Fuzzy Controller based TCP-Vegas Enhancement for Congestion Control

Doaa H. Esawi
Industrial Electronics and Control
Faculty of Electronic Engineering
Menoufia
engdoaaelnady2005@yahoo.com

Gamal Attiya
Computer Science
Faculty of Electronic Engineering
Menoufia
gamal_mahrouce@yahoo.com

Gaber Allam
Industrial Electronics and Control
Faculty of Electronic Engineering
Menoufia
Gaber.ebrahim@el-eng.menofia.edu.eg

***Abstract*— Due to the serious effectiveness of congestion problem in the Internet performance, congestion control has the most concern in the network community. Several End-to-End mechanisms were developed to overcome this problem. However, most of the existing mechanisms adapt the sending rate at the sender, when detecting congestion, without considering the network status. This behavior degrades the Internet performance. This paper presents a new fuzzy controller to adjust the sending rate at the sender dynamically based on the network load. The intended controller is employed to enhance the TCP-Vegas and the performance is evaluated by using the well-known Network Simulator NS-2. The results indicate that the intended controller the AT&T real network increases the throughput and decreases both the packet loss and packet delay.**

***Keywords*— Congestion Control, Fuzzy Logic, TCP, TCP-Vegas, NS2 .**

I. Introduction

During the past few years, several changes have been made to the Transmission Control Protocol (TCP) [1], and different mechanisms have been implemented to avoid congestion problem [2]. The most popular mechanisms include TCP Tahoe [3], Reno , New-Reno [12], Sack [6] and Vegas [4,5,7,8,10]. However, most of the existing mechanisms adapt the sending rate at the sender without considering the network load. Briefly, when detecting congestion, some mechanisms adjust the congestion window size to one segment and go back to slow start phase. Other mechanisms adjust the congestion window to slow-start threshold and go back to congestion avoidance phase. This behavior decreases the network throughput especially at high traffic load because the protocol does not use the available network capacity.

This paper presents a new fuzzy controller to enhance the TCP-Vegas for congestion control. The intended controller changes the sanding rate at the sender dynamically based on the network load. It detects the load by using the Round Trip Time (RTT) and then adjusts the congestion window size that in turn adjusts the sending rate. The achievement of the intended TCP-Vegas fuzzy controller is evaluated by using the well-known Network Simulator NS-2.

The remainder is regular as follows, Section 2 describes the behavior of the TCP Vegas while section 3 presents the problems of TCP Vegas. Section 4 describes the intended Fuzzy based TCP congestion controls while section 5 presents performance evaluation. Finally, the paper conclusions.

II. TCP Vegas

TCP Vegas is prefectly different from TCP Tahoe, Reno and their modifications. It uses packet delay, rather than packet loss, as a signal to detect congestion at an incipient stage and then determine the sending packet rate at the sender . In other words, TCP Vegas develops the congestion avoidance and fast retransmission phases of TCP Reno to linearly increase or decrease the window size at the sender conformity to the observed Round Trip Time (RTT) of the packets that it has already sent. Commonly, Vegas works by observation the difference between the expected and the actual flow rates and adjusts the congestion window. It first sets Base RTT to the smallest measured RTT, and calculates the expected flow rate according to

$$Expected = cwnd / BaseRTT \quad (1)$$

Where, cwnd is the current window size.

In addition, it calculates the actual flow rate as follows.

$$Actual = cwnd / RTT \quad (2)$$

Where, RTT is computed by the difference between the time at which the ACK comes back and the sending time of the packet.

Then, it computes the difference between the expected and the actual flow rates as follows.

$$Diff = (Expected - Actual) * base\ RTT \quad (3)$$

Finally, it updates the cwnd continuously to:

$$cwnd + 1; \quad if\ Diff < \alpha$$

$$cwnd - 1; \quad if\ Diff > \beta$$

$$cwnd; \quad otherwise$$

Where, α and β are two thresholds more than zero [10] .

In TCP Vegas, the difference is used to adapt the window size. TCP Vegas defines two threshold values ; α and β such that $(\alpha < \beta)$. When Diff < α, Vegas increases the window size linearly during the next RTT. If Diff > β, reduced then Vegas decreases the window size linearly during the next RTT. Otherwise, it leaves the window size unchanged. This means that, if the observed RTT becomes large, the network is encounter congestion, causing TCP Vegas to reduce its window size. In addition, if the observed RTT becomes small, the network is not encounter congestion causing TCP Vegas to increase its window size.

III. Problem statement

TCP Vegas has some problems that have critical impacts on its performance ,When route of a connection changes, the RTT increases and Vegas misreads it as the outcome of the network congestion and accordingly decreases its own sending rate to one segment in some mechanisms and to the threshold value in other mechanisms . This behavior inefficiently utilizes the available capacity of the network. In case of no congestion, it leads to a considerable decrease of network throughput since the number of packets to be transmitted is less than the available capacity of the network.

When a flow reaches to the network later than other flows and faces with congested queues, it wrongly assesses the measured RTT as its initial Base RTT. This means that while other flows decrease their sending rates due to running congestion, this flow does not sense the congestion and inequitable increases its sending rate .

To overcome these problems, a new design is required to adjust the sending rate at the sender according to the available bandwidth of the network at any time.

IV. Intended Design

In this section, a new congestion control design is developed to enhance the TCP Vegas. The main idea of the intended design is to employ fuzzy logic capabilities to dynamically adapt the congestion window (cwnd) that in turn adjust the sending rate. To do this, the controller changes the window size based on five distance values between the Diff value and the two thresholds instead of changing the window size according to the comparison between the Diff with the two fixed thresholds only. That is, if the distance between the Diff value and thresholds is very low, low, medium, high or very high, the controller changes the window size very low, low, high or very high, respectively.

Generally, designing fuzzy controller requires awareness of the effect of the membership functions on the efficiency of the controller. In other words, the challenging tasks associated with fuzzy controller are to choose appropriate membership functions, shape of membership functions i.e. triangular, Gaussian etc., inference mechanism, minimum rule base and the most convenient fuzzifier and defuzzifier .

Research results have shown that singleton fuzzifier, triangular and trapezoid input membership functions, center average defuzzifier and product inference engine works best for the fuzzy controller [9,11,13,14].

Fuzzy based TCP congestion control presents a fuzzy based TCP congestion controller using a Single Input – Single Output (SISO) fuzzy controller. The input parameter to the controller is Diff. The output of fuzzy controller is Adapt that used to adjust the value of congestion window rightly. The input and output Fuzzy sets of the linguistic variables are shown in Fig. 1 and Fig. 2, respectively.

Fig. 1 the input fuzzy sets of Diff.

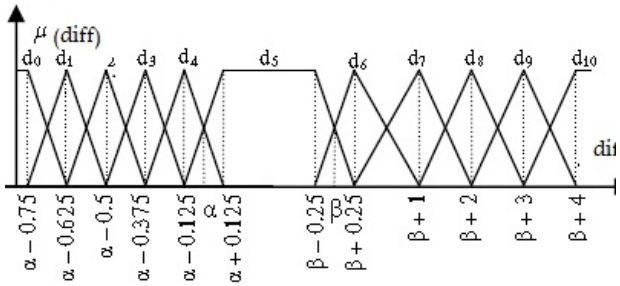
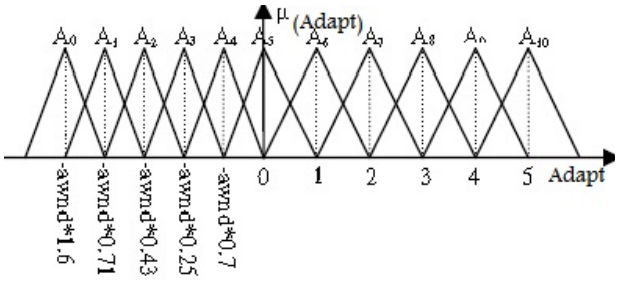


Fig. 2 the output fuzzy sets of Adapt



In Fig. 2, the output of fuzzy controller, Adapt is added to the current window size and the new window size is purposed.

$$\text{Adapt} = f(\text{diff}) = \frac{\sum_{l=0}^{10} \overline{A}^{l} . \mu_{d^{l}}(\text{diff})}{\sum_{l=0}^{10} \mu_{d^{l}}(\text{diff})} \quad (4)$$

Where, *l* is the rule number and A is the center of fuzzy set used in the part of l^{th} rule.

Using the singleton fuzzifier, suppose the inference engine to be provided with a precise and specific observation and center of average defuzzifier, the final output of fuzzy controller is calculated .

The intended fuzzy controller uses the following linguistic rules in its rule-base:

Rule 0: if diff is d0 then Adapt A10

Rule 1: if diff is d1 then Adapt A9

Rule 2: if diff is d2 then Adapt A8

Rule 3: if diff is d3 then Adapt A7

Rule 4: if diff is d4 then Adapt A6

Rule 5: if diff is d5 then Adapt A5

Rule 6: if diff is d6 then Adapt A4

Rule 7: if diff is d7 then Adapt A3

Rule 8: if diff is d8 then Adapt A2

Rule 9: if diff is d9 then Adapt A1

Rule 10: if diff is d10 then Adapt A0

V. Performance Evaluation

To evaluate the achievement of the intended fuzzy controller, it is coded in C++ and implemented in the Network Simulator (NS2) . In addition, different designs; simple and real AT&T design, are used to test the performance. In each designing, comparison is done considering throughput, congestion window, packet delay and packet loss.

A. program code

```
// calc actual and expect thruput diff, Adapt
int delta=int((expect-v_actual_)*v_baseRTT_+0.5);
int opt , i;
double up,den;
switch(opt)
{
case (1):
if (delta < (v_alpha_ - 0.75)) return 1;
if (delta >= (v_alpha_ - 0.625)) return 0;
return (-0.125 * delta) + (0.125 * v_alpha_) + 0.078125;
break;
case(2):
if (delta < (v_alpha_ - 0.75)) return 0;
if (delta >= (v_alpha_ - 0.5)) return 0;
if (delta < (v_alpha_ - 0.625)) return (0.125 * delta)-(0.125 *
v_alpha_ ) + 0.09375 ;
return (-0.125 * delta) + (0.125 * v_alpha_ ) - 0.0625 ;
break;
case(3):
if (delta < (v_alpha_ - 0.625)) return 0;
if (delta >= (v_alpha_ - 0.375)) return 0;
if (delta < (v_alpha_ - 0.5)) return (0.125 * delta)-(0.125 *
v_alpha_ ) + 0.078125 ;
return (-0.125 * delta) + (0.125 * v_alpha_ ) - 0.046875 ;
break;
case(4):
if (delta < (v_alpha_ - 0.5)) return 0;
if (delta <= (v_alpha_ - 0.125)) return 0;
if (delta < (v_alpha_ - 0.375)) return (0.125 * delta)-(0.125 *
v_alpha_ ) + 0.0625 ;
return (0.25 * delta) - (0.25 * v_alpha_ ) + 0.03125 ;
break;
case(5):
if (delta < (v_alpha_ - 0.375)) return 0;
if (delta >= (v_alpha_ + 0.125)) return 0;
if (delta < (v_alpha_ - 0.125)) return (0.25 * delta)-(0.25 *
v_alpha_ ) + 0.09375 ;
return (-0.25 * delta) + (0.25 * v_alpha_ ) + 0.03125 ;
break;
case(6):
if (delta < (v_alpha_ - 0. 125)) return 0;
if (delta >= (v_beta_ + 0.25)) return 0;
if ((v_alpha_ + 0.125< delta >v_beta_ - 0.25)) return 1 ;
if (delta > (v_beta_ - 0.25)) return (-0.5 * delta)+(0.5 *
v_beta_ ) + 0.125 ;
return (0.25 * delta) - (0.25 * v_alpha_ ) + 0.03125 ;
break;
case(7):
if (delta < (v_beta_ - 0.25)) return 0;
if (delta > (v_beta_ + 1.0)) return 0;
if ( delta < (v_beta_ + 0.25)) return (0.5 * delta)-(0.5 *
v_beta_) + 0.125 ;
return (-0.75 * delta) + (0.75 * v_beta_ ) + 0.75 ;
break;
case(8):
if (delta < (v_beta_ + 0.25)) return 0;
if (delta > (v_beta_ + 2.0)) return 0;
if (delta < (v_beta_ + 1.0)) return (0.75 * delta)-(0.75 *
v_beta_) + 0.1875 ;
return (-1.0 * delta) + v_beta_ + 2.0 ;
break;
case(9):
if (delta < (v_beta_ + 1.0)) return 0;
if (delta > (v_beta_ + 3.0)) return 0;
if (delta < (v_beta_ + 2.0)) return delta - v_beta_ - 1.0 ;
```

```
return (-1.0 * delta) +  v_beta_  + 3.0 ;
break;
case(10):
if (delta < (v_beta_+ 2.0)) return 0;
if (delta > (v_beta_+ 4.0)) return 0;
if (delta < (v_beta_+ 3.0)) return delta - v_beta_ - 2.0 ;
return (-1.0 * delta) +  v_beta_  + 4.0 ;
break;
case(11):
if (delta < (v_beta_+ 3.0)) return 0;
if (delta > (v_beta_+ 4.0)) return 1.0;
return  delta -  v_beta_  - 3.0 ;
break;
}
//fuzzy conclusion
switch(i)
{
case(1):up =up + delta*(1/cwnd_);
break;
case(2):up =up +delta*(1.5/cwnd_);
break;
case(3):up =up +delta*(2/cwnd_);
break;
case(4):up =up +delta*(2.5/cwnd_);
break;
case(5):up =up +delta*(3/cwnd_);
break;
case(6):up =up +delta*(-1/cwnd_);
break;
case(7):up =up +delta*(-1.5/cwnd_);
break;
case(8):up =up +delta*(-2/cwnd_);
break;
case(9):up =up +delta*(-2.5/cwnd_);
break;
case(10):up =up +delta*(-3/cwnd_);
break;
}
//defuzzification
if ((den==0)||(cwnd_<=1)){
            cwnd_=cwnd_+1;
} else if (cwnd_ <= ssthresh_){
/* slow-start (exponential) */
cwnd_ =cwnd_ + (up/den)*(2-(cwnd_/ssthresh_));
}else{
cwnd_=cwnd_+(up/den)*(ssthresh_/cwnd_);
//fuzzy congestion avoidance
}
```

B. Simple Design

In this section, a simple design, shown in Fig. 3, is used to discussion the network execution when applying the intended Fuzzy-controller.

Fig. 3 Simple Network.

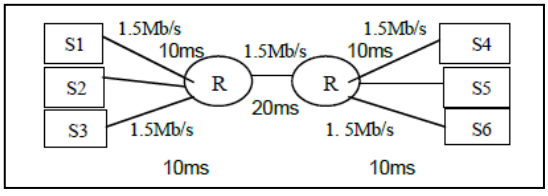


Fig. 4 Throughput versus simulation time.

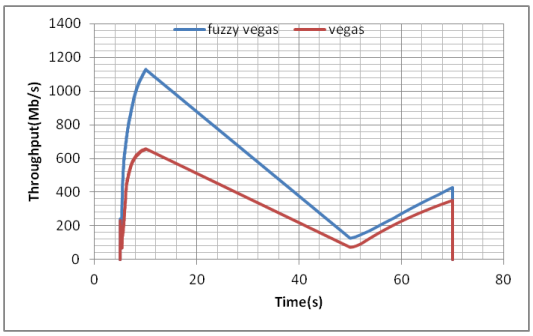


Fig. 5 cwnd versus simulation time.

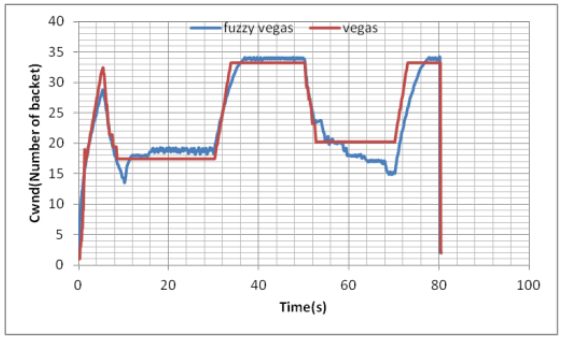


Fig. 6 the average delay versus simulation time.

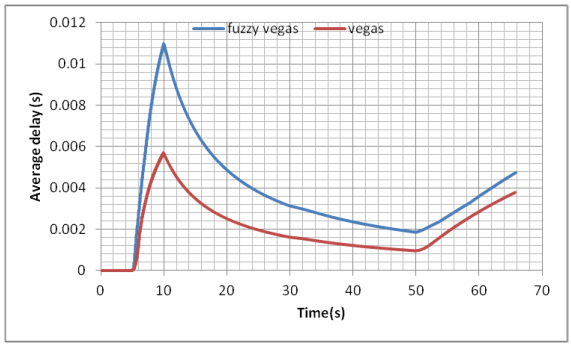


Fig. 4 shows the network throughput for the TCP-Vegas and TCP Fuzzy Vegas versus the simulation time while Fig. 5 shows the modification of cwnd versus the simulation time for both of them.

From the Fig. 4, the intended Fuzzy Vegas a chives a much better throughput than that of Vegas. This is because the intended Fuzzy Vegas efficiently use the available bandwidth at any time. Fig. 6 shows the average delay for the TCP-Vegas and TCP Fuzzy Vegas versus the simulation time.

Fig. 7 AT&T network topology

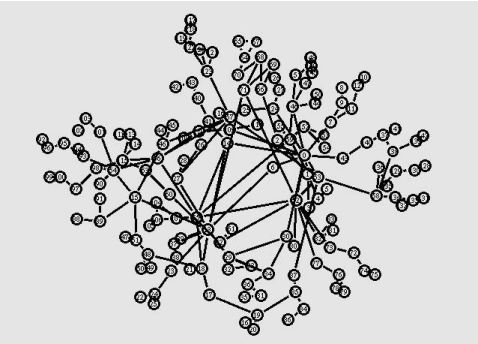


Fig. 8 Throughput versus simulation time.

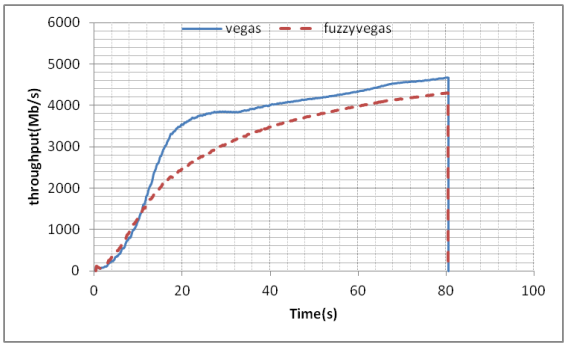


Fig. 9 cwnd versus simulation time.

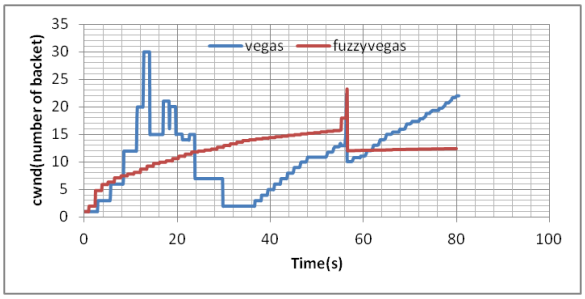


Fig. 10 average delay versus simulation time.

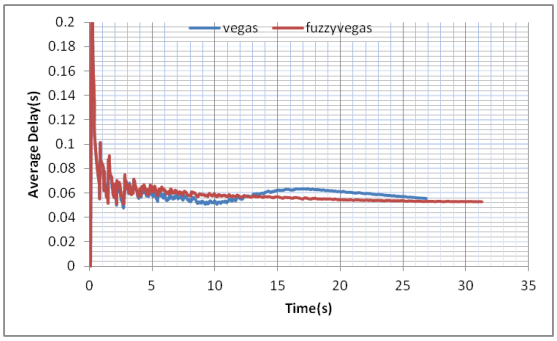


Fig. 11 average packet losses.

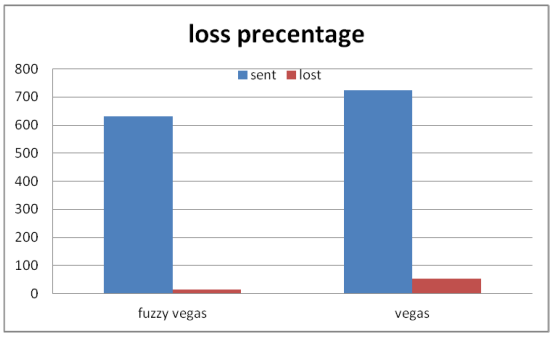
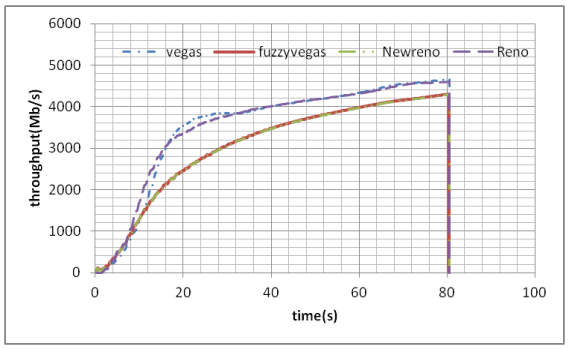


Fig. 12 Throughput versus simulation time.



C. Real Design

In this section, the AT&T real network, shown in Fig. 7, is used to discussion the inteded Fuzzy-controller. The design contains 166 nodes and 180 links. The simulation time is 80 seconds.

Fig . 8 and Fig . 9 show the throughput and the behavior of cwnd of the TCP Vegas and the intended Fuzzy Vegas. Fig. 10 shows the average delay for the TCP-Vegas and TCP Fuzzy Vegas versus the simulation time. Fig. 11 shows the average packet losses for the TCP-Vegas and TCP Fuzzy Vegas versus the simulation time.

Fig. 12 shows the measured throughput for the TCP Taheo, Reno, NewReno, Vegas and the intended TCP Fuzzy Vegas versus the simulation time.

VI. Conclusions

In this research work, an efficient fuzzy based TCP congestion controller is developed to avoid network collapse. The algorithm dynamically estimates the available network bandwidth at any time and adapts the sending rate at the TCP sender according to the changes in traffic load. The proposed mechanism efficiently utilizes available capacity of the network as it adapts the sending rate dynamically. The results indicate that the proposed controller real network is more effective than TCP Vegas real network.

References

[1] J. Postel, “Transmission Control Protocol,” IETF RFC 793, September 1981.
[2] J. Nagle, “Congestion control in IP/TCP Internetworks,” Request for Comments (RFC) 896, Internet Engineering Task Force, January 1984.
[3] V. Jacobson, “Congestion Avoidance and Control,” ACM SIGCOMM Computer Communication Review, Vol. 18, No. 4, pp. 314-329, August 1988.
[4] L. S. Brakmo, S. W. O’Malley and L. L. Peterson, “TCP Vegas: New Techniques for Congestion Detection and Avoidance” Proceedings of ACM SIGCOMM, London, August 31-September 2, pp. 24–35, 1994.
[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] U. Hengartner, J. Bolliger and Th. Gross, “TCP Vegas Revisited “ Proceedings of the IEEE Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2000), 2000.
[8] Richard J. La, Jean Walrand, and Venkat Anantharam,“ Analysis and comparison of TCP Reno and Vegas” 2001.
[9] P. Carbonell, Z. P. Jiang, and S. S. Panwar, “Fuzzy TCP: A Preliminary Study”, Proceedings of the 15th IFAC World Congress (IFAC 2002), Barcelona, Spain, July 21-26, 2002.
[10] Cheng P. Fu and Soung C. Liew, “A Remedy for Performance Degradation of TCP Vegas in Asymmetric Networks” IEEE COMMUNICATIONS LETTERS, VOL. 7, NO, 1, pp. 42-44, JANUARY 2003.
[11] H. Nejad, M. Yaghamaee, H. Tabatabaee “Modified Fuzzy TCP: Optimizing TCP congestion control”, IEEE 2006 .
[12] 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.
[13] Zainab T. Alisa and Sara Raad Qasim, “A Fuzzy based TCP Congestion Control for Wired Networks,” International Journal of Computer Applications (0975–8887), Vol. 89, No.4, pp. 36-42, March 2014. Article (CrossRef Link)
[14] Doaa H. Esawi, Gamal Attiya, Mohammad El-Bardini "Intelligent Controller based Host-to-Host Congestion Control: Fuzzy Logic Controller", 2019 7th International Japan- Africa Conference on Electronics, Communications, and Computations, (JAC-ECC), 2019.