A NEW METHOD TO IMPLEMENT AF_{ij} AT CORE ROUTERS IN DIFFSERV NETWORKS

LE HUU LAP[1], VU NHU LAN [2], NGUYEN HONG SON[3]

[1] *Vice Director of PTIT, Hanoi, Vietnam*

[2] *Institute of Information Technology Hanoi, Vietnam*

[3] *Faculty of Information Technology II PTIT Ho Chi Minh city branch, Vietnam*

Abstract. Nowadays, the DiffServ architecture is considered the most prevalent solution of QoS provision in IP networks. What is the best way to implement this architecture is still a question without answer. The most complicated part in DiffServ implementation belongs to $AFBHP$. Indeed, the complication of AF comes not only from its architecture, but also from its goals. This paper will propose a method to implement AF_{ij} of $AFBHP$ at core routers in DiffServ networks. Using this method, we can make AF subclasses (AF_{ij}) easily. Moreover, the performance of the system will be better since the mechanism has congestion control ability and fair sharing between subclasses in an AF class.

Tóm tắt. Hiện nay, kiến trúc dịch vụ khác biệt được xem như một giải pháp đảm bảo chất lượng dịch vụ trên mạng Internet. Cách thực thi tốt nhất kiến trúc kiểu này còn là một vấn đề để ngỏ. Thành phần phức tạp nhất trong việc thực thi dịch vụ khác biệt thuộc về $AFBHP$. Tính phức tạp của AF không chỉ là vấn đề kiến trúc mà còn là mục tiêu của nó. Bài báo đề xuất một phương pháp để thực thi AF_{ij} của $AFBHP$ ở trong lõi bộ định tuyến trong mạng dịch vụ khác biệt. Sử dụng phương pháp này, ta có thể chia lớp AF thành các lớp con AF_{ij} một cách dễ dàng. Tuy nhiên, hiệu năng của hệ thống sẽ tốt hơn khi cơ chế có khả năng điều khiển tắc nghẽn và chia sẻ công bằng giữa các lớp con trong lớp AF.

1. INTRODUCTION

The DiffServ architecture is based on a network model implemented over a complete Autonomous System (AS) or domain. All traffic entering and flowing through the domain are managed by clear and consistent rules. Depending on [2], IP packets entering network at edge routers are classified and aggregated into different groups called Behavior Aggregates (BAs). Inside the domain, packets belonging to the same BA are forwarded according to previously established rules. Like this, the real work is creating classes of flows that travel through networks. Each flow is treated along the domain according to the class to which it belongs. Thus, the treatment of routers for classes is also established and takes forms called Per Hop Behaviors (PHB). PHB are implemented by allocating resources quite differently inside routers. Currently, only two PHB are adopted and deployed, they are $EFPHB$ and $AFBHP$. AF PHB is more intricate than $EFPHB$. Indeed, [3] specifies that AF PHB has four classes, called AF classes. Each AF class is further divided into three priority levels, which will be called AF subclasses and denoted AF_{ij}. PHBs and AF classes are implemented by assigning resources of the system, such as buffer, bandwidth, service time, etc. There are numerous proposals and evaluations on implementing PHB for IP DiffServ Networks by various authors. For example, in [4] authors implemented $EFPHB$ by using CBQ (Class Based Queuing) and proved that it is more efficient than priority scheduling or WRR (Weighted Round Robin) proposed by Christian Worm Mortensen. Some others, such as in [5] also implemented $AFPHB$ by using CBQ, just called AFCBQ. Recently, in [6] used HTB (Hierarchical Token Bucket) proposed by Martin Devera to implement AF, and named AFHTB. [6] also used the packet dropping mechanism of RED for implementing AF subclasses. By this way, the packet dropping probability depends on the average queue length, q_{ave}, and with determined maximum threshold/minimum threshold, the higher q_{ave} gets, the bigger the probability is. So, adding various values into q_{ave} will make different behaviors of the queue, which are truly AF subclasses. However, the method depends much on RED. It is nearly impossible to select such RED parameters that the impact on network performance and on stability of the system doesn't get worse. The proposed method in this paper bases on a controllable queue management (CQM) scheme instead of using RED, an active queue management (AQM) algorithm. The essence of the method is a mechanism, which includes controllers and token buckets as proposed in [1]. This mechanism also uses packet discarded levels to make difference of services (drop precedence) but these time token buckets are responsible for dropping packets in urgent situations.

As mentioned above, the paper will focus on implementing AF subclasses in each AF class. The implementation not only satisfies the requirements of AF class specification, but also uses maximum capacity of the system without congestion and shares allocated resources of the system between subclasses in each AF class.

The rest of the paper is structured as follows. In section II, we present the principle of implementation and operation inside the mechanism. This section not only explains how to make various drop precedences (AF_{ij}), but also guides to share resources between them. In the next section, we clarify the method by establishing the dynamics of the mechanism. In section IV, we simply validate the proposed method by a computer simulation. The results of simulation also show that the system is always protected from congestion and can obtain the highest utility. In the final section, we present our conclusions and mention some relative issues for the future work.

2. THE PRINCIPLE OF IMPLEMENTATION AND OPERATION

The mechanism applies the results from [1], and the principle scheme is shown in Figure 1.

The mechanism has two separate modes, namely the free mode and the constrained mode. The free mode is corresponding to switches K at 0 position that connects K to the constant token buckets (CTBs). CTB is a new component defined in this mechanism. CTB is a token bucket that fully contains tokens and never losses tokens. Once the system in the free mode, IP packets from all AF subclass are forwarded without any restriction. This mode aims to satisfy needs of AF subclasses traffic conditioned at edge routers (also to match AF specification) while traffic load enters the core router not heavily. It also doesn't conflict with the shaper at edge routers in the normal situations. When the system falls into urgent situations, the AF monitor switches Ks into 1 position that connects K to the second token bucket. At that time, the mechanism operates in the constrained mode. In this mode, AF subclasses traffic is governed by corresponsive token bucket in cooperating with a PI controller. The controller takes the current queue length, and then generates an adequate token rate so that congestion doesn't happen. The token bucket will drop or mark arriving IP packets if there are not enough tokens at that time. Drop level of each token bucket is truly drop precedence and different from others. According to AF class specification, there are three drop precedences in each AF class, thus in the implementation there are three corresponsive token bucket-controller sets. Inside the system, the referential queue size, q_{refi}, is used as the key parameter in order to make various drop levels. The bigger q_{ref} is, the lower the drop level is, or the lower the drop precedence is. Hence, assign $q_{ref1} > q_{ref2} > q_{ref3}$ if AF_{i1}, AF_{i2} and AF_{i3} are the Gold service, Silver service and Bronze service respectively. The specific values of these q_{ref} depend on the desired differences between AF subclasses. However, there is a new problem concerning buffer sharing between subclasses. What will happen when lower priority levels are disabled ($r_i(t) = 0$) but still contain full bucket of tokens? The answer is still packets from these token buckets continue to flow into the common buffer. This will overflow the buffer in case of all lower priority levels be disabled. To overcome this problem, follow the proposition below.

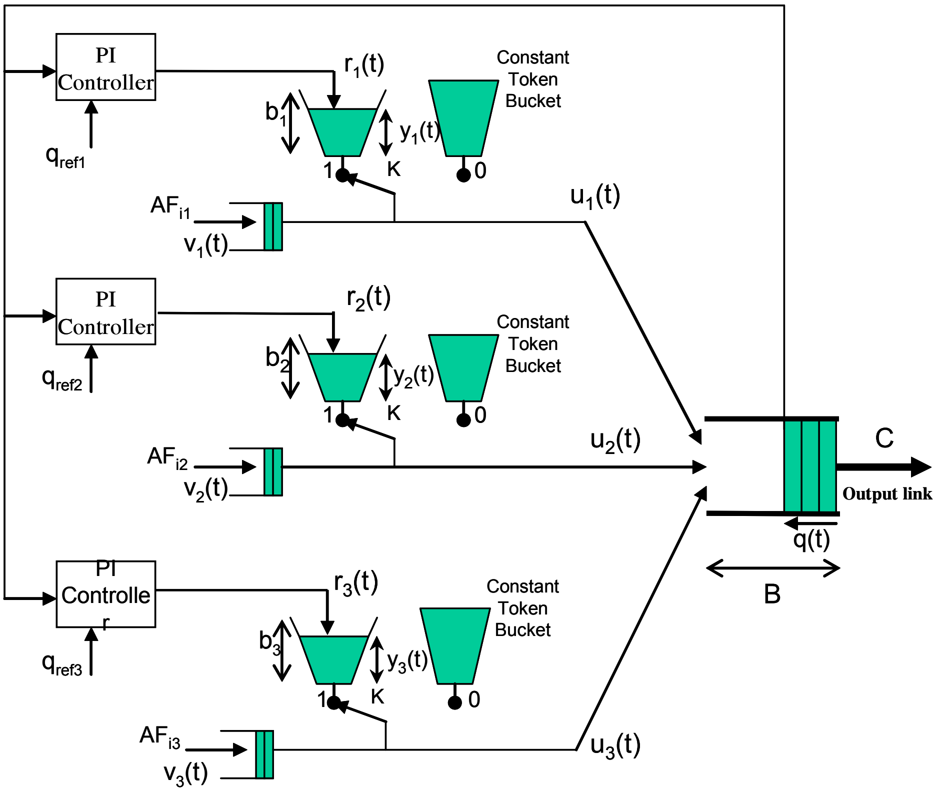


Figure 1. The principle scheme to implement AF subclasses

Proposition 1. *Suppose the sharing buffer has the size of B packets and size of buckets are respectively b1, b2 and b3 tokens. For the sake of protecting buffer from overflowing,* q_{ref1} *should be configured as*

$$q_{ref1} = B - \sum_{1}^{3} b_i.$$

This simply makes a surplus space in the buffer for standby purpose.

Now, we would like to explain how alter operating mode in this mechanism. In the queue, we define a threshold, called alterative threshold. When the queue length is equal to or smaller than the threshold, the system is in normal state, and thus it operates in the free mode. But if the queue length is greater than the threshold, the system is considered in urgent situation, and it operates in the constrained mode. The reason we define the threshold is simply that the queue surely has a certain size in the normal operating condition. This assures to exploit complete bandwidth of C. The AF monitor is responsible for keeping trace of the queue length, comparing with the threshold and switching between two modes.

It's no need to say any more about resource sharing between subclasses in an AF class in the free mode, but this is a really problem in the constrained mode that we have to solve. Indeed, when the higher priority subclass is idle, its resource lays waste while the lower priority subclasses may need some more. Like this, it is important to find the way of an idle subclass concedes its unused resource to lower priority subclass. The proposed way in this implementation leans on the possibility of redistributing q_{ref} for subclasses of AF monitor. The monitor is aware of an idle AF_{ij} by checking the ingress buffer of corresponsive token bucket and if see it empty. Hence, it raises every lower q_{ref} to a next higher q_{ref}, namely if AF_{i3} is idle, no need to do any thing but if AF_{i2} is idle, assign $q_{ref3} = q_{ref2}$, and if AF_{i1} is idle, assign $q_{ref3} = q_{ref2}$ and $q_{ref2} = q_{ref1}$, especially, if both AF_{i1} and AF_{i2} are idle, assign $q_{ref3} = q_{ref1}$. Whenever an AF_{ij} engages traffic again, these q_{ref} must be adjusted appropriately.

Proposition 2. *In the constrained mode, every* q_{ref} *of* AF_{ij} *should be assigned and adjusted as following Algorithm 1.*

Algorithm 1.

```
Initial
{Declare variables}
mode, start, $i$;
Main
mode = true;
start = true;
event = false;
Loop
If (mode and start) then
  begin
  assign ($AF_i$)
```

```
      start = false
      end
    Check (event)
    if (event) then
      begin
      assign($AF_i$);
      update($AF_i$);
      event=false;
      end
    End
    Assign ($AF_i$)
      {   $q_{ref1} = value1$;
          $q_{ref2} = value2$;
          $q_{ref3} = value3$;
      }
    Update ($AF_i$)
      {   for $i = 1$ to 2 do
            if ($tb\_buffer[i] = false$) then
              for $j = 3$ downto $i + 1$ do
                $q_{refj} = q_{ref(j-1)}$
      }
    Check (event)
      {   for $i = 1$ to 3 do
            begin
            $current\_tb\_buffer[i] = true$
            if (the $buffer(i)$ is empty) then $current\_tb\_buffer = false$
            if ($current\_tb\_buffer[i]$) $xor$ ($tb\_buffer[i]$) then
              begin
              $event[i] = true$;
              $tb\_buffer[i] = current\_tb\_buffer[i]$
              end
            end
      $Event = event[1]$ or $event[2] or event[3]$
      }
```

3. THE DYNAMIC MODEL OF SUBCLASS IMPLEMENTED MECHANISM

According to [12] each token bucket have two parameters concerned. The first parameter r is the rate of flow that pours tokens into the bucket. The second parameter b indicates the height of bucket or exactly, this is the maximum amount of tokens which can be contained in that bucket. In other applications of token bucket, the parameter r is fixed but it is a varying parameter in our approach, denoted $r_i(t)$. $r_i(t)$ is governed by a control mechanism which bases on current free space of the sharing buffer. The number of tokens in the bucket at the time t is $y_i(t),\ 0 \leqslant y_i(t) \leqslant b$. Packets arrive token bucket at rate of $v_i(t)$. The rate of packets forwarded from token bucket to output buffer is called $u_i(t)$. Following fluid flow model mathematically represents this:

$$\begin{aligned} \dot{q}(t) &= -1(q(t) > 0)C + \sum_{1}^{3} u_i(t), \\ u_i(t) &= 1(v_i(t) > 0)\Big[r_i(t) + \frac{y_i(t)}{T}\Big]. \end{aligned} \tag{1}$$

where, T is the continuous transmitted time. The outgoing link has capacity of C, a constant in IP DiffServ networks.

From (1), we find that $u(t) \approx 1(v(t) > 0).r(t)$ if T is a considerable time. With any time scale, we always have:

$$u_i(t)_{\max} = 1(v_i(t) > 0)[r_i(t) + y_i(t)] \tag{2}$$

with T gets the value of unit.

Considering $u_i(t) = u_i(t)_{\max}$ as a general case because it is the case filling buffer at the greatest speed.

Therefore, the dynamic of the bottleneck queue is given by:

$$\dot{q}(t) = -1(q(t) > 0)C + \sum_{i=1}^{3} 1(v_i(t) > 0)[r_i(t) + y_i(t)]. \tag{3}$$

Reference to operating model as shown in Figure 1, we find that it doesn't seem like any impact on the system when $v_i(t) = 0$. In Addition, the buffer is not empty in the constrained mode. From that, (3) can be rewritten as follow:

$$\dot{q}(t) = -C + \sum_{i=1}^{3} [r_i(t) + y_i(t)]. \tag{4}$$

Performing a Laplace transform on the differential equation (4):

$$q(s) = -\frac{C}{s^2} + \sum_{i=1}^{3} \Big(\frac{r_i(s)}{s} + \frac{y_i(s)}{s}\Big) \tag{5}$$

The linear dynamics is illustrated in a block diagram form in Figure 2.

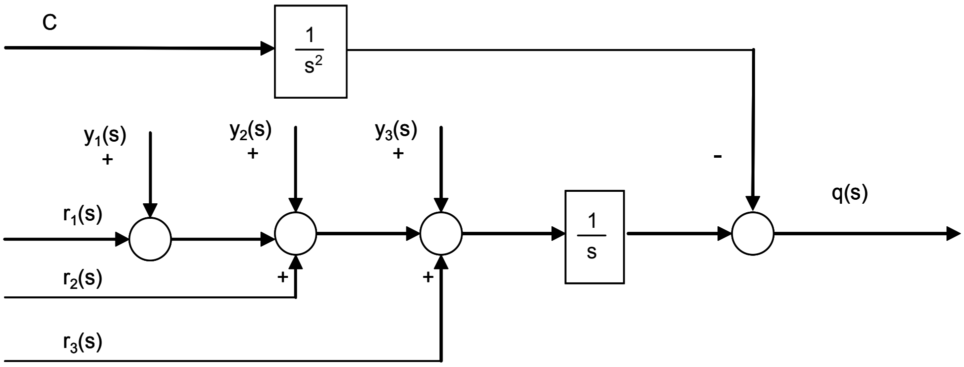


Figure 2. Block diagram of the bottleneck queue at core router

The basic effect of the token bucket parameters $r(t)$ and b is that the amount of packets sent $P(T)$ over any interval of time T obeys the rule:

$$P(T) \leqslant r.T + b. \tag{6}$$

Hence, $y_i(t) = b_i$ corresponds with the case of maximum amount of packets entering the buffer. Thus, we simply consider $y_i(t) = b_i$ as the worst case and replace $y_i(t)$ by b_i in calculations later.

The rate $r_i(t)$ must be controlled so that the buffer is never congested and obtain the highest utility level. We use a controller, which controls $r_i(s)$ according to free space part of buffer. The dynamics of system is illustrated in Figure 3. This is a feedback control mechanism without delay because of every component at the same place (at core router).

Depending on [the possibility of], we use PI controller for controllers in our implementation, it takes the form of transfer function:

$$G(s) = K_p\Big(1 + \frac{1}{T_1 s}\Big) \tag{7}$$

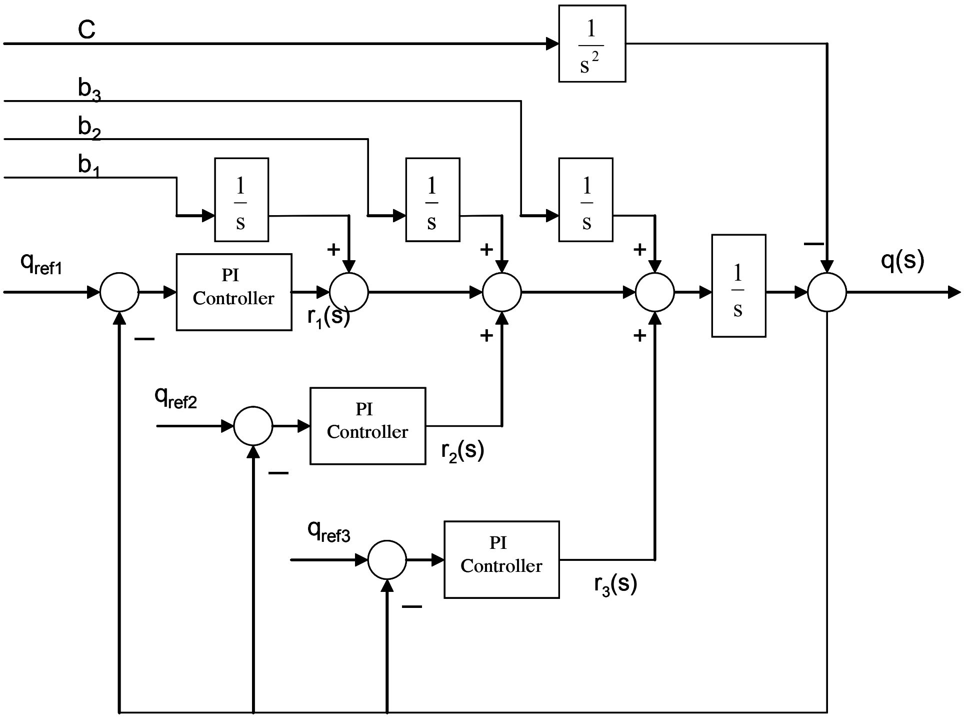


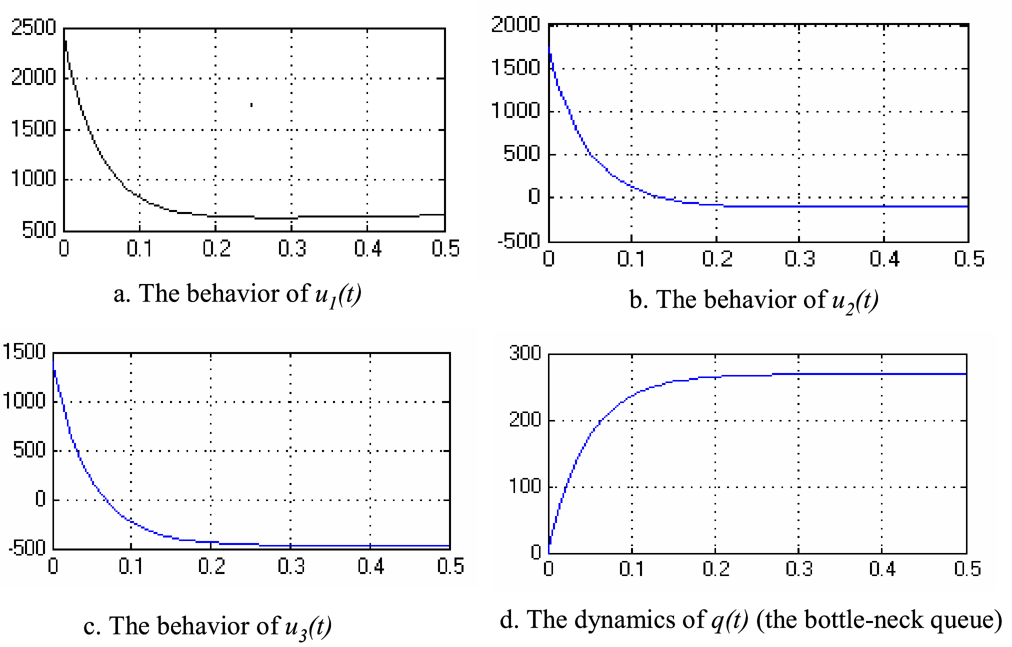
Figure 3. Block diagram of the control mechanism

No delay time in this control mechanism is an advantage for designing the controller. It can be completely designed by the normal way for the system without delay. A PI design involves choosing the value of the gain K_p and integrated constant T_I. There are several methods to determine these parameters such as the first Ziegler-Nichols method, the second Ziegler-Nichols method, the Chien-Hrones-Reswick method, the Kuhn method, etc. In this paper, we don't focus on how to design an optimal controller for the system, instead of that, we chose $K_p \approx 7$ and $T_I = 1$ in the next computer simulating section.

4. SIMULATION

This section describes the results of simulation on computer, taken from Matlab software. It shows the behavior of the bottleneck queue in a given system. In this simulation, assuming, the buffer has a size of 500 packets; all token buckets can contain up to 50 tokens; the rate of output link $C = 150$ packets per second. Choose $q_{ref1} = B-(b_1+b_2+b_3) = 500-150 = 350$; $q_{ref2} = 250$ and $q_{ref3} = 200$.

From the results of computer simulation by using Matlab, we see easily that $u_2(t)$ and $u_3(t)$ go quickly down crossing zero (actually equal to zero, corresponding to all packets be dropped), as illustrated in Figure 4b & 4c. In this, $u_3(t)$ goes down faster than $u_2(t)$ and also reaches zero level earlier. This means that AF_{i3} drops packet absolutely before AF_{i2} does the same affair. We also recognize that the rate of $u_1(t)$ of AF_{i1} decreases but still greater than 500 as shown in Figure 4a, this because it has the most priority. Next, the dynamics is illustrated in Figure 4d shows that although the numerous amount of coming traffic are available, the queue length will increase and stabilize at a finite level. The buffer or the bottle-neck queue is protected from congestion.



a. The behavior of $u_1(t)$

b. The behavior of $u_2(t)$

c. The behavior of $u_3(t)$

d. The dynamics of $q(t)$ (the bottle-neck queue)

Figure 4. The behaviors of some parameters in the mechanism

5. CONCLUSION

The new method to implement AF_{ij} in IP DiffServ networks has been presented. This is a way for making AF classes and their drop precedences easily. By pre-configuring q_{ref}, each active packet flow can be treated appropriately. It completely satisfies the AF specifications about architecture and operation. The proposed mechanism not only guarantees about congestion control, but also gives the highest utility. The simulation on computer shows that the system is completely stable. In addition, the mechanism also gives the way to share resources between subclasses. It contributes a certain degree to resolve the resource-sharing problem in DiffServ architecture. The performance of the system depends on some factors such as the type of the controller, the performance of the controller, the sample time, the b parameter of token bucket, the shape of traffic etc. Therefore, the performance of the system needs to be studied more details. These issues will be dealt in the future.

REFERENCES

[1] Lap Le Huu, Lan Vu Nhu, Son Nguyen Hong, The possibility of using token bucket for dropping/marking packets at core routers in IP networks, *Journal of Computer Science and Cybernetics* **21** (3) (2005) 201–207.

[2] R. Braden, D. Clark, and S. Shenker, Integrated services in the internet architecture: an overwiew, *RFC 1633* (1994).

[3] S. Blacke, D. Clark, M. Carlson, E. Davies, Z. Wang, and W. Weiss, An architecture for differentiated service, *RFC 2475* (1998).

[4] S. Floy and V. Jacobson, Random early detection gateways for congestion avoidance, *IEEE/ACM Transactions on Networking* **1** (4) (1993) 397–413.

[5] J. Kidambi, D. Ghosal, B. Mukherjee, Dynamic token bucket: A fair bandwidth allocation algorithm for high-speed networks, *IEEE/ACM Transactions on Networking* **1** (4) (1999) 24–29.

[6] C. V. Hollot, Vishal Misra, Don Towsley, and Wei-Bo Gong, A control theoretic analysis of RED, *Proceedings of IEEE/ INFORCOM* 2001.

[7] C. V. Hollot, V. Misra, D. Towsley, and W. B. Gong, On designing improved controllers for aqm routers supporting tcp flows, *Procs. INFOCOM*, 2001.

[8] V. Misra, W. Gong, and D. Towsley, Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED, *Procs. ACM SIGCOMM*, 2000.

[9] Y. Chait, C. V. Hollot, V. Misra, D. Towsley, H. Zhang, and C. S. Lui, Throughput guarantees for TCP flows using adaptive two color marking and multi-level AQM, *Procs. INFOCOM*, 2002 (837–844).

[10] N. U. Ahmed*, QUN. Wang, and L. Orozco Barbosa, Systems approach to modeling the token bucket algorithm in computer networks, *Mathematical Problems in Engineering* **8** (3) (2002) 265–279.

[11] S. Shenker, J. Wroclawski, General characterization parameters for integrated service network elements, *RFC 2215*, IETF (September 1997).

[12] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, Assured forwarding PHB group, *RFC 2597* (June 1999).

[13] V. Jacobson, K. Nichols, K. Poduri, An expedited forwarding PHB, *RFC 2598* (June 1999).

Received on August 7 - 2003