CONGESTION CONTROL OF TCP-WESTWOOD USING AQM-PI CONTROLLER

Amirhossein Abolmasoumi[1], Saleh Sayyad Delshad[1] and Mohammadtaghi Hamidi Beheshti[1*]
[1] Control and Communication Networks Lab., Electrical Engineering Dept., Tarbiat Modares University, Tehran, Iran
a_masoumi@modares.ac.ir ,sayyaddelshad@gmail.com
mbehesht@modares.ac.ir

ABSTRACT

This paper concerns TCP Westwood (TCPW), a recently developed modification of TCP, in combination with random early detection (RED), PI and self-tuning PI queue management. Using the fluid-flow model of TCPW and separating the model dynamics through block diagram simplification, it is arrived at an approximate and simplified new model. Based on this approximate model, a PI-AQM (Active Queue Management) controller is designed. The parameters of considered PI controller are tuned through estimating the network parameters. Numerical simulations show that bandwidth utilization and the queue stabilization in the self-tuning PI is improved as compared with the PI and RED algorithms.

KEYWORDS

Congestion control, PI-AQM, TCP Westwood, PI controller, Self-tuning PI, RED

1. INTRODUCTION

Congestion usually occurs in computer networks routers. There are several reasons for its occurrence and propagation. [1] Investigates the cause and conditions of propagating the congestion in TCP networks. To overcome the problems caused by congestion, the TCP always uses a network congestion avoidance algorithm which includes various aspects of an additive-increase-multiplicative-decrease (AIMD) scheme, with other schemes such as slow-start in order to achieve congestion avoidance. Two such variations are those offered by TCP Tahoe [2] and Reno [3]. To Our best knowledge, many researchers have proposed the mathematical models describing the behavior of these TCP variations. For example, in [4], an analytical fluid flow model for Reno has been proposed and in [5] an AQM controller has been designed.

TCP-Westwood (TCPW) has been recently proposed as a source side modification of TCP [6-9]. The main target of TCPW is achieving larger portion of the window bandwidth. The main idea is to estimate, through the Acks stream, the bandwidth successfully obtained by the source. In fact, similar to two previous methods, here, the AIMD method in window management is used. In addition, when a packet loss occurs, the sender resets the congestion window and slow start threshold based on Rate Estimation (RE) value. In [9], a discrete time dynamic feedback model for a simple network with TCPW connections and random early detection (RED) gateway is proposed. Also the nonlinear dynamics of the TCPW/RED network are analyzed and its parameter sensitivities are studied. Moreover, in [10] the cause and conditions of propagating the congestion in TCP networks has been investigated.

In active queue management (AQM), by maintaining dropping/marking probabilities, the routers probabilistically drop or mark packets before the queue is full. In recent years, many AQM approaches have been reported in the literature. For example see [10-14]. In [15] several active queue management algorithms with respect to their abilities of

* Corresponding author

DOI : 10.5121/ijcnc.2011.3212

maintaining high resource utilization, identifying and restricting disproportionate bandwidth usage, and their deployment complexity are analyzed.

The main goal of this paper is to design an AQM controller for the router side of the TCP Westwood network so that it can overcome the queue variations and instability. Here, we use a proportional integral controller with adjustable gains where the considered gains can be updated to tolerate the variations in network parameters. As a result, the proposed controller will be able to overcome the problem of abrupt changes in network parameters. In [16] the behavior of the overall TCP Westwood and queue management system with RED algorithm as AQM has been analyzed. However, RED is a classic and simple algorithm and lacks the high performance provided by the proportional and Integral controllers (see [5]). Here, we simplify the TCP Westwood model through block diagram analysis, and then we design an AQM-PI controller to control the congestion in the network.

The paper is organized as follows. In section 2, dynamical model of TCPW will be studied. In section 3, "Windows Widening" and "Windows Narrowing" ratios will be introduced. We will also show the important role of these parameters on simplification and analytical study of TCP/Congestion dynamics. In section 4, the PI controller will be synthesized to effectively set the queue size at the desired level. Next, in the section 5, a self-tuning method is used to adjust the PI parameters in order to overcome the variations in network parameters (such as the network load and the link capacity). Finally, in the last section, NS simulations are performed on RED, PI and self-tuning PI algorithms. In comparison to other algorithms, it is shown that the self-tuning PI outperforms in adjusting the queue size as well as maximizing the link bandwidth utilization.

2. TCP WESTWOOD ALGORITHM AND ITS FLUID FLOW MODEL

TCPW is a source-side only modification of TCP New Reno. Similar to any other version of TCP, in TCP Westwood, the sender maintains a congestion window, limiting the total number of unacknowledged packets that may be in transit end-to-end. To avoid congestion collapse, TCP makes a Slow Start when the connection is initialized. It starts with a window of two packets. For every packet ACKed the congestion window is doubled. When the congestion window exceeds a threshold (*ssthresh)*, or a packet is lost, the algorithm enters a new state, called congestion avoidance. As long as non-duplicate ACKs are received, the congestion window is additively increased by one every round trip time [17].

In TCP Westwood, the sender computes the rate estimation (RE) value. RE is defined as a share of bottleneck bandwidth achieved by the connection. In other words, RE is the rate at which data is successfully delivered to the TCP receiver. More precisely, $cwin=RE \cdot RTT_{min}$, where RTT_{min} is the round trip propagation delay and *cwin* is the current window size. In [16], an analytical fluid flow model for TCPW has been proposed. Most previous network bottleneck dynamics are modeled with the variables, window size (w) and instantaneous queue size (q) (see [5]). In TCPW, *RE* is also added as a third variable. Dynamical equations describing the system are:

$$\begin{aligned} \dot{q} &= \frac{Nw}{R} - C \\ \dot{w} &= \frac{w_R(1-p)}{Rw} - \frac{w_R p_R}{R}(w - RE.d) \\ T.R\dot{E} + RE &= \frac{w_R(1-p)}{R} \end{aligned} \tag{1}$$

where N is the number of TCP sessions, R is the round trip time, C is the link capacity, p is the marking probability, d represents propagation delay and T is the time constant of the filter. Finally, w_R and p_R stand for values in one RTT before. Window dynamic is actually defined by:

$$\dot{w} = x_R \frac{1-p}{w} - x_R p\left(w - RE.d\right) \tag{2}$$

where $x=w/R$ is the average transmission rate of packets by the sender.

Comparing (2) with the equations given for Reno [5], it is turn out that additive increasing portion is unchanged from TCP-Reno while decreasing portion is changed. The additive term will increase the window size by $1/w$ for each positive Ack received and it is multiplied by the rate of such Acks ($x_R\left(1-p\right)$).

TCPW and Reno differ in decreasing term. For each loss, in TCPW, the window is set to $RE*d$ instead of dividing by 2. Equivalently, the window is decreased by $w - RE*d$ multiplied by $x_R p$ (the rate of losses). In the third equation of (1), RE is computed based on the rate of arriving packets to receiver and is smoothed by a low-pass filter. Senders are assumed to be the same in window size. Also, here, it is assumed that only a single link is congested ((1) is not valid for the case of multiple congested links). Figure (1) shows the network for which, (1) holds.

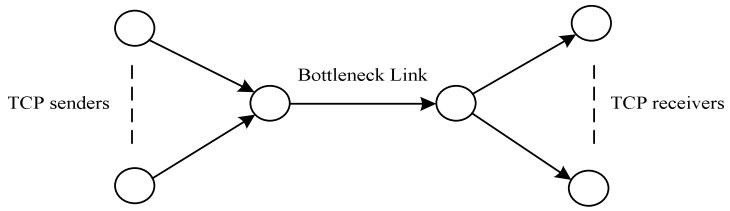


Figure 1. Network Topology in describing equations (3)

3. WINDOWS WIDENING RATIO- WINDOWS NARROWING RATIO

When the TCP sender receives three duplicate Acks, in Reno action, it halves the window and goes to recovery phase. So, we can define the "Windows Narrowing Ratio" in recovery phase to be 0.5. Also we define the value of "Windows Widening Ratio" to be 2. Widening ratio has an important role in the performance of TCP algorithm. With fixing the widening and narrowing ratio, adaptability for the sender side algorithm will decrease.

Windows widening ratio in TCPW is not a fixed value. Defining windows widening ratio, we get the block diagram shown in Figure (2), in which, window dynamics is separated from Rate Estimation (RE) and the queue dynamics. Let h and $1/h$ be defined as widening and narrowing ratios, respectively. According to Reno algorithm, h has the constant value of 2, while in TCPW it is a function of window size and RE. Assuming the marking probability p to be small and w_R/w equals to unity [5], windows dynamics in TCPW may be described by:

$$\dot{w} = \frac{1}{R} - \frac{w^2}{hR} p_R \tag{3}$$

Apart from h value, (3) is expectedly similar to the window dynamic in Reno. As it is mentioned, h is varying in TCPW while it is a constant in Reno. h is directly estimated using RE and window size.

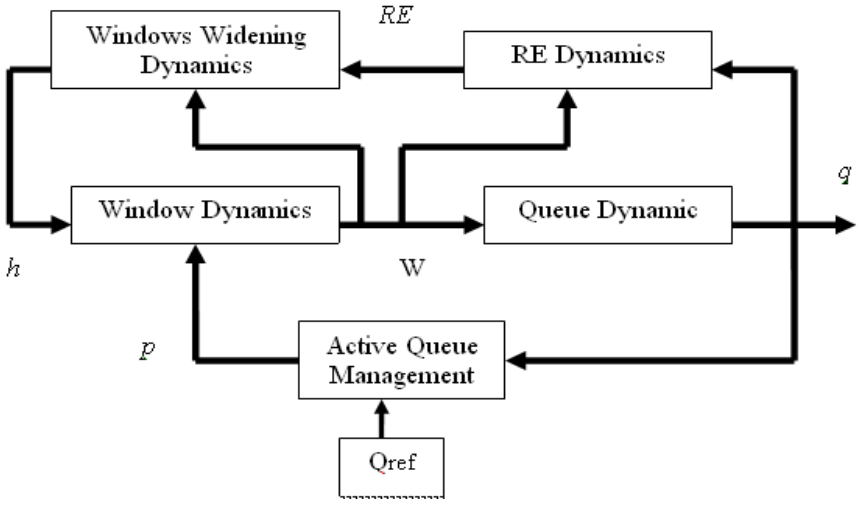


Figure 2. The TCPW congestion structure

In order to eliminate high frequency variations, the widening ratio is passed through a low-pass filter. Then, the dynamic of h is given by:

$$\dot{h} = -k_h h + k_h \frac{w}{w - RE.d} \tag{4}$$

in which k_h is the cut-off frequency of the filter. It is recommended that k_h be selected large enough to prevent over smoothing h. The complete system dynamics will be as follows:

$$\begin{aligned} \dot{q} &= \frac{Nw}{R} - C \\ \dot{w} &= \frac{1}{R} - \frac{w^2}{hR} p_R \\ R\dot{E} &= -\frac{1}{T} RE + \frac{1}{T} \frac{w}{R} \\ \dot{h} &= -k_h h + k_h \frac{w}{w - RE.d} \end{aligned} \tag{5}$$

Note that above equations are valid during the congestion avoidance phase, where the window size is increased each time an Ack is received in a RTT. The decreasing term appears as multiplicative factor similar to AIMD algorithm. Equations (5) are actually, the starting point in congestion control problem. By linearizing the nonlinear model around the equilibrium point as the following:

$$\dot{q},\dot{w},\dot{RE},\dot{h}=0\Rightarrow\begin{cases}w_0=\dfrac{R_0C}{N}=\sqrt{\dfrac{h_0}{p_0}}\\ RE_0=\dfrac{C}{N}\\ h_0=\dfrac{R_0}{R_0-d}\\ R_0=\dfrac{q_0}{C}+d\end{cases}\tag{6}$$

we get the linear model:

$$\begin{cases}\delta\dot{w}=-\dfrac{2N}{R_0^2C}\delta w-\dfrac{R_0C^2}{h_0N^2}\delta p(t-R_0)+\dfrac{1}{h_0R_0}\delta h\\ \delta\dot{q}=\dfrac{N}{R_0}\delta w-\dfrac{1}{R_0}\delta q\\ \delta\dot{RE}=-\dfrac{1}{T}\delta RE+\dfrac{1}{TR_0}\delta w-\dfrac{1}{TR_0N}\delta q\\ \delta\dot{h}=-k_h\delta h-\dfrac{k_hNd}{C\left(R_0-d\right)^2}\delta w+\dfrac{k_hNR_0d}{C\left(R_0-d\right)^2}\delta RE\end{cases}\tag{7}$$

where, $\delta(.)$ is the deviation of variable (.) from the equilibrium point. Figure (3) shows the block diagram of linearized model in which dynamics of each variable is separated. From Figure (3) it can be understood that in addition to dynamics of instantaneous queue size and window size, the poles *1/T* and k_h are affecting the behavior of the system. Also, the effect of h_0 parameter in the forwarding transfer function should be noted.

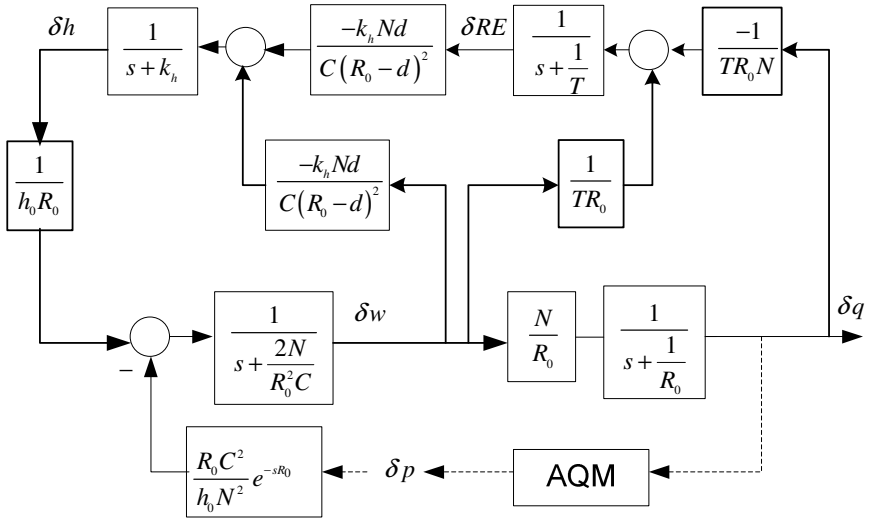


Figure 3. System Block Diagram

4. AQM-PI ALGORITHM

PI structures have been previously suggested in the literature; (see [5]). Because of their generality and simplicity in design, simple PI controllers seem to be an appropriate candidate for active queue management in network routers. The input-output relationship of PI controller in Laplace domain is as follows:

$$C(s) = K_p\left(1 + \frac{1}{T_i s}\right) \tag{8}$$

As shown in Figure (4), the congestion control problem is a feedback control problem. Replacing AQM transfer function by *C(S)* and after some simplifications, we get the block diagram in Figure (4).

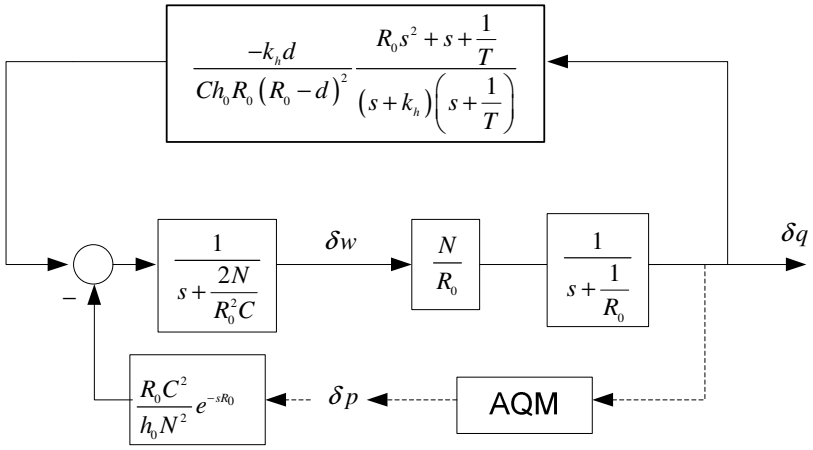


Figure 4: Simplified System Block Diagram

Figures (3), (4) show that the feedback from *q* to *w* is not entirely through the AQM algorithm and that parameters *RE* and *h* have contributions to this feedback as well. With appropriately selecting poles in the upper branch of Figure (4) and considering k_r (in (9)) as a small gain, we can neglect the effect of upper branch feedback, resulting in familiar Reno diagram.(See Figure (5)).

$$k_r = \frac{-k_h d}{Ch_0 R_0 (R_0 - d)^2} \tag{9}$$

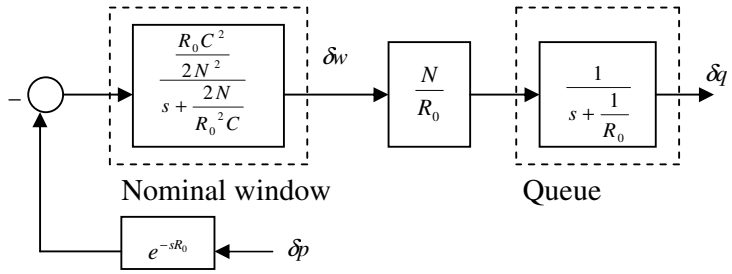


Figure 5: System Block Diagram Using TCP-Reno

The only difference between TCPReno/AQM and TCPW/AQM diagram is parameter h_0 in the feedforward transfer function. There are two ways of adjusting PI parameters. In the first approach, the upper branch needs not to be omitted. The diagram of Figure (4) may be simulated by an appropriate simulator. Ziegler-Nichols algorithm [19] can provide acceptable PI parameter values through this method, the control parameters are given by:

$$K_p = 0.45K_{cr}$$
$$T_i = \frac{1}{1.2}P_{cr} \tag{10}$$

K_{cr} is critical state gain and P_{cr} is the corresponding time period. This is essentially an empirical approach for setting the controller parameters and requires correct and precise system response, while bearing Ziegler-Nichols deficiencies [19].

The second approach is to use Reno design. PI parameters are computed by:

$$K_p = h_0\beta_{pi}\sqrt{\beta_{pi}^2+1}\frac{N}{R_0^2C^2}$$
$$K_i = h_0\frac{N}{R_0^2C}K_p \tag{11}$$

in which β_{pi} is responsiveness factor selected by designer. ($0< \beta_{pi}<0.85$) [5]. Validity of the parameters given by (11) can be explained by the system diagram of Figure (4), through which the open loop characteristic equation is obtained as:

$$\left(s+\frac{2N}{{R_0}^2C}\right)\left(s+\frac{1}{R_0}\right)(s+k_h)\left(s+\frac{1}{T}\right)-K_r\left(R_0s^2+s+\frac{1}{T}\right)=0 \tag{12}$$

Note that we can choose the poles of estimation dynamics (i.e. *h* and *RE* estimation) such that it will simplify the problem. As mentioned, k_r is a small coefficient when the network is in a normal condition. We assume this coefficient to be negligible. In this case the low frequency open loop system poles are the four poles of window dynamic, queue dynamic, *h* estimation and *RE* estimation. Then we can choose the poles k_h and *1/T* large enough so that two other poles of window and queue dynamics have dominant behavior. We can essentially assume these dominant poles to be the open loop poles and neglect the effect of k_h and *1/T*.

There is an important problem in this approximation that should be noted. The coefficient k_r is affected by the pole k_h. If we choose k_h too large to minimize its effect on open loop behavior, then k_r will increase, too. As result, our approximation will not be precise. Therefore, we should select it large enough to limit its effect on open loop behavior of the system, but not too large that it will undermine our approximation. By applying the mentioned modifications, Figure (4) will become very similar to Figure (5). These two block diagrams just differ in *2/h* coefficient in the forward transfer function. To solve this problem, we can simply transfer its effect to the AQM controller transfer function.

5. SELF-TUNING ALGORITHM

AQM behavior is influenced by variations in main network parameters such as link capacity and number of TCP sessions. Generally, these parameters do not have static values, but in some conditions, it is possible to assume their variations negligible. Most of AQM methods are designed for networks with limited parameter variations.

The need for adaptability of AQM controller is completely considerable since in some conditions it is possible to lose stable control action because of the changes in network

parameters. For instance, unresponsive traffics like UDP or short lived TCP flows could be mentioned. These traffics change the link capacity experienced by long lived flows. However, the physical capacity of link may be fixed like a pipe. But actually it is divided to many virtual links by different mechanisms. In this case, the capacity of virtual links will be variable and the assumption of fixed link capacity will not be valid anymore. It is also true about TCP load; the assumption of static value for the number of TCP sessions will be incorrect, too. Changes in the number of TCP sessions cause deterioration of AQM performance.Without online parameters updating, the controller has to be designed in a worst case scenario leading to low quality performance and weak adaptability. So, the online estimation of network parameters will be necessary.

In this section, we will perform continuous adjusting action for existing AQM design. This is the concept of "Self-tuning AQM". The actions to be done in self tuning algorithm are: 1- parameter estimation (Network parameters are *R*, *N* and *C*) and 2- AQM controller adjusting. For actual design of self tuning AQM it will be used of "effective RTT" [20] calculated by different mechanisms. Link capacity is obtained directly by keeping track of departed packets. Also, term *N/R* could be estimated by measuring packet marking probability *p*. From (5) we have:

$$\dot{w} = \frac{1}{R} - \frac{w^2}{hR} p_R$$

and in equilibrium point we have:

$$\left.\begin{aligned} 0 &= \frac{1}{R} - \frac{{w_0}^2}{h_0 R} p_0 \\ 0 &= \frac{N w_0}{R} - C \end{aligned}\right\} \Rightarrow \sqrt{\frac{p}{h_0}} = \frac{N}{RC} \qquad (13)$$

Equation (13) is related to TCPW algorithm and is independent of AQM control rule. However, (13) is derived in steady state, we use it to estimate *N/RC* in transient state. In fact, estimated terms are *C* and *N/RC*. In [20, 21] two methods of designing adaptive algorithm for congestion control have been proposed. In previous section PI coefficients were computed. Using these coefficients and the simplified model of TCPW, the design of PI controller could be improved to self-tuning scheme.

As the first step, AQM dynamics should be expressed in terms of network variables. Then a method for continuous estimation of parameters and a rule for automatic adjusting of controller should be given. We use "certainty equivalence principle″ [22] for this purpose. In Figure (6), the block diagram of self-Tuning adaptive structure is shown. The AQM-PI controller formula is in the form of (8) in which K_p and K_i are obtained from (11) ($K_i=K_p/T_i$). Also, β_{pi} is the AQM responsiveness factor determined by the designer ($0<\beta_{pi}<0.85$) [5].

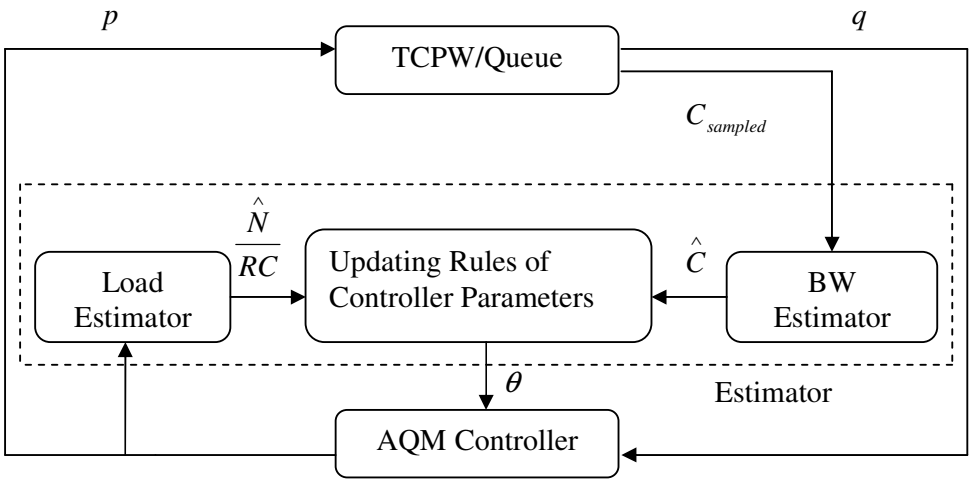


Figure 6. Block diagram of self-Tuning adaptive structure

5.1. Link Capacity Estimation

To estimate actual link capacity, we periodically compute the ratio of departed packets to time period of sampling and to smooth high frequency transient variations we use a low-pass filter with cut-off frequency K_c as the following:

$$\dot{\theta}_c = -K_c\theta_c + K_c\hat{C} \tag{14}$$

5.2. Round Trip Time Estimation

In the case of small queuing delay, the propagation time could be an estimation of RTT. We could use it as the "effective" RTT or an estimate of RTT made from samples of the SYN packets. (see [20]).

5.3. Flow Number (TCP Load) Estimation

Equation (13) is used to estimate *N/RC* in the TCPW/AQM system. Determining *C* and *N/RC* is sufficient for adjusting PI controller. Estimation and smoothing the *N/RC* parameter will be as follows (see [20]):

$$\dot{\theta}^n_{rc} = -K^n_{rc}\dot{\theta}^n_{rc} + K^n_{rc}\sqrt{\frac{p}{h}} \tag{15}$$

where in (15), K^n_{rc} is the cut-off frequency of low-pass filter. By determining θ_c and θ^n_{rc} through equations (14) and (15), controller coefficients are computed as below:

$$K_p^{\theta} = h\beta_{pi}\sqrt{\beta_{pi}^{2}+1}\,\frac{\theta_{rc}^{n}}{R\theta_C}$$
$$K_i^{\theta} = h\frac{\theta_{rc}^{n}}{R}K_i^{\theta} \tag{16}$$
$$p = K_i\int\left(q-q_{ref}\right)dt + K_p\left(q-q_{ref}\right)$$

The overall system equations are rewritten as follows:

$$\begin{cases} \dot{w} = \dfrac{1}{R} - \dfrac{w^2}{hR}p_R \\ \dot{q} = \dfrac{Nw}{R} - C \\ R\dot{E} = -\dfrac{1}{T}RE + \dfrac{1}{T}\dfrac{w}{R} \\ \dot{h} = -K_h h + K_h\dfrac{w}{w-RE.d} \\ \dot{\theta}_{rc}^{n} = -K_{rc}^{n}\theta_{rc}^{n} + K_{rc}^{n}\sqrt{\dfrac{p}{h}} \\ \dot{\theta}_C = -K_C\theta_C + K_C\hat{C} \\ p = K_i\int\left(q-q_{ref}\right)dt + K_p\left(q-q_{ref}\right) \end{cases} \tag{17}$$

To guarantee the overall stability of (17), the time constant for the self-tuning algorithm should be large enough as compared with the time for AQM response [20].

6. SIMULATION RESULTS: RED, PI AND SELF-TUNING PI SIMULATION

In order to investigate the effectiveness of the proposed algorithm, we will use NS2 which is custom in the network simulation. It is assigned ftp type traffic to 60 source nodes. Time intervals between file transfers are chosen to be NS random. Bottleneck link bandwidth is 0.5Mbps and the other bandwidths are 1000Mbps. Reference queue size for PI and Self-tuning PI methods is 175 packets. Propagation delay of link in which AQM is done is 70ms and other links have the delay of 20 ms. Buffer size of the router and average packet size are limited to 800 packets and 500 bytes, respectively. In other links drop-tail is used as queue control algorithm. For the RED algorithm *gentle* property [23] is active. *minth* and *maxth* will be 150 and 200 packets and other RED parameters are set to be the *default* values in NS. The ideal performance of each method is when the queue length is fixed and the maximum available bandwidth is achieved. The fluctuation in queue available bandwidth is also undesirable. Consider three different scenarios defined as follows:

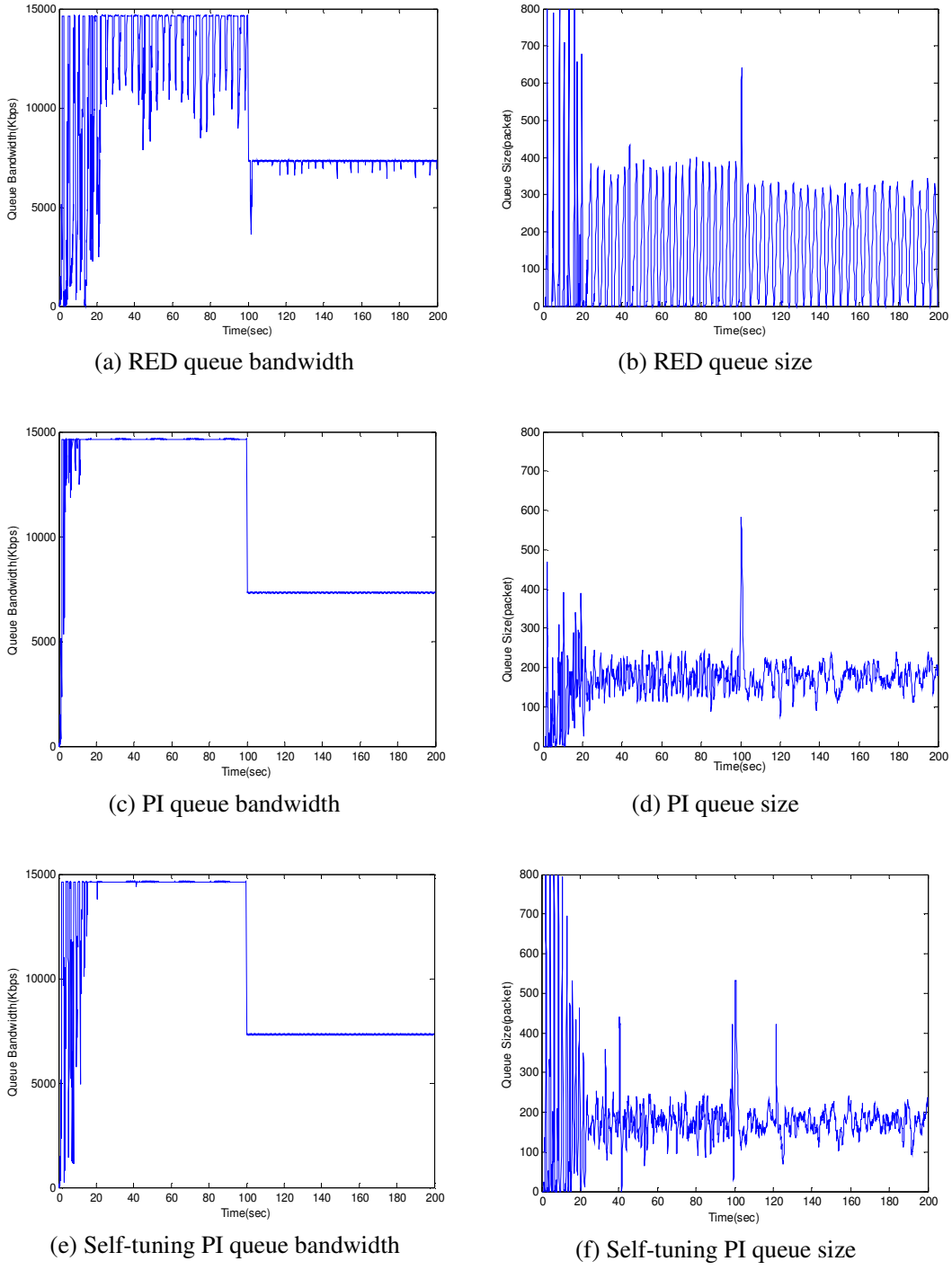
a) Change in Link Capacity

The initial capacity of the bottleneck link is 3750 packets per second (equals to 15Mbps). PI parameters will be adjusted to keep the queue size at 175 packets. The simulation time is 200 sec. We run two experiments: first, at time *t=100sec* we change the capacity to 90 Mbps and second, the capacity is set to 7.5 Mbps. Figures (7) and (8) show queue size variations and output queue bandwidth variations in the bottlenecked router. Superior performance of Self-tuning PI as compared with rather weak performance of RED in achieving bandwidth and queue size control is shown in Figure (7) and (8). In the first case, when we change the link capacity to 7.5 Mbps, PI and Self-tuning PI act well. But in the second experiment (link capacity set to 90

Mbps) only Self-tuning PI can keep the queue size at desired level and utilizes the available resource in good manner.

b) Change in TCP load (flow numbers or TCP sessions)

In this experiment, the link capacity will be kept fixed at 15Mbps and at time *t=100*s the number of flows changes from 60 to 30. Figure (9) shows the performance of the three methods in this situation.



(a) RED queue bandwidth

(b) RED queue size

(c) PI queue bandwidth

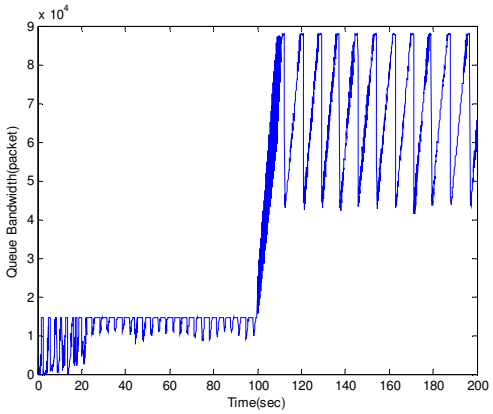
(d) PI queue size

(e) Self-tuning PI queue bandwidth

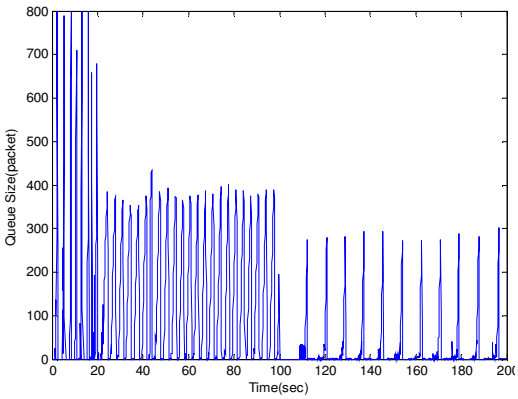
(f) Self-tuning PI queue size

Figure 7. Queue size and bandwidth variations

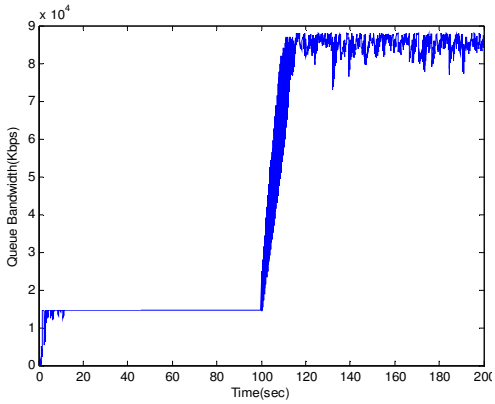
(At time *t=100*s the link capacity is changed to 7.5Mbps)



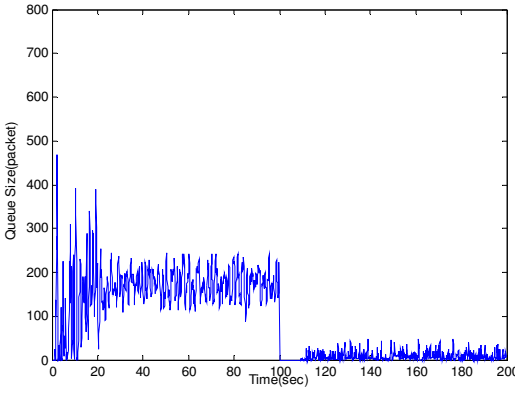
(a) RED queue bandwidth



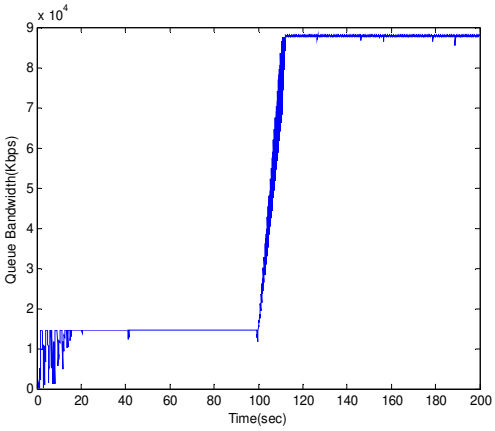
(b) RED queue size



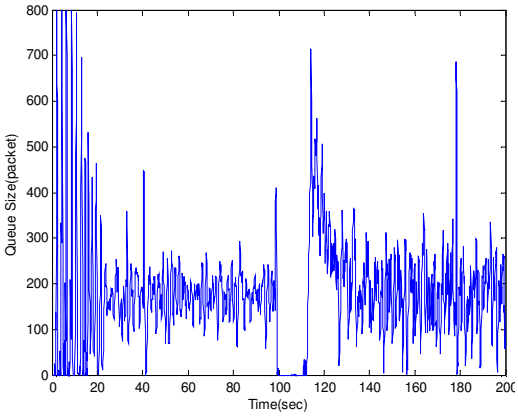
(c) PI queue bandwidth



(d) PI queue size



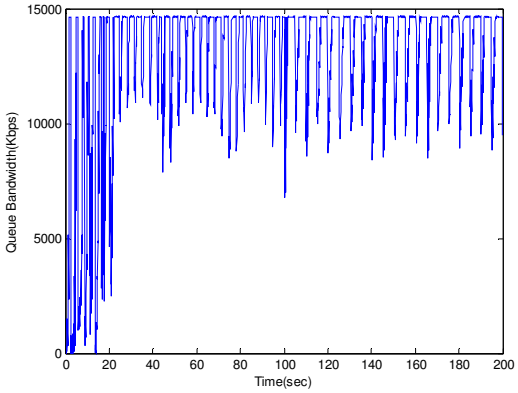
(e) Self-tuning PI queue bandwidth



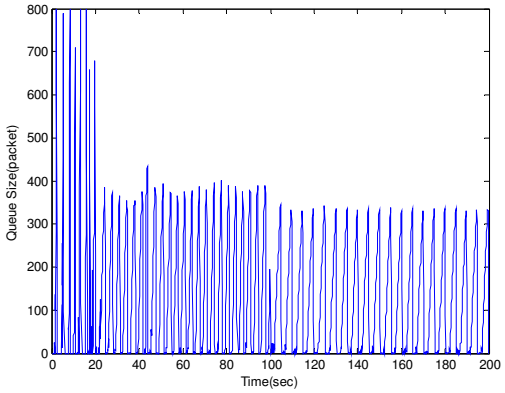
(f) Self-tuning PI queue size

Figure 8. Queue size and bandwidth variations
(At time *t=100*s the link capacity has changed to 90Mbps)

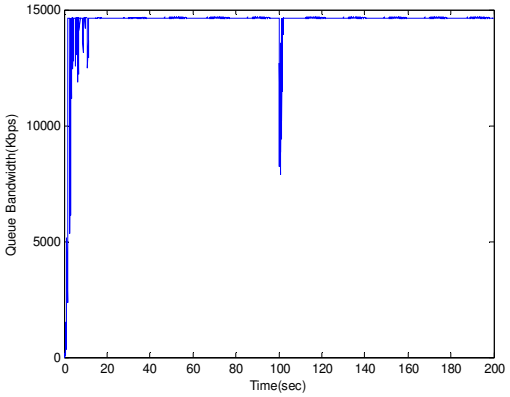
Figure (9) shows again the better performance of PI and Self-tuning PI as compared with the poor performance of RED in the case of TCP load variation. The load variation cause that the predefined parameters of RED and PI could adjust the queue and achieve the bandwidth. It can also be seen from Figure (9) that the PI performance is still better than RED.



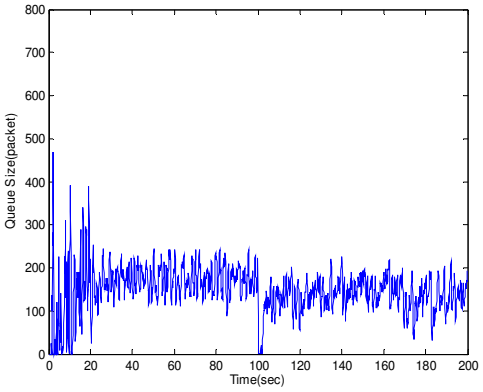
(a) RED queue bandwidth



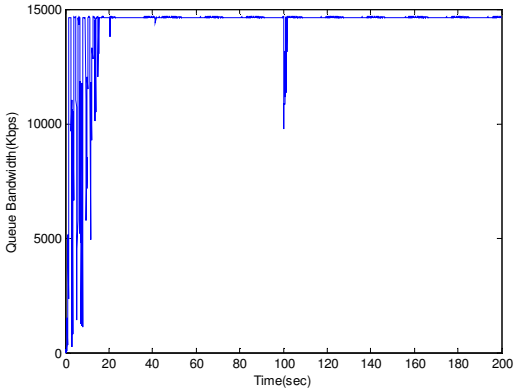
(b) RED queue size



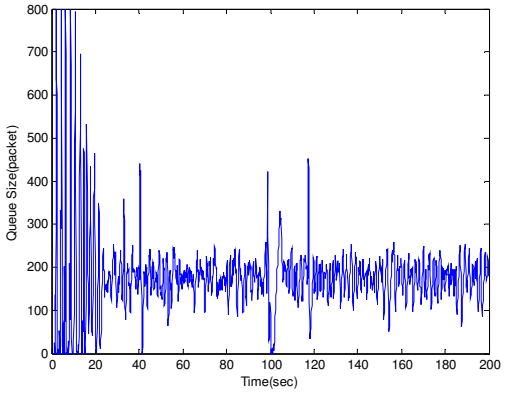
(c) PI queue bandwidth



(d) PI queue size



(e) Self-tuning PI queue bandwidth

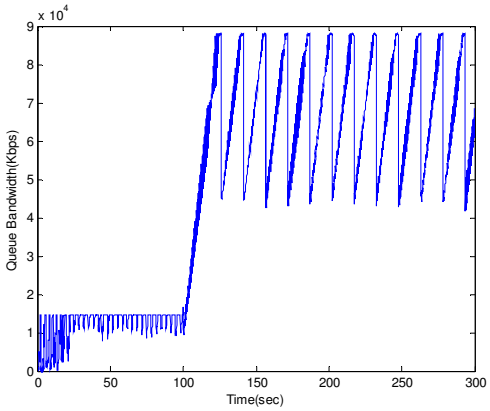


(f) Self-tuning PI queue size

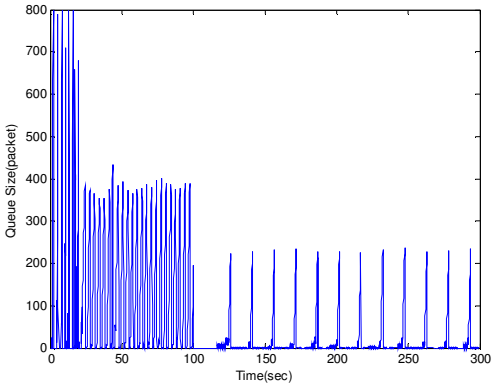
Figure 9. Queue size and bandwidth
(At time 100s TCP flow number has changed from 60 to 30)

c) Link Capacity and Flow Number both Changed

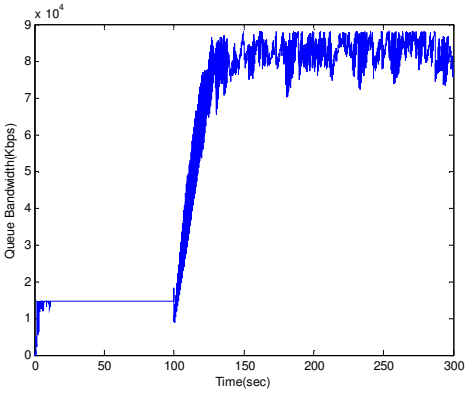
Here, both of the link capacity and flow number are changing. At time t=100s link capacity is changed to 90Mbps and number of flows is set to 30. Figure (10) shows the queue size and bandwidth variations for the three algorithms.



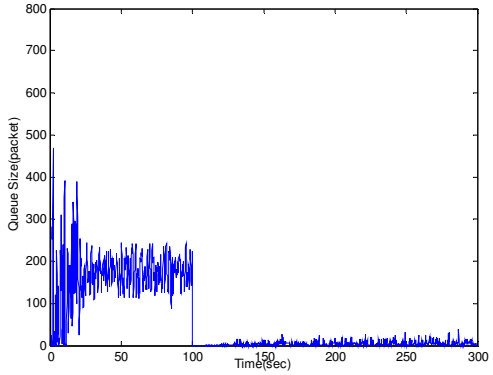
(a) RED queue bandwidth



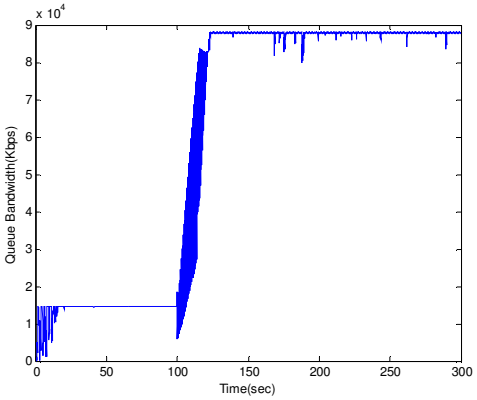
(b) RED queue size



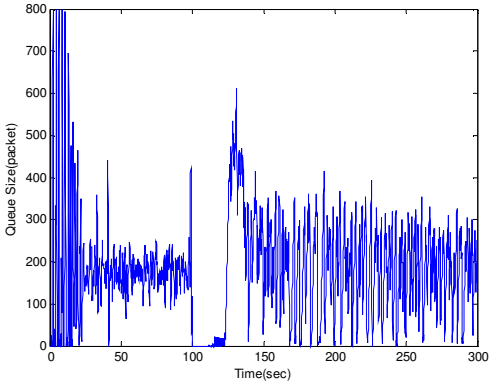
(c) PI queue bandwidth



(d) PI queue size



(e) Self-tuning PI queue bandwidth



(f) Self-tuning PI queue size

Figure 10. Queue size and bandwidth
(At time *t=100*s TCP flows number is set to 30 and link capacity is changed to 90 Mbps)

As seen in Figure (10), when there are drastic changes in network parameters, only self-tuning algorithm can almost keep the queue size at desired level and utilizes the existing bandwidth. In this situation, PI and RED totally loose the control of the queue length. As results, they cannot effectively achieve the available bandwidth. In fact, Figure (10) clearly shows the effect of self-tuning property on the performance of the AQM methods. Also, as Figure (10) shows, PI acts better than RED. It is because the PI gains design method is rather robust in comparison with that of RED.

7. CONCLUSION

In this paper, we proposed an effective approach to simplify the TCPW model and tuning the AQM/PI controller. To this end, first two parameters window widening/narrowing ratios were defined. Then TCPW dynamics were separated and an effective approach to easily determine the PI controller coefficients was presented. Also, an adaptive self-tuning approach was introduced to automatically adjust the PI parameters due to changes in network conditions. It was shown that PI and self-tuning PI have a superior performance in comparison with weak performance of RED in achieving bandwidth and queue size control. The NS results also clarified that in drastic changes in network parameters it is just self-tuning PI which can fix the queue size at desired level and efficiently utilize the available bandwidth.

REFERENCES

[1] Ohsaki H., Sugiyama K. and Imase M., *Congestion Propagation among Routers with TCP Flows*, International Journal of Computer Networks & Communications, vol. 1, no. 2, pp. 112-127.

[2] Jacobson V., *Congestion avoidance and control*, proceeding of SIGCOM88, ACM, Agust 1988, pp 314-329.

[3] Jacobson V., *Modified TCP congestion avoidance algorithm,*ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail, April 1990.

[4] Misra V., Gong W. B., and Towsley D., *Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED,* In Proc. ACM/SIGCOMM, 2000, pp 151-160

[5] Hollot C. V., Misra V., Towsley D. and Gong W., *Analysis and Design of Controllers for AQM Routers Supporting TCP Flows*, IEEE Transactions on Automatic Control, Vol 47, June 2002, pp 945-959.

[6] Wang R., Valla M., Sanadidi M.Y. and Gerla M., *Adaptive bandwidth share estimation in TCP Westwood*, in: Proceedings of IEEE Globecom, Taipei, 2002

[7] Wang R., Valla M., Sanadidi M.Y., Ng B.and Gerla M., Efficiency*/friendliness tradeoffs in TCP Westwood*, In: IEEE Symposium on Computers and Communications,Taormina, Italy, July 2002.

[8] Mascolo S., Casetti C., Gerla M., Sanadidi M.Y. and Wang R., *TCP Westwood: bandwidth estimation for enhanced transport over wireless links*, In: Proceedings of Mobicom, 2001.

[9] D. Ding, J. Zhu, X. Luo, L. Hung and Y. Hu, *Nonlinear dynamics in Internet congestion control model with TCP Westwood under RED,* Journal of China Universities of Posts and Telecommunications, pp. 53-58, Aug. 2009.

[10] F. Ren, C. Lin and X. Yin, *Design a congestion controller based on sliding mode variable structure control, Computer* Communications, vol. 28, pp. 1050-1061, 2005.

[11] J. Wang, L. Rong and Y. Liu, *Design of a stabilizing AQM controller for large-delay networks based on internal model control,* Computer Communications, vol. 31, pp. 1911-1918, 2008.

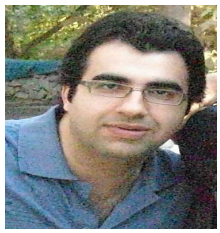
[12] J. Aweya, M. Ouelette, D. Y. Montuno and K. Felske, *Design of rate-based controllers for active queue management in TCP/IP networks, Computer* Communications, vol. 31, pp. 3344-3359, 2008.

[13] S. M. Alavi and M. J. Haeri, *Robust active queue management design: A loop-shaping approach,* Computer Communications, vol. 32, pp. 324-331, 2009.
[14] B. Marami, M. Haeri, *Implementation of MPC as an AQM controller,* Computer Communications, vol. 33, pp. 227-239, 2010.
[15] Ali Ahmad G.F., Banu R., *Analyzing the performance of Active Queue Management algorithms,* International Journal of Computer Networks & Communications, Vol. 2, No. 2, March 2010.
[16] Chen J., Paganini F., Sanadidi M.Y., Wang R. and Gerla M., *Fluid-flow analysis of TCP Westwood with RED*, Computer Networks, April 2005.
[17] From Wikipedia, the free encyclopedia, *TCP congestion avoidance algorithm, http://en.wikipedia.org/wiki/TCP_congestion_avoidance_algorithm.*
[18] Floyd S. and Jacobson V., *Random early detection gateways for congestion avoidance,* IEEE/ACM Trans. Networking, vol. 1, pp. 397–413,Aug. 1993
[19] Ogata K., *Modern control engineering*, 3rd Ed. 1997.
[20] H. Zhang, C. V. Hollot, D. Towsley and V. Misra, *A Self-Tuning Structure for Adaptation in TCP/AQM Networks*, IEEE GLOBECOM 2003.
[21] S. Kunniyur and R. Srikant. *Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management*, in Proc. of ACM SIGCOMM 2001, Aug. 2001.
[22] Astrom K. J., Wittenmark B., Adaptive control, Addison-Wesley, 2nd Ed.
[23] Floyd S. , *Recommendation on using 'gentle_' variant of RED,* IEEE/ACM Trans. Networking, Mar 2002.

Authors

Amir Hossein Abolmasoumi received his B.S. degree in Electrical Engineering from University of Tehran in 2004 and his M.S. in Control Engineering from Tarbiat Modares University of Tehran in 2007. He is now the Ph.d. student in Control Engineering in Tarbiat Modares University. His research topics include stochastic switched systems, traffic control and management in computer networks.

Mohammad T.H. Beheshti received his B.S. degree in Electrical Engineering from University of Nebraska, Lincoln in 1984 and his M.S. and PhD in Electrical Engineering from Wichita State University, Wichita, KS. in 1987 and 1992 respectively. He is currently with the department of ECE at Tarbiat Modares University, Tehran, Iran. His research interests are robust optimal control of singularly perturbed systems and quality of service of communication systems.



Saleh Sayyad Delshad received the B.S. degree in Electrical Engineering from Islamic Azad University in 2006 and he got his M.Sc. in Control Engineering from Tarbiat Modares University, Iran in 2010. His research areas include nonlinear control, robust control and applications of fractional calculus in engineering.