Improving Host-to-Host Congestion Control Protocols by Dynamic Bandwidth Estimation of the Network

Marwa O. Al-Enany
B.SC. in Computer Science and Engineering,
Faculty of Electronic Eng.,
Menoufia University, Egypt.

Gamal Attiya
Department of Computer Science and Engineering,
Faculty of Electronic Eng.,
Menoufia University, Egypt.

Nagy W. Meseha
Department of Electronics and Electrical Communication Eng,
Faculty of Electronic Eng.,
Menoufia University, Egypt.

ABSTRACT
Congestion is one of the major problems that affects on throughput, delay, losses and other performance metrics of the network. During the last decade, several congestion control protocols have been proposed to overcome this problem. The most widely protocols are TCP Tahoe, Reno, New Reno, Vegas and SACK. In this paper, a new approach is developed to enhance most of the existing host-to-host congestion control protocols. The main idea is to adjust the congestion window size (cwnd) dynamically according to the available bandwidth of the network. In the proposed strategy, instead of increasing the cwnd size linearly by the AIMD, the cwnd is increased according to the available bandwidth of the network. Also, instead of decreasing the cwnd to half of its size as congestion happens, the cwnd is decreased to latest value that was used effectively without losses. The proposed approach is implemented in the TCP Tahoe, Reno, Newreno, Vegas and SACK and the performance is evaluated by using the network simulator NS-2 considering a realistic network topology generated by the GT-ITM.

General Terms
Computer Networks, Network Protocols.

Keywords
TCP, Congestion Control, Congestion Avoidance, Dynamic Bandwidth Estimation.

1. INTRODUCTION
With the dramatic growth and significant increase of the internet, an important problem called congestion is arising. Congestion occurs when the number of transmitted packets by the sender exceeds the capacity of the network. Generally, congestion has significant impact on the network performance. If congestion happens, the network throughput will decrease, packet delay and losses will increase and that causes the network performance to degrade [1].

Over the last decade, several congestion control protocols have been proposed by researchers to overcome the congestion problem and to improve the network performance. They try to keep number of packets being transmitted below the level at which performance falls off. The proposed congestion control protocols may be classified into two categories; (i) Network-assisted congestion control protocols, taken by routers and (ii) Host-to-Host congestion control protocols, taken by the Transmission Control Protocol (TCP) at the end hosts and are mostly achieved in transport layer. This paper concerns with the Host-to-Host congestion control protocols.

TCP is a reliable window based connection oriented end-to-end protocol. It ensures reliability by making the receiver to acknowledge the segments that it receives. It sets a timer whenever it sends a segment. If it does not receive an acknowledgement from the receiver within the 'time-out' interval, then it retransmits the segment. Many congestion control mechanisms were developed based on the TCP to overcome the congestion problem. TCP Tahoe is the first proposal developed by Jacobson and Karles [2]. Then, various versions were developed based on Tahoe including; TCP Reno [3], Newreno [4], TCP Vegas [5] and SACK [6]. In [7], the effects of using the End to End protocols in the Internet are studied.

Recently, several modifications are released aiming to overcome the congestion problem [8-22]. However, most of them use the Additive Increase Multiplicative Decrease (AIMD) strategy for adjusting the congestion window (cwnd). This strategy is inefficient in terms of network utilization and unfair in throughput. This is because, the AIMD strategy blindly updates the congestion window (cwnd) size statically by a fixed value regardless the network status. In this paper, a new strategy is proposed to dynamically adjust the cwnd size based on the available bandwidth of the network. The aim is to overcome the congestion problem, increase the network throughput and decrease packet delay and losses.

This paper is organized as follows. The Host-to-Host congestion control protocols are described in Section 2. Section 3 presents the problem of the current congestion control protocols while the proposed strategy is introduced in Section 4. Section 5 presents the simulation results with comparative study between the network behavior under the current congestion mechanisms and under the proposed modifications. Finally, the concluding remarks and the direction for future work are presented in Section 6.

2. HOST-TO-HOST CONGESTION CONTROL
The main idea of the Host-to-Host congestion control protocols is based on implementing some mechanisms at the end hosts of the network rather than in the intermediate nodes (routers) to organize packets flow in the network. In this section, the most widely Host-to-Host congestion control protocols; namely; TCP Tahoe, Reno, New Reno, SACK and Vegas, are presented.

2.1 TCP Tahoe
TCP Tahoe is the first developed congestion control protocol. It consists of three mechanisms: Slow Start, Congestion Avoidance, and Fast Retransmit [2].

Slow Start operates at the sender by employing two windows, called congestion window (cwnd) and advertised window [7]. When a new connection is established, the cwnd is initialized to one segment. The sender then starts by transmitting one segment and waiting for ACK. Each time an ACK is received, the cwnd is increased by one segment. That is, the cwnd is incremented from one to two, and the two segments are sent. Then, the congestion window is increased to four when each of those two segments is acknowledged, and so on. This provides an exponential growth behavior [7]. This behavior continues until the cwnd reaches slow start threshold (ssthresh) or packet loss detection. Once the cwnd reaches the ssthresh, TCP goes into congestion avoidance. While, if a packet loss occurs, the ssthresh is set to half of the current cwnd (the multiplicative decrease), the cwnd is set to 1 and the slow start begins again.

Slow Start Algorithm:

Initial: cwnd = 1;
For (each packet Acked)
cwnd++;
Until (congestion event, or, cwnd > ssthresh)

Congestion avoidance starts when the cwnd reaches the ssthresh. This Mechanism is used to slow the increasing rate of the cwnd, where the cwnd increases by one segment every Round Trip Time (RTT). This phase uses the AIMD strategy and continues until congestion is detected. That is, on each successful ACK, the cwnd is increased by 1/cwnd (Additive Increase), implying linear growth instead of exponential growth. But, if congestion is detected by timeout, the ssthresh is set to one half of the current cwnd (Multiplicative Decrease) and the cwnd is reset to one segment, which automatically puts the sender into Slow Start mode. If congestion was detected by duplicate ACKs, the Fast Retransmit algorithm is invoked.

Congestion Avoidance Algorithm:

/* slow start is over and cwnd > ssthresh */
Every Ack:
cwnd = cwnd + (1/cwnd) /* normal operation */
Until (Timeout or 3 DUPACKs) /*loss occurred*/

Fast Retransmit speeds up the retransmission process. It starts when congestion is detected by 3 duplicate ACK. This phase performs a retransmission of what appears to be missing segment, without waiting for a retransmission timer to expire.

Fast Retransmit Algorithm:

After receiving 3 DUPACKs;
Resend lost packet; /*avoid waiting timeout*/

2.2 TCP Reno

TCP Reno is the most widely adopted Internet protocol [3]. It employs four transmission phases; Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery. In Reno, the behavior the fast retransmit of the TCP Tahoe is modified to include the fast recovery phase. The fast retransmit and fast recovery algorithms are usually implemented together [8] as follows. When 3 duplicate ACK is received, Reno retransmits the missing segment (Fast Retransmit), sets the ssthresh to one-half the current cwnd, and sets cwnd to ssthresh plus 3. This inflates the congestion window by the number of segments that have left the network and which the other end has cached. Each time another duplicate ACK arrives, increment cwnd by one segment. This inflates the congestion window for the additional segment that has left the network. When receiving new ACK that acknowledges new data, Reno sets cwnd to ssthresh and works in congestion avoidance.

Fast recovery algorithm:

After fast retransmit;
*/*do not enter slow start*/*
ssthresh=cwnd/2;
cwnd = 3 + ssthresh;
Each DACK received;
cwnd++;
send new packet if allowed;
After new ACK;
cwnd=ssthresh;
return to congestion avoidance;

Note that: TCP Reno improves TCP performance in the case of a single packet loss within the same window of data. However, performance of TCP Reno suffers in the case of multiple packet losses within the same window of data [4].

2.3 TCP NewReno

TCP NewReno is an enhanced version of the TCP Reno to combat multiple packet losses from a single window without entering into fast recovery multiple times as Reno [4]. In entering fast recovery, the ssthresh is set to one-half the current congestion window (cwnd) but no less than two segments and the cwnd is set to the ssthresh plus three. TCP NewReno continues in fast recovery until all the packets which were outstanding during the start of the fast recovery have been acknowledged. This strategy helps to combat multiple losses without entering into fast recovery multiple times as TCP Reno or causing timeout. During fast recovery, the TCP NewReno distinguishes between a partial Ack and a full Ack. A partial acknowledgement is considered as an indication that the packet following the acknowledged one in the sequence space has been lost and should be retransmitted. Therefore, TCP NewReno immediately retransmits the other lost packet indicated by the partial acknowledgement and remains in fast recovery. On the other hand, a full acknowledgement acknowledges some but not all of the outstanding data. TCP NewReno exits fast recovery when all data in the window is acknowledged. When the sender receives new acknowledgement, it exits that phase putting its cwnd to ssthresh and start congestion avoidance algorithm. Note that, whenever a timeout occur, the ssthresh is set to one half of the current congestion window and the congestion window is set to one and the sender enters into the slow-start phase. In NewReno,

Partial ACK → stay in fast recovery.
Full ACK → Exit fast recovery.

2.4 TCP Vegas

TCP Vegas emphasizes packet delay, rather than packet loss, as a signal to determine the rate at which packets may send [5]. It detects congestion at an incipient stage based on the increasing Round-Trip Time (RTT). If the RTT is too small, then the sending rate on the connection is less than the bandwidth available while if the value is too large then it will overrun the connection. Generally, Vegas works by monitoring the difference between the expected and the actual flow rates and adjusts the congestion window as follows:

Expected flow rate = current cwnd / base RTT.
Actual flow rate = current cwnd /RTT.
Difference = (Expected – Actual)*base RTT.

cwnd continuously updated to:

cwnd+1; if difference $< \alpha$
cwnd-1; if difference $> \beta$
cwnd; otherwise

Where, α, β are thresholds more than 0 [5].

2.5 TCP SACK

Selective Acknowledgment (SACK) is a strategy that makes the receiver to inform the sender about all segments that have arrived successfully. So, the sender retransmits only the segments that have actually been lost [6]. SACK ads to the ACK an option field containing a pair of sequence numbers for blocks of data received out of order with maximum size 40 bytes [6]. When multiple packets dropped from one window, the sender can selectively resend lost packets depending on the sack option field in the acknowledgement. SACK adds variable called pipe to the Fast Recovery algorithm to represent the estimated number of packets that stand in the path. The TCP sender sends packets when the pipe variable < cwnd. Every time it receives an ACK it reduces the pipe by 1 and every time it retransmits a segment it increments it by 1.

3. PROBLEM STATEMENT

From the above discussion, it is clear that, most of the proposed protocols depend on the Additive Increase Multiplicative Decrease (AIMD) strategy in adjusting cwnd and so the sending rate. The AIMD strategy makes the cwnd to be increased by one packet per window for each acknowledge in the congestion avoidance phase. Also, the AIMD strategy forces the cwnd to set to half of its value as long as a packet drop is detected. This behavior inefficiently utilizes the available capacity of the network. In case of no congestion, it leads to a significant decrease of network throughput since the number of packets to be transmitted is less than the available capacity of the network. On the other hand, when congestion is detected, it blindly halves the cwnd which in turn decreases the packet sending rate and so decreases the network utilization.

To overcome this problem, a new strategy is required to adjust the sending rate at the end hosts according to the available bandwidth of the network at any time.

4. PROPOSED STRATEGY

This section presents a new strategy for adjusting (increasing and decreasing) the cwnd size dynamically based on the available bandwidth of the network. The main idea is to estimate the available bandwidth of the network at any time and then adjust the sending rate at the TCP sender according to the estimated bandwidth. This strategy could be used instead of the AIMD strategy to improve the Host-to-Host congestion control protocols.

4.1 Dynamic Increasing Strategy

The main idea of dynamic increasing is to estimate the available bandwidth of the network with each arrival of an ACK at the sender. Then, depending on the bandwidth ratio (BW_{ratio}) between the current estimated bandwidth ($BW_{current}$) and the previous estimated bandwidth ($BW_{previous}$), increase the congestion window (cnwd).

To estimate the bandwidth at any moment k, let an ACK_k is arrived at the sender at time t_k. This implies that the corresponding amount of data d_k has been received by the receiver. So the bandwidth b_k that used by the connection to transfer the data d_k can be measured as:

$$b_k = d_k / (t_k - t_{k-1})$$

Where, t_{k-1} is the arrival time of the previous ACK at previous moment k-1.

The bandwidth estimation (BWE) starts at the congestion avoidance phase where the increase of cwnd is very slow (1/cwnd) and applied as follows.

Modified Congestion avoidance

```
/*cwnd > ssthresh*/
For every new ACK:
    Estimate BWE;
    Set BW_current = BWE;
    BW_ratio=BW_current/BW_previous;
    BW_previous = BW_current;
    If (1 <= BW_ratio < 1.5)
        cwnd = cwnd + 1/cwnd ;
    Else If (BW_ratio >= 1.5)
        cwnd = cwnd + 2/cwnd ;
    Else if (BW_ratio < 1)
        cwnd = cwnd + 0 ;
Until (timeout or 3 DUPACKs);
```

4.2 Dynamic Decreasing Strategy

The main idea of dynamic decreasing is to avoid decreasing the cwnd to half of its value when detecting congestion but decrease the cwnd to an average value between the current value and the last value before congestion. To do so, with each ACK, a variable called $cwnd_p$ is set to the present value of the cwnd before its updating. If congestion is detected, the cwnd is set to the average value between the current value of the cwnd and the previous value that stored in the variable $cwnd_p$. Decreasing the cwnd to the average value maintains the network throughput, since this behavior guarantees a large number of packets in the cwnd.

/*in case of no congestion*/

```
cwnd_p = cwnd;
Update cwnd;
if  congestion
        /* Do not use AIMD */
cwnd= int (cwnd_p + cwnd) / 2;
```

5. SIMULATION RESULTS

To study the effect of the suggested modifications on the behavior of the Host-to-Host congestion control protocols, the cwnd size of the existing protocols is adapted by the new proposed strategy. Then, the network performance is tested by using the network simulation NS2 [22] and the results are compared with that obtained by applying the original protocols considering different topologies; simple and real AT&T topology. The network topology is created by using the topology generator GT-ITM [23]. In each topology, the comparison is done by considering throughput, packet delay, and losses. In this study, the main widely existing congestion control protocols; Tahoe, Reno, Newreno, and SACK are considered in the evaluation.

5.1 Scenario 1: Simple Topology

The simple topology consists of 12 nodes (6 sources and 6 destinations) and two routers (N0 and N1), as shown in Figure 1. The link between the routers has bandwidth of 1.5 Mbps and delay of 50ms and acts as a bottleneck link of this topology. TCP connections are established between the sources and the sinks to transfer File Transfer Protocol (FTP) application. The FTP application starts at the first second and ends at 20s.

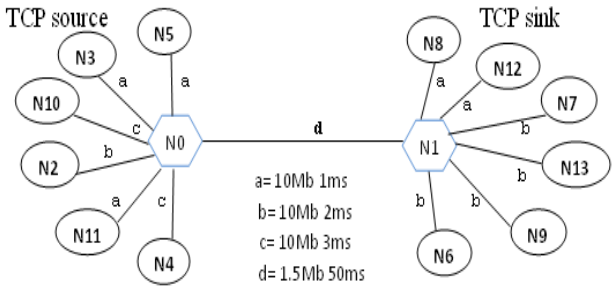


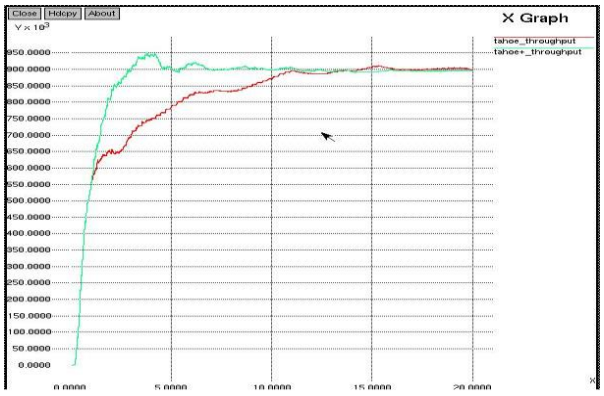
Figure 1: Simple Network Topology

5.1.1 Effect of modifications on Throughput

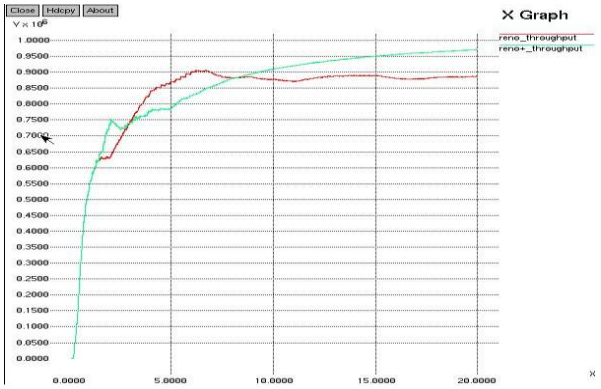
This section presents the effect of the proposed modifications on throughput. Throughput is used to identify the number of packets sent by the source and received by the destination correctly. In other words, it means the sum of the data that are delivered to all terminals over the time in the network. Figure 2 shows the network throughput of both the original and modified congestion control protocols after adding the proposed modifications. Figure 2(a) shows the throughput of Tahoe and Tahoe+, Figure 2(b) shows the throughput of Reno and Reno+, Figure 2(c) shows the throughput of NewReno and NewReno+, while Figure 2(d) shows the throughput of SACK and SACK+. From the figures, the network throughput is increased in the case of applying the proposed modifications with the existing protocols. This indicates that the network throughput is improved by changing the cwnd size according to the estimated bandwidth.

5.1.2 Effect of modifications on Delay

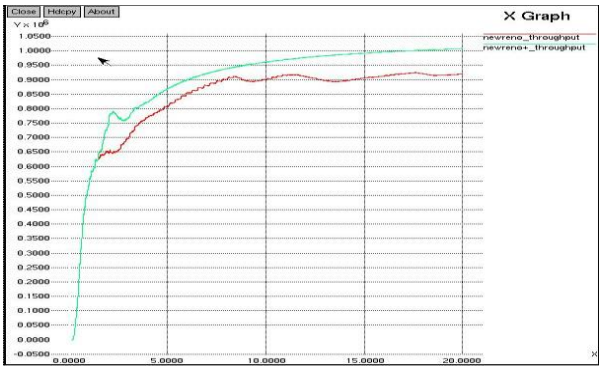
This section presents the effect of the proposed modifications on packet delay. Figure 3 shows the packet delay of both the original and modified congestion control protocols after adding the proposed modifications. Figure 3(a) shows the delay of Tahoe and Tahoe+, Figure 3(b) shows the delay of Reno and Reno+, Figure 3(c) shows the delay of NewReno and NewReno+, while Figure 3(d) shows the delay of SACK and SACK+. The figures show that the average delay when considering modifications is less than that of applying the original protocols.



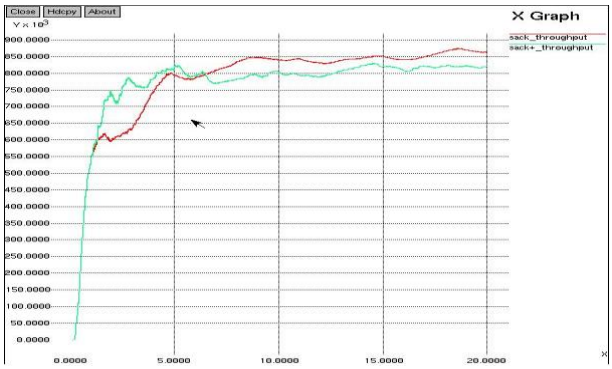
(a) Tcp Tahoe and Tahoe+



(b) Tcp Reno and Reno+

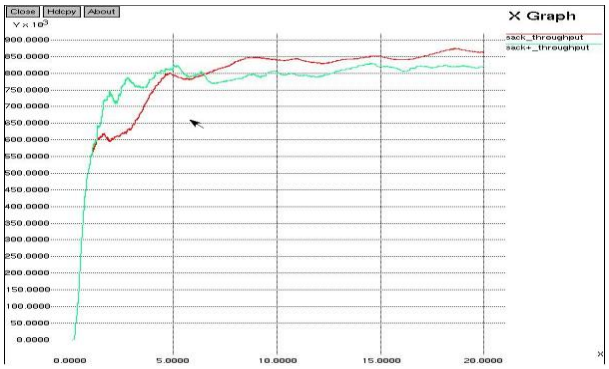


(c) Tcp Newreno and Tcp Newreno+

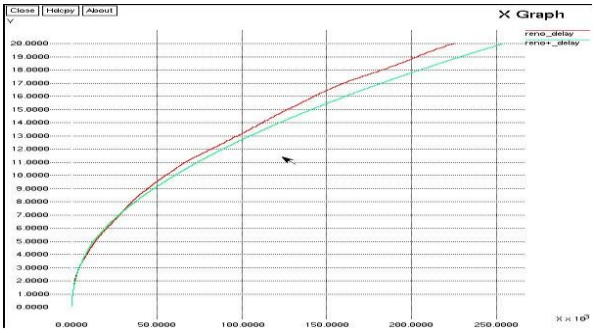


(d) SACK and SACK+

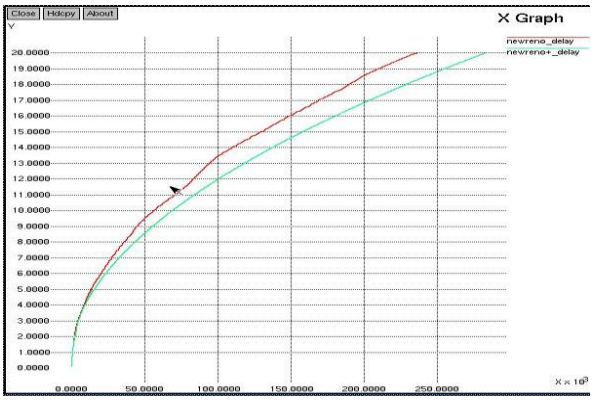
Figure 2: Throughput



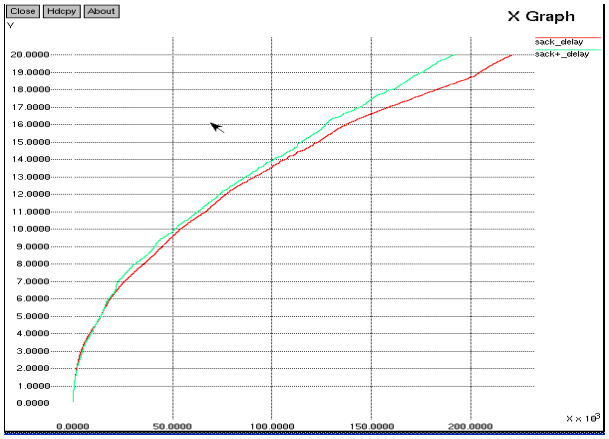
(a) Tcp Tahoe and Tahoe+



(b) Tcp Reno and Reno+



(c) Tcp Newreno and Tcp Newreno+

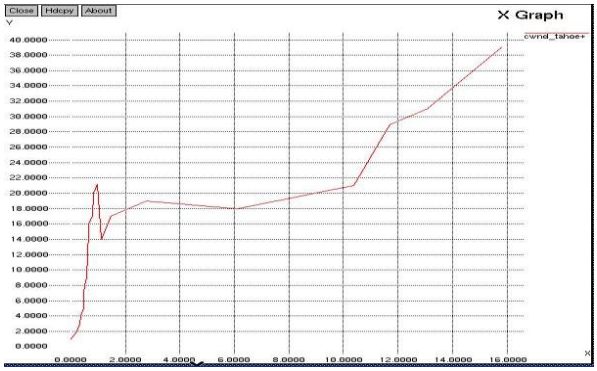


(d) SACK and SACK+

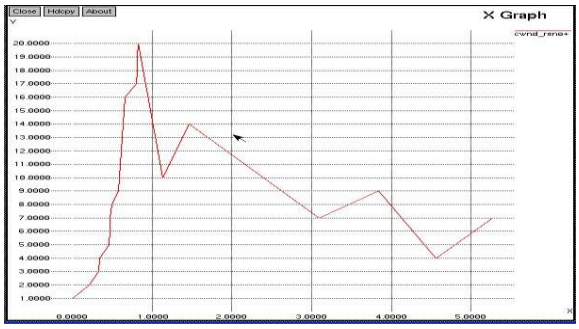
Figure 3: Packet Delay

5.1.3 Effect of modifications on cwnd

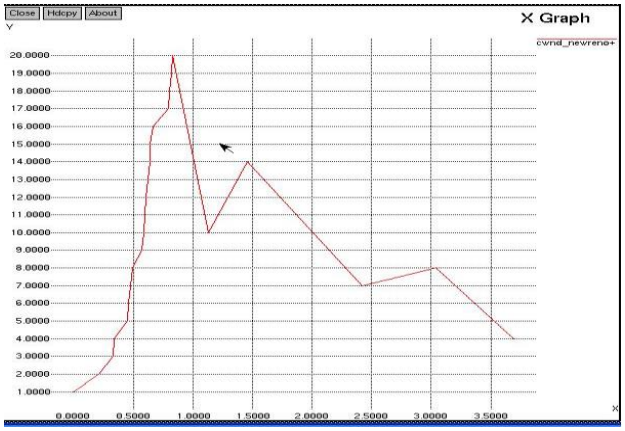
Congestion window (cwnd) is a flow control window imposed at the sender to prevent the sender from sending more data than the network can accommodate. TCP sender dynamically increases or decreases its window size according to the degree of network congestion. Figure 4 shows the behavior of the cwnd under the proposed modifications. Figure 4(a) shows the cwnd of Tahoe+, Figure 4(b) shows the cwnd of Reno+, Figure 4(c) shows the cwnd of NewReno+, while Figure 4(d) shows the cwnd of SACK+. The figures show that the proposed modifications force the cwnd size to increase according to the available bandwidth and it does not increase by one segment as in AIMD strategy. Also, the proposed modifications force the cwnd size to decrease according to the available bandwidth and it does not decrease to half of its value as in AIMD strategy.



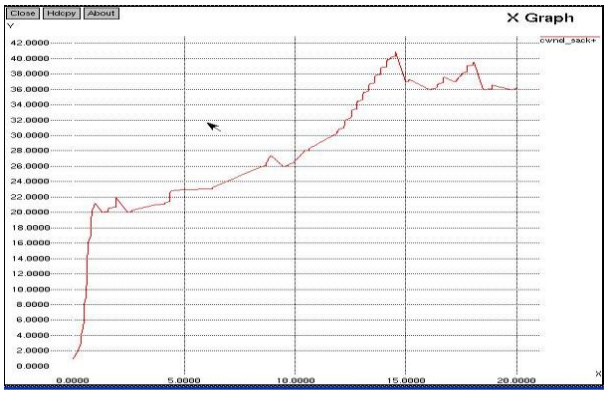
(a) cwnd of Tcp Tahoe+



(b) cwnd of Tcp Reno+



(c) cwnd of Newreno+

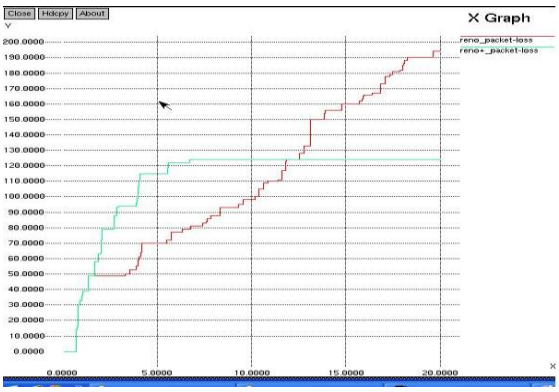


(d) cwnd of sack+

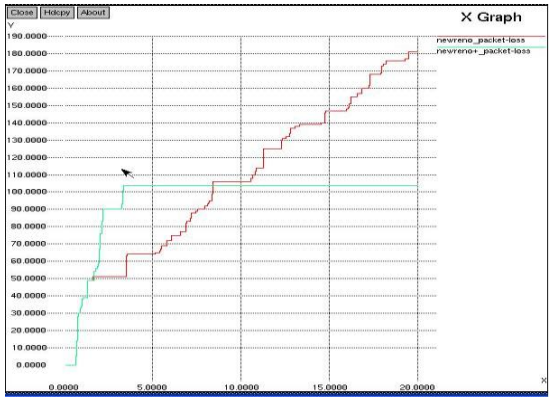
Figure 4: behavior of cwnd of modified protocols

5.1.4 Packet loss

Figure 5 shows the packet loss of both the original and modified congestion control protocols after adding the proposed modifications. Figure 5(a) shows the losses of Reno and Reno+ while Figure 5(b) shows the losses of NewReno and NewReno+. The figures show that the average losses when considering the proposed modifications are less than that of applying the original protocols.



(a) Reno and Reno+



(b) Newreno and Newreno+

Figure 5: packet loss

5.2 Scenario 2: Real topology

In this section, a realistic topology is used to test the performance of the proposed strategy. The AT&T real network topology is created by using the generator GT-ITM, as shown in Figure 6. The topology contains 166 nodes and 189 links with 65 TCP connections in addition to 5 UDP connections [16]. The simulation time is 40 seconds.

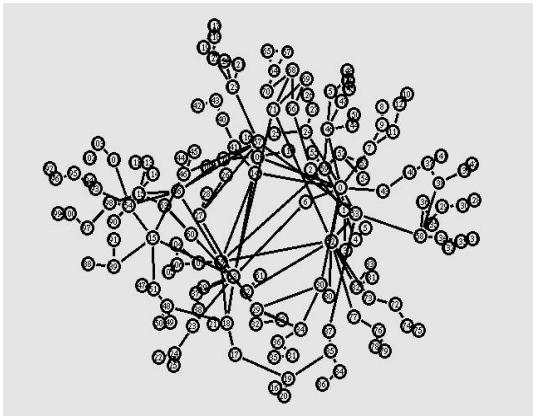
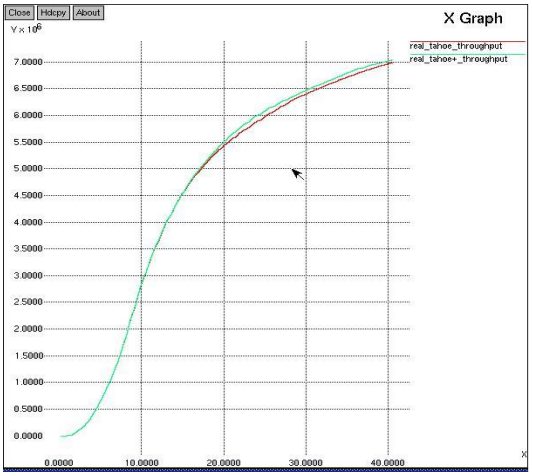


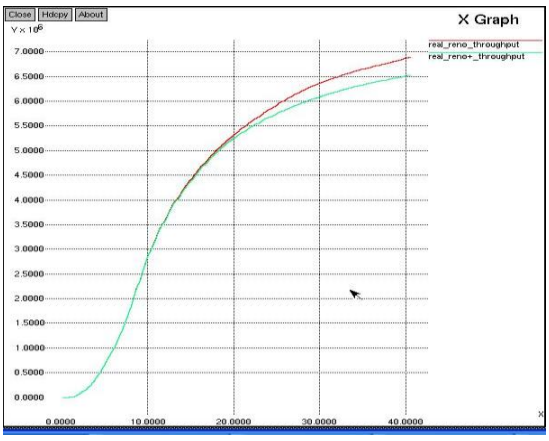
Figure 6: AT&T network topology

5.2.1 Throughput

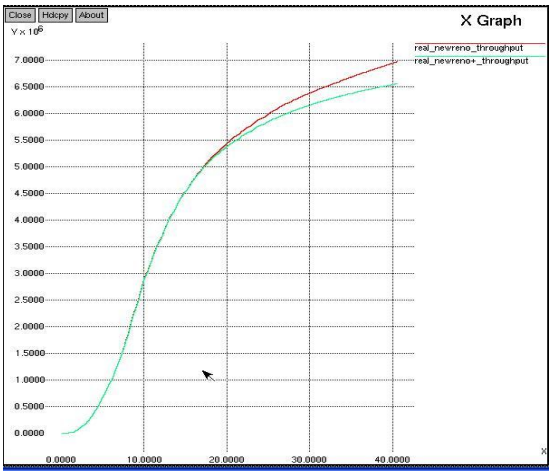
Figure 7 shows the network throughput of both the original and modified congestion control protocols after adding the proposed modification. Figure 7(a) shows the throughput of Tahoe and Tahoe+, Figure 7(b) shows the throughput of Reno and Reno+, Figure 7(c) shows the throughput of NewReno and NewReno+, while Figure 7(d) shows the throughput of SACK and SACK+. From the figures, the network throughput is increased in the case of applying the proposed modifications with the existing protocols.



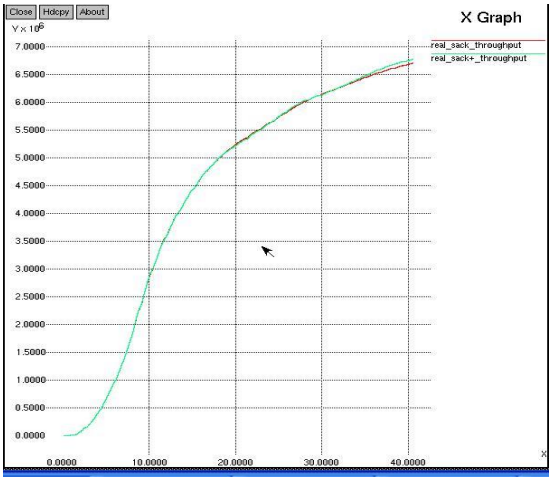
(a) Tahoe and Tahoe+



(b) Reno and Reno+



(c) Newreno and Newreno+

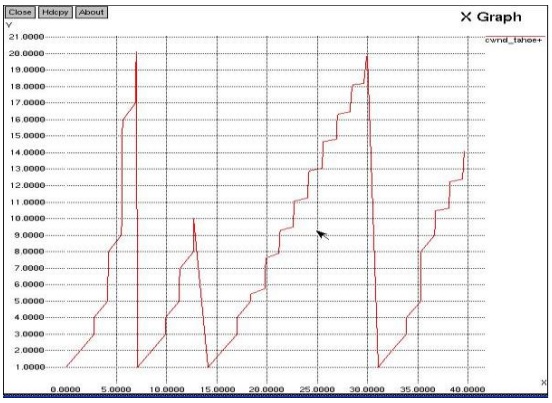


(d) Sack and Sack+

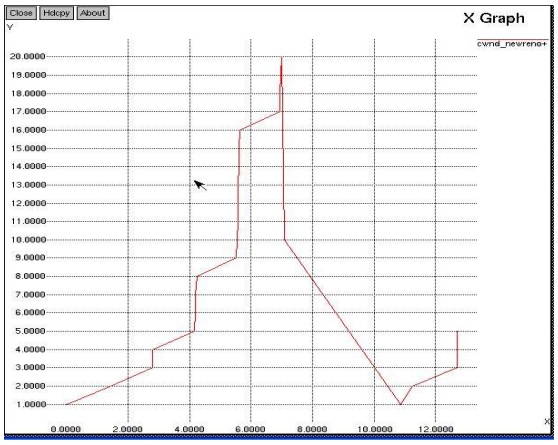
Figure 7: Throughput

5.2.2 Congestion window

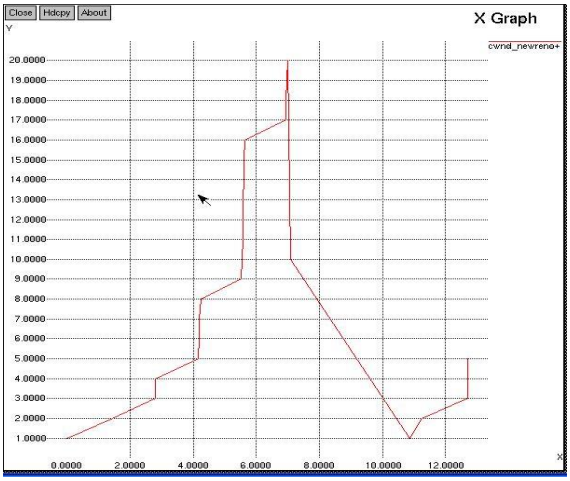
Figure 8 shows the behavior of the cwnd under the proposed modifications. Figure 8(a) shows the cwnd of Tahoe+, Figure 8(b) shows the cwnd of Reno+, Figure 8(c) shows the cwnd of NewReno+, while Figure 8(d) shows the cwnd of SACK+. The figures show that the proposed modifications force the cwnd size to increase rapidly because it increases according to the available bandwidth and it does not increase by one segment as in AIMD strategy.



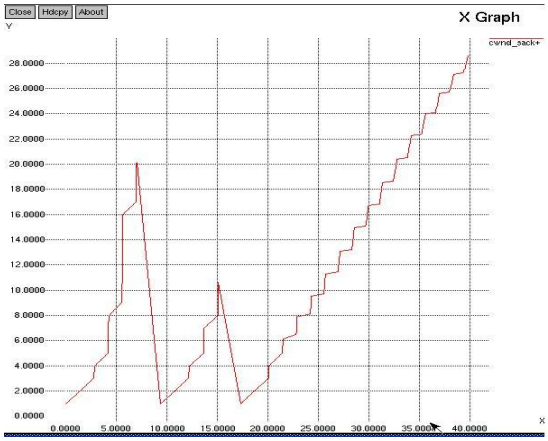
(a) Tahoe cwnd behavior



(b) Reno cwnd behavior



(c) Newreno cwnd behavior



(d) SACK cwnd behavior

Figure 7: cwnd behavior

6. CONCLUSIONS

In this paper, an efficient strategy is developed to dynamically estimate the available bandwidth of the network at any time and then adapts the sending rate at the TCP sender accordingly. The proposed strategy is implemented in the most widely proposed protocols; Tahoe, Reno, NewReno, and SACK instead of the AIMD strategy and then the network performance is tested by using the NS2. The results show that the proposed strategy efficiently utilizes the network capacity as it adapts the sending rate dynamically according to the available bandwidth of the network.

7. REFERENCES

[1] J. Nagle, "Congestion control in IP/TCP Internetworks," Request for Comments (RFC) 896, Internet Engineering Task Force, January 1984.

[2] V. Jacobson, and M. J. Karels, "Congestion Avoidance and Control," Proceedings of ACM SIGCOMM, Vol.18 (4), pp. 314-329, August 1988.

[3] V. Jacobson, "Berkeley TCP Evolution from 4.3-Tahoe to 4.3 Reno," Proceedings of the 18th Internet Engineering Task Force, University of British Columbia, Vancouver, BC, Aug. 1990.

[4] J. Hoe, "Start-up Dynamics of TCP's Congestion Control and Avoidance Schemes," Master Theses, Massachusetts Institute of Technology, 1995.

[5] L. Brakmo and L. Peterson, "TCP Vegas: End-to-End Congestion Avoidance on Global Internet," IEEE Journal on Selected Areas in Communications, Vol. 13, No. 8, pp. 1465-1480, 1995.

[6] M. Mathis, J. Mahdavi, S. Floyd and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, Internet Engineering Task Force, October 1996.

[7] S. Floyd, and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet", IEEE/ACM Transactions on Networking, Vol.79 (4), pp. 458-472, August 1999.

[8] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 3782, April 2004.

[9] D. Roman, K. Yevgeni, and H. Jarmo, "TCP NewReno Throughput in the Presence of Correlated Losses: The Slow-but-Steady Variant", IEEE International Conference on Computer Communications INFOCOM, pp. 1- 6, April 2006.

[10] Cheng-Yuan Ho, Yaw-Chung Chen, Yi-Cheng Chan, Cheng-Yun Ho, "Fast retransmit and fast recovery schemes of transport protocols: A survey and taxonomy," Computer Networks, Vol. 52, pp.1308–1327, 2008.

[11] Hanaa A. Torkey, Gamal M. Attiya and I. Z. Morsi, "Performance Evaluation of End-to-End Congestion Control Protocols", Minufiya Journal of Electronic Engineering Research (MJEER), Vol. 18, No. 2, pp. 99-118, July 2008.

[12] Alexander Afanasyev, Neil Tilley, Peter Reiher and Leonard Kleinrock, "Host-to-Host Congestion Control for TCP," IEEE Communications Surveys & Tutorials, 2010.

[13] Kolawole I. Oyeyinka, Ayodeji O. Oluwatope, Adio. T. Akinwale, Olusegun Folorunso, Ganiyu A. Aderounmu, and Olatunde O. Abiona, "TCP Window Based Congestion Control Slow-Start Approach," Communications and Network, Vol. 3, pp.85-98, , May 2011.

[14] Jeff Edmonds, "On the competitiveness of AIMD-TCP within a general network", Theoretical Computer Science, Vol. 462, pp. 12–22, 30 November 2012.

[15] Hanaa Torkey, Gamal Attiya and Ibrahim Z. Morsi, "Modified Fast Recovery Algorithm for Performance Enhancement of TCP-NewReno", International Journal of Computer Applications, Volume 40, No.12, pp. 30-35, February 2012.

[16] Gamal Attiya, "New Strategy for Congestion Control based on Dynamic Adjustment of Congestion Window", International Journal of Computer Science Issues, Vol. 9, Issue 2, pp. 368-377, March 2012.

[17] Tharwat Ibrahim, Gamal Attiya and Ahmed Hamad, "Fuzzy Based Tuning Congestion Window for Improving End-to-End Congestion Control Protocols", International Journal of Computer Applications (0975–8887), Vol. 87, No. 1, pp. 1-8, February 2014.

[18] Hanaa Torkey, Gamal ATTIYA, Ahmed Abdel Nabi, "An Efficient Congestion Control Protocol for Wired/Wireless Networks", International Journal of Electronics Communication and Computer Engineering, Volume 5, Issue 1, pp. 77-81, 2014. ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

[19] Sharma, Neeraj; Mann, Manish; Thakur, Ravinder, "TCP Congestion Control in Wired com Wireless Networks", International Journal of Computer Applications, Vol. 88, p34-37, Feb 2014.

[20] Rana A., Jennifer Cecilia, "Enhanced TCP Friendly Congestion Control Protocol," International Journal of Computer Theory & Engineering, Vol. 6, Issue 1, p39-42, Feb 2014.

[21] Lei Niu, Feng Wang, Dongdong liu and Bo Guo "A dynamic adjustment algorithm of slow-start threshold based on RTT", Applied mechanics and materials, Vol. 8, pp 782-785, 2014.

[22] K. Fall, and K. Varadhan, "The ns Manual (formerly ns Notes and Documentation)", UC Berkeley, LBL, USC/ISI, and Xerox PARC, December 2006.

[23] GT-ITM "Georgia Tech Internetwork Topology", http://www.cc.gatech.edu/project/gtitm.