HOST TO HOST CONGESTION CONTROL USING TCP ENHANCED NEWRENO

K.UshaRani¹, M.Prudhvi Teja², K.Mounika³, J.M.K.Naidu⁴

¹Assistant Professor, LENDI INSTITUTE OF ENGINEERING TECHNOLOGY, Vizianagaram
²,³,⁴Student, LENDI INSTITUTE OF ENGINEERING TECHNOLOGY, Vizianagaram.

Abstract

Todays Internet is relying on using efficient congestion control[1] mechanisms which has got lot of importance in controlling congestion problems. Congestion control is required not only to prevent congestion collapse in the network, but also to improve network utilization. Without congestion control, a sending node may continue transmitting packets that may be dropped later due to congestion collapse[1]. This paper presents a modified fast recovery algorithm to enhance the performance of the most widespread congestion control protocol[1] TCP-NewReno. Transmission Control Protocol (TCP) is an important transport layer 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 lied to a serious problem called congestion collapse. After observing a series of congestion collapse, several congestion control algorithms are proposed and incorporated into the TCP to resolve the congestion collapse problem. In 1988, several innovative congestion control algorithms were introduced into TCP. This TCP version is called TCP Tahoe. It includes three algorithms namely Slow Start, Congestion Avoidance[2] and Fast Retransmit. A Fast Recovery algorithm was added to Tahoe to form a new TCP version called TCP Reno. TCP Reno is a reactive congestion control scheme that uses packet loss as an indicator for congestion. In order to probe the available bandwidth along the end-to-end path, the TCP congestion window (cw) is increased until a packet loss is detected, at which point the congestion window is halved and a linear increase algorithm takes over until further packet loss is experienced. TCP newReno reduces its window size to half irrespective of the congestion in the network. On the other hand TCP EnhNewReno adjusts its window size based on the network status. It also transfers more packets to the destination.

Keywords: Tahoe, Reno, NewReno, Enhanced NewReno

1. Introduction

TCP: TRANSMISSION CONTROL PROTOCOL is connection oriented protocol. TCP have an acknowledgement. TCP is reliable, slower and it has a port no. 6 eg: HTTP, FTP, SMTP. 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 lied to a serious problem called congestion collapse [1]. 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. Without congestion control, a sending node could be busy transmitting packets that may be dropped later due to congestion collapse. After observing a series of congestion collapse, several congestion control algorithms are proposed and incorporated into the TCP to resolve the congestion collapse problem.

Fig1: Hierarchial Structure of TCP Congestion Control Mechanisms
1.1 Tahoe: In 1988, several innovative congestion control algorithms were introduced into TCP. This TCP version is called TCP Tahoe[3]. It includes three algorithms namely Slow Start, Congestion Avoidance and Fast Retransmit[4].

1.2 TCP Reno: Two years later, a Fast Recovery algorithm was added to Tahoe to form a new TCP version called TCP Reno[3]. TCP Reno is a reactive congestion control scheme that uses packet loss as an indicator for congestion. In order to probe the available bandwidth along the end-to-end path, the TCP congestion window (cw) is increased until a packet loss is detected, at which point the congestion window is halved and a linear increase algorithm takes over until further packet loss is experienced.

Generally, the congestion window is used to limit the amount of data that the sender can inject into the network in order to prevent the source from overrunning the capacity of the network. In Reno, the TCP sender changes its congestion window size according to the congestion control algorithms; Slow-Start, Congestion Avoidance, Fast Retransmit and Fast Recovery. In , the authors have shown that the TCP Reno may periodically generate packet loss by itself and cannot efficiently recover multiple packet losses from a window of data. Moreover, the Additive Increase and Multiplicative Decrease (AIMD) strategy of TCP Reno leads to periodic oscillations in the aspects of the congestion window size, round-trip delay, and queue length of the bottleneck node. Indeed, the oscillation may induce chaotic behavior in the network, thereby adversely affecting overall network performance.

1.3 TCP NewReno: To alleviate the performance degradation problem of packet loss, many researchers attempted to refine the Fast Retransmit and the Fast Recovery algorithms of the TCP Reno . In , a 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. That is, TCP New Reno modifies the sender behaviour during Fast Recovery algorithm, where, it continues in Fast Recovery until all the packets which were outstanding during the start of the Fast Recovery have been acknowledged. Although the additional modifications to the Fast Recovery algorithm improve the performance of TCP NewReno, it has been found that the TCP NewReno is inefficient in terms of utilization of link capacity and unfair in throughput.

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. */ After fast retransmit, do not enter slow start */

Recovery phase:

/* After fast retransmit, do not enter slow start */

1. NewReno is inefficient in terms of utilization of link capacity and unfair in throughput.
2. Halves its congestion window irrespective of the state of the network.
3. Packet reordering occurs when there is a

TCP NEW RENO

TCP ENHANCED NEW RENO

1. NewReno is inefficient in terms of utilization of link capacity and unfair in throughput.
2. Halves its congestion window with respect to the state of the network.
3. Packet reordering occurs when there is a
even no packet loss by receiving more than three duplicate acknowledgments.

| 4. No reduction of Congestion window but sets to half of the current size. | 4. Congestion window is reduced when packet is occured. |
| 5. Data Transmission rate is moderate. | 5. Data Transmission rate is high. |

1.5 APPLICATIONS
1. Virtual private network (To find information)
2. Rexec (Remote Execution Server) protocol that enables a client user to submit system commands to a remote system
3. Talk protocol

2. TECHNIQUES
The EnewReno is formed by using the main three algorithms; Slow Start, Congestion Avoidance\[2\], and Fast Retransmit, in addition to the modified Fast Recovery algorithm.

2.1 SLOW START AND CONGESTION AVOIDANCE PHASE:
Initially, the congestion window (cw) size is taken as one and increases linearly for each acknowledgement from receiver and continues till slow start thresh (sst) is reached.

\[
\begin{align*}
\text{Initial: } \text{cw} & = 1; \\
\text{For (Acked of each packet) } \\
\text{cw} & = \text{cw} + 1; \\
\text{Until (cw} & > \text{sst) }
\end{align*}
\]

After sst, the packet enters into congestion avoidance phase to slow the increasing rate of the cwnd. In this phase, the window size increases linearly by one segment for every round trip time (RTT) as long as congestion is detected in the network. At this point, to resolve the network congestion the transmission rate is reduced.

\[
\begin{align*}
\text{/* cw} & > \text{sst and slow start is finished */} \\
\text{Each Ack: } \\
\text{cw} & = \text{cw} + (1/cw) \\
\text{Until (3 DUPACKs or Timeout )}
\end{align*}
\]

2.2 FAST RETRANSMISSION
During Congestion Avoidance, the network congestion is signaled by the reception of 3 duplicate acknowledgements\[7\] or by retransmission time out (RTO). If the congestion is indicated by timeout, the sender enters into slow start again. If congestion is indicated by duplicate acknowledgements\[7\], the sender goes into the fast retransmit\[4\] mode to retransmit the lost packets\[10\].

\[
\begin{align*}
\text{/* After receiving 3 DUPACKs */} \\
\text{Again send lost packet: } \\
\text{Evoke Fast Recovery Phase}
\end{align*}
\]

2.3 MODIFIED FAST RECOVERY
After the packet loss is occurred, the packet is entered into fast recovery phase\[5\] to reduce those losses of packets to give maximum throughput\[8\] and also reduces the packet delay.

\[
\begin{align*}
\text{/* After fast retransmit; do not enter slow start */} \\
\text{cw} & = \text{max} \{2, \text{cw} - \text{Avg}_{\text{num}}\}; \\
\text{Avg}_{\text{no}} & = \text{cw}/\text{ReTTavg}*(A-\text{ReTT}) \\
\text{sst} & = \text{max} \{2, \text{cw}\}; \\
\text{cw} & = \text{sst} + 3; \\
\text{Each DACK received; } \\
\text{cw} & = \text{cw} + 1; \\
\text{If allow Send new packet;}
\end{align*}
\]

\[
\begin{align*}
\text{After partial Ack: } \\
\text{Stay in fast recovery; } \\
\text{Again transmit next lost packet (per each ReTT);}
\end{align*}
\]

\[
\begin{align*}
\text{After Full Ack: } \\
\text{cw} & = \text{sst}; \\
\text{Quit Fast Recovery; } \\
\text{Evoke Congestion Avoidance Phase;}
\end{align*}
\]

\[
\begin{align*}
\text{ii-When TimeOut: } \\
\text{cw} & = \text{max} \{2, \text{cw} - \text{Avg}_{\text{num}}\}; \\
\text{sst} & = \text{max} \{2, \text{cwnd}_{\text{c}}\}; \\
\text{cw} & = 1;
\end{align*}
\]
Evoke Slow Start Phase;

3. METHODOLOGIES

3.1 ARCHITECTURE

In slowstart phase initially window size will be taken as 1. When each packet ack received congestion will be increased until the congestion event occur or cw>sst. After it will be entered in to congestion avoidance phase, in this cw>sst. When every ack occur cw will be equals to the sum of cw and inverse of cw until the time out occur or 3dup ack occured. After this, retransmit phase will be occurred when 3 dup Acks are received it shows packet loss so resend the lost packet. After it enter in to the fast recovery phase, initially cw will be taken as max[2,(cw-AVGnum)]. AVGnum means that (Ratio of the cw and RetTavg)*(RetTT1-RetTT2) value and sst value will be taken as max{2,cw} and cw=sst+3. When each DACK will be recived then cw will be increased. If it allow new packet will be send. In this we have occurred two types of acks. 1: partial ack and 2: full ack. If partial ack will be occur then stay in fast recovery phase and retransmit the lost packet. If full ack will be occur then cw=sst and exit from fast recovery phase. After that enters in to congestion avoidance phase. When time out occurred then cw=max[2,(cw-AVGnum)] and sst=max{2,cw}. After, window size will be taken as 1 now it again goes to initial phase i.e slow start phase. This process will be repeated.

3.2 GRAPHS
3.3 DESCRIPTION
In slow start phase, the cw size is initially taken as one and goes on increasing till sst is reached. The point at which the packet enters sst+1 becomes the congestion avoidance phase. In this phase, if time out occurs or 3 duplicate ACK’s occur indicates that the packet is lost.

After the avoidance phase enter into fast retransmit phase to resend the lost packets. After this phase, reduce the cw by half and continue sending segments at this reduced level. By doing this, the sender has too many outstanding packets and has no knowledge of how to send the packets in case duplicate acknowledgements are received.

4. CONCLUSION
In this work a modified Fast Recovery algorithm, ENewReno is proposed to improve the performance of the TCP NewReno. The mechanism is developed by adapting the congestion window of the TCP sender based on the level of congestion in the network. This level is determined by using the Round Trip Time (RTT) that represents as an indicator for the traffic loads on the network. Incorporating the modified Fast Recovery algorithm[9] with the TCP NewReno improves its performance against both the throughput and the packet delay because of transferring more packets to the destination. Although the additional modifications to the Fast Recovery algorithm improve the performance of the proposed mechanism, Additional enhancements should be considered in the future work to improve the EnewReno against performance.

REFERENCES

Khandapu Usharani received B.Tech degree from JNTU University Kakinada and M.Tech degree from Andhra University. Presently working as an Assistant Professor in Lendi Institute of Engineering and Technology, Jonnada, Vizianagaram.

M.Prudhvi Teja pursuing B.Tech final Year in Lendi Institute of Engineering And Technology (JNTUK) Jonnada, Vizianagaram.

K.Mounika pursuing B.Tech final Year in Lendi Institute of Engineering And Technology (JNTUK) Jonnada, Vizianagaram.

J.Murali Krishnam Naidu pursuing B.Tech final Year in Lendi Institute of Engineering And Technology (JNTUK) Jonnada, Vizianagaram.