Modified Algorithm for Performance Enhancement of TCP- NewReno

Punith R
Computer Science and Engineering
IIT Bhubaneswar
Bhubaneswar, India
Email: pr17@iitbbs.ac.in

Snigdha Kyathari
Computer Science and Engineering
IIT Bhubaneswar
Bhubaneswar, India
Email: ks22@iitbbs.ac.in

***Abstract*—One of the keys to the success of the Internet is relying on using efficient congestion control mechanisms. Congestion control is required not only to prevent congestion collapse in the network, but also to improve network utilization. TCP is the most widely used transport protocol by the Internet community. It is used by applications e-mail, WWW, and file transfer. TCP provides reliable transfer of stream of bytes between a pair of processes.There are different TCP implementations and each modifies the basic congestion control mechanism slightly. This paper presents a modified fast recovery algorithm to enhance the performance of the most widespread congestion control protocol-TCP-NewReno. The proposed mechanism is evaluated by using the network simulator NS-2 and compared TCP-Newreno. The simulation results show that the proposed mechanism improves the performance of the TCP-NewReno against packet delay (Forwarding delay and Jitter) and packet drop ratio. Detailed algorithm, the results and analysis are discussed in the various sections below.**

***Index Terms*—TCP Congestion Control, slow start, congestion avoidance, fast retransmit, fast recovery**

I. Introduction

Transmission Control Protocol (TCP) has been the dominant transport protocol for reliable data transfer over the Internet. It supports most of the popular Internet applications, such as the World Wide Web, file transfer and e-mail. However, the rapid growth of the Internet and the increasing demand of different traffics over the Internet lead to a serious problem called congestion collapse. This problem occurs when the aggregate demand for resources exceeds the available capacity of the network.

Congestion is generally bad for network users, applications and network performance. When a packet encounters congestion, there is a good chance that the packet is dropped, and the dropped packet wasted precious network bandwidth along the path from its sender to its destination. Congestion control is thus required to prevent congestion collapse in the network and improve the network performance.

Several congestion control algorithms are proposed and incorporated into the TCP to resolve the congestion collapse problem. TCP Tahoe includes three algorithms namely Slow Start, Congestion Avoidance, and Fast Retransmit. In Reno, packet loss is used as an indicator for congestion and the TCP sender changes its congestion window size according to the congestion control algorithms Slow-Start, Congestion Avoidance, Fast Retransmit and Fast Recovery.

However, the authors have shown that the TCP Reno cannot efficiently recover multiple packet losses from a window of data. Another congestion control mechanism, called TCP NewReno, is developed using an augmented Fast Recovery algorithm to overcome the problem of TCP Reno and combat multiple packet losses from the same transmission window without entering into Fast Recovery multiple times.

The problem with NewReno is that, within Fast Recovery algorithm, it halves its congestion window irrespective of the state of the network as long as a packet loss is detected. Another problem arises with NewReno is that when there are no packet losses, but packets are reordered by more than three duplicate acknowledgements; NewReno mistakenly enters Fast Recovery, and halves its congestion window.

This paper presents a modified algorithm to improve the TCP-NewReno performance. We have used different adjustments to the congestion window (cwnd) of the TCP sender to obtain a minimum delay in packet transfer. The proposed mechanism is evaluated by using the network simulator NS2 and compared with the other existing mechanisms. The simulation results show that the proposed mechanism improves the performance of the TCP-NewReno against packet delay and packet drop ratio.

The rest of this paper is organized as follows; Section 2 presents an overview of the most widespread congestion control protocol TCP-NewReno. The proposed mechanism is described in Section 3 while Section 4 presents the simulation results and discussions. Finally, the paper conclusions and the direction for future work are given in Section 5

II. TCP NewReno

Modern TCP implementations incorporate various congestion control algorithms to adapt the sending rate at the sender site in order to overcome the congestion collapse. This section briefly describes the main algorithms of the TCP NewReno; Slow-Start, Congestion Avoidance, Fast Retransmit and Fast Recovery.

A. Slow Start and Congestion Avoidance

In TCP NewReno, when a TCP connection begins, the Slow Start algorithm initializes a congestion window (cwnd) to one

segment. The sender is then start transmitting packets based on the window size. For each acknowledgement returned from the receiver, the congestion window is increased by one segment. This behavior continues until the cwnd arrives to the slow start threshold (ssthresh). At this point, the NewReno enters into Congestion Avoidance phase to slow the increasing rate of the cwnd. During congestion avoidance, the congestion window increases linearly by one segment every round trip time (RTT) as long as the network congestion is not detected. The implementation of the Slow Start and the Congestion Avoidance algorithms is built in a manner so that the increasing rate of the cwnd goes on until an indication for congestion occurrence is reached. At this point, the transmission rate should be slowed down to resolve the network congestion.

B. Fast Retransmit and Fast Recovery

During Congestion Avoidance, the reception of duplicate acknowledgement or the expiration of retransmission timer can implicitly signal the sender that the network congestion is occurred. So, the sender has to slow down its transmission rate. If the congestion was indicated by a timeout, the ssthresh is set to one half of the current congestion window and the congestion window is set to one segment and the sender enters into the Slow Start phase. If the congestion was indicated by duplicate acknowledgements, the TCP sender goes into the Fast Retransmit mode to retransmit what appears to be lost packet without waiting for the retransmission timer to expire. Then, the sender sets the ssthresh to half of the current congestion window and the new congestion window to the new ssthresh plus the number of received duplicate acknowledgements and enters into the Fast Recovery phase. Upon entering Fast Recovery, the sender continues to increase the congestion window by one segment for each subsequent duplicate ACK received. The intuition behind the Fast Recovery algorithm is that duplicate ACKs indicate the reception of some segments by the receiver, and thus can be used to trigger new segment transmissions. The sender transmits new segments if permitted by its congestion window. During Fast Recovery, the TCP NewReno distinguishes between a "partial" ACK and a "full" ACK. A full ACK acknowledges all segments that were outstanding at the start of fast recovery, while a partial ACK acknowledges some but not all of this outstanding data. On receiving a partial ACK, NewReno retransmits the segment next in sequence based on the partial ACK, and reduces the congestion window by one less than the number of segments acknowledged by the partial ACK. This window reduction, referred to as partial window deflation, allows the sender to transmit new segments in subsequent RTTs of Fast Recovery. The NewReno continues in Fast Recovery until all the packets which were outstanding during the start of the Fast Recovery have been acknowledged. On receiving a full ACK, the sender sets the congestion window (cwnd) to ssthresh, terminates Fast Recovery, and resumes Congestion Avoidance.

C. NewReno Implementation

Slow Start Algorithm:
Initial: cwnd = 1;
For (each packet Acked)
  cwnd++;
Until (congestion event, or, cwnd $\geq$ ssthresh)
Congestion Avoidance Algorithm:
/* slow start is over and cwnd $\geq$ ssthresh */
Every Ack:
  cwnd = cwnd + (1/cwnd)
  Until (Timeout or 3 DUPACKs)

Fast Retransmit Algorithm:
/* After receiving 3 DUPACKs */
Resend lost packet;
Invoke Fast Recovery algorithm

Fast Recovery Algorithm:
/* After fast retransmit; do not enter slow start */
ssthresh = cwnd / 2;
cwnd = ssthresh + 3;
Each DACK received;
cwnd ++;
Send new packet if allow;
After receiving an Ack:
  If partial Ack;
    Stay in fast recovery;
    Retransmit next lost packet (one packet per RTT);
  If Full Ack;
    cwnd = ssthresh;
    Exit fast recovery;
    Invoke Congestion Avoidance Algorithm;
  When Timeout:
    ssthresh = cwnd /2;
    cwnd = 1;
    Invoke Slow Start Algorithm;

III. Proposed Algorithm

As mentioned earlier, the problem with both Reno and NewReno is that within Fast Recovery algorithm, TCP halves its congestion window irrespective of the state of the network. Another problem with NewReno is that when there are no packets lost but packets are reordered by more than three duplicate acknowledgments, NewReno mistakenly enters Fast Recovery, and halves its congestion window. Since the TCP's congestion window controls the number of packets that a TCP sender can send over the network at any time, hence, the process of setting the congestion window to half of its value make the TCP NewReno inefficient in terms of utilization of link capacity. The proposed mechanism avoids these problems by modifying the Fast Recovery algorithm of the TCP-NewReno. The basic idea is to adjust the congestion window (cwnd) of the TCP sender based on the level of congestion in the network so as to allow transferring more packets to the destination.

A. Modification of congestion window

The congestion window is modified to increase to power of three instead of two to facilitate hitting the threshold quicker. Also, during congestion avoidance, the linear increase constant is increased to four from one to push more packets. Similarly, the congestion window is reduced to three-fourths during fast recovery to try to push more packets.

IV. Simulation Results

To test the behavior of the proposed congestion control mechanism, it is coded in C++ and incorporated into the Network Simulator NS-2 to be used as the transmission control protocol in the simulated network. The new algorithm is formed by using the main three algorithms; Slow Start, Congestion Avoidance, and Fast Retransmit, in addition to the modified Fast Recovery algorithm.

Figure 1 shows the network topology that is used for the simulation. The topology has four TCP connections between 4 senders and 4 receivers. The network links are labeled with their bandwidth capacity and propagation delay.

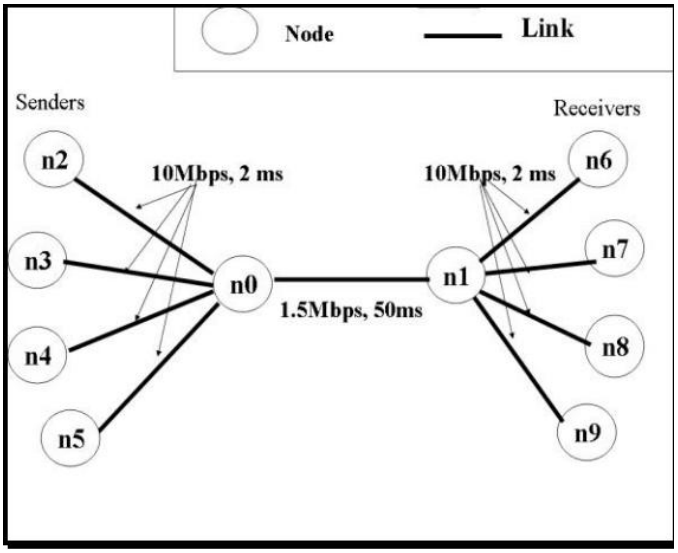

Fig. 1. Network Topology

Figure 2,3 shows the behavior of congestion window (cwnd) for the congestion control mechanism- NewReno, and the new algorithm respectively. The figures show the change of the congestion window with time. With TCP NewReno, when the packet losses are detected; these protocols set the window size to 50% of the current size. In the modified algorithm, every time the packet losses are detected, it brings down the window size by 3/4th of the current size.

The table below shows the comparison of various attributes between Newreno and the modified Algorithm. By observing the table, we can clearly notice the significant drop in delay and packet drop ratio. congestion window size is more than the corresponding Newreno algorithm's size almost always. The adjustment of the congestion window size is made so the there is minimum congestion in the network and also the

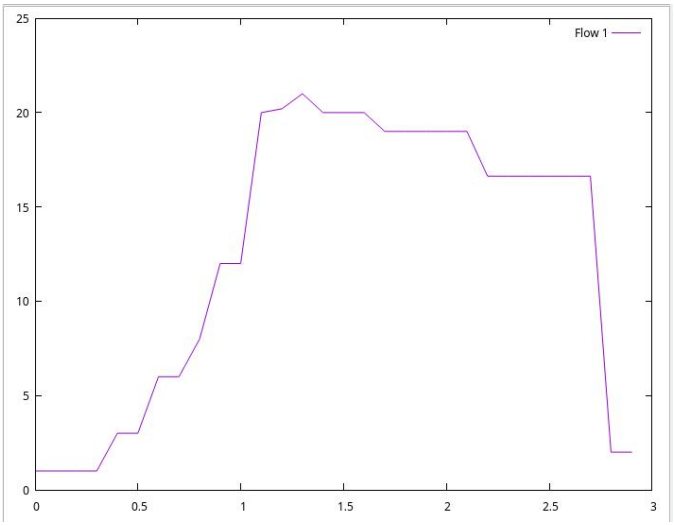

Fig. 2. Congestion window vs Simulation time

TABLE I
Comparison Table

Algorithm	Forwarding Delay	Jitter Delay	PDR
NewReno	0.11918	0.339585	0.0566038
Modified Algorithm	0.0767876	0.233606	0.0486726

number of packets sent is moderately high, thus, reducing the average forwarding and jitter delay. The no. of packets dropped have also reduced significantly.

V. Conclusion and Future Work

A modified Fast Recovery algorithm is proposed to improve the performance of the TCP NewReno. The mechanism is developed by modifying the congestion window of the TCP sender by not drastic measures to maintain better performance. The proposed mechanism is evaluated by using the network simulator NS2 and compared with both the TCP NewReno. The simulation results shown that, incorporating the modified Fast Recovery algorithm with the TCP NewReno improves its performance against packet drop ratio, forwarding delay and jitter delay because of transferring more packets to the destination.

Although the additional modifications to the Fast Recovery algorithm improve the performance, the proposed mechanism is slightly inefficient in terms of throughput. Additional enhancements should be considered in the future work to improve against throughput.

Acknowledgments

We wish to extend our sincere gratitude to Dr. Padmalochan Bera for all the support, encouragement and inspiration provided. We also thank Mr. Bata Krishna Tripathy and Mr. Kamalakanta Sethi for their help during the laboratory hours which provided the basic foundation in computer networks.