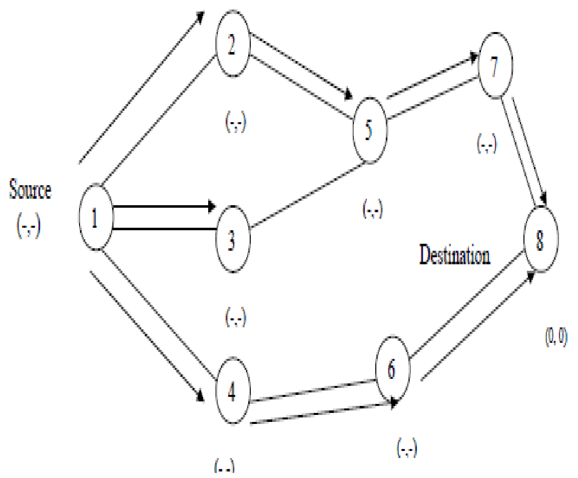
Each node has: –

- Logical time of a link failure
- The unique ID of the node that defined the new reference level
- A reflection indicator bit 0= original level, 1=reflected level
- A propagation ordering parameter to order nodes relative to reference level
- The unique ID of the node

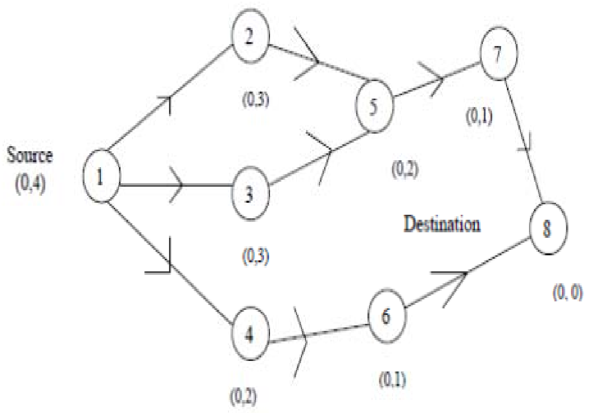
The first three elements collectively represent the *reference level*. A new reference level is defined whenever a node loses its last downstream link due to a link failure. The last two values define a *height* with respect to the reference level.

The **route creation algorithm** starts with the height (propagation ordering parameter) of destination set to 0 and all other node's height set to NULL (i.e. undefined). The source broadcasts a QRY packet with the destination node's id in it. A node with a NONE-NULL height responds with a UPD packet that has its height in it. A node receiving a UPD packet sets its height to one more than that of the node that generated the UPD. A node with higher height is considered upstream and nodes with lower height downstream. In this way a directed acyclic graph (DAG) is constructed from source to the destination. Figure 2.5 illustrates a route creation process in TORA [9]. As shown in figure 2.a, node 5 does not propagate QRY from node 3 as it has already seen and propagated QRY message from node 2. In figure 2.b, the source (i.e. node 1) may have received a UPD each from node 2 or node 3 but since node 4 gives it lesser height, it keeps that height.

When a node moves, the DAG route is broken; route maintenance is needed to re-establish a DAG for the same destination. When the last downstream link of a node fails, it generates a new reference level. This results in the propagation of that reference level by neighbouring nodes as shown in figure 3. Links are reversed to reflect the change in adapting to the new reference level. This has the same effect as reversing the direction of one or more links when a node has no downstream links.



(A) Propagation of QRY message through the network



(B) Height of each node updated as a result of UPD messages

Figure 3: Route creation in TORA

In the route erasure phase, TORA provides a broadcast clear packet (CLR) throughout the network to erase invalid outes.

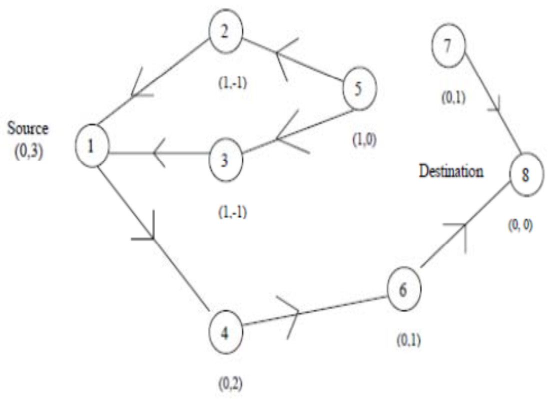


Figure 4. Re-establishing route on failure of link 5-7. The new reference level is node5

II. Related Work

There are many simulation-based investigations of the problematic aspects of TCP in wireless network environments. For example, [9] 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 [10] 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 [11] 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.

There are also many surveys exploring the problematic aspects of TCP over wireless networks (e.g. [12, 13]). Some of them analyze existent solutions for dealing with

TCP throughput degradation due to handovers and lossy links. In [12] the authors conclude that selective acknowledgements and explicit loss notifications can significantly improve performance. Several works propose models for TCP throughput prediction, e.g., [14, 15, 16, 17, 18]. An overview of these works can be found in [14, 15]. The authors of [16] present a model for studying the effect of general packet loss caused by network congestion on TCP Reno.

It predicts the throughput of a TCP connection as a function of loss rate and RTT. It assumes that if a packet is lost, all the remaining packets transmitted until the end of that round will be lost as well. The impact of mobility on the performance of TCP has drawn a lot of attention, in the last years. For example, the authors of [22] analyze packet drop scenarios in cellular networks, which are attributed to handovers, to poor wireless link conditions or to congestion. They suggest incorporating a finite state Markov channel model into the TCP flow control in order to adapt the response of the sender to the real cause of the packet drop. They show that this model indeed improves TCP performance in cellular networks.

In [19], the authors calculate the probabilities for packet loss, taking into account network congestion and the loss caused by handovers. Then, they incorporate these probabilities into the prediction equations from the Amherst model [16]. In [22], the authors concentrate on calculating loss probabilities and finding the throughput as a function of the amount of lost data.

With the increasing capabilities of mobile devices and of the data rates offered by mobile networks, mobile multimedia services over TCP become popular. Examples to these services are Smooth Streaming by Microsoft [17] and Live HTTP Streaming by Apple [18]. An overview of the multimedia streaming standards for 3GPP mobile networks is presented in [19]. In [20], the authors present an analytical model for evaluating the performance of multimedia streaming over TCP. They explore how various network parameters, such as delay, loss rate and retransmission timeout, effect the throughput of TCP streaming applications. They also show that TCP streaming provides good performance when the available network bandwidth is roughly twice the video bit-rate. In [21], the authors address the issue of limiting the latency introduced by TCP. They discuss the importance of low latency to streaming applications and show how such latency can be obtained using dynamic adaptation of the TCP sender's buffer.

In [21], the authors discuss the importance of TCP-friendliness to adaptive streaming in mobile networks. Using simulations, they show that the ability of a TCP-friendly protocol to dynamically adapt the bit-rate of the stream can significantly improve various performance indicators in mobile networks, such as loss rate, delay and buffer space.

In several papers the authors propose schemes to be used by the intermediate node to inform the sender of route failure [23, 24]. With Explicit Link Failure Notification (ELFN) [24], a node that detects a link failure notifies the TCP sender. The sender then freezes its retransmission timer and periodically sends a probing packet until it receives

• The Route-control mechanism informs the sending node not only about the need to retransmit a packet, but also about the need to take actions due to the change of route. EPLN does not provide this information, because with source-routing the sending node is supposed to know when a new route is used.

• EPLN plays a role when a route is broken. In contrast, the Route-control mechanism helps when a route changes.

III. TCP Detection Of Broken Order And Response With Time Stamp Option (Tcp- Bo) Algorithm

TCP-BO algorithm is an end-to-end layered proposed solution to improve the performance of TCP in MANET. It deals with TCP layer in TCP/IP architecture. In the TCP implementation in wired networks, TCP assumes that packet losses are mainly due to congestion in the network. As a result, when a packet loss is indicated, it enters into congestion control and packet recovery algorithms. This may not be suitable in a wireless network. Because it maintains multiple routes between sender and receiver, and a significant amount of packet losses could be due to delivery of broken order packets and packet loss. The TCP-BO algorithm is a modification in TCP-SACK both at the receiver and sender side, aiming to provide some possible solution by responding for detection of broken-order packets and changes the time at which congestion control algorithm is invoked.

Once broken-order packet is detected, The TCP receiver informs the TCP sender by setting broken-order bit in the TCP header. On the reception of broken order bit, for a time period T the TCP sender: -

1- Temporarily disables some of the state variables like Duplicate Acknowledgment (DUPACK) and RTO.

2- Adjusts the Congestion Window Size (CWND) based on the network condition.

3- Send one new packet.

Otherwise,

- If out-of-order packet is not detected by the receiver and the receiver sends three DUPACKs. On the reception of the first DUPACK at the sender side, for a time period T disable triggering of duplicate acknowledgment algorithm and send one new packet.
- If within the disabling period the packet that is out-of-order is not reached, call congestion control algorithm.

A. TCP-BO Algorithm

Algorithm 1. Pseudo-code of broken-order packet detection at the receiver side

```
1- Retrieve the time stamp (present_pkt_ts_) from the incoming packet, which is sent by TCP sender.
2- Retrieve the previously saved packet time stamp (saved_time_) that is recorded in TCP receiver.
3- Compare the present packet time stamp with the previously received packet time stamp, i.e.:
If (present_pkt_ts_ < saved_time_) {
```

```
1- Set broken order option bit (bo_option_) to 1
2- Send broken-order option bit (bo_option_) with the ACK packet to be acknowledged to the TCP sender.
3- Put the value of present packet time stamp (present_pkt_ts_)
into saved time (saved_time_) for the next comparison.
4- Then reset bo_option_ = 0, present_packet_ts_ = 0 and other necessary variables.
// bo_option_ a variable which should be put in the TCP
header (tcp.h)
}
Else {
In -sequence packet is received
Send ACK (same as TCPSINK does)}
```

Algorithm 2. Pseudo-code of broken-order packet response at the sender side

```
1 – Retrieve the received acknowledgment header and check for broken-order option bit (bo_option_).
If (bo_option bit is set to 1) {
For a time period T, The TCP Sender disables its state variables
i.e.
For (Disable period = RTT) {
1 - Set number of duplicate acknowledgment (dupacks_) to zero.
2 – Set the retransmission time out (timeout_) to false.
3 – Adjust the congestion window size (cwnd_) by invoking the
Open congestion window algorithm.
4 – Send one new packet.
}
}
Else if (Three duplicate ACK) {
For (Disable period = RTT)
{
- Invoke send one algorithm in order to send one new Packet.
If (disable period expires)
{
Call fast retransmission and fast recovery algorithms.
}
}
}
Else if (standard ACK is received)
{
Do as standard TCP-SACK does
}
```

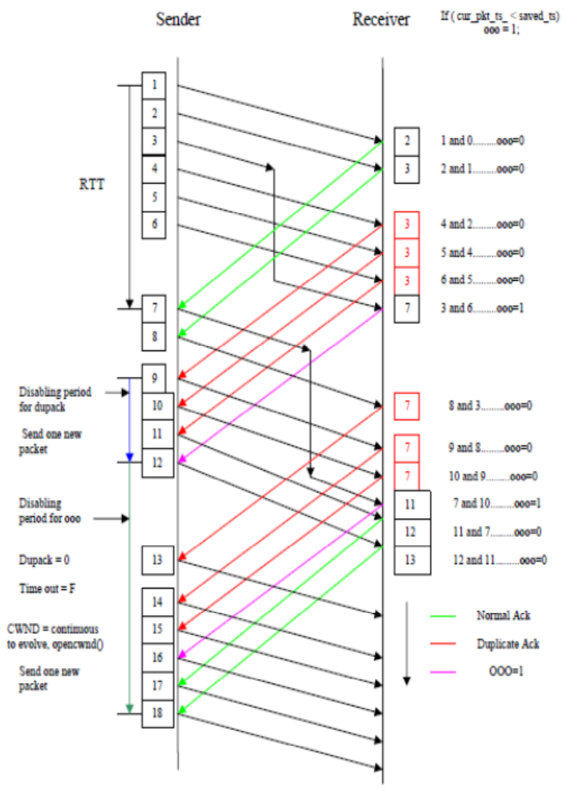


Figure 5: TCP-BO Broken order detection of packet and response mechanism

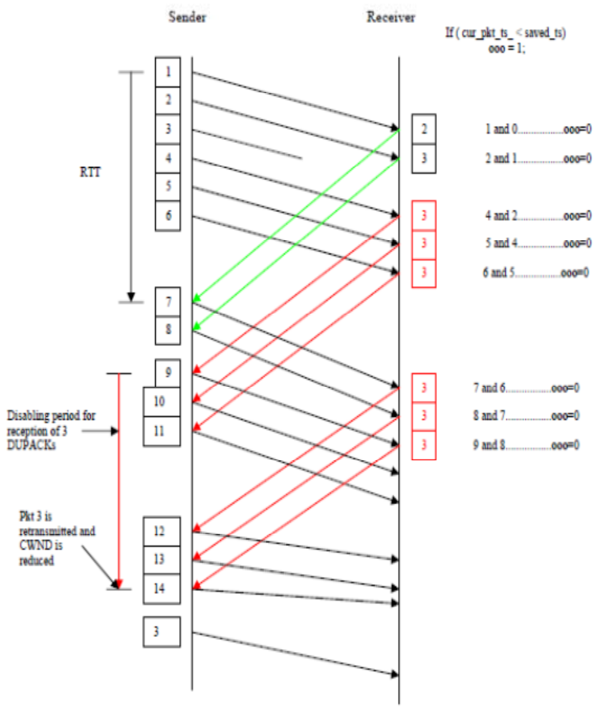


Figure 6. TCP-BO packet loss detection

IIII. PROPOSED METHODOLOGY

In this paper, 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. Proposed TCP-BO algorithm improves the performance of TCP by detecting and responding for out-of-order packet delivery. TCP-BO algorithm tries to reduce unnecessary invoking of congestion control algorithm and retransmission of packets.

A. Implementation of TCP-BO algorithm using NS-2

The implementation of TCP-BO algorithm is based on one of the TCP version known as TCP-SACK modules both at the receiver and sender side. At the TCP receiver side the recv () method, which is called whenever new packet is received, and the ack () method, which is processed whenever an acknowledgment is prepared to send to the TCP sender, are mainly used to implement the proposed algorithm. Both the methods are modules of the SACK at the TCP receiver and sender side. Necessary modifications are made in the agent named class TCPSINK and its recv () function, which is the main reception path for packets and provides various other necessary methods. One variable known as ooo_option has been included in the TCP header format so as to inform the sender about the detection of out of order bit. At the sender side the main modification is done on the receive method. This method should check for two variables, namely ooo_option bit and scoreboard. Up on the reception of ooo_option bit it checks for the value of this bit, if it is one, some of the variables like number of duplicate acknowledgment (numdupack_), and retransmission timeout (timeout_) will be disabled and opencwnd () will be called so as the congestion window to evolve. Lastly, send_one () function will be called, which sends one new packet in order to utilize the network effectively. If ooo_option bit is set to 0 and numdupack_ becomes increments, the TCP sender will check this increment from the variable known as scoreboard, that is it Checks for a duplicate ACK, which Check the SACK block actually acknowledges new data or not and one the increment of numdupack_ = 1, TCP sender will disable triggering of dupack () function, instead it will call sendone() algorithms so as to send one new packet, if within the disabling period the packet which is lost or out of order is not come, dupack () function will be called, which retransmit the lost packet and reduce the congestion window size cwnd_. Figure shows the implementation hierarchy and the important C++ classes for TCP-BO implementation are shown with darker shade.

Table I *Salient Simulation Parameters*

Parameter	Value
Simulation time	150 Sec
Simulation area	1000m x 1000m
Routing Protocol	TORA
No. of nodes	20
Packet size	512 Bytes
Max queue length	50
TCP-Variant	TCP-SACK
MAC type	IEEE-802.11
Proposed Algorithm	TCP-BO

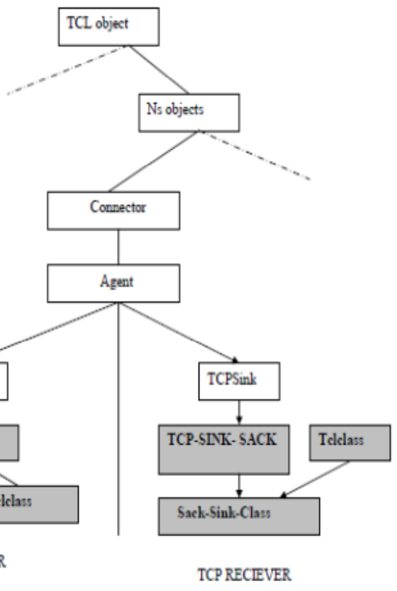


Figure 7. Implementation Hierarchy Of NS2

In Figure 6, between 100 and 110 simulation seconds, the congestion window size reduced a lot, which is the effect of out-of-order delivery of packets. This reduces the TCP's sender CWND that affects the maximum number of packets that can be sent by the TCP sender (throughput). 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. Figure 6, shows the effect of proposed TCP-BO algorithm that reduce unnecessary invoking of congestion control algorithm and retransmission of packets.

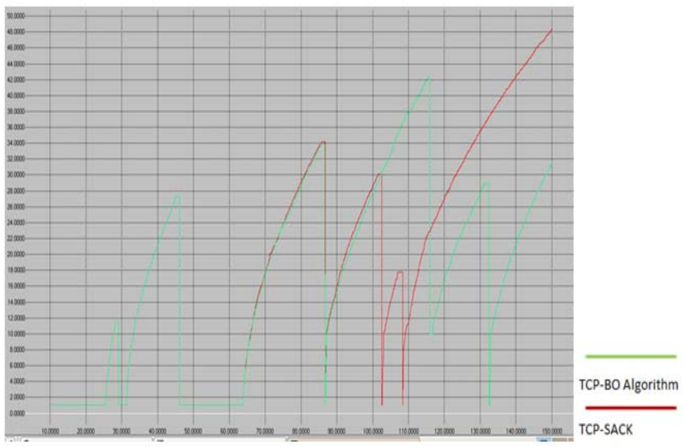


Figure 7: Congestion window Vs simulated time.

Figure 7, shows the size of congestion window for simulated time that is somehow good in case of TCP-BO algorithm relative to base TCP-SACK.

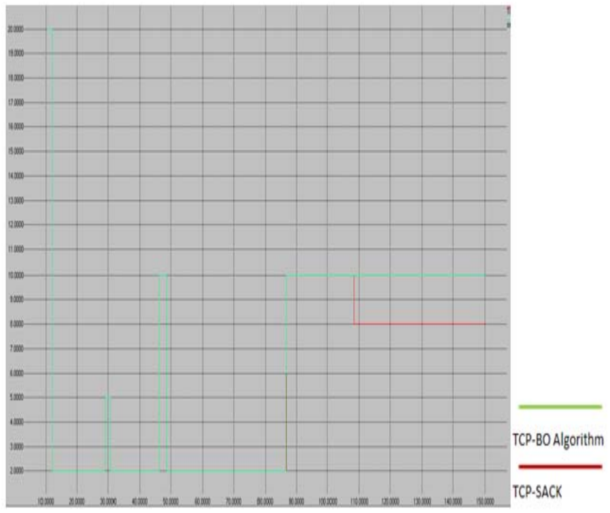


Figure 8 : Slow start threshold versus simulation time.

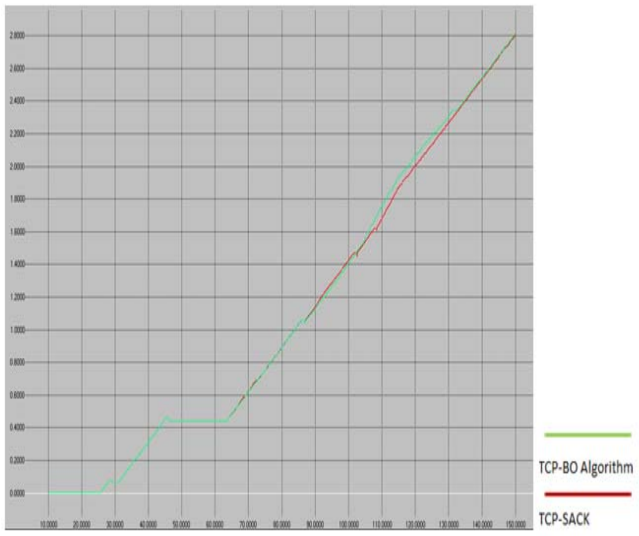


Figure 8: Sequence number versus simulation time.

V. Conclusion And Future Work

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. Proposed TCP-BO algorithm improves the performance of TCP by detecting and responding for out-of-order packet delivery. TCP-BO algorithm tries to reduce unnecessary invoking of congestion control algorithm and retransmission of packets. All measures taken by the TCP-BO algorithm improve the bandwidth utilization and increase the performance of routing protocol (TORA) as compared to TCP-SACK.

Further in this direction our aim is to compare the performance of other routing protocols under proposed TCP-BO congestion avoidance mechanism by avoiding unnecessary retransmission.

References

[1] K.Sunderesan, V.Anantharaman, R.SivaKumar, "ATP reliable transport protocol for adhoc networks," Proceeding of 4th ACM international Symposium on Mobihoc, June 2003pp 64-75.

[2] B. Bakshi, P. Krishna, N.H. Vaidya, and D.K. Pradhan, "Improving Performance of TCP over Wireless Networks," Proc. 17th Int'l Conf. Distributed Computing Systems (ICDCS), May 1997.

[3] M. Gerla, K. Tang, and R. Bagrodia, "TCP Performance in Wireless Multi Hop Networks," Proc. IEEE Workshop Mobile Computing Systems and Applications, Feb. 1999.

[4] G. Holland and N.H. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks," Proc. ACM MOBICOM Conf., pp. 219-230, Aug. 1999.

[5] J.P. Monks, P. Sinha, and V. Bharghavan, "Limitations of TCP ELFN for Ad Hoc Networks," Proc. Workshop Mobile and Multimedia Comm., Oct. 2000.

[6] T.D. Dyer and R. Bopanna, "A Comparison of TCP Performance over Three Routing Protocols for Mobile Ad Hoc Networks," Proc. ACM MOBIHOC 2001 Conf., Oct. 2001.

[7] Sami Iren and Paul D Amer, "The Transport Layer tutorial and survey" ACM computing survey, vol 31, No. 4, December 1999.

[8] Yue Fang , McDonald A .B "Cross layer performance effect of path coupling in wireless adhoc network power and throughput implications of IEEE 802.11 MAC "conference 2002,21st IEEE International , pp 281-290.

[9] A. Gurtov, "Effect of delays on TCP performance," in *Working Conference on EmergingPersonal Wireless Communications*, 2001, pp. 87–108.

[10] A. Lahanas and V. Tsaoussidis, "Experimental evaluation of TCP-Probing in mobile networks," *Journal of Supercomputing*, vol. 23, pp. 261–279, 2002.

[11] J. Schuler and T. Schwabe, "A comparison of the performance of four TCP versions during mobile handoff," *IEEE MWCN*, Sept. 2002.

[12] H. Balakrishnan and et al., "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 756–769, Dec. 2000.

[13] V. Tsaoussidis and I. Matta, "Open issues on TCP for mobile computing," *Wireless Communications and Mobile Computing*, vol. 2, pp. 3–20, 2002.

[14] E. Altman, K. Avrachenkov, and C. Barakat, "A stochastic model of TCP/IP with stationary random losses," *IEEE/ACM Transactions On Networking*, vol. 13, no. 2, pp. 356–369, April 2005.

[15] M. Goyal, R. Guerin, and R. Rajan, "Predicting TCP throughput from non-invasive network sampling," *INFOCOM*, 2002.

[16] J. Padhye, V. Firoiu, D. F. Towsley, and J. F. Kurose, "Modeling TCP Reno performance: a simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 133–145, 2000.

[17] A. Kumar, "Comparative performance analysis of versions of TCP in a local network with a lossy link," *IEEE/ACM Transactions on Networking*, vol. 6, no. 4, pp. 485–498, 1998.

[18] V. Argyriou, Antonios; Madisetti, "WLC47-1: Modeling the effect of mobile handoffs on TCP and TFRC throughput," *GLOBECOM*, 2006.

[19] A. Zambelli, "IIS smooth streaming technical overview." [Online]. Available: http://www.microsoft.com/downloads/

[20] A. Goel, C. Krasic, and J.Walpole, "Low-latency adaptive streaming over TCP," *ACM TOMCCAP*, vol. 4, August 2008.

[21] K. Tappayuthpijarn, G. Liebl, T. Stockhammer, and E. Steinbach, "Adaptive video streaming over a mobile network with TCP-friendly rate control," *IWCMC*, pp. 1325– 1329, 2009.

[22] F. Hu and N. Sharma, "The quantitative analysis of TCP congestion control algorithm in third-generation cellular networks based on FSMC loss model and its performance enhancement," *INFOCOM*, 2002.

[23] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "feedback-based scheme for improving TCP performance in ad hoc wireless neworks," *IEEE Pers. Comm. Mag.*, vol. 8, no. 1, Feb. 2001.

[24] G. Holland and N. H. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," in *MobiCom*, Aug. 1999.