ENSC 835: COMMUNICATION NETWORKS

Evaluation of TCP congestion control mechanisms using OPNET simulator

Spring 2008

FINAL PROJECT

REPORT

LAXMI SUBEDI

http://www.sfu.ca/~lsa38/project.html

lsa38@cs.sfu.ca

Acknowledgment

I would like to express my gratitude to Professor Ljiljana Trajković for providing us the environment and opportunity to conduct project. I heartily appreciate her encouragement, guideline and feedback presenting wide range of thesis works, conference papers and ideas to pursue project. I would like to thank all my classmates for their support and mutual discussion to solve problems.

Finally I would like to thank system administrators from Department of Engineering Science for helping us to set up the environments and troubleshoot problems.

Abstract

Transmission Control Protocol (TCP) is the major protocol that provides reliable delivery of packets suitable for various application protocols such as FTP, HTTP, SMTP and SSH. TCP provides error-free data transfer, proper control of data flow, and manage congestion. However, TCP requires enhancements in order to reliably handle loss, minimize errors, and successfully manage congestion in wireless and high-speed network. Various TCP flavors have been developed to avoid congestion, such as Tahoe, Reno, SACK, NewReno, Vegas, Hybla, BIC, and CUBIC.

This project observes, analyzes, and compares congestion window maintenance and recovery process by different TCP flavors namely: Reno, SACK, and NewReno. For this purpose, three different topologies with different scenarios are simulated using Opnet simulator. For each scenario, three different algorithms are implemented. The maintenance of congestion window by each algorithm on different cases (e.g no packet drop, single drop, multiple drops, disconnection, and congested network) along with the corresponding file download response time are observed and compared. The result indicates that all three algorithms maintain congestion window in a similar way in case of no packet loss and all packets loss. During multiple packets loss SACK and NewReno gain congestion window rapidly than Reno does. The short file download response time of SACK shows that it behaves better in case of multiple packets loss. All three algorithms are insensitive to the link disconnection and hence reduce the congestion window to minimum. But among three algorithms, SACK recovers congestion window faster than Reno and NewReno in case of short disconnection period. In case of heavily congested network, none of the algorithms improves the congestion window consistently. This is due to the cumulative behavior of each algorithm in different cases. However, the overall performance of SACK is better than Reno and NewReno.

Table of Contents

List of Tables

List of Figures

I. Introduction

1.1 Overview

Transmission control protocol (TCP) is mostly used protocol in Internet. It is widely used as connection oriented transport layer protocol that provides reliable packet delivery over an unreliable network. In theory, TCP is independent of the underlying network layer so the design of various TCPs is based on wired network. However, in practice these implementations do not work perfectly as in wired network for all sort of underlying networks. So the phenomenon of congestion in TCP has huge concern in present networking scenario.

Different congestion control algorithms have been proposed based on the situation for wired network where the congestion is the only cause for the loss of packet. However, wireless reveals higher bit error rates due to various factors like weather conditions, obstacles, multipath interferences, mobility of wireless end devices which can lead to the loss of packets. Various techniques have been proposed to improve congestion and reduce the non-congestion related loss on TCP's performance. For example, Reno, SACK, NewReno, Vegas, BIC, and CUBIC are end-to-end control approaches; Snoop-TCP is link layers control approach, M-TCP and I-TCP are split connection approaches. Among various approaches the end-to-end technique seems to be the most promising scheme as they do not require expensive changes in the intermediate nodes to improve performance.

This project performs a comparative study of various congestion control algorithms that implements end-to end control approaches to improve congestion in case of wired as well as link disconnected network. The analysis on behavior of different algorithms for different situations is performed on the models developed using network simulator.

1.2 Project objective

The objective of the project is to observe, analyze and compare congestion window maintenance and recovery process by different congestion control algorithms and avoidance mechanisms by modeling a network using network simulator package, OPNET 11.0 tool.

1.3 Project scope

Among various available and purposed congestion avoidance algorithms; this project implements Reno, SACK, and NewReno, end-to-end approaches, to observe their congestion control mechanism and congestion window recovery process. In order to determine their performance under different conditions, simple client server topology is built using Opnet 11.0. The custom applications that use TCP (ftp, database) are used for different simulation scenarios. The congestion window and file download response time are observed for various packet loss. The mechanism to maintain congestion window in case of link disconnection and heavily congested network is also observed and compared.

1.4 Organization of document

General introduction, project objective and scope is described in section one. The relevant background theory, some useful definition, and discussion of relevant literature are discussed in section two. Section 3 describes the implementation of the project, different simulation scenarios followed by the analysis of result. Discussion and final conclusion are described in section four. The limitations and suggested future work are in section five. Section six contains the list of references followed to achieve this project.

II. Background theory and Literature review

2.1 Definitions

TCP is connection-oriented point-to-point protocol. It provides services like flow control, reliable, error free data transmission as well as congestion control suitable for various application protocols such as FTP, HTTP, SMTP and SSH. About 90% internet traffic is carried by TCP protocol in today's heterogeneous network. [8] However, TCP was developed base on wired network properties so it has become weak in case of hybrid network. The problem arises due to the design assumption of TCP based on wired network and the difference in properties of wired and wireless network. The problems like throughput degradation, inefficient network resource utilization and excessive interruption of data transmission. Among various problems congestion is one of them.

TCP uses sliding window protocol for end to end flow control where receiver advertises the window that determines the amount of data that sender can send. But in case of routers and slower links between the sender and receiver, intermediate routers placed the packets in their buffer. However, the buffer is of finite size. So there exists a problem when the intermediate router run out of memory and drop the packets. TCP protocol uses congestion window that determines the number of bytes which can be outstanding at any time in the link. This helps to stop the link between two places from getting overloaded with too much traffic and dropping packets. The size of this window is calculated by estimating congestion between two places. Once this size, the maximum number of bytes that can be transmitted without acknowledgment of sent packets, is calculated then it helps to minimize router buffer overflow. Theoretically, the size of the window, to a large extent, controls the speed of transmission by stopping the transmission until the acknowledgment is received in case of congestion.

However, in case of end-to-end control approach, network layer does not provide any support and information regarding congestion to the transport layer to control it. The congestion in the network is inferred by packet loss and delay only (triple duplicate or timeout).

Accordingly, TCP decreases its congestion window. There are various approaches to control congestion window in the network like slow start algorithm, Addititive Increase and Multiplicative Decrease (AIMD), Fast retransmit and Fast recovery. Different TCP flavors use these general congestion control algorithms to regulate the sending rate as a function of perceived congestion. These basic algorithms are described below:

Slow Start

When a new TCP connection is established between host and client, congestion window is initialized to 1 Maximum segment size (MSS). The sender starts transmitting by sending 1 MSS. When it received its acknowledgment then it sends two segments and so on. This process increases TCP sender's congestion window exponentially by doubling its value of congestion window at every round trip time (RTT). Thus this increases the sending rate rapidly and hence helps to utilize the available bandwidth effectively. TCP sender continues to increase its sending rate exponentially until there is a loss event (congestion) or it reaches the size of advertised receiver window, whichever is minimum. This phase is known as the TCP slow start phase where TCP increases its congestion window by 1 MSS for every acknowledgement received.

Additive Increase Multiplicative Decrease

According to the multiplicative decrease approach, when a sender perceived the loss event it reduces its congestion window (cwnd) by reducing the transmission rate of packets into the network. The amount of decrease of the sender congestion window depends upon the nature of congestion perceived by the sender. When the congestion occurs due to a packet loss (received duplicate ACKs for same packets), it decreases the congestion window to the half of the current value of the congestion window. If another congestion loss occurs it reduces the congestion window to half of the current value (which becomes one-fourth of the previous value). It this way, the congestion window value continues to drop. But the maximum drop of window size is 1 MSS.

According to additive increase, when TCP sender does not perceive congestion then it increases the congestion window by increasing transmission rate. It increases the congestion

window when acknowledgment packet arrives for previously yet-to-be acknowledged data. It increases the congestion window by 1 MSS for every round trip time (RTT). The increase is slow for additional use of the bandwidth in the end to end path.

Hence this process of additive increase in congestion window when TCP sender perceives congestion free path and multiplicative decrease when it perceives congestion is known as Additive Increase and Multiplicative Decrease algorithm.

Congestion Avoidance

TCP maintains a variable called slow start threshold (ssthresh) at which point the exponential increase of window stops. When the congestion window reaches to ssthresh value, TCP sender starts the congestion avoidance phase. During this phase it starts to increase the TCP window linearly (additive increase) instead of exponentially which results in the increase of 1 MSS every RTT. This phase is known as the congestion avoidance phase where the congestion is avoided by slowly increasing the congestion window.

Fast Retransmit and Fast Recovery

With Fast retransmit and recovery algorithm, the congestion due to retransmission timeout or due to packet loss is differentiated and acts accordingly in order to maintain maximum utilization of resources.

In case of fast retransmit algorithm, if a receiver receives a data segment that is out of order, it immediately sends a duplicate acknowledgement to the sender. If the sender receives three duplicate acknowledgements, it assumes that the data segment indicated by the acknowledgements is lost and immediately retransmits the lost segment.

In case of Fast recovery, after receiving three duplicate acknowledgments for the same TCP segment, the TCP sender infers that a packet has been lost and retransmits the packet without waiting for a retransmission timer expires according to the fast retransmit algorithm. When sender TCP receives three duplicate acknowledgments, it reduces the congestion window to half of the current congestion window rather than reducing the congestion window to 1MSS. Then it starts to increase the congestion window as per congestion avoidance algorithm. So

fast recovery algorithm helps to resumes higher channel utilization and connection throughput quickly.

Different TCP flavors are developed based on the above basic algorithms. Few of the TCP flavors that use these basic algorithms, are Reno, SACK, and NewReno.

Reno

TCP Reno is most widely adopted TCP protocol scheme. It uses four transmission phases; slow start, congestion avoidance, fast recovery, and fast retransmit. Reno employs a sliding – window based flow control mechanism allowing the sender to advance the transmission window linearly by one segment upon reception of an ACK, which indicates the last in-order packet received successfully by the receiver. When the packet loss occurs at a congested link due to buffer overflow at the intermediate router either the sender received three duplicate acknowledgments, or the senders retransmission timeout timer expires. These events activate TCP's fast retransmit and recovery by which the sender reduces its congestion window size to half and linearly increases congestion window as in congestion avoidance, resulting in a lower transmission rate to relieve the link congestion.

In Reno, fast recovery is entered by TCP sender after receiving an initial threshold of three duplicate acknowledgments. Once three duplicate acknowledgments are received, the sender retransmits one packet and reduces its congestion window by one half. Instead of slow start, the Reno sender use additional incoming duplicate acknowledgment to clock subsequent outgoing packet. So the useable window of user is a receiver advertised window or sum of sender congestion window and number of duplicate acknowledgment, whichever is minimum. Once it enters the fast recovery, it retransmits a single packet and remains in the fast recovery until it receives duplicate acknowledgments then transmits a new packet for additional received duplicate acknowledgment. [1]

Reno fast recovery algorithm improves its behavior when a single packet is dropped from a window of data. Reno sender retransmits one dropped packet per RTT at most. But it suffers in performance when multiple packets are dropped from a window of data. [1]

SACK

The SACK TCP allows a TCP receiver to acknowledge out-of-order segment selectively rather than cumulatively acknowledging the last correctly received, in order segment. It uses the same algorithms for increasing and decreasing the congestion window, and makes minimal changes to the other congestion control algorithm. SACK behaves similarly as Tahoe and Reno TCP which are robust in case of out of order packets arrival, and uses the retransmit timeouts as the final recovery method. Reno suffers in performance when multiple packets are dropped from a window of data. So SACK option helps to improve the performance in case of multiple packets loss. Hence the main difference between Reno and SACK arises when multiple packets are dropped from one window of data. As in Reno, the SACK TCP implementation enters Fast recovery when data sender receives three duplicate acknowledgments. The sender transmits a packet and cuts the congestion window in half. During the fast recovery, SACK maintains a variable called pipe that represents the estimated number of packets outstanding in the path. The sender only sends new or retransmitted data when the estimated number of packet in the path is less than the congestion window. The variable pipe is incremented by one when the sender either sends a new packet or retransmits an old packet. It is decremented by one when the sender receives the dup ACK packet with a SACK option. [1]

NewReno

NewReno is a modification of Reno. NewReno improves retransmission during the fast recovery phase of Reno. In Reno, sender remains in fast recovery until either a retransmission timeout or until all of the data outstanding when it entered the fast retransmit has been acknowledged. So problem of multiple fast retransmit from a single window of data can only occur after a RTO. NewReno is able to detect multiple packet losses so it is more efficient that Reno where multiple packet loss occur. [2]

NewReno exhibits all four phases of slow start, congestion avoidance, fast retransmit and fast recovery. It also enters into fast retransmit when it receives three duplicate acknowledgments. But it differs from Reno where NewReno doesn't exit fast-recovery until all the data which were unacknowledged at the time it enters fast recovery are acknowledged.

Thus it overcomes the problem faced by Reno of reducing the congestion window multiples times. The fast retransmit phase is similar to that of Reno. The difference in the fast recovery phase which allows for multiple re-transmissions in NewReno. Whenever NewReno enters fast recovery it notes the maximum segment which is outstanding. When NewReno receives new acknowledgment it acts in two ways as follows:

- If it acknowledges all the segments which were outstanding when it entered fast recovery then it exits fast recovery and sets congestion window to slow start threshold and continues congestion avoidance phase. [6]

- If the acknowledgment is a partial acknowledgment then it perceives that next segment in line was lost and it re-transmits that segment and sets the number of duplicate acknowledgment received to zero.[6] Partial acknowledgments are the acknowledgment segments that do not acknowledge all the information that has been sent up to the moment when they are issued.

It exits Fast recovery when all the data in the window is acknowledged. NewReno performs as well as SACK at low packet error rates, and substantially outperforms Reno at high error rates. [6].

2.2 Literature review

A considerable sum of researches has been conducted to study the effect of different algorithms to recover and maintain congestion window in various scenarios. Since the behavior of hybrid network is unpredictable, none of the algorithms works perfectly in all cases and scenarios. Algorithms which are theoretically better do not perform perfectly in real system as these algorithms do not consider the effect of all parameters associated in the network. Most of the time the nature of all the variables associated with the network cannot be predicted exactly. Obviously, the algorithms are tested in the simulated network with assumptions and hence the effect of algorithms in real environments is different. Despite the facts, various algorithms have been developed to improve the performance of the network and maximum utilization of end-to-end resources.

Among various algorithms and TCP flavors, I conducted study on Reno, SACK, and NewReno as these are the algorithms that are mostly used in today's internet. According to Sally Floyd and et al., NewReno which is a modified version of Reno in absence of SACK avoids some of Reno's performance problems when multiple packets are dropped from a window of data. But it still imposes limits to the ultimate performance of TCP in the absence of SACK. The paper states that, without selective acknowledgments, TCP implementations are constrained to either retransmit at most one dropped packet per round-trip time, or to retransmit packets that might have already been successfully delivered. So SACK is better in case of multiple packet drops. [1] It does not mention about the case in which the disconnection might happen, the frequent known case for wireless network.

In contrary to the previous paper, R. Paul and et al. states that SACK shows smaller bandwidth utilization in case of congested link and have lower goodput than Reno and NewReno. Hence, SACK performance deteriorates in case of congested network. In case of non-congested network all three algorithms performance is comparable [3].

Similarly Lee and et al. compare the performance of Reno, NewReno, and Selective Acknowledgement (SACK) using ns-2 simulator and conclude that increasing the bandwidth-delay product leads to performance degradation, regardless of TCP versions and the bottleneck buffer size. NewReno outperforms Reno and SACK when no packet losses occur during the slow-start phase. [9]

F. Anjum and L. Tassiulas investigate the behavior of the various Transmission Control Protocol (TCP) algorithms over wireless links with correlated packet losses. They conclude that the performance of NewReno is worse than the performance of Tahoe in many situations because of the inefficient fast recovery method of NewReno. Under certain conditions the performance depends not only on the bandwidth-delay product but also on the nature of timeout, coarse or fine. From the ns–simulation they conclude that the performance of SACK is the best and the most robust over the wireless channel. [10].

Hence, different studies show variation in final result for the same algorithm. This variation may be due to the consideration and assumption of different parameters and network topologies.

2.3 Motivation

The purpose of developing all these algorithms is to maintain and recover the congestion window in case of congestion in order to increase the utilization of network resources without overwhelming intermediate routers and thus provide fastest response to the client. So I perform the comparative study of mostly used algorithms namely, Reno, SACK and NewReno. The widely used commercial simulator, Opnet 11.0 is used to create and simulate network. The mechanism and response of Reno, SACK, and NewReno to adjust the congestion window is observed and compared for different scenarios as described in the next section. Discussion and final conclusion are made based upon the analysis of performed literature review and observed results.

III. Implementation and analysis

Simulation Setting

The study of congestion window of three TCP flavors; Reno, SACK, and NewReno are performed on three different network models built using network simulator, Opnet 11.0, as described in the following sections. The simple client server model is used to observe the performance of each algorithm in case of different amount of packet loss. The client server model with disconnected link node is used to observe the performance of algorithms in case of link disconnection at various interval of time using the failure recovery node. The topology with multiple clients and multiple servers is used to model and generate heavy traffic in the network and to observe the overall performance of three above mentioned algorithms in case of congestion.

3.1 Simple client server topology

The simple client server model is shown in figure 1. The model consists of a client and a server connected with 1.5 Mbps line. A packet discarder is placed between the server and client to impose packet loss on data transferred from server to client. The scenario is set to transfer file of size 3 MB from server to client using ftp application.

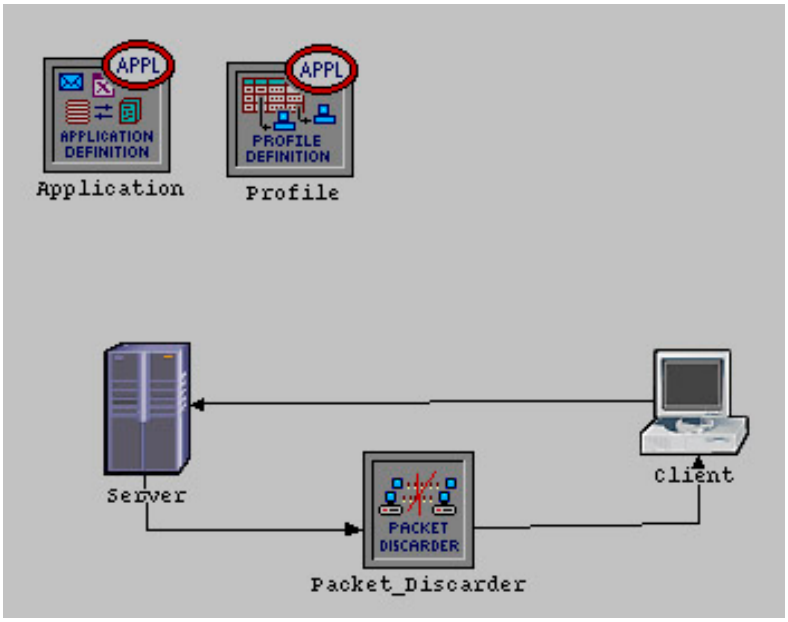

Figure 1: Simple client server network model

Three different TCP flavors are implemented for each different scenario: one packet loss, two packets loss, five packets loss, and all packet loss in the interval of 0.5 seconds. For all scenarios, parameters with standard values are set. Various variables like receiver buffer size, initial RTO, and minimum RTO are adjusted to the standard values. The table 4 below shows variables and the corresponding values set.

Table 1: Chosen parameters

Parameters	Values
Slow start initial count	1
Dup Ack Threshold	3
Initial RTO	1.0
Minimum RTO	0.5
RTT Gain	0.125
Deviation	0.25
RTT Deviation Coefficient	4
Receiver Buffer	65535

Network is simulated for 2 minutes (actual simulation time is 5 minutes). TCP congestion window statistic for the server and FTP download response time statistic for the client are observed for all algorithms for each scenario.

3.2 Simulation result: Simple client server topology

Scenario 1: **No packet loss**

When no packet is discarded in the network i.e. no packet is lost in the network, congestion window is observed as shown in figure 2. The graph reflects that all three algorithms; Reno, SACK, NewReno follows the slow start phase in the beginning of connection and as it reaches the slow start threshold (which is equal to the receiver window size of 65 KB), TCP sender follows the congestion avoidance phase. From figure 3, it indicates that the file download response time by all three algorithms is same when there is no congestion at all.

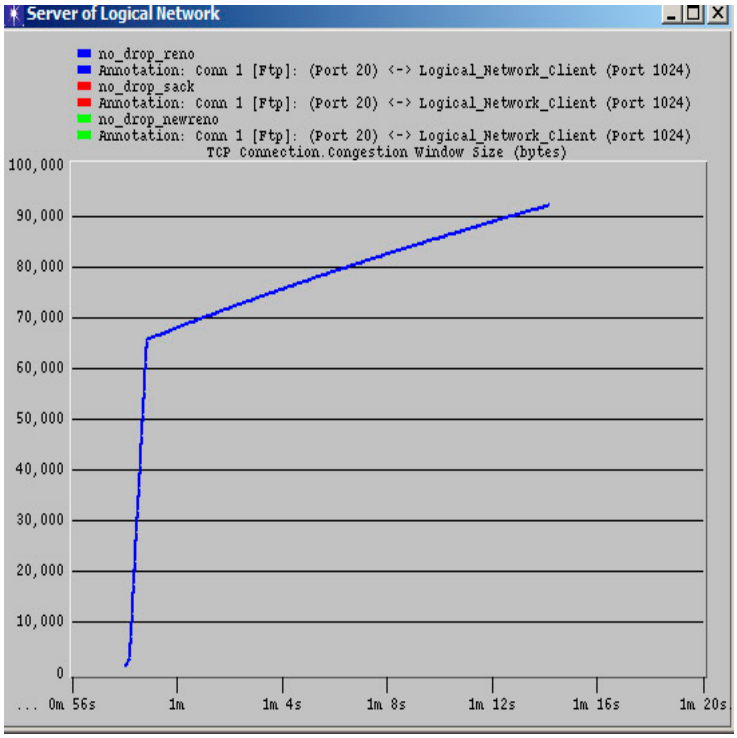

Figure 2: Congestion window for no packet loss

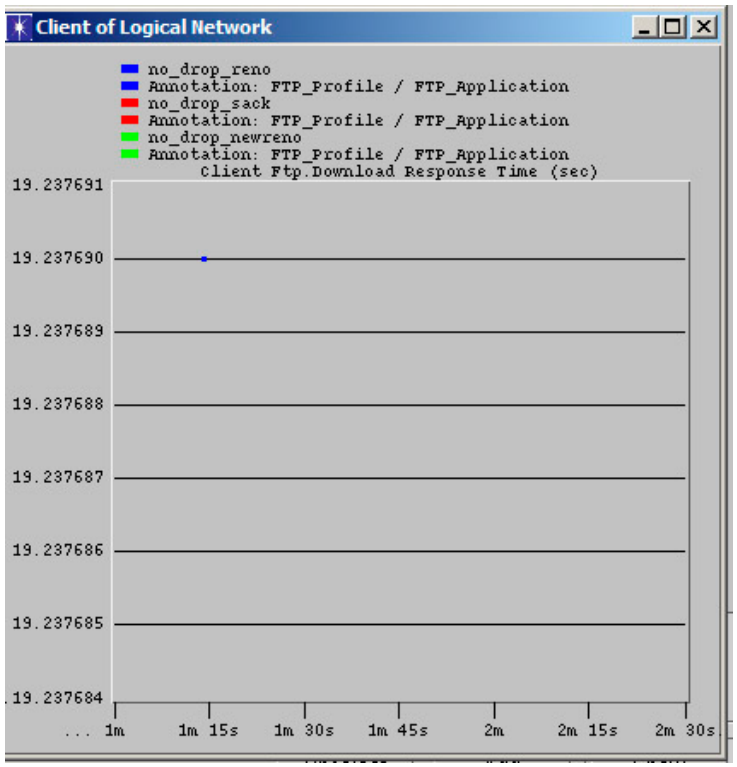

Figure 3: File download response time for no packet loss

Scenario 2: One packet loss

When a single packet is lost in the network, each TCP flavor behaves differently to improve the congestion window. The slow start phase is observed in the beginning of connection. As it reaches the slow start threshold TCP sender follows the congestion avoidance phase.

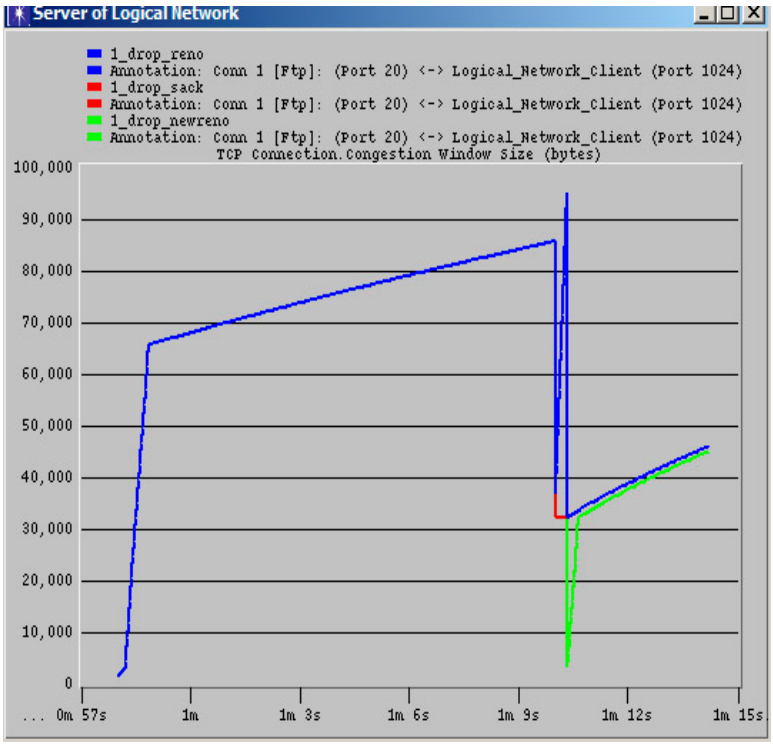

Figure 4: Congestion window for one packet loss

When a single drop event occurred, Reno reduces the congestion window to half of the current congestion window and enters the fast recover phase. As it receives the new acknowledgment, the window drops down and enters into the congestion avoidance phase. Whereas NewReno drops down the window to the predefined value and enters into the congestion avoidance phase. Since SACK uses selective acknowledgment, it reduces it congestion window to half of the current congestion window and it starts to recover using congestion avoidance phase. The graph indicates that SACK starts to recover congestion window sooner than Reno and NewReno starts to recover the window at latest. However, figure 5 indicates that the download response time for all three algorithms is same.

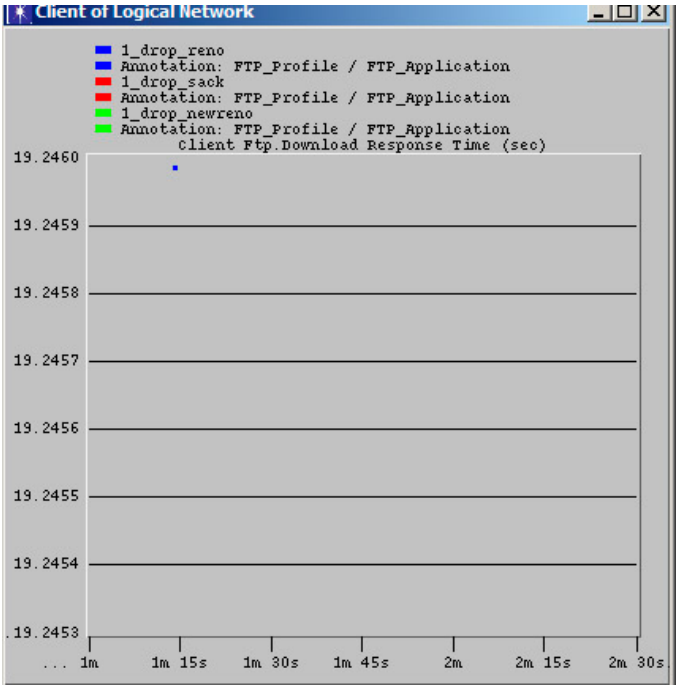

Figure 5: File download response time for one packet loss

Scenario 3: Two packets loss

When multiple packets are lost, each algorithm behaves differently to recover congestion window. The slow start phase is observed in the beginning of connection and as it reaches the slow start threshold TCP sender follows the congestion avoidance phase. When multiple drops occur, the basic principle followed by each algorithm is as described in one drop event. However, Reno reduces the congestion window to 1 MSS and starts to recover after SACK and NewReno as shown in figure 7. SACK starts to recover the congestion window sooner than Reno and NewReno. But NewReno leads Reno to recover the congestion window.

In case of multiple packet drops, the file download response time varies for each algorithm which is as shown in figure 8. SACK provides the fastest download file than NewReno and Reno is the slowest among three.

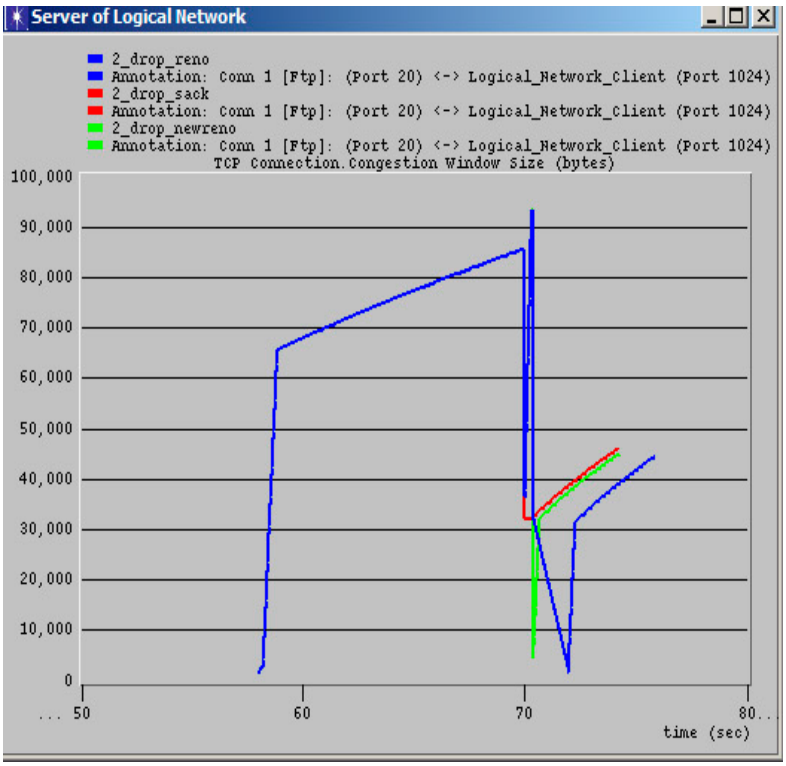

Figure 6: Congestion window for two packets loss

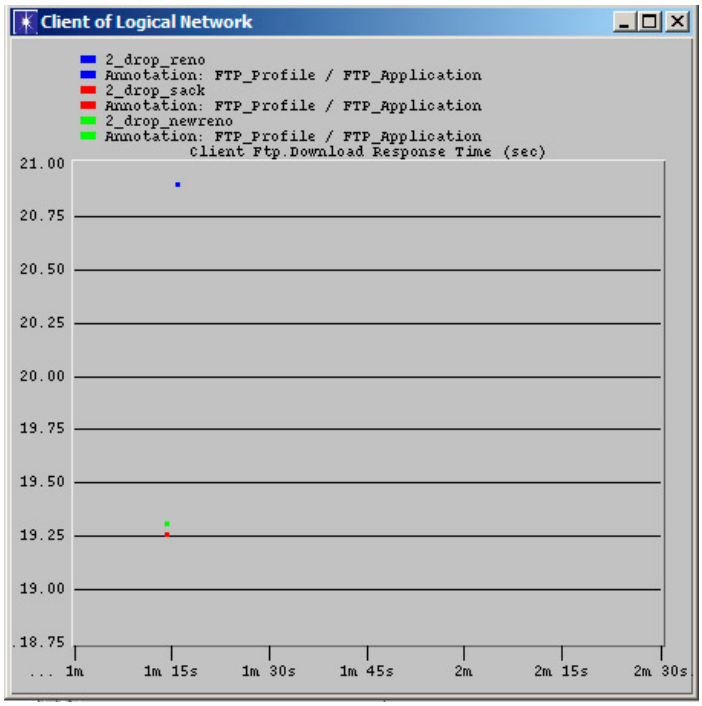

Figure 7: Client download response time for two packets loss

Scenario 4: Five packets loss

When five packets are lost, each algorithm has similar behavior as in case of two packets drops. However, the congestion window recovery time is larger than in case of two packets loss case. Similarly, the file download response time varies where SACK provides the fastest download file than NewReno and Reno is the slowest among three.

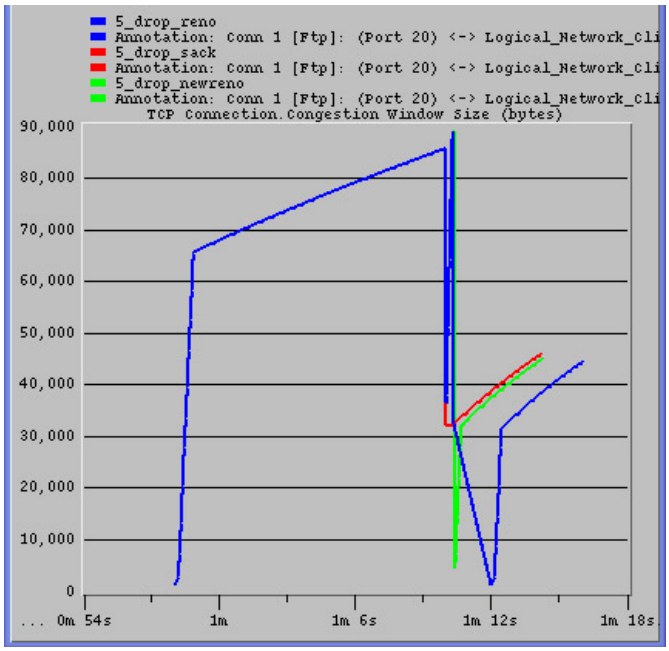

Figure 8: Congestion window for five packets loss

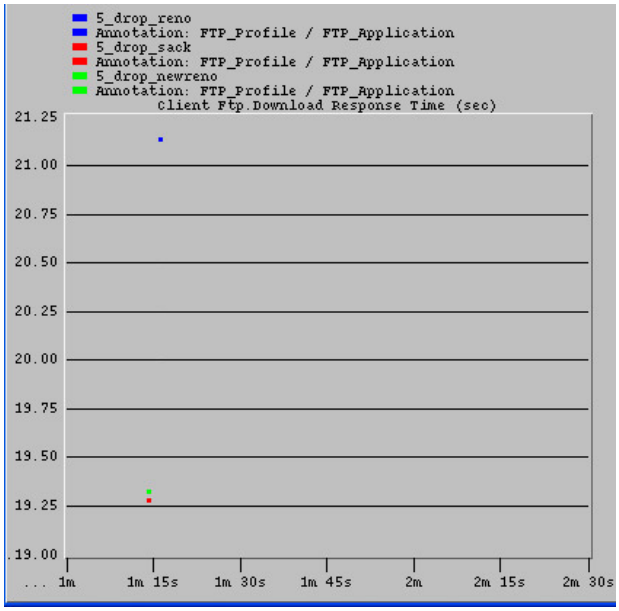

Figure 9: Client download response time for five packets loss

Scenario 5: All packets loss

The wireless scenario is modeled dropping all the packets from the packet discarder for 0.5 seconds so that the scenario behaves as if client and server are disconnected dropping all the packets. Figure 10 indicates that the slow start phase occurred in the beginning. As it reaches the slow start threshold, TCP sender follows the congestion avoidance phase. When all the packets are dropped, all three algorithms causes congestion window reduce to 1 MSS. All of these algorithms recover congestion window similarly when a disconnection between server and client is recovered after a short duration of time. Figure 11 indicates that the file download response time for all three algorithms is same when there is disconnection. In addition, all these algorithms reduce the congestion window in similar manner.

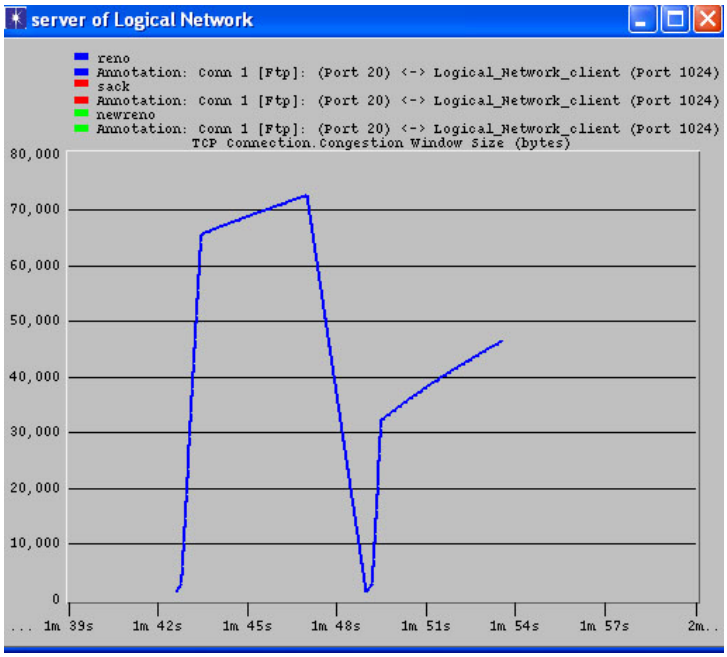

Figure 10: Congestion window for all packets loss

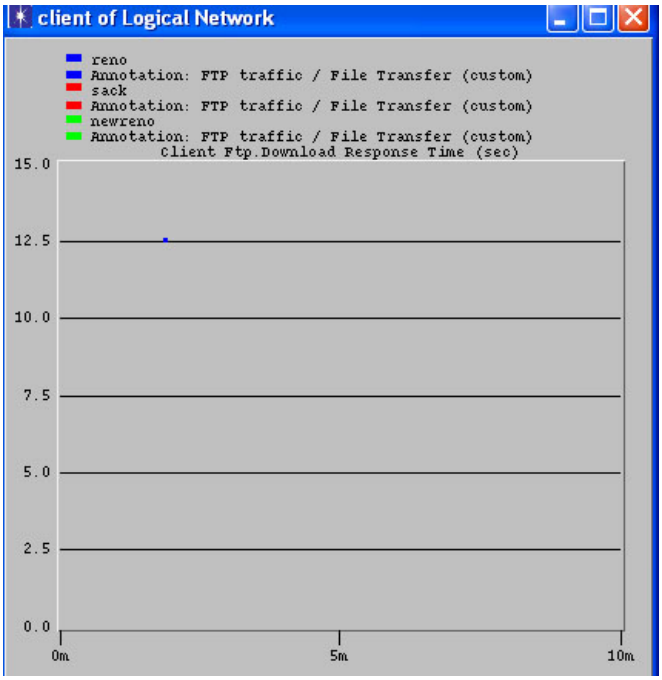

Figure 11: Client download response time for all packets loss

3.3 Topology with client, server and disconnection node

The wireless scenario is further analyzed using a topology which consists of two subnets. Each subnet consists of client and server connected to router with 100 Mbps link. The routers are connected with the backbone Internet (IP cloud) by 45 Mbps link. The network model is shown in figure 12. The link between client and router is disconnected with failure recovery node for different time intervals (0.05 s, 0.1 s, 0.2 s and 10 s), considering the handoff and link disconnection period to study the behavior of different algorithm with respect to disconnected time. File of size 15 MB is transferred using ftp application from server to client. Network is simulated for 10 minutes where actual simulation time is 14.5 minutes. TCP congestion window statistic for the server and FTP download response time statistic for the client are observed and compared in order to determine the relation with disconnection time.

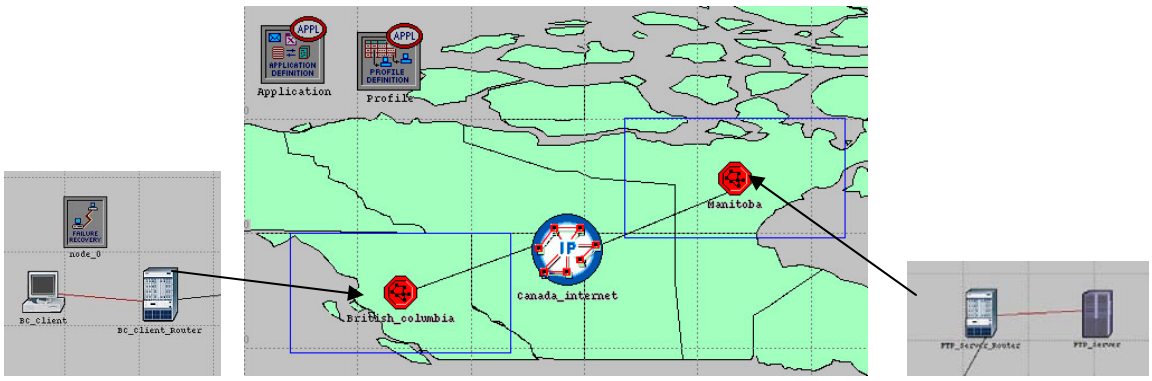

Figure 12: Topology to simulate disconnected network

3.4 Simulation result: Topology with client, server and disconnection node

Scenario 1: 0.05s, 0.1s and 0.2s disconnection period

When the link is disconnected for 0.5s, 0.1s, and 0.2s interval of time, all three algorithms reduces the congestion window. However, SACK starts to recover window prior to NewReno and Reno in all three disconnection interval of time. Whereas Reno drops the congestion window to 1MSS and starts to recover it after SACK and NewReno start to recover the congestion window which is shown in figure 13. The mechanism to recover window seems to be similar to that in case of multiple packet drop case for all three TCP flavors. However, all three algorithms are incapable of maintaining and distinguishing the loss due to the link failure since they all drop the congestion window regardless of the cause of loss. In addition to this, SACK completes the file download faster than NewReno. Reno takes longer time to download the file. The download response time is shown in figure 14. It seems that the variation in disconnection period has different impact in file download response time for each algorithms so another simulation with larger disconnection period of 10 sec is simulated.

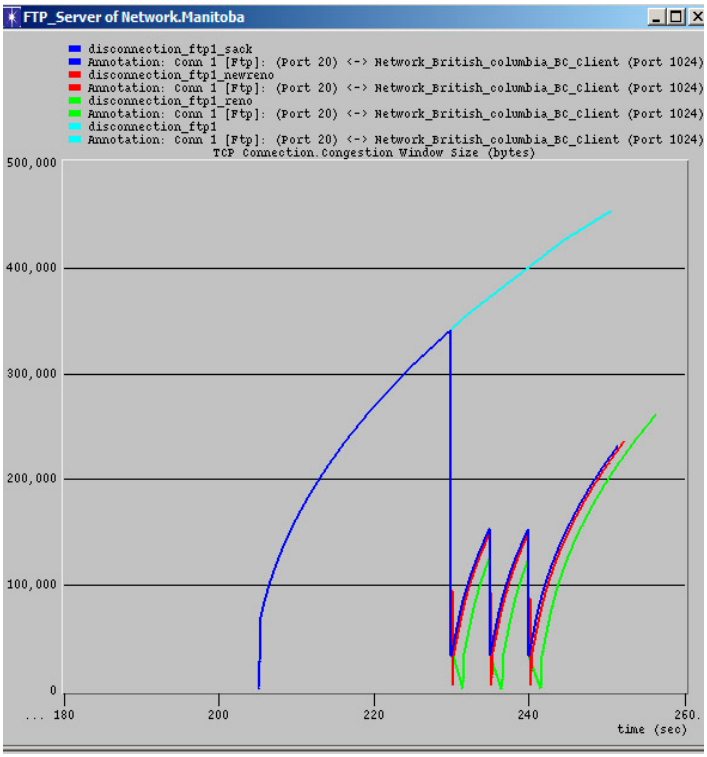

Figure 13: Congestion window for 0.05s, 0.1s, and 0.2s disconnection

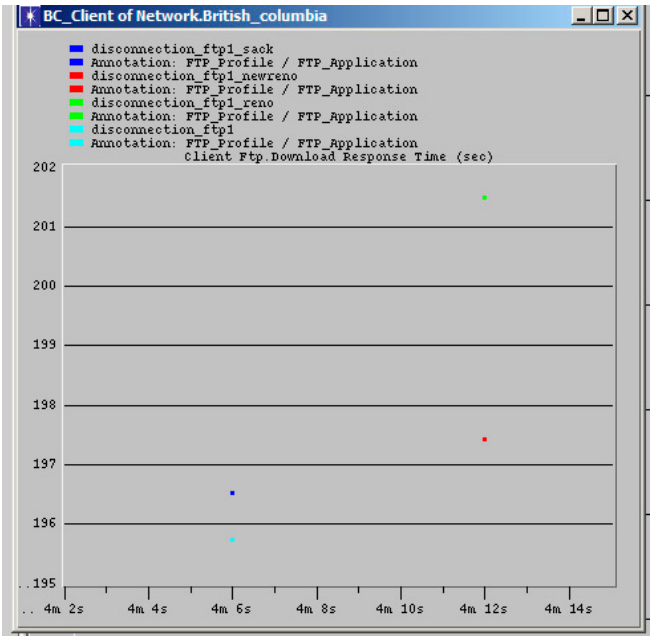

Figure 14: Client download response time with 0.05s, 0.1s, and 0.2s disconnection

Scenario 2: 10s disconnection period

In case of disconnection for 10 seconds, the congestion window drops down to 1 MSS for all three TCP flavors and starts to recover at the same time. It remains in 1 MSS for 10s since there is no acknowledgment from clients during that disconnection period. Figure 15 shows the congestion window for all three TCP flavor. From file download response time in figure 15, it is noted that the download time is same for all three algorithms.

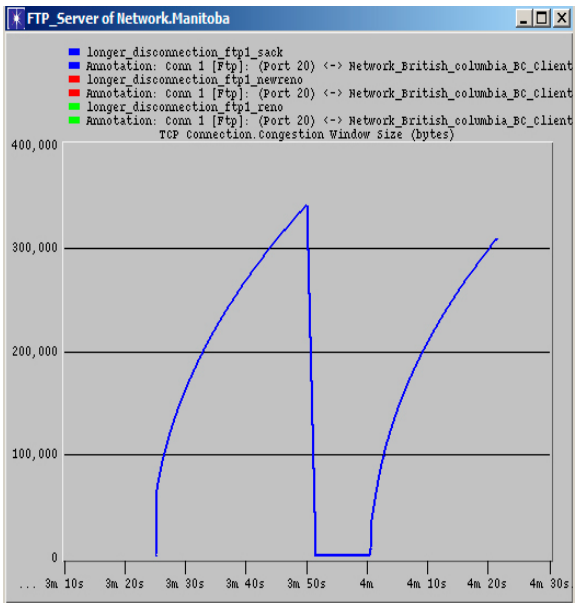

Figure 15: Congestion window for 10s disconnection

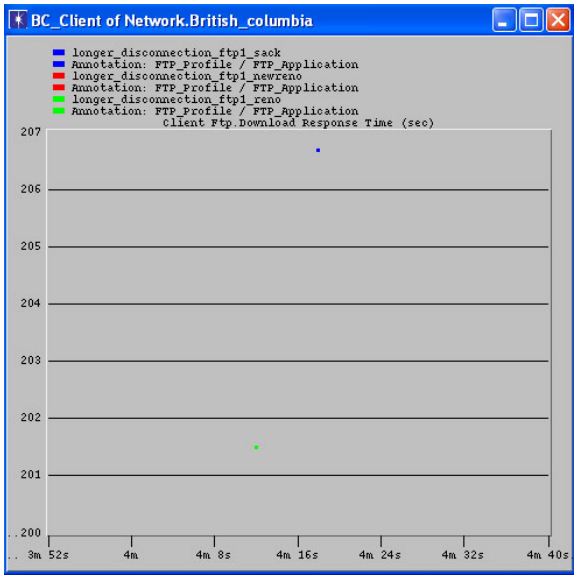

Figure 16: Client download response time with 10s disconnection

3.5 Multiple clients and multiple servers topology

The network with four different subnets consisting of six clients and two servers is built to generate a congested network and to observe the overall performance of Reno, SACK and NewReno. Ethernet clients and servers are connected with the corresponding router with 100 Mbps link and routers are connected with Internet cloud by 45 Mbps links. Two different applications, ftp and database, are customized and profile for each client is created. The topology is as shown below. The network in simulated for 5 minutes to observe congestion window for all three algorithms (actual simulation time is 1 hr 32 minutes).

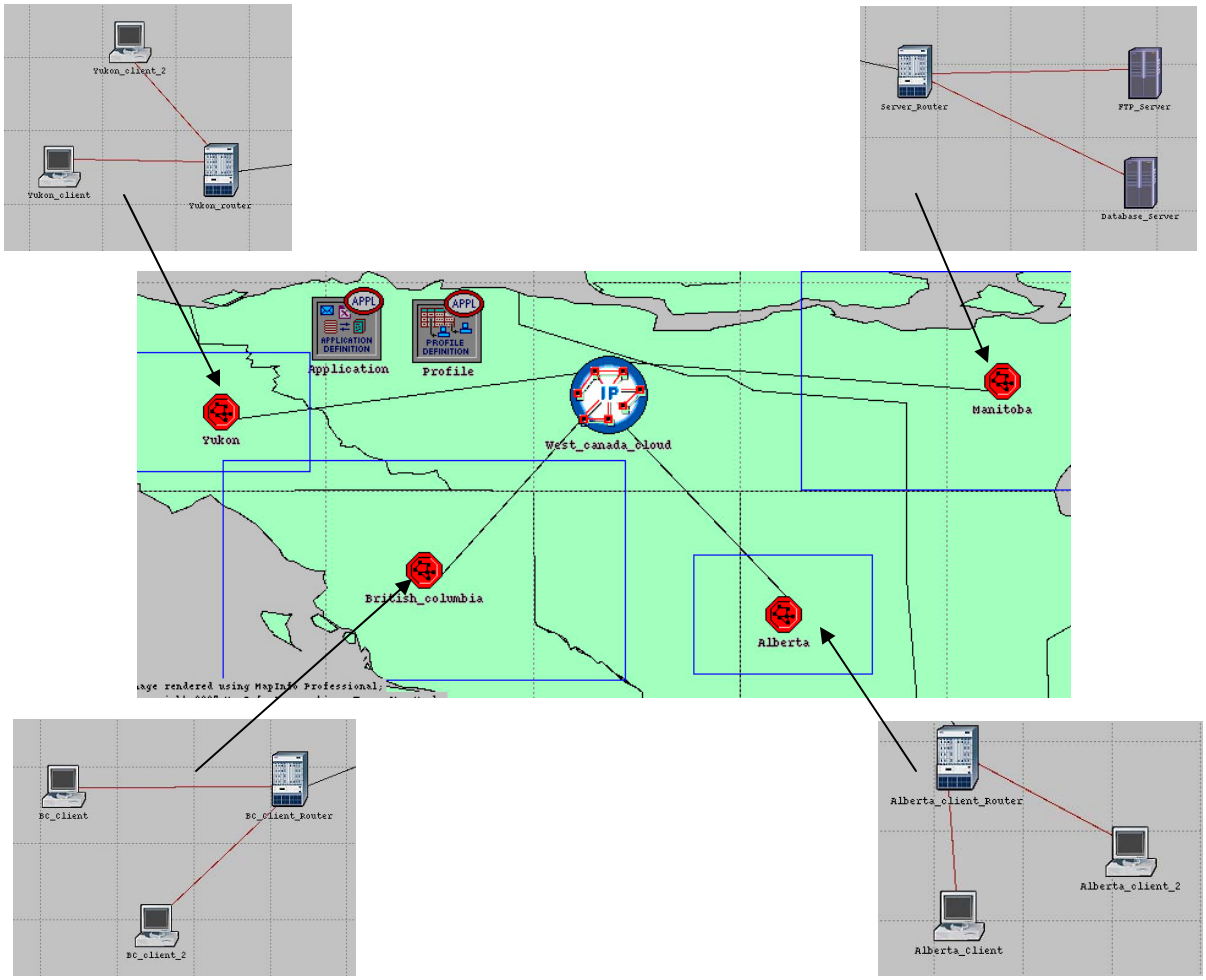

Figure 17: Multiple clients and multiple servers topology

3.6 Simulation result: Multiple clients and multiple servers topology

Scenario 1: Heavy congestion

When heavily congested network is simulated and statistics are observed, the inconsistent behavior of congestion window recovery by all three algorithms is observed. So simulation is repeated with different seed values and numbers of congestion window sample from different simulation are observed. Still it is noted that for different connection port, all three algorithms shows different magnitude of congestion window. In one case Reno maintains the highest congestion window (observed for port 1025) whereas in another case NewReno maintains the largest congestion window (observed for port 1027) among three. Similarly, SACK shows the largest congestion window in another case (observe for port 1027) which are shown in figure 18, 19 and 20. This behavior is due to varying nature of these algorithms in maintaining congestion window for different loss case noted in case of simple client and server topology and accumulation of all sort of losses during the simulation period in case of heavily congested network. So, further investigation for the cause of such behavior is to be performed in future work.

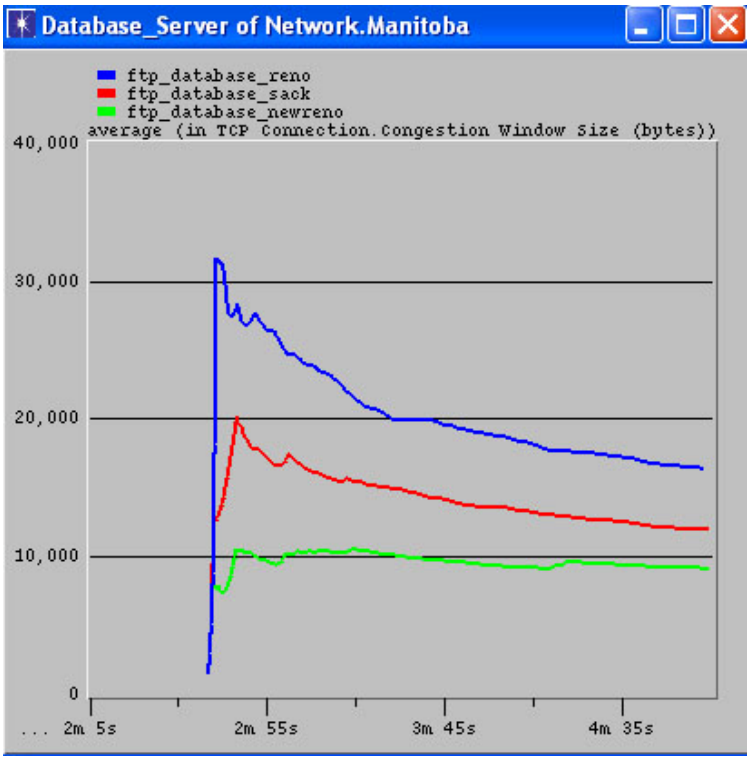

Figure 18: Congestion window connected to port 1025

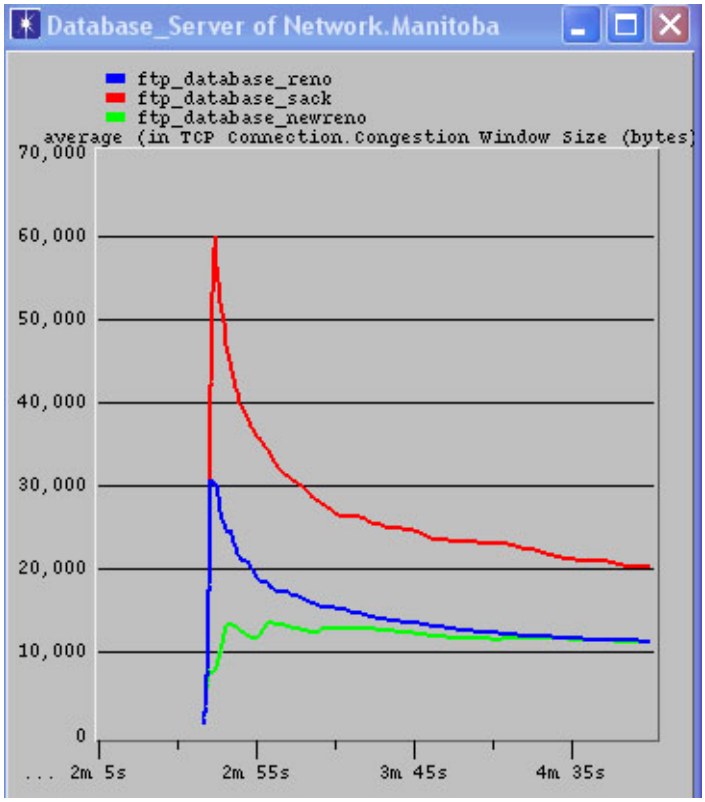

Figure 19: Congestion window connected to port 1026

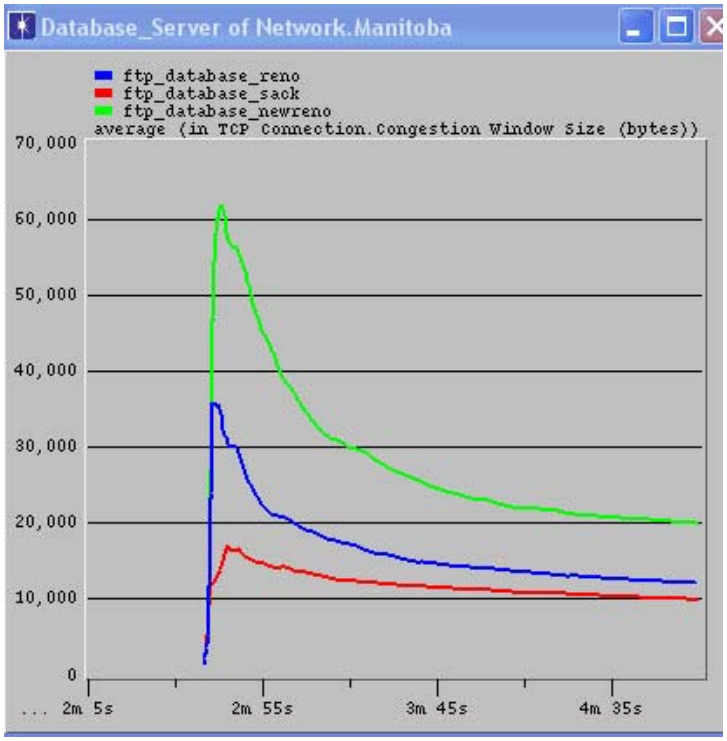

Figure 20: Congestion window connected to port 1027

Scenario 2: Low congestion

When inconsistent behavior of all three algorithms is observed in case of heavily congested network, the network is simulated with very low congestion. The simulation is repeated number of times with different seeds. The SACK performance is observed superior to that of Reno and NewReno. The congestion window is shown in figure 21.

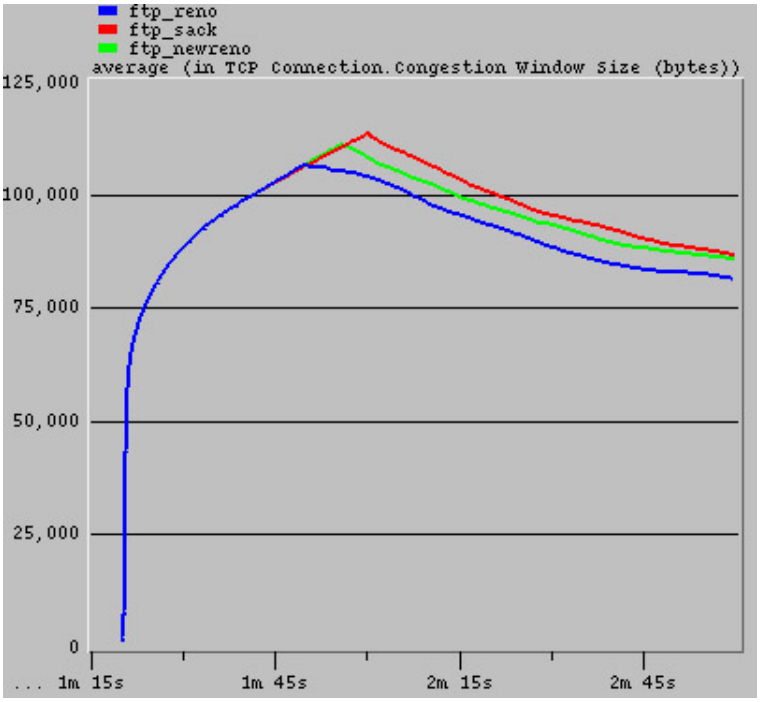

Figure 21: Congestion window for lightly congested network

IV. Discussion and Conclusion

Reno, SACK, and NewReno are implemented on network simulated using Opnet simulator and recovery of congestion windows in case of congestion is observed. The performance of each TCP flavor is compared using ftp download response time. From simple client server topology, it is noted that the congestion window is consistent with theoretical design of all three algorithms. All algorithms show similar congestion window behavior in case of no packet loss and all packets loss for the interval of 0.5 second. In case of multiple packets loss, SACK recovers the congestion window faster than rest of the two algorithms. The ftp download response time is summarized in table 2 below. It is noted that, in case of multiple packets drops, SACK transfer the same file sooner than Reno and New Reno.

Table 2: FTP download response time for simple client server topology

Number of packet loss	FTP download response time in seconds		
	Reno	SACK	NewReno
None	19.13	19.13	19.13
1	19.23	19.23	19.23
2	20.90	19.25	19.28
5	21.13	19.26	19.31
All	20.83	20.83	20.83

From the client server model with disconnection node, it is noted that all three TCP flavors are incapable to distinguish the loss due to link failure and hence reduce the congestion window to minimum. From congestion window recovery process, it is observed that the Reno, SACK, and NewReno recover the congestion window differently in case of short disconnection intervals of 0.05s, 0.1s, 0.2. However the congestion window recovery process is similar in case of longer disconnection period of 10s. The ftp download response time shown in table 3 indicates that SACK downloads file faster than Reno and NewReno for shorter disconnection period

Table 3: FTP download response time for client server with disconnected node

Disconnection interval in seconds	FTP download response time in seconds		
	Reno	SACK	NewReno
0.05, 0.1, and 0.2	201.46	196.50	197.41
10	206.53	206.53	206.53

However, in multiple client and server network posing heavy traffic, SACK, Reno, and NewReno shows inconsistent and conflicting congestion window recover process. The performance of each algorithm is different at different port. So, the network is simulated with very low congestion. The simulation is repeated number of times with different seeds. The SACK performance is observed superior to that of Reno and NewReno.

.

V. Limitations and Future Work

1. For more detailed analysis, the drop time interval can be changed for various time periods and the performance of Reno, SACK, and NewReno for each drop event can be observed.

2. The wireless loss is modeled with disconnected network. So, in future wireless node with disconnection behavior can be modeled for more detailed analysis.

3. The cause of inconsistent behavior of these TCP flavors in case of heavily congested network can be further analyzed.

4. New algorithms can be implemented to see the performance in comparison to these basic algorithms.

VI. References

[1] S. Floyd and K. Fall, "Simulation Based Comparisons of Tahoe, Reno and Sack TCP", *ACM Computer Communication Review*, 1996, Vol.26, No.3: 5 – 21.

[2] S. Floyd and T. Henderson "The NewReno Modification to TCP's Fast Recovery Algorithm" *RFC 2582*, Apr 1999.

[3] R. Paul and Lj. Trajkovic, "Selective-TCP for wired/wireless networks," *Proc. SPECTS 2006*, Calgary, AL, Canada, Aug. 2006, pp. 339 – 346.

[3] W. G. Zeng and Lj. Trajkovic, "TCP packet control for wireless networks," *Proc. IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob 2005)*, Montreal, Canada, Aug. 2005, pp. 196 – 203, vol. 2.

[4] Z. Chen and M.M. Ali, "The performance of TCP congestion control algorithm over high-speed transmission links," *IEEE Canadian Conf., Department of Electrical and Computer Engineering*, Concordia University, Montreal, Canada, May 2004, pp.1371 – 1374 Vol.3.

[5] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," *Proc. ACM/IEEE Int. Conf. on Mobile Computing, Networking*, Seattle, Washington, United States, 1999, pp. 219 – 230.

[6] RFC [online]. Available: http://www.ietf.org/rfc.html.

[7] Q. Shao and Lj. Trajkovic, "Measurement and analysis of traffic in a hybrid satellite-terrestrial network," *Proc. SPECTS 2004*, San Jose, CA, July 2004.

[8] I. Khalifa and Lj. Trajkovic, “An overview and comparison of analytical TCP models,” (invited session) *Proc. IEEE Int. Symp. Circuits and Systems*, Vancouver, British Columbia, Canada, May 2004, vol. V, pp. 469 – 472.

[9] H. lee, S. Lee and, Y.Choi, “The influence of the large bandwidth-delay product on TCP Reno,NewReno, and SACK”, *Proc. Information Networking Conference*, Oita, Japan, 2001, pp. 327 – 334.

[10] F. Anjum and L. Tassiulas, “Comparative study of various TCP versions over a wireless link with correlated losses”, *IEEE/ACM Transactions on Networking (TON)*, NJ, USA, June 2003, Vol. 11, Issue 3, pp. 370 – 383.

Appendix