Fuzzy based Tuning Congestion Window for Improving End-to-End Congestion Control Protocols

Tharwat Ibrahim
Department of Computer Systems
Faculty of Computer and Information,
Benha University

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

Ahmed Hamad
Department of Computer Systems
Faculty of Computer and Information,
Ain-Shams University

ABSTRACT
Transmission Control Protocol (TCP) is the transport-layer protocol widely used in the internet today. TCP performance is strongly influenced by its congestion control algorithms which limit the amount of transmitted traffic based on the estimated network capacity to avoid sending packets that may be dropped later. In other words Congestion Control is Algorithms that prevent the sender from overloading the network. This paper presents a modified fuzzy controller implementation to estimate the network capacity which reflected by congestion window size. Fuzzy controller use Round Trip Time "RTT" as network traffic indication as well as current window size and slow start threshold "ssthresh" as currently occupied bandwidth indicator. NS2 used as a simulation tool to compare proposed fuzzy approach with most widespread congestion control protocols including; TCP-Tahoe, Reno, New Reno, and Sack. Simulation results show that the proposed mechanism improves the performance against throughput, packet drop, packet delay, and connection fairness.

General Terms: Computer Networks, Network Protocols.
Keywords: Network Protocols, TCP, Congestion control, NS2, Fuzzy logic

1. INTRODUCTION
Congestion is a serious problem for today's wide-area networks, i.e., the Internet. It occurs when many sources sends data packets to a router whose output capacity is less than the sum of the inputs [1]. The result of congestion is that, some packets may be dropped. Congestion control is thus required to avoid congestion collapse and enhance network performance. Without congestion control, a source node could be busy transmitting packets that may be dropped later because of congestion collapse. Over years, continuous efforts are carried out to avoid the problem of congestion collapse. The most widespread mechanism is that provided by the Transmission Control Protocol (TCP). TCP is a window based, connection-oriented, reliable data transfer protocol. It provides byte stream service on the top of the Internet Protocol (IP). TCP has been refined several times during the last years to ensure the internet stability and improve the internet performance. In 1988, a congestion control protocol, called TCP Tahoe, was initiated [2]. It initiates by slow start mechanism, then enters into congestion avoidance when the window size reaches a threshold value, and finally enters into fast retransmit mechanism when detects congestion. In 1990, the TCP Tahoe is modified by adding a fast recovery mechanism. The modified protocol is called TCP-Reno [3]. In Reno, when duplicate acknowledgements arrive at the TCP sender, it enters into fast recovery instead of switching to slow-start as Tahoe. Additional modifications to TCP Reno are done and several protocols are developed. These include; NewReno [4] and SACK (TCP with Selective Acknowledgement) [5]. In TCP-NewReno, a slight modification is added to the TCP-Reno implementation to improve the performance during the fast recovery phase [4]. TCP-SACK only modifies the fast recovery algorithm of TCP-Reno keeping the other algorithms unchanged [5]. Although many congestion control protocols were developed, they use packet dropping as an indication of network congestion. Also, most of the developed protocols use the Additive Increase Multiplicative Decrease (AIMD) strategy to change the packet sending rate at the TCP sender [6, 7]. But, this strategy inefficiently utilizes the available capacity of the internet. With no congestion, the AIMD strategy linearly increases the congestion window (cwnd) size, while it halves the cnwd as well as congestion is detected without regarding the current state of the network. In this paper, a new approach based on fuzzy logic is developed to enhance the end to end congestion control protocols. The basic idea is to adapt the congestion window size, at the TCP sender, dynamically based on the estimated capacity of the network, instead of using the AIMD strategy as all the previous protocols. The proposed strategy is embedded in the TCP Tahoe, Reno, New Reno and Sack, and then evaluated by using the network simulator NS2. The obtained results are compared with that obtained by the most widespread congestion control protocols; TCP Tahoe, Reno, New Reno and Sack. The simulation results indicate that the proposed approach improves the network performance against throughput, packet drop, packet delay and fairness. The rest of this paper is organized as follows. Section 2 introduces the basics of the most widespread congestion control protocols; Tahoe, Reno, NewReno and Sack. Section 3 describes the congestion problem. Section 4 presents the proposed fuzzy approach. The effect of proposed fuzzy approach on congestion avoidance is studied in Section 5 by using NS2. Finally, the conclusions are listed in Section 6.

2. CONGESTION CONTROL PROTOCOLS
This section presents the most widespread congestion control protocols including; TCP-Tahoe, TCP-Reno, TCP-NewReno and TCP-Sack.

2.1 TCP Tahoe
Tahoe introduce in 1988, as the first congestion control protocol, to overcome the problem of congestion collapse. It has three main algorithms called; slow start, congestion avoidance, and fast retransmit.

2.1.1 Slow Start
Slow start is a way to initiate data flow across a connection by gradually increases the amount of data in transient [6]. At the beginning of the connection establishment phase, the congestion window (cwnd) is initialized to one segment. The congestion window then increases by one segment for each

acknowledgement returned, i.e., the cwnd is effectively doubled per RTT (exponential increase). The incremental of the cwnd continues until it arrives to the slow start threshold (ssthresh), or detects a packet loss [8].

```
                    /* slow start */
Initially:      cwnd = 1;

                For each newly acknowledged segment:
                   If (cwnd < ssthresh)
                      cwnd = cwnd + 1;   /*exponential increase*/
                Until (congestion occur, or, cwnd >= ssthresh)
```

2.1.2 Congestion Avoidance

Congestion avoidance begins after slaw start, when the cwnd reaches the ssthresh. In this phase, the cwnd increases linearly by one segment every Round Trip Time (RTT) to avoid possible congestion. The increasing rate of the cwnd continues until congestion event is detected [8]. At this point, the transmission rate should be slowed down, as follows: (i) If the congestion is detected by 3 duplicate ACKs, the TCP sender invokes the fast retransmit phase because it believes that a segment has been lost. (ii) If the congestion is detected by the timeout expired, the TCP sender got the slow start phase.

```
                    /* Congestion Avoidance */
For each newly acknowledged segment:
          /*cwnd increases by 1 per RTT */
          cwnd += 1/cwnd;          */Additive increase*/
Until (timeout or 3 Duplicate ACKs)
If 3 Duplicate ACKs:
     cwnd = cwnd/2;                /* Multiplicative decrease */
     ssthresh = cwnd;
     Invoke fast retransmit phase
If Timeout:
     ssthresh = cwnd/2;            /* Multiplicative decrease */
     cwnd = 1;
     Go to slow start phase
```

2.1.3 Fast Retransmit

In fast retransmit, the sender retransmits what seems to be missing segment, without waiting for timeout expiration, and enters into slow start phase after setting the cwnd to 1 [9]. Figure 1 shows the behaviour of the cwnd during different phases. At beginning, the cwnd is set to 1 and increases in exponential manner until reaches the ssthresh. Then the congestion avoidance phase starts and the cwnd increases in linear manner until occurrence of congestion event due to packet loss. The new ssthresh is set to half of cwnd and the cwnd is set to 1, and then the protocol enters into the slow start phase.

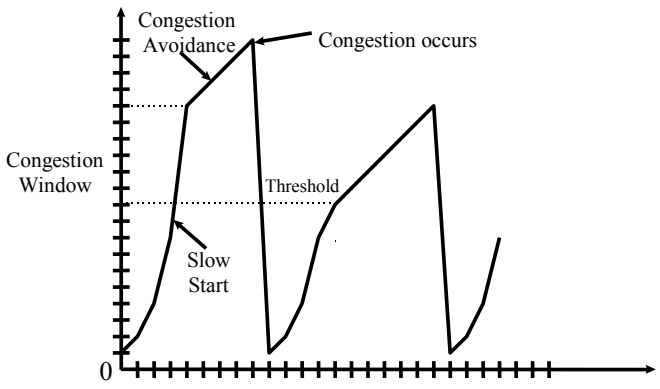

Fig 1: The cwnd behaviour

2.2 TCP Reno

Reno is the improvement of TCP Tahoe. It includes a fast recovery algorithm to TCP Tahoe.

2.2.1 Fast Recovery Algorithm:

In TCP Reno, the sender switches to the fast recovery phase after fast retransmit, when it receives 3 duplicate acknowledgements, instead of switching to slow-start as Tahoe. The reason for going to fast recovery instead of slow start is to allow the sender to work in congestion avoidance phase. This behaviour improves the network throughput as more packets will be transmitted [9, 10].

```
                    /* Fast Recovery */
/* after retransmission of missed packets, don't enter slow start
but enter fast recovery */

     ssthresh = cwnd/2
     cwnd= 3 + ssthresh
     Return to congestion avoidance phase
```

Briefly, when the TCP sender detects congestion, it must slow down its transmission rate. Generally, the sender may detect congestion as a result of the reception of 3 duplicate acknowledgements or the expiration of retransmission timer. If the congestion is detected by a timeout, the congestion window is reset to one segment and the sender enters into Slow Start mode. But, if the congestion is detected by 3 duplicate acknowledgements, fast retransmit and then fast recovery algorithms are invoked. At that time, the congestion window at the TCP sender is set to one half of the current window size, with a minimum value of at least two segments. During fast retransmit, after setting the value of cwnd and ssthresh, the sender retransmits what seams to be lost, then, enters the fast recovery mode. When the sender receives a fresh acknowledgement, it exits that phase and starts congestion avoidance phase.

2.3 TCP NewReno

NewReno provides a slight modification to the TCP-Reno to overcome the problem arises with Reno whenever multiple packets are lost from the same transmission window [4]. In other words, NewReno enhances the sender's behaviour during the fast recovery mechanism to eliminate the Reno's wait for a retransmit time-out whenever multiple packets are lost from the same transmission window [10]. The new medications make the sender to continue in fast recovery until all the packets which were outstanding during the start of the fast recovery have been acknowledged. This behaviour overcomes the problem of multiple losses without entering into fast recovery multiple times or causing timeout. In this case, two types of acknowledgements are used; partial ACK and full ACK. The partial ACK is considered as an indication that the packet following the acknowledged one has been dropped from the same transmission window, and therefore, TCP NewReno immediately retransmits the other lost packet indicated by the partial acknowledgement and remains in fast recovery. The full ACK is considered as an indication that the entire data packet in the window is acknowledged and the exit fast recovery.

```
     After ACK
     If (partial ACK)
     then stay in fast recovery and transmit one packet every RTT
     If (full ACK)
     then exit fast recovery
```

2.4 TCP Sack

TCP SACK (TCP with Selective Acknowledgement) is an extension to the TCP Reno. It only modifies the fast recovery algorithm of Reno keeping the other algorithms unchanged [5]. Similar to NewReno, TCP SACK handles multiple packet losses from the same window but it has a better estimation capability for the number of outstanding packets. In TCP-Reno, a Duplicate ACK means that the receiver received data out of order. In TCP-Sack, a duplicate ACK carries the same information and also carries information on what other data segments that have been received out-of-order. The sender starts the data recovery mechanism after receiving 3 duplicate ACKs. TCP-Sack adds to the packet an option field that contains a pair of sequence numbers which describe the block of data that was received out-of-order. A TCP sender uses Sack options to build a table of all the correctly received data segments, thus it knows exactly which parts are missing to retransmit them together during a recovery phase. The main drawbacks of this protocol is that the receiver side needs some modification and difficult to implement.

3. PROBLEM STATEMENT

From the above discussion, several congestion control protocols have been developed to prevent congestion collapse. However, it is clear that, all the previous protocols are developed based on 4 mechanisms; slow start, congestion avoidance, fast retransmit and fast recovery. Indeed, the developed protocols use packet drop as the indication of network congestion. Also, they use the AIMD (Additive Increase, Multiplicative Decrease) strategy to change the packet sending rate at the sender. The slow start approach blindly changes the sending rate in static manner without regarding the current state of the network. Briefly, at beginning, the slow start begins by setting the cwnd to one segment and then duplicate the cwnd every RTT. This behaviour inefficiently utilizes the available bandwidth at the beginning of this phase. Indeed, it needs many RTT to reach the optimal operating point that allows using the available bandwidth. Also, the exponential growth of the cwnd may cause severe buffer overflow and so many packets may be dropped at the moment of congestion event. Also, the Additive Increase Multiplicative Decrease (AIMD) strategy that used during the congestion avoidance phase inefficiently utilizes the available capacity of the network. In the case of no congestion, the AIMD linearly increases the congestion window (cwnd) size at the TCP sender. This in turn does not utilize the network capacity efficiently because the number of packets to be transmitted is less than the available capacity. On the other hand, when congestion is detected, the AIMD halves the congestion window size without regarding the current state of the network which in turn decreases the packet sending rate and so decreases the network utilization.

To overcome this problem, the sending rate should be adapted as a function of the available capacity of the network. In other words, a new approach is required at the TCP sender, instead of the AIMD strategy, to adopt the sender sending rate based on the available network capacity at any time.

4. PROPOSED ALGORITHM

This section presents a new approach, called fuzzy controller, to adapt congestion window size at the TCP sender dynamically based on the available capacity of the network. This approach may be embedded in all the previous protocols, instead of the AIMD strategy, to improve their performance. In other words, the fuzzy controller could be applied instead of both the slow start and the congestion avoidance phases. The basic idea of the proposed fuzzy controller is that, the congestion window size at the TCP sender is determined dynamically at any time based on the traffic load in the network. The proposed fuzzy controller changes the behaviour of the cwnd from static exponential increase during slow start and static linear increase during congestion avoidance to dynamic non-analytical approaches and more intelligence behaviour that based on the network traffic load and current cwnd to ssthresh gap. The following sections first present the basic concepts of fuzzy logic and then describe the proposed fuzzy controller.

4.1 Fuzzy logic

Fuzzy logic is one of the Computational Intelligence (CI) tools that may be used in situations where traditional control theoretic approaches cannot be used because of the difficulties in obtaining a formal analytical model. Fuzzy Logic Control (FLC) may be viewed as a way of designing feedback controller in which analytical models are not easily obtainable or the model itself, if available, is too complex and possibly highly nonlinear [11,12]. The FLC has been applied successfully for controlling systems. It basically consists of four parts including; a fuzzifier, a defuzzifier, an inference engine and a fuzzy rule base, as shown in Figure 2 [11]. As in many fuzzy control applications, the input data are usually crisp, so a fuzzification is necessary to convert the input crisp data into a suitable set of linguistic value that is needed in inference engine. In the rule base, a set of fuzzy control rules, which characterize the dynamic behaviour of system, are defined. The inference engine is used to form inferences and draw conclusions from the fuzzy control rules.

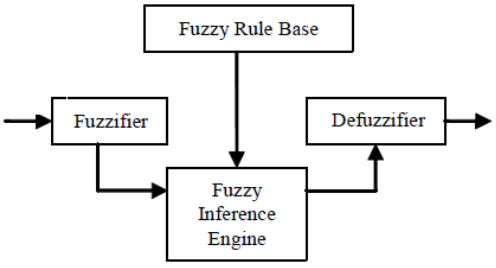

Fig 2: Basic configuration of fuzzy system

4.2 Proposed Fuzzy Controller

The proposed fuzzy controller adapts the size of congestion window using fuzzy system. The system is based on the current cwnd, the ssthersh, the average RTT and the network traffic load that indicated by Last RTT. The proposed fuzzy controller has two input parameters sc and dr, and a fuzzy controller output parameter dcwnd. Where, sc = (ssthresh-cwnd) / ssthresh and dr = (average RTT- RTT) / (0.5(average RTT + RTT)) [13]. The sc parameter is the first FLC Member Ship Function (MSF). It indicates the gap between the ssthresh and the cwnd and ranges from 0 to 1, as shown in Figure 3. The value 0 means ssthresh = cwnd (denoted by L), the value 1 means cwnd = 0 (denoted by H) and the value 0.5 means ssthresh = 2 cwnd (denoted by M).

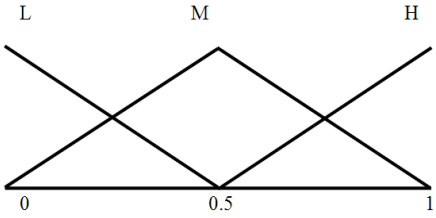

Fig 3: The sc Member Ship Function

The dr parameter is the second FLC MSF. It indicates the network traffic and ranges from 0 to 2, as shown in Figure 4, where, 0 reflects low traffic and 2 reflects high traffic.

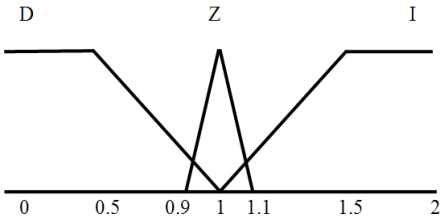

Fig 4: The dr Member Ship Function

In proposed fuzzy approach, the dr parameter reflects the following; from 0.5 to 1 means network traffic decrease "D", from 1 to 1.5 network traffic increase "I", and from 0.9 to 1.1 the network traffic stable "Z". The proposed fuzzy controller uses singleton fuzzifier, product interference engine, centre average defuzzifier [13] and the proposed fuzzy rules is shown in Table 1.

In the proposed approach, fuzzy sets centres range from 1 to 8, as shown in Table 1. The final output is multiplied by weight function (wf), as shown in Figure 5. The wf is added as a block in the FLC as shown in Figure 6 with the final output.

wf = 2 - (cwnd / ssthresh) during slow start

wf = (ssthresh / cwnd) during congestion avoidance

Table 1: Fuzzy Rules

sc			dr			dcwnd	
L	M	H	D	Z	I		
1	0	0	1	0	0	S-	1
1	0	0	0	1	0	S	1.5
1	0	0	0	0	1	S+	2
0	1	0	1	0	0	M-	3
0	1	0	0	1	0	M	4
0	1	0	0	0	1	M+	5
0	0	1	1	0	0	L-	6
0	0	1	0	1	0	L	7
0	0	1	0	0	1	L+	8

At the beginning of slow start wf duplicate the FLC output and at the end of slow start wf = 1 but during congestion avoidance wf is Inverse relationship = ssthresh / cwnd.

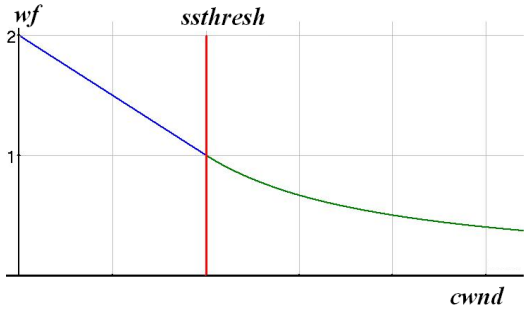

Fig 5: weight function (wf)

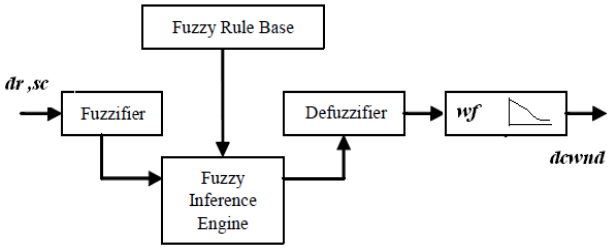

Fig 6: proposed fuzzy system

5. PERFORMANCE EVALUATION

To study the effect of the suggested fuzzy controller on the behaviour of TCP, the old cwnd of the TCP agent is replaced with the new strategy. Then, its behavior is tested using the network simulation tool NS2 [14] and compared with the original TCP protocols considering different topologies; simple and real AT&T topology. In each topology, the comparison is focused on throughput, losses, packets delay and fairness.

5.1 Simple topology

The simple topology constructed from 6 nodes; two sources (N1 and N2), two destinations (N3 and N4), and two routers (R1 and R2), as shown in Figure 7. The link between the routers has bandwidth of 1.7 Mbps and delay of 20ms and acts as the bottleneck link of this topology. A TCP connection is established between the source N1 and the sink N3 to transfer File Transfer Protocol (FTP) application. A UDP connection is established between the source N2 and the sink N4 to transfer Constant Bit Rate (CBR) application. The source nodes N1 and N2 are connected with R1 by a link has 2 Mbps bandwidth and 10 ms delay. The destination nodes N3 and N4 are connected with R2 by a link has 2 Mbps bandwidth and 10 ms delay. Total simulation time is 20 seconds. The FTP application on the TCP connection starts at first second and ends at 20s. The CBR application for UDP connection starts at second 8 and stop at second 12. This scenario is used to study the behavior of the congestion control mechanisms in case of suddenly traffic is added and suddenly removed and also during sharing same link with UDP connection.

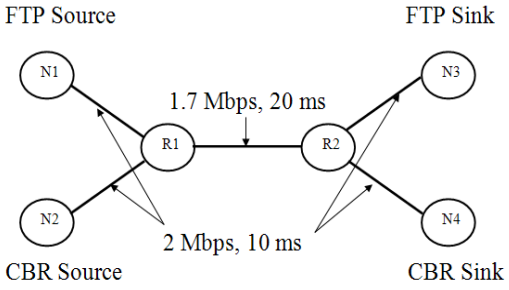

Fig 7: Simple Network Topology

Figure 8 shows the instantaneous throughput of the original congestion control protocols; Taheo, Reno, NewReno and Sack. The figure shows that the highest utilization of network capacity is achieved by the TCP NewReno.

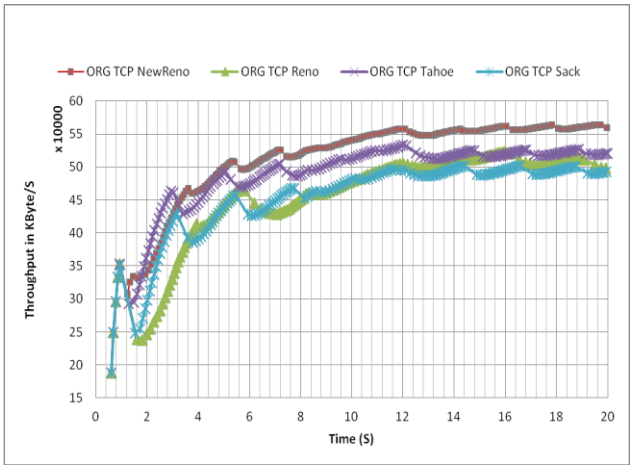

Fig 8: Throughput of original protocols

Figure 9 shows the instantaneous packet losses of the original congestion control protocols; Taheo, Reno, NewReno and Sack. The figure shows that the lowest packet loss is achieved by the TCP Sack.

Table 2 shows the average values of throughput, packet losses and delay of the original congestion control protocols and original protocols under different modification steps of the proposed algorithm. In Table 2;

ORG: original protocols without modifications.

M1: proposed algorithm in [13].

M2: same M1 with proposed fuzzy rules in Table 1 and wf Figure 6.

M3: same M2 with proposed MSF Figure 4.

M4: same M3 with average RTT calculated from the 3 previous RTT with more weight for last RTT, average RTT = (3*previous1 RTT + 2*previous2 RTT + previous3 RTT)/6

M5: same M3 with average RTT = (previous1 RTT + previous2 RTT +previous3 RTT)/3

M6: same M3 with average RTT = smoothed RTT which is the average of RTT from time of connection established.

From Table 2; the simulation results show that the proposed modification M4 provides the best network utilization overall other modifications.

Table 2: Simulation results under different modifications

		ORG	M1	M2	M3	M4	M5	M6
Throughput [kbps]	NewReno	4026.29	4138.52	4262.41	4284.18	4300.24	4180.8	4294.42
	Reno	3552.05	3589.41	4102.01	4170.14	4115.52	3997.86	4144.92
	Tahoe	3730.95	3819.63	3986.11	3997.36	3978.29	3936.91	4110.44
	Sack	3517.44	3617.83	3793.49	3815.89	3773.41	3806.14	3848.78
Losses [kbps]	NewReno	14.75	16.15	13.43	12.32	10.73	25.67	14.19
	Reno	13.78	8.63	9.75	10.8	9.13	17.34	8.91
	Tahoe	15.39	13.1	8.63	9.76	8.99	11.49	6.84
	Sack	10.92	7.89	7.26	8.65	8.49	9.01	13.84
Delay [s]	NewReno	0.0542	0.0548	0.0551	0.0552	0.0551	0.0546	0.0559
	Reno	0.0537	0.054	0.0548	0.0549	0.0547	0.0541	0.0551
	Tahoe	0.0546	0.0541	0.0549	0.055	0.055	0.0551	0.0556
	Sack	0.0542	0.0537	0.0546	0.0549	0.0547	0.0549	0.0553

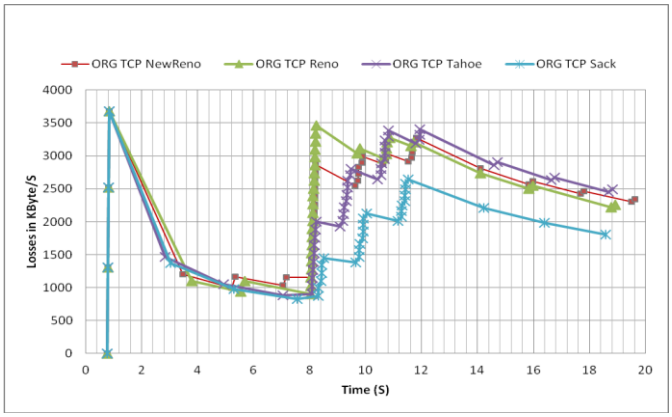

Fig 9: Packet losses of original protocols

Figure 10, 11, and12 show the instantaneous throughput, packet dropping and delay of the original TCP NewReno and the TCP NewReno including M4 modification. From the figures, embedding the proposed modification M4 into the TCP NewReno improves its performance against throughput, packet dropping and packet delay.

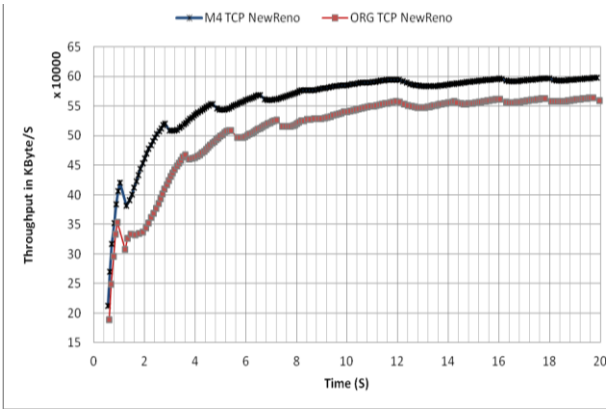

Fig 10: Throughput of NewReno and NewReno with M4

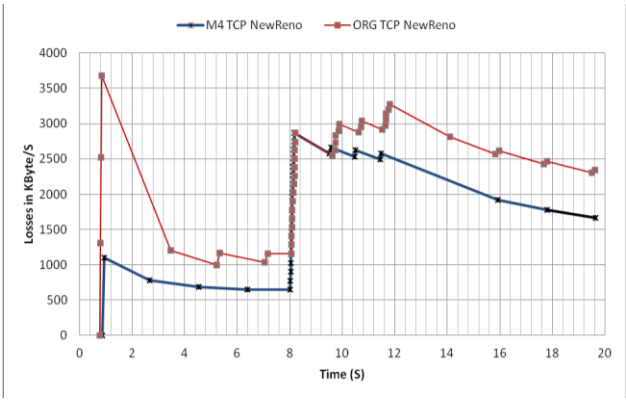

Fig 11: Packet losses of NewReno and NewReno with M4

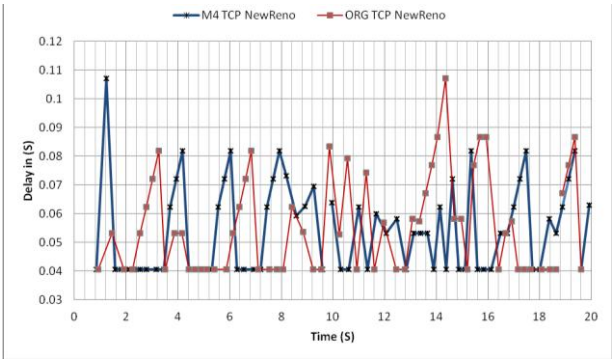

Fig 12: Packet delay of NewReno and NewReno with M4

Figure 13, 14, and 15 show instantaneous throughput, packet dropping and delay of the TCP NewReno including M1 modification and the TCP NewReno including M4 modification.

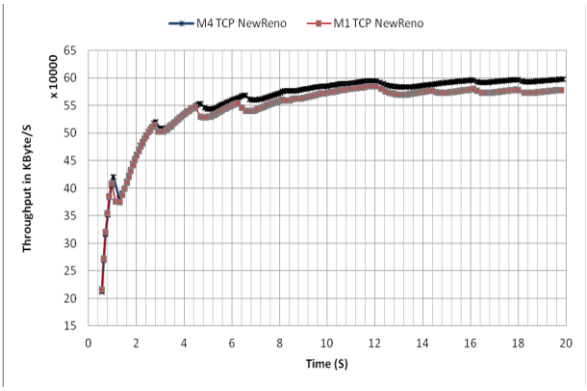

Fig 13: Throughput of NewReno with M1 and NewReno with M4

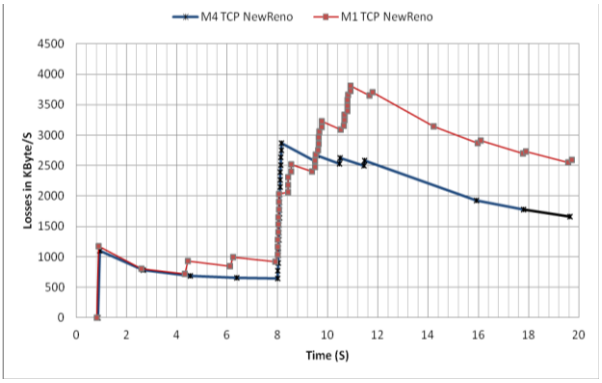

Fig 14: Packet losses of NewReno considering M1 and M4

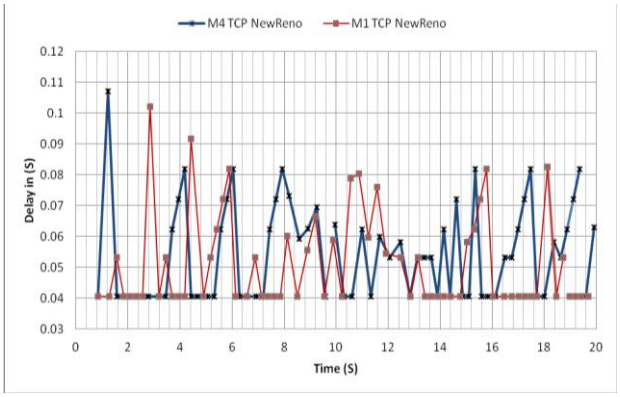

Fig 15: Packet delay of NewReno with M1 and M4

Figure 16 shows the behavior of cwnd of the original TCP NewReno and the TCP NewReno including M4 modification. Figure 17 shows the behavior of cwnd of the TCP NewReno with M1 and the TCP NewReno including M4 modification. From the figures, embedding the proposed modification M4 into the TCP NewReno improves its performance.

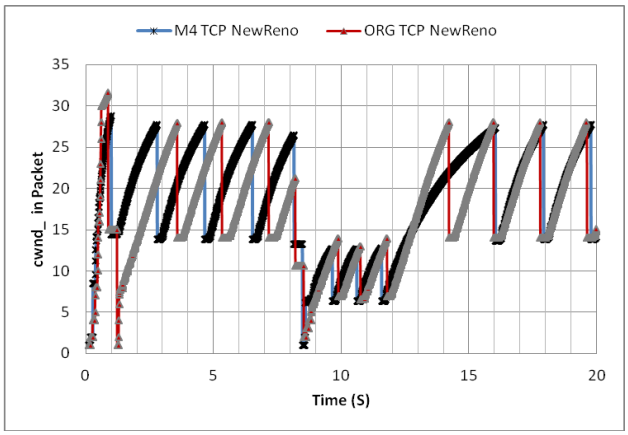

Fig 16: cwnd of NewReno and NewReno with M4

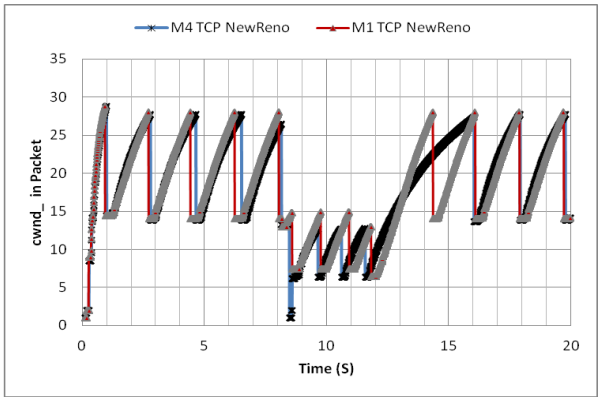

Fig 17: cwnd of NewReno with M1 and NewReno with M4

Table 3 shows average throughputs for TCP and UDB flows, Fuzzy M4 approach enhance TCP as shown and UDB is the same because of CBR application which generate constant traffic.

Table 3 Total, TCP and UDB average throughput

Throughput [kbps]		Newreno	Reno	Tahoe	Sack
ORG	Total	4026.29	3552.05	3730.95	3517.44
	TCP	3724.74	3250.86	3430.19	3214.62
	UDB	301.55	301.19	300.76	302.82
Fuzzy	Total	4300.24	4115.52	3978.29	3773.41
	TCP	3995.78	3808.2	3674.27	3466.9
	UDB	304.46	307.32	304.02	306.51

5.2 Fairness Study

Although the proposed modification improves the protocols in terms of link utilization and packet delay, the protocol must also be fair against different flows. To measure fairness, the following Fair Index [15] is used.

$$\text{FairnessIndex} = \frac{\left(\sum x_i\right)^2}{n\left(\sum x_i^{\,2}\right)}$$

Where, xi is the throughput of the ith flow and n is the total number of flows. The fairness index of a system ranges from 0 to 1, with 0 being totally unfair and 1 being totally fair [15].Figure 18 shows the network topology that used to study the fairness between 2 TCP flows. Each flow transfers FTP application. The two FTP start and end at the same time, and share the same bottleneck connection between R1 and R2.

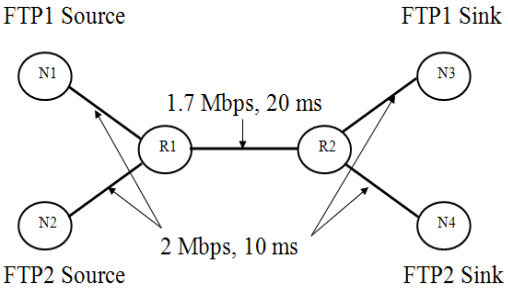

Fig 18: Network Topology with 2 TCP flows

Tables 4, 5 and 6 show the average throughput, the average packet losses, and Fairness index respectively, for the original protocols with and without the proposed fuzzy controller. From the tables, embedding the fuzzy approach into the protocols enhance total network utilization, decrease packet losses and achieve fairness between the different network flows.

Table 4: average throughput with and without fuzzy controller

Throughput [kbps]		Newreno	Reno	Tahoe	Sack
ORG	Total	4604.03	4411.57	4458.97	4345.05
	TCP1	2292.57	2746.92	2560.17	2810.34
	TCP2	2311.46	1664.65	1898.8	1534.71
Fuzzy	Total	4713.87	4681.63	4584.06	4578.22
	TCP1	2359.47	2478.48	2029.34	2856.51
	TCP2	2354.4	2203.15	2554.72	1721.71

Table 5: Average losses with and without fuzzy controller

Losses [kbps]		Newreno	Reno	Tahoe	Sack
ORG	Total	24.56	27.78	28.93	25.37
	TCP1	13.34	14.76	15.93	14.56
	TCP2	11.22	13.02	13	10.81
Fuzzy	Total	16.77	16.11	17.84	15.75
	TCP1	8.6	8.31	10.88	8.32
	TCP2	8.17	7.8	6.96	7.43

5.3 Real topology

In this section, a realistic topology is used to test the performance of the proposed strategy. The topology generator GT-ITM is used to produce the AT&T real network shown in Figure 19. The topology contains 166 nodes and 189 links with 65 TCP connections in addition to 5 UDP connections [16]. The simulation time is 45 seconds.

Table 6 Fairness Index with and without fuzzy controller

Fairness Index		Newreno	Reno	Tahoe	Sack
ORG	Throughput	0.999983166	0.943232022	0.978473717	0.920648735
	Losses	0.992604099	0.996092189	0.989846721	0.978618652
Fuzzy	Throughput	0.999998843	0.996553221	0.987034808	0.942117107
	Losses	0.99934297	0.998998815	0.953942068	0.996817017

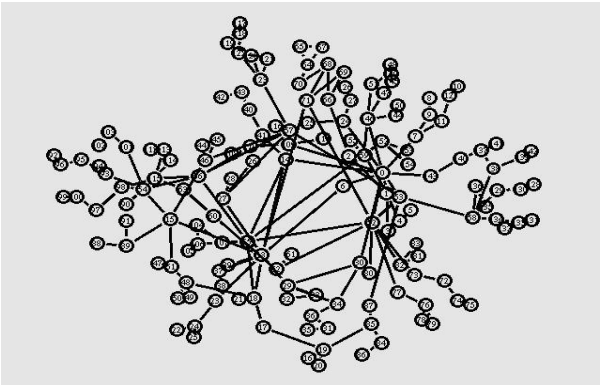

Fig 19: AT&T network topology

Figure 20, 21, and 22 presents the average throughput, the average packet losses, and average packet delay respectively, for the original protocols with and without the proposed fuzzy controller. From the figures, the proposed fuzzy approach enhances throughput, reduces both the average packet dropping rates and average packet delay.

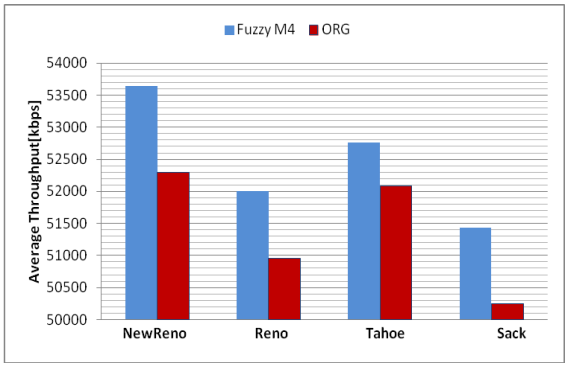

Fig 20: Average throughput with and without modification

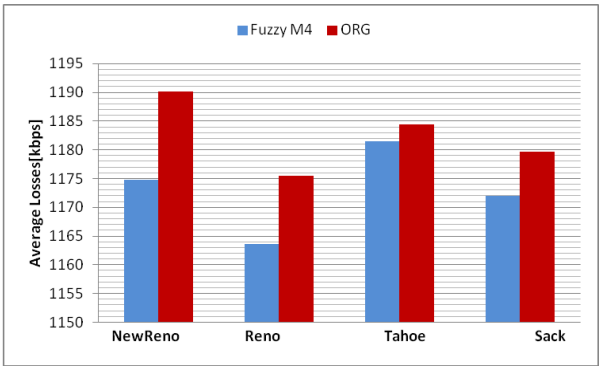

Fig 21: Average packet lost with and without modification

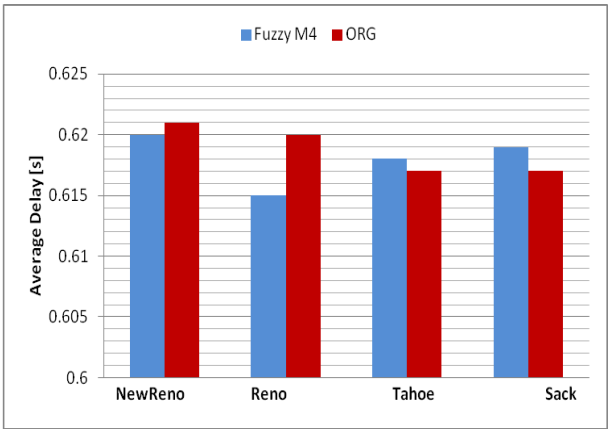

Fig 22: Average packet delay with and without modification

6. CONCLUSIONS

In this paper, a fuzzy logic controller is proposed to adapt the congestion window size dynamically based on the available network capacity at any time of operation. This approach is developed to be used instead of the AIMD strategy that blindly adapts the congestion window size. The proposed strategy is embedded into the most widespread congestion control protocols including, Tahoe, Reno, NewReno, and Sack. The performance behaviour of the proposed protocol is tested and compared with the original protocols considering simple and real network topologies by using the network simulator NS2. The simulation results show that, the proposed fuzzy approach improves the original end-to-end congestion control protocols against throughput, packet dropping rates and packet delay.

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, "Congestion Avoidance and Control," ACM SIGCOMM Computer Communication Review, Vol. 18, No. 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] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 3782, April 2004.

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

[6] W. Stevens, no, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC 2001, January 1997.

[7] 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.

[8] 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.

[9] 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.

[10] S. Floyd, T. Henderson, A. Gurtov, Y. Nishida "The NewReno Modification to TCP's Fast Recovery Algorithm" RFC 6582, April 2012.

[11] Adel Nadjaran Tousi, Mohammad Hossien Yaghmaee, " A Fuzzy Based TCP Congestion Controller" international symposium on telecommunications, PP. 641-646, September 10-12 2005

[12] Deepa Jose, R.R.Mudholkar "Congestion Control in TCP/IP Using Fuzzy Logic" IJMIE, Volume 2, Issue 5 ISSN: 2249-0558, PP. 539-544, May 2012.

[13] H. Nejad, M. Yaghamaee, H. Tabatabaee "Modified Fuzzy TCP: Optimizing TCP congestion control", IEEE 2006

[14] NS2 Network Simulator, http://www.isi.edu/nsnam/ns/

[15] Deepa Jose, Dr.R.R.Mudholkar, "Congestion Control in TCP/IP Using Fuzzy Logic", IJMIE Volume 2, Issue 5 ISSN: 2249-0558, PP. 568-576, May 2012.

[16] H. Natiq James, Z. Ahmed Zukarnain, M. Shamamla Subramaniam, "Fairness of the TCP-Based New AIMD Congestion Control Algorithm" Journal of Theoretical and Applied Information Technology, KOM Technical Report 2002, JATIT.