A Distributed Throttling Approach for Handling High Bandwidth Aggregates

Chee-Wei Tan, Princeton University
Dah-Ming Chiu, John C.S. Lui, The Chinese University of Hong Kong
David K.Y. Yau, Purdue University

Abstract

*Public-access networks need to handle persistent congestion and overload caused by high bandwidth aggregates that may occur during times of flooding- based DDoS attacks or flash crowds. The often unpredictable nature of these two activities can severely degrade server performance. Legitimate user requests also suffer considerably when traffic from many different sources aggregates inside the network and causes congestion. This paper studies a "*family*" of algorithms that "proactively" protect a server from overload by installing* rate throttles *in a set of upstream routers. A control-theoretic approach is used to obtain an "optimal" control setting that achieves throttling in a distributed and fair manner by taking important performance metrics into consideration, such as minimizing overall load variations. Using ns-2 simulations, we show that our proposed algorithms (1) are highly adaptive by avoiding unnecessary control parameter configuration, (2) provide max-min fairness for any number of throttling routers, (3) respond very quickly to network changes, (4) are extremely robust against extrinsic factors beyond the system control, and (5) are stable under given delay bounds. Most importantly, our approach provides a* systematic *way to handle high bandwidth aggregates by viewing them as a generic congestion control problem with relevance to many network applications.*

I. Introduction

This paper proposes a control-theoretic approach to design a *class* of distributed throttling algorithms for handling high bandwidth aggregates in the Internet. High bandwidth aggregates, as well as persistent congestion, can occur in the current Internet even when links are appropriately provisioned, and whether flows use conformant end-to-end congestion control or not. Two prime examples of high bandwidth aggregates that have received a lot of recent attention are *distributed denial-of-service* (DDoS) attacks and *flash crowds*[12].

In a DDoS attack, attackers send a large volume of traffic from different end hosts and direct the traffic to overwhelm the resources at a particular link or server. This high volume of traffic will significantly degrade the performance of legitimate users who wish to gain access to the congested resource. Flash crowds, on the other hand, occur when a large number of users concurrently try to obtain information from the same server. In this case, the heavy traffic may overwhelm the server and overload nearby network links. An example of flash crowds occurred after the 9/11 incident [12], when many users tried to obtain the latest news from the CNN web site.

Notice that network congestion caused by high bandwidth aggregates cannot be controlled by conventional flow-based protection mechanisms. It is because these aggregates may originate from

many different end hosts, each traffic source possibly of low bandwidth. One approach to handle high bandwidth aggregates is to treat it as a resource management problem and protect the resource *proactively* (i.e., ahead of the congestion points) from overload.

Consider a network server, say S, experiencing a DDoS attack. To protect itself from overload, S can install a *router throttle* at a set of upstream routers that are several hops away. The throttle specifies the maximum rate (in bits/s) at which packets destined for S can be forwarded by each router. An installed throttle will generally change the load experienced at S. S can then adjust the throttle rate in multiple rounds of feedback control until it achieves its load target. In [28], the authors propose a particular throttle algorithm, which we called the additive-increase-multiplicative-decrease (AIMD) control algorithm, and show how such server-based adaptation can effectively handle DDoS attacks, while guaranteeing fairness between the deployment routers.

Although the AIMD throttle algorithm in [28] is shown to work under various attack scenarios, it also raises several important issues that need to be addressed, for example, what are the proper parameters so as to guarantee the system is stable and to converge quickly to the steady state. Addressing these issues, this paper leverages control-theoretic analysis to design a *class* of distributed feedback control throttle algorithms. Our goal is to derive a throttle algorithm that is (i) highly *adaptive* (by avoiding unnecessary control parameters that would require configuration), (ii) able to *converge* quickly to the fair allocation, (iii) highly *robust* against extrinsic factors beyond the system's control (e.g., dynamic input traffic, and number/locations of current attackers), and (iv) *stable* under given delay bounds. Specifically, whereas the AIMD algorithm is proposed and extensively evaluated in [28], in this paper we more fundamentally look at the feedback control problem and show how it is possible to achieve system stability and fast convergence while minimizing the number of control parameters. While our discussion uses flooding DDoS attacks for context, our results are generally *applicable for network congestion control involving a large number of distributed contributing sources*. DDoS attacks complicate the network congestion control issue because additional detection mechanism such as statistical filtering [8], [13], [16], [21], [27] has to work jointly with overload control. In general, a robust defense mechanism has to minimize the disruption in terms of the amount of attackers' traffic getting through to the server, and the time to bring down the congestion level without any knowledge of the number of zombies and the adopted attack distribution.

The balance of the paper is organized as follows. In Section II, following the discussion in [28], we introduce the background and basic terminology of adaptive router throttling. We will also comment on the limitations of the previous approach, and motivate the need for a formal control-theoretic approach in the design of a stable and adaptive control system. In Section III, we present a class of distributed throttle algorithms, and discuss their feedback control properties. In Section IV, we provide formal analysis of the essential properties of our algorithms. Section V reports experimental results that illustrate various desirable properties of the algorithms in practice. Related work are discussed in Section VI. Section VII concludes the paper.

II. **Background of Router Throttling**

In [28], the authors present a distributed approach to defend against flooding-based DDoS attacks (i.e., one form of high bandwidth aggregates). They view the problem as a general resource management problem of protecting a server system from having to deal with excessive requests arriving over a global network. The approach is *proactive*: Before aggressive packets can converge to overwhelm a server, one can activate a subset of routers along the forwarding paths to regulate the contributing packet rates[1] to more moderate levels. The basic mechanism is for a server under attack, say S, to install a *router throttle* at a set of upstream routers several hops away. The throttle limits the rate at which packets destined for S can be forwarded by a router. Traffic that exceeds the rate limit will then be dropped at the router.

An important element in the proposed defense system[28] is to determine appropriate throttling rates at the defense routers such that, globally, S exports its full service capacity U_S (in kb/s) to the network, but no more. These appropriate throttle rates should depend on the current demand distributions and therefore must be negotiated dynamically between server and the defense routers. The negotiation approach is *server-initiated*. A server operating below the designed load limit needs no protection, and need not install any router throttles. As server load increases and crosses the designed load limit U_S, however, the server may start to protect itself by installing a rate throttle at the defense routers. If a current throttle rate fails to bring down the load at S to below U_S, then the throttle rate can be further reduced. On the other hand, if the server load falls below a low-water mark L_S (where $L_S < U_S$), then the throttle rate is increased to allow more packets to be forwarded to S. If an increase does not cause the load to significantly increase over some observation period,

[1] All traffic rate and server load quantities are in unit of Mbps, unless stated otherwise.

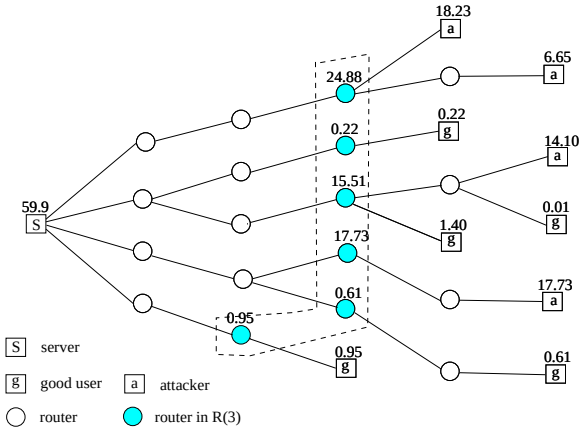

Fig. 1. Network topology illustrating $R(3)$ deployment points of router throttle. A square node represents a host and a round node represents a router. The host on the far left is the target server S. The deployment routers in $R(3)$ are shaded. The number above each host (or router) denotes the traffic rate at which the host (or router) sends to S.

then the throttle is removed. The goal of our throttle algorithms is to keep the server load within $[L_S, U_S]$ whenever a throttle is in effect.

As mentioned, router throttling does not have to be deployed at *every* router in the network. Rather, the deployment points are *parameterized* by a positive integer k, and are within the administrative domain of S. We denote this set of deployment points by $R(k)$. $R(k)$ contains all the routers that are k hops away from S and also routers that are less than k hops away from S but are directly connected to an end host. To illustrate this concept, Fig. 1 shows an example network topology in which a square node represents a host and a round node represents a router. The host on the far left is the target server S. The deployment routers in $R(3)$ are shaded. Notice that the bottom-most router in $R(3)$ is only two hops away from S, but is included because it is directly connected to a host. The number above each host (or router) denotes the traffic rate at which the host (or router) sends to S. For this example, we use $U_S = 22$ and $L_S = 18$. Since the total offered rate of traffic to S is $\rho = 59.9$, S will be overloaded without protection.

In fairly allocating the server capacity among the routers in $R(k)$, a notion of *level-k max-min fairness* is introduced in [28]:

Definition *1* (level-k max-min fairness) *A resource control algorithm achieves* **level-k max-min fairness** *among the routers $R(k)$, if the allowed forwarding rate of traffic for S at each router is the router's max-min fair share of some rate r satisfying $L_S \leq r \leq U_S$.*

Level-k max-min fairness is achieved by a fair throttle algorithm based on *AIMD* feedback con-

AIMD Fair Throttle Algorithm

```
/* initialize the aggregated traffic rate */
ρlast := −∞;
r_s := (L_s + U_s)/f(k);/*initialize throuttle rate*/
While (1)
    sends current rate-r_s throttle to R(k);
    monitor traffic arrival rate ρ for time window W;
    If (ρ > U_S) /* throttle not strong enough */
        /* further restrict throttle rate */
        r_s := r_s/2;
    elif (ρ < L_S) /* throttle too strong */
        If (ρ − ρlast < ε)
          remove rate throttle from R(k);
          break;
        else
    /* Relax throttle by additive step */
          ρlast := ρ;
          r_s := r_s + δ;
        fi;
    else
        break;
    fi;
end while;
```

Fig. 2. AIMD fair throttle algorithm specification.

trol. The AIMD algorithm (specified in Fig. 2) is run by the server. It installs at each deployment router a throttle rate r_s, which is the *"leaky bucket"* rate at which each router can forward traffic to S. Traffic that exceeds the allowed rate will be dropped by the routers. Note that an installed throttle has low runtime overhead since the cost is to maintain a leaky-bucket throttle for each victimized server.

In Fig. 2, r_s is the current throttle rate to be used by S. It is initialized to $(L_S + U_S)/f(k)$, where $f(k)$ is either some small constant, say 2, or an estimate of the number of throttle points needed in $R(k)$. The monitoring window W is used to measure the incoming aggregate rate. It is set to be somewhat larger than the maximum round trip time between S and a router in $R(k)$ so as to smooth out the variation of incoming aggregate rate. In section III-D, we examine the impact of the window size W in mitigating DDoS attack traffic. A constant additive step, δ, is used to ramp up r_s if a throttle is in effect and the current server load is below L_S. Each time S computes a new throttle rate using the algorithm, the throttle will be installed in $R(k)$, causing the loading at S to change. The algorithm may then continue in the while loop that iteratively adjusts r_s so that the server load converges to and stays within $[L_S, U_S]$ whenever a throttle is in effect.

To illustrate the AIMD throttle algorithm, consider the example shown in Fig. 1, with $U_S = 22, L_S = 18$, and a step size $\delta = 1$. We initialize r_s to $(L_S + U_S)/4 = 10$. Table I(a), reproduced from [28], shows the AIMD algorithm in action. When the algorithm is first invoked with throttle rate 10, the aggregate load at S drops to 31.78. Since the server load still exceeds U_S, the throttle rate is halved to 5, and the server load drops below L_S, to 16.78. As a result, the throttle rate is increased to 6, and the server load becomes 19.78. Since 19.78 is within the target range $[18, 22]$, the throttle algorithm terminates. When that happens, the forwarding rates of traffic for S at the deployment routers (from top to bottom in the Fig. 1) are 6, 0.22, 6, 6, 0.61, and 0.95, respectively. This is the max-min fair allocation of a rate of 19.78 among the deployment routers, showing that level-k max-min fairness is achieved (in the sense of Definition 1).

Round	r_s	ρ
1	10	31.78
2	5	16.78
3	6	19.78

(a)

Round	r_s	ρ
1	10	31.78
2	5	16.78
3	5.15	17.23
4	5.30	17.68
5	5.45	18.13

(b)

TABLE I

Example of using the AIMD algorithm. (a) $\delta = 1$ (b) $\delta = 0.15$

Although the AIMD algorithm is shown to be effective under various attack scenarios, there remain some "unresolved problems". In particular, the algorithm requires the *careful* choice of several control parameters, including the monitoring window W, step size δ, an initial estimate of the number of throttles $f(k)$, and the load limits U_S and L_S. The choice of these parameters will affect system stability and the speed of convergence to the stable load. For example, had we picked a smaller δ in the above example, it would have taken longer for the system to converge. This is shown in Table I(b). Furthermore, the parameters can be affected by a number of extrinsic factors that are beyond the algorithm's control – e.g., the number of routers in $R(k)$, the delay of messages from S to the routers in $R(k)$, and changes in the offered traffic to S. As pointed out in [28], if the control parameters are not set properly, the system's behavior may oscillate. In the following section, we will address these robustness and stability issues by developing a "*family*" of distributed throttling algorithms that can effectively and automatically handle the feedback control problem.

III. A Family of Adaptive Fair Throttle Algorithms

In this section, we describe several algorithms with increasing sophistication. There are multiple objectives for these throttle algorithms:

1. **Fairness**: achieve max-min fairness among the $R(k)$ routers
2. **Adaptiveness**: use as few configuration parameters as possible
3. **Fast convergence**: converge as fast as possible to the fair operating point for static or dynamic load
4. **Stability**: algorithm is stable with respect to any rate input distribution from the $R(k)$ routers and robust against inaccurate or unknown information of network parameters

The throttle algorithm terminates if the aggregate input rate is within a pre-defined region $[L_s, U_s]$. In general, the region contains C_0 where C_0 is a desired capacity. By default, we assume C_0 to be the mid-point between L_s and U_s. Each throttle algorithm is basically a probing algorithm that tries to determine a *common* throttle rate, r_s, to be sent by the server to all the $R(k)$ routers using negative feedback [5], [19]. Negative feedback is the process of feeding back the input a part of the system output. The principle of negative feedback thus implies that when the server asks the routers to decrease, we should ensure that the total load at the server will not increase, and when the server asks the routers to increase, the total load at the server will not decrease [5]. They are all designed to converge to the desired fairness criterion. The difference between the algorithms is in the way the server computes r_s, which will affect the speed of convergence and the stability of the system. Particularly, we summarize our algorithms as follows.

1. Binary Search Fair Algorithm: A simple algorithm that satisfy the first three objectives of a fair throttle algorithm for static load only.
2. Proportional aggregate control (PAC) with number of throttle estimate: An algorithm that satisfies all the objectives of a fair throttle algorithm.
3. Proportional-aggregate-fluctuation-reduction (PAFR) control fair algorithms: Improves the PAC algorithm by considering stability issues and tracking ability.

A. Binary Search (BS) Fair Algorithm

The possible range for the throttle rate r_s is $[0, U_s]$, the two extremes corresponding to an infinite number of throttles and one throttle, respectively. A familiar binary search algorithm [6] can be

applied: Reduce the search range by half at every iteration of the algorithm (please see Fig. 3). Like the AIMD algorithm, this algorithm uses the aggregate rate ρ to determine the direction of adjustment. When the aggregate rate is more than U_s, r_s is reduced by half (same as before). However, when the aggregate rate is less than L_s, r_s is increased to the mid-point between the current r_s and U_s. This change removes the need to configure δ. The increase and decrease rules are symmetrical, each time reducing the search range by half. The pseudo-code of binary search is shown in Fig. 3. In the algorithm, notice that we initialize r_s to $(U_s + L_s)/N$, where $N = |R(k)|$. This corresponds to the guess that half of the $R(k)$ routers will be dropping traffic because of throttling.

Round	r_s	ρ
1	10	31.78
2	5	16.78
3	7.50	24.28
4	6.25	20.53

(a)

Round	r_s	ρ
1	5.63	24.11
$\vdots$	$\vdots$	$\vdots$
∞	5.0	22.22

(b)

Round	r_s	ρ
1	5.63	24.11
$\vdots$	$\vdots$	$\vdots$
7	5.01	22.25
8	2.51	12.77
9	3.76	18.50

(c)

TABLE II

Trace of the throttle rate and server load using the Binary Search algorithm. (a) Binary Search for stationary load (b) Binary Search becomes unstable without re-initialization (c) The algorithm corrects itself using $\epsilon = 0.05$.

The problem of optimal bandwidth probing assuming a known upper bound has been considered earlier by [15]. In their slightly different problem formulation of searching for available bandwidth by a single flow, they conclude that (a generalized version of) the binary search algorithm is near-optimal for certain reasonable cost functions of overshooting and undershooting.

The performance of binary search can be illustrated using the same example we considered in Section II. The results are shown in Table II(a). Notice that binary search converges quickly for *stationary* load.

The basic binary search algorithm assumes that the traffic loading is static. As noted by Karp *et al.* [15], when the traffic conditions are dynamic, the situation is more complicated. They propose and analyze alternative algorithms for bounded variations in the traffic conditions. In our case, traffic can vary drastically since attackers may often vary their attack rates to reduce the chance of being revealed. The binary search algorithm must therefore detect the situation that the shrunken

Binary Search (BS) Fair Throttle Algorithm

```
r_s ≐ (U_s + L_s)/N; /* Initialize r_s, N = |R(k)| */
ρ_last := −∞;
high := U_s;
low := 0;
While(1)
    sends current rate-r_s throttle to R(k);
    monitor traffic arrival rate ρ for time window W;
    If (ρ ≥ U_s)
        high := r_s;
        If (ρ_last − ρ < ε)
            low := 0; /* re-initialize BS */
    elif (ρ ≤ L_s)
        low := r_s;
        If (ρ − ρ_last < ε)
            high := U_s; /* re-initialize BS */
        fi;
    fi;
    ρ_last := ρ;
    r_s := (high + low)/2;
end while;
```

Fig. 3. Binary Search fair throttle algorithm specification.

search range for r_s can no longer deal with the changed traffic conditions, and *re-initialize* the search range accordingly. Such a modification is incorporated in the algorithm given in Fig. 3. However, it takes time to discover the need for re-initializing the search, and it often takes more time than necessary for the new search to converge. For example, Table II(b) shows that, if the last two sources in Fig. 1 with offered rates of 0.61 and 0.95 both change to 3.5 after round 4 in Table II(a), both sources are still within the rate throttle limit of r_s, but r_s cannot change to a value lower than low $= 5.0$ without re-initialization, and the system will be overloaded indefinitely. Table II(c) shows that re-initialization corrects the algorithm by resetting the variable low. Although the binary search algorithm is *fair, fully adaptive* and *fast*, it is not entirely satisfactory for dynamic traffic and packet delay.

B. Proportional aggregate control (PAC) with estimate on the number of throttles

The previous algorithms directly adjust r_s adaptively, without using the magnitude of the overshoot (or undershoot) of the aggregate rate ρ beyond the target range. We now explore algorithms that do make use of this information, as well as an *estimate* of the number of routers in $R(k)$ that actually drop traffic. For clarity of exposition, we will call a deployment router whose offered traffic rate exceeds the throttle rate (and hence will drop traffic because of throttling) an *effective*

Estimator for the number of throttles n in $R(k)$: est(ρ, r_s)

$\psi := 0.25$; /* $0 < \psi \leq 1$. */
$n_{\max} := \min(\lceil \rho / r_s \rceil, |R(k)|_{\max})$;
$n := \min(n_{\max}, \lceil ((1-\psi) \times n_{last}$
$+\psi \times (\rho - \rho_{last})/(r_s - r_{s,last})) \rceil)$;
$n_{last} := n$;
$\rho_{last} := \rho$;
$r_{s,last} := r_s$;
return n;

Fig. 4. Estimating the number of effective throttles n in $R(k)$. This algorithm assumes that an upper bound of the total number of routers in $R(k)$, $|R(k)|_{\max}$ is known.

throttling router. We also call the throttle at such a router an *effective throttle*. If the number of effective throttles is n, then it is reasonable to set the change of r_s to

$$\Delta r_s = \frac{|\rho - C|}{n} \tag{1}$$

where C represents the target rate (a value within $[L_S, U_S]$) of aggregate traffic at S.

Since the traffic distribution at the routers is arbitrary and can be dynamically changing, it is not possible to know the exact number of effective throttling routers. Furthermore, as r_s is adjusted, the number of these routers changes as well. By monitoring the past traffic and r_s, however, we can estimate n. The idea is to keep track of the last throttle rate r_{last} and last aggregate rate ρ_{last}. This allows us to estimate n as

$$n = \frac{|\rho - \rho_{last}|}{|r_s - r_{last}|}.$$

A pseudo-code implementation is shown in Fig. 4. Exponential averaging is added to minimize inaccuracies due to fast changing components of the traffic load. Given the estimator for the number of effective throttles, we can compute a suitable change to r_s using Eqn. (1). This is the essence of the new algorithm, as shown in Fig. 5.

With the new algorithm, we wish also to address our fourth objective, namely system *stability* in the face of delay between a throttle request sent by the server and the throttle taking effect at a deployment router. This objective can be achieved by trying to correct only a *fraction* K_P of the discrepancy between the aggregate rate and the rate target. The fraction K_P should be *proportional* to the discrepancy; hence, we also refer to such control as *Proportional Aggregate Control* (PAC). We will show in Section IV how one can choose automatically (i.e., without the extra burden of

Proportional Aggregate Control (PAC) Algorithm

```
r_s ≐ (U_s + L_s)/N; /* Initialize r_s, N = |R(k)| */
ρ_last := −∞
While(1)
        sends current rate-r_s throttle to R(k);
        monitor traffic arrival rate ρ for time window W;
        If (ρ ≥ U_s)
                C := L_s;
        elif (ρ ≤ L_s)
                If (ρ − ρ_last < ε)
                        remove rate throttle from R(k);
                        break;
                else
                        C := U_s;
                        ρ_last := ρ;
                fi;
        else
                 break;
        fi;
        φ := −K_P × (ρ − C);
        r_s := r_s + φ/est(ρ, r_s);
end while;
```

Fig. 5. Proportional aggregate control fair throttle algorithm specification.

tricky parameter configuration) an appropriate value for K_P based on the maximum delay between the server and a deployment router.

Another detail in the pseudo-code is how to pick the value of C in Eqn. (1). The value should be in the target control range of $[L_S, U_S]$. Our current implementation sets the target capacity C to L_S when we have overload and to U_S when we have underload. By advertising a larger aggregate rate discrepancy, we can maximize the likelihood that the resulting target rate will end up in the target range.

The PAC algorithm is applied to the same example used before. The results are shown in Table III. An estimate of the number of effective throttles is given by the change in server load over the change in throttle rate, $\triangle\rho/\triangle r_s$. For stationary traffic demand, the rate of convergence is monotonic and relatively fast. The speed of convergence depends on the setting of K_P. However, the setting of K_P can also affect system stability. This is discussed next.

C. Proportional-aggregate-fluctuation-reduction (PAFR) control fair algorithms

We now introduce one more optimization to the PAC algorithm presented in the last section. When there is a change in the offered load at the routers in $R(k)$, a higher proportional gain can

Round	r_s	ρ	$\triangle\rho/\triangle r_s$
1	10	31.78	–
2	8.28	26.62	3
3	6.84	22.30	3
4	6.12	20.14	3

TABLE III

Trace of the throttle rate and server load using the PAC algorithm. Target capacity used is $L_s = 18$. Here, $K_P = 1/2$ so it reduces the search interval by half at every round.

Proportional-Aggregate-Fluctuation-Reduction Algorithm

$r_s \doteq (U_s + L_s)/N$; /* Initialize r_s, $N = |R(k)|$ */
$\rho_{last} := -\infty$
While(1)
  sends current rate-r_s throttle to $R(k)$;
  monitor traffic arrival rate ρ for time window W;
  If ($\rho \geq U_s$)
    $C := L_s$
  elif ($\rho \leq L_s$)
    If ($\rho - \rho_{last} < \epsilon$)
      remove rate throttle from $R(k)$;
      break;
    else
      $C := U_s$;
    fi;
  else
    break;
  fi;
  $\phi := -K_P \times (\rho - C) - K_D \times (\rho - \rho_{last})$
  $r_s := r_s + \phi/\text{est}(\rho, r\_s)$;
  $\rho_{last} := \rho$;
end while;

Fig. 6. Proportional-Aggregate-Fluctuation-Reduction fair throttle algorithm specification.

adapt faster to the change. However, such "strong" control may result in large fluctuations in the throttle rate, which may be annoying to the end users and may cause instability. On the other hand, a "weak" control causes prolonged congestion at the server. Hence, the goal is to reduce fluctuations while still achieving fast convergence. The stability problem that arises when a high proportional gain is used can be mitigated by considering a derivative control parameter, K_D, as shown in Fig. 6. As we will show in Section IV, it turns out that K_D can be optimized to a constant value, independent of the operating conditions, for a given design time *cost function*. Hence, K_D does not introduce any extra burden of parameter configuration. The new algorithm is called a *Proportional-aggregate-fluctuation-reduction* (PAFR) controller.

In this case, if $\rho > U_s$ or $\rho < L_s$, the total expected change in throttled traffic rate in all of $R(k)$ at the next control interval is

$$\triangle\rho'_{i+1} = -K_P(\rho_i - C) - K_D\triangle\rho_i; \quad \triangle\rho'_1 = 0, \tag{2}$$

where $\triangle\rho_i$ is the actual change and $\triangle\rho'_i$ is the expected change in the control interval i.

The first term is necessary for the mismatch regulation and the second term tracks the rate of change in the feedback. The first term has the largest impact on the aggressiveness of the probe used in the algorithm. A smaller K_P will make the system slower to converge. But a larger K_P may result in larger overshoots, sometimes even causing the system to become unstable – i.e., the system oscillates indefinitely around the band $[L_s, U_s]$ without convergence. The second term gives a signal proportional to the rate at which the mismatch changes. It is more responsive to changes in the mismatch than the first term. This second term also has a damping effect on the amount of rate throttle fluctuation. A larger K_D implies more damping. It helps the stability of the system, and allows the proportional gain to be increased. As we shall see in the simulation results, the term can reduce the amplitude of overshoot significantly during transient response while maintaining fast convergence.

Table IV shows the PAFR algorithm for $K_P = 0.73$ and $K_D = 0.48$ using the previous example. An estimate of the number of effective throttles is given by the change in server load over the change in throttle rate, $\triangle\rho/\triangle r_s$. Although this simple example does not show any improvement in convergence time, the simulation results in Section V will clearly illustrate such improvements. In addition, the PAFR algorithm tracks the target rate range better. The tracking metric will be made precise in the next section.

Round	r_s	ρ	computed $\triangle\rho'_i$	$\triangle\rho/\triangle r_s$
1	10	31.78	0	–
2	7.48	24.22	-10.06	3
3	7.18	23.32	-0.91	3
4	6.05	19.93	-3.38	3

TABLE IV

Trace of the throttle rate and server load using the PAFR algorithm. Target capacity used is $L_s = 18$.

D. Setting the window size W

The criterion for the right window size is twofold– obtain a more accurate estimate of the effective number of throttled routers n which improves the adaptive throttle algorithms, and enforce rate limiting. For example, the filter at each throttle may employ some form of deterministic or probabilistic policy that drops packet according to specific features in the packet. Further, it is recommended in [21] that a filter policy cannot be completely memoryless while enforcing rate limiting, i.e., in order for such kind of a filter to operate effectively based on its past measurement of the number of packets in a window length W, the throttle rate r_s cannot change too drastically in consecutive windows. Similarly, to obtain an accurate estimate of n, W cannot be too small. On the other hand, a large W affects the convergence time of the overload control system. To cope with this problem, we use a moving window with a parameter which is a function of the maximum round trip time in the system to obtain a smooth measurement of the incoming aggregate rate ρ. Specifically, we define $\lambda = 2D/W$ where D is the average round trip delay from each router in $R(k)$. Averaging between consecutive windows, we have $\overline{\rho}_i = \lambda\rho_{i-1} + (1-\lambda)\rho_i$ where the subscript denotes the i-th window. We verify by simulations in Sec. V that the above approximation works very well in measuring the aggregate rate which is used as an input for our throttle algorithms.

IV. An Analysis of the Fair Throttle Algorithms

In this section, we use a control-theoretic approach to study router throttling. A fluid flow model is used to analyze the stability of the PAC and PAFR fair throttle algorithms. We represent the protected server S as a single bottleneck node where $y(t)$ denotes the aggregate amount of traffic from $R(k)$ to S. $y(t)$ serves the same role as ρ_i in our algorithm specifications. In our analysis, we assume that D is upper bounded by D_0. For both the PAC and PAFR algorithms, at each control interval i, the expected change in aggregate throttle rate is computed as in Eqn. (2) where $K_D = 0$ for the PAC algorithm.

However, the offered traffic at each router varies over time. Therefore, $y(t)$ will be larger than the expected change if (1) the load at some previously unthrottled routers increases, but is still less than the throttle limit, or (2) additional routers are added to $R(k)$ while the throttle rates are changing. On the other hand, $y(t)$ is less than the expected change if (1) the offered load

at some previously throttled routers fall significantly below the throttle limit, or (2) some routers are removed from $R(k)$ while the throttle rates are changing. These factors can be viewed as uncertainties at the input of the system. Here, the system refers to maintaining the aggregate input traffic rate within the bounds $[L_S, U_S]$. Stability is defined in the *bounded-input-bounded-output* sense (See [19], pp. 319). Further, estimation of network parameters such as the number of throttles may introduce additional uncertainty to the system. Indeed, the purpose of using feedback is to combat uncertainty. An important question is how fast we should adapt the change in throttle rate while maintaining a stable system. A properly designed negative feedback controller has the characteristic of rejecting disturbance in the closed loop, and is therefore the focus of this section.

We next introduce a fluid model that governs the system we are analyzing. In our fluid model, we treat the target capacity to be a constant value, C. By default, C can be the mid-point between L_S and U_S. The input traffic rate at S is given by

$$\frac{\mathrm{d}y(t)}{\mathrm{d}t} = -K_P(y(t-D) - C) - K_D(\mathrm{d}y(t-D)/\mathrm{d}t).$$

The parameters K_P and K_D are the proportional and differential control gains, respectively. Selecting appropriate parameters is part of the design of the throttle algorithms. We first consider the PAC algorithm where the change in the search range strictly converges to zero if the traffic is stationary. Our first result shows that the amount of feedback is inversely proportional to the amount of delay that the system can tolerate.

Theorem *1: The system is asymptotically stable using proportional control if and only if* $0 < K_P < \min(\pi/(2D_0), 1)$ *where* D_0 *is the maximum delay in the system.*

Proof: please refer to the Appendix. ∎

Remark 1: When $K_P = 1$, the exact mismatch between $y(t)$ and C is fed back. It is intuitive that the system is most responsive. However, the system is also the least tolerant to time delay (refer to the Appendix for the proof). Hence, we note that there is a tradeoff between the system speed of response and robustness to time delay.

Remark 2: Although a maximum proportional gain is associated with the maximum delay, recall that over-estimating the number of throttles in $R(k)$ has the effect of reducing the proportional gain. In this sense, a PAC algorithm with a more aggressive gain can be considered. For example, by assuming the maximum number of routers in $R(k)$ that will be reacting to the change of control,

we have a form of doubly conservative PAC algorithm.

Next we introduce a *cost function* which measures the effectiveness of a fair throttle algorithm, and is given as

$$J = \sum_{i=1}^{T_s} (x_i)^2 \tag{3}$$

where x_i is the corresponding mismatch between C and ρ at control interval i, and T_s is a given terminating interval of the algorithm. Note that x_i is positive if the system is overloaded, or negative if the system is underloaded, hence the use of the squared function. Eqn. (3) is similar to the proposed cost function (gentle cost function) in [15] where both overloading and underloading are penalized. In this paper, we treat the cost of overloading and underloading to be the same.

Note that T_s plays the same role as *settling time* in classical control theory (See [19], pp. 272), which is defined as the time required for the system response to decrease and stay within a specified percentage of its final value (e.g., a frequently used value is 5%). This criterion is tolerant to *small* but punitive to *large* errors at every control interval. In other words, a smaller J is preferred. For a given small range $[L_s, U_s]$ (a very small specified percentage of its final value), a sufficiently long T_s and offered traffic demand, the metric J is useful for differentiating the performance of various algorithms in the most stringent manner. An over-conservative throttle algorithm tends to have larger J at most of the control intervals. Furthermore, an algorithm that causes large variation in throttle rate r_s often results in a larger index J.

The design of the PAC and PAFR algorithms is based on a time domain optimization approach where appropriate control parameters K_P and K_D are selected by minimizing J subject to the constraint that the integral exists in the face of a unit step input. In our context, a unit step input is equivalent to an external disturbance that occurs when the server load is within the band $[L_s, U_s]$, e.g., traffic demand changes in $R(k)$ or n changes (e.g., due to re-deployment of the defense perimeter). Hence, r_s has to be re-adjusted. Without loss in generality, we consider an infinitesimal control interval and let $T_s \to \infty$. Since x_i has the magnitude of the control error, we obtain the following integral of squared error, (ISE),

$$J = \int_0^\infty x^2(t) \mathrm{d}t. \tag{4}$$

The ISE belongs to a family of integral cost functionals for the time domain optimization method in

control theory, which is used to determine optimal fixed order controller parameters for a particular input. For a proportional controller, we have the following result.

Theorem *2: The optimal $K_P(D)$ that minimizes Eqn. (4) is $(\sqrt{3}-1)/D$ for a delay D and the optimal proportional control, $K_P(D_0)$ stabilizes the system for all $D \leq D_0$ where D_0 is the maximum delay in the system.*

Proof: please refer to the Appendix. ∎

For the PAFR fair throttle algorithm, we have the following main result.

Theorem *3: The system is asymptotically stable using the PAFR algorithm and, J in Eqn. (4) is minimized if and only if $K_P(D) = 0.80/D$ for a delay D, and the optimal PAFR algorithm $K_P(D_0)$ stabilizes the system for any delay $D \leq D_0$ where D_0 is the maximum delay in the system and $K_D = 0.48$.*

Proof: please refer to the Appendix. ∎

Remark 3: For both optimal PAC and PAFR algorithms, only a single parameter K_P needs to be configured in view of the maximum delay in the $R(k)$ routers.

Remark 4: The above theorems imply that, if the control parameters are designed to stabilize the system for a fixed maximum delay D_0, $R(k)$ router deployment using the same control parameters is feasible anywhere in the network as long as the maximum delay in the $R(k)$ routers is less than D_0. Furthermore, Theorems 2 and 3 are valid for any number of routers in $R(k)$.

Corollary *1: An optimal PAFR algorithm that minimizes Eqn. (4) always has a smaller J than an optimal PAC algorithm for all $D > 0$.*

Proof: please refer to the Appendix. ∎

Corollary 1 implies that the PAFR algorithm outperforms the PAC algorithm in minimizing rate mismatch given that the fair throttle algorithms run for a sufficiently long time. This fact is further corroborated by the simulation results in the next section. The throttle rate $r_s(t)$ at each router is a function of the aggregate feedback and the estimated number of throttles, $f(r_s(t))$. The differential equation of the throttle rate is thus given by $\mathrm{d}r_s(t)/\mathrm{d}t = (-K_P x(t-D) - K_D \mathrm{d}x(t-D)/\mathrm{d}t)/f(r_s(t))$ where $f(r_s(t)) \approx (\mathrm{d}y(t)/\mathrm{d}t)/(\mathrm{d}r_s(t)/\mathrm{d}t)$. Assuming that $f(r_s(t))$ is a constant value, then the dynamics of $r_s(t)$ is described by the inverse Laplace transform of

$A(K_P + K_D s)e^{-sD}/(s + (K_P + K_D s)e^{-sD})$ where A is some constant and $K_D = 0$ for the PAC algorithm. Hence, for small delay D, $r_s(t)$ changes approximately (negatively) exponentially with a time constant of $1/K_P$ when the system is (overloaded) underloaded.

V. **Simulation Results**

In this section, we study the performance of the proposed fair throttle algorithms in terms of performance metrics such as convergence time, rate throttle fluctuation and adaptation to different network parameters using ns-2[26]. We denote the instantaneous offered traffic rate at router i as $T_i(t) \forall\, i$. For all experiments, we set $L_s = 100$Mbps and $U_s = 115$Mbps. The initial throttle rate r_s is set as 200 with $W = 3D$, and the low pass filter time constant of *est()* is set as 0.25, unless noted otherwise.

Experiment A: Performance of fair throttle algorithms for heterogeneous network sources with constant input: In the experiments, we consider 50 heterogeneous sources. In the first experiment, all sources have constant input of offered traffic rate $T_i(t) = 30 \forall\, i$. Ten of these sources changes their sending rate to 1Mbps and 4Mbps at $t = 50$ and $t = 100$ respectively. The network delay D between S and each of the constant sources is 100ms. The network parameter setting is the same as in [28]. The corresponding cost function J and settling time T_s for each algorithm are reported under the figures for the input disturbance at $t = 50$ only if the system is stable. Fig. 7 shows the aggregate server load when AIMD ($\delta = 0.02$Mbps, $\delta = 0.2$Mbps and $\delta = 0.4$Mbps), binary search ($\epsilon = 1$Mbps), PAC ($K_P = 0.5$) and PAFR ($K_P = 0.5, K_D = 0.48$) algorithms are used respectively. From the figure, we observe that the PAFR algorithm is stable, converges, and has a much less over/under-shoot.

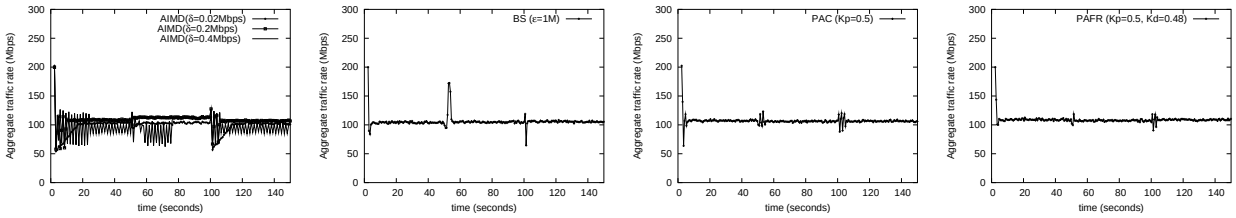

Fig. 7. (a) AIMD algorithm with step size of $\delta = 0.02M$ ($T_s = 4.0s, J = 74.1Mb$), $\delta = 0.2M$ ($T_s = 2.5s, J = 50Mb$) and $\delta = 0.4M$ ($T_s = 42.7, 664.3Mb$). (b) Binary Search algorithm. $\epsilon = 1M$ ($T_s = 6.6s, J = 183.9Mb$). (c) PAC ($K_P = 1/2$) algorithm ($T_s = 4.8s, J = 38.7Mb$). (d) PAFR algorithm with $K_P = 0.5$ and $K_D = 0.48$ ($T_s = 1.5s, J = 18.6Mb$).

Experiment B: Performance of fair throttle algorithms for heterogeneous network sources

with varying input: Next we repeat the above experiment, but the first thirty sources are constant sources of $T_i(t) = 4$Mbps for $i = 1, \ldots, 30$. The next ten sources are sinusoidal sources where $T_i(t) = 1.5\sin(0.16t) + 2.5$Mbps for $i = 31, \ldots, 40$. The network delay for each of these sinusoidal sources is 50ms. The last ten sources are square pulse sources wherein $T_i(t) = 4$Mbps for the time interval $20k \leq t \leq 10(2k+1)$ and $T_i(t) = 1$Mbps for the time interval $10(2k+1) \leq t \leq 20(k+1)$, $i = 41, \ldots, 50$ and $k \in \{0, 1, 2, \ldots\}$. The network delay for each of these square pulse sources is 50ms. Fig. 8 shows the aggregate server load for the second experiment.

Based on the above experiments, we can make the following observations.

Remark 1: The stability and performance of the AIMD fair throttle algorithm is dependent on the step size used. The system is stable but slow for small δ, but unstable when δ is large. Further, there is a significant overshoot in the initial transient response for all the three different step sizes.

Remark 2: The binary search algorithm is oscillatory when the number of sources is large, as in the experiment. Significantly large overshoots are observed before the algorithm finally terminates, and are due to the large variation in throttle rate r_s as shown in Fig. 9. In the dynamic traffic case, slow response to re-initialization is the main cause for the large overshoots between $t = 50$ and $t = 100$. In general, this effect can be mitigated by a gradual release of r_s.

Remark 3: The PAC ($K_P = 0.5$) fair throttle algorithm is comparatively faster and stable, but there are large overshoots when there is a disturbance to the system at $t = 50$ and $t = 100$. As a result, the convergence time might be longer as shown in Fig. 8.

Remark 4: The PAFR fair throttle algorithm is more robust to external disturbances with much smaller overshoots, and has fast convergence time. From Fig. 10, we note that the estimator algorithm is extremely effective in estimating accurately the number of throttles in $R(k)$ at $t = 50$ and $t = 100$. Fig. 9 compares the rate throttle variation for the PAC and PAFR algorithms. We observe that damping provides excellent performance in reducing the rate throttle variation.

Experiment C: Effect of system parameters on Binary Search (BS) fair algorithm: Here we evaluate an improved version of Binary Search algorithm using the estimator algorithm. Re-initializing the control parameters in the Binary Search algorithm which leads to a gradual change in r_s can be achieved by taking into account the current aggregate rate mismatch and the total number of effective throttles. Specifically, if we have (under)/overload, we let

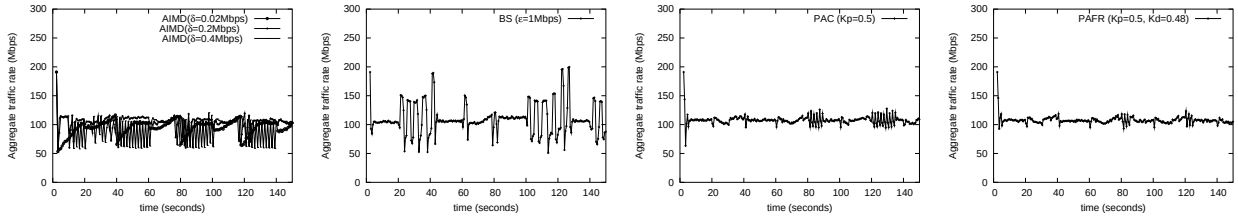

Fig. 8. (a) AIMD algorithm with step size of $\delta = 0.02M$ ($T_s = 9.3s, J = 114Mb$), $\delta = 0.2M$ ($T_s = 2.0s, J = 28.1Mb$) and $\delta = 0.4M$ ($T_s = 16.1s, J = 236.4Mb$). (b) Binary Search algorithm. $\epsilon = 1M$ ($T_s = 9.8s, J = 229.2Mb$). (c) PAC ($K_P = 1/2$) algorithm ($T_s = 2.4s, J = 12.9Mb$). (d) PAFR algorithm with $K_P = 0.5$ and $K_D = 0.48$ ($T_s = 1.3s, J = 7.1Mb$).

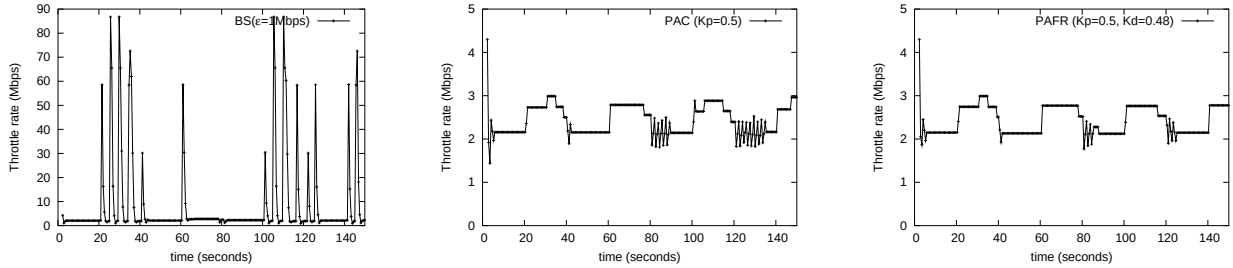

Fig. 9. (a) Binary Search algorithm exhibits large rate throttle variation. (b) PAC ($K_P = 0.5$) algorithm. Significantly large rate throttle variation is observed in the transient response. (c) PAFR algorithm with $K_P = 0.5$ and $K_D = 0.48$. Damping provides better transient response while maintaining excellent response time.

$\mathbf{If}(\rho - \rho_{\mathbf{last}} < \epsilon)$

$(\text{high})\text{low} = \min(\max(r_s + (C - \rho)/est(\rho, r_s), 0), C);$

fi;

We repeat the second experiment using the improved Binary Search, and observe the system response when the step disturbance occurs at $t = 50$. Fig. 11 shows the effect of different ϵ on T_s and J of the improved Binary Search algorithm and as shown in Fig. 11(a) and (d), if ϵ is too small or too big, the convergence time is much longer. If ϵ is too small, the algorithm takes unnecessarily longer to detect that re-initialization is required, and the system transient response is characterized by very large overshoots. If ϵ is too large, the algorithm becomes more sensitive to small perturbation in the offered load. In comparison to the previous experiment, it is not surprising

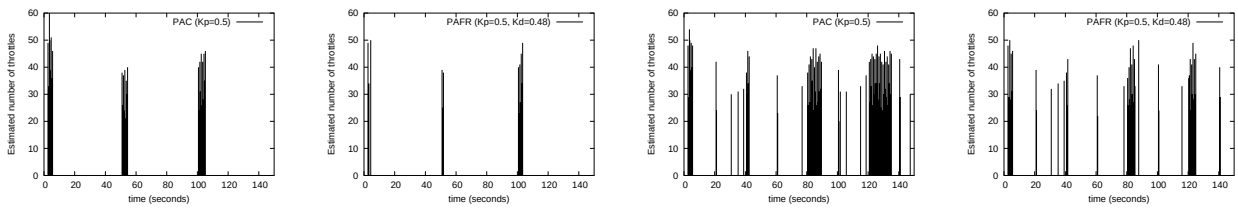

Fig. 10. Estimating the number of throttles in $R(k)$ using the *est()* algorithm.

that gradual re-initialization will stabilize the Binary Search algorithm. In fact, by making use of delayed information on the aggregate mismatch, the gradual re-initialization mechanism brings the improved Binary Search a step closer and comparable to a PAC algorithm. However, in general, the system response is sensitive to the parameter setting of ϵ, delay and offered traffic load. Hence, the performance of the improved Binary Search is not satisfactorily as compared to the PAC or PAFR fair throttle algorithm.

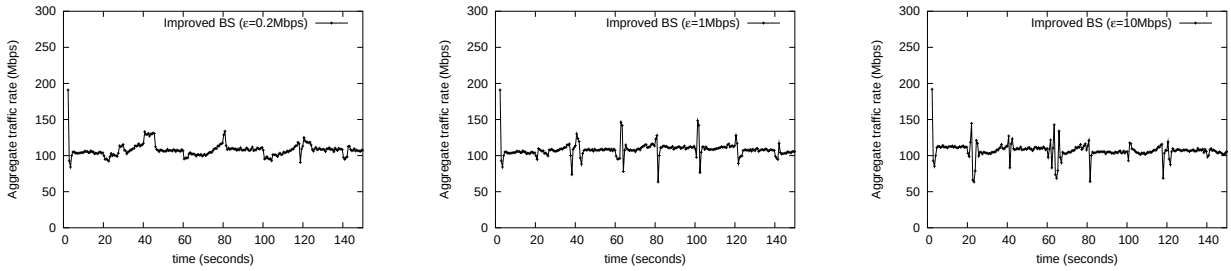

Fig. 11. Improved Binary Search (BS) algorithm with (a) $\epsilon = 0.2M$ ($T_s = 3.7s, J = 27.0Mb$), (b) $\epsilon = 1M$ ($T_s = 1.7s, J = 13.2Mb$), and (c) $\epsilon = 10M$ ($T_s = 2.3s, J = 28.7Mb$).

Experiment D: The effect of parameters U_s and L_s on the fair throttle algorithms: We repeat the second experiment, but increase the range $[L_s, U_s]$ to $[100\text{Mbps}, 125\text{Mbps}]$. Fig. 12 shows the aggregate server load for binary search, PAC ($K_P = 0.5$) and PAFR algorithms, respectively. From EXP B and D, we note that a smaller range of $[L_s, U_s]$ makes binary search unstable, again due to the need for repeated re-initializations and delays. The PAC algorithm is stable, but overshoots remain in the transient response, while the PAFR algorithm maintains its excellent performance with fast convergence time. Generally speaking, a narrower range $[L_s, U_s]$ is more demanding for the throttle algorithms.

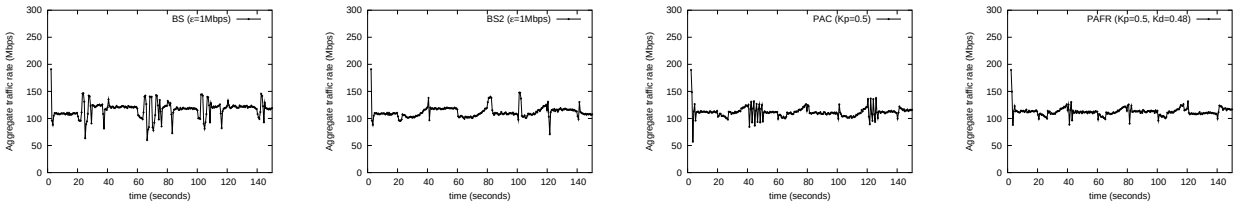

Fig. 12. (a) Binary Search (BS) with $\epsilon = 1M$ ($T_s = 3.9, J = 47.0Mb$). (b) Improved binary search with $\epsilon = 1M$ ($T_s = 1.5s, J = 12.9Mb$). (c) PAC algorithm with $K_p = 1/2$ ($T_s = 3.2, J = 19.4Mb$) (d) PAFR algorithm with $K_p = 0.5, K_D = 0.48$ ($T_s = 1.0s, J = 7.2Mb$). As compared to EXP B, a larger $[L_s, U_s]$ reduces the cost J.

Experiment E: Performance of fair throttle algorithms for heterogeneous network sources with stochastic varying input: In this experiment, we introduce stochastic elements in the load

level of the $R(k)$ routers. The presence of stochastic elements represents closely realistic traffic with background flows in a network, and we can expect to see larger variation in the input with lower stability margin. The delay in each link is uniformly distributed from $1s$ to $3.5s$, and $W = 2s$. There are altogether 50 routers in $R(k)$. Fifteen routers have sinusoidal varying sources–Five with rate of $0.5\sin(0.16t) + 3$Mbps, five with rate of $0.5\sin(0.16t + 0.5\pi) + 1.5$Mbps and five with $\sin(0.16t + 1.5\pi) + 3.5$Mbps. Ten routers have square pulses that vary between 1.5Mbps and 2.5Mbps with a period of $80s$. Twenty five routers have *Gaussian* varying sources–Ten with mean 3Mbps and variance 1Mbps, ten with mean 3.5Mbps and variance 0.5Mbps and five with mean 4Mbps and variance 1Mbps.

We perform the experiment for two scenarios: In the first scenario, S assumes $|R(k)|_{\max}$ as 50. As shown in Fig. 13(a)-(c), the response of the control system is relatively smooth for small K_p in the presence of large delays and large variation in the sources. Larger K_p makes the system sensitive to noise in the load level. Fig. 13(d) shows that *est()* predicts that there are close to 50 effective throttles in the system. The second scenario assumes a smaller $|R(k)|_{\max} = 10$. As shown in Fig. 13(a)-(c) and 14(a)-(c), the responses for $K_P = 0.05$ and $K_P = 0.1$ are relatively smoother than that for $K_P = 0.2$. However, as shown in Fig. 14(d), even though the number of routers can be grossly underestimated, the PAFR algorithm can still function properly and meet the target load. We repeat the experiment for the case of $|R(k)|_{\max} = 50$ and increase the low pass filter time constant to $\psi = 0.95$. As shown in Fig. 15(d), a larger ψ tracks a larger variation in the number of effective throttle routers, but the equilibrium points are not affected much as shown in Fig. 15(a)-(c).

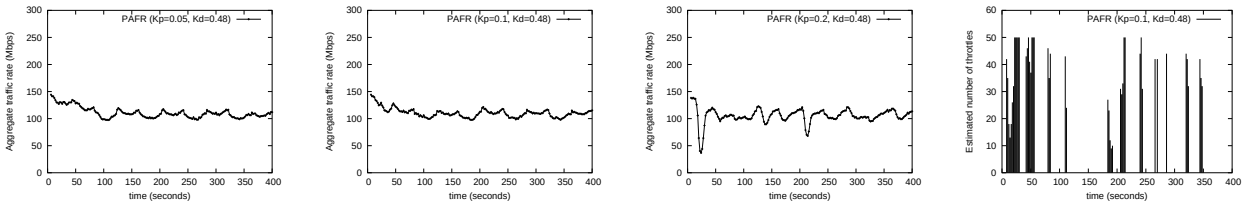

Fig. 13. Performance of the PAFR throttle algorithm with an estimate that $|R(k)|_{\max} = 50$, $K_D = 0.48$ as K_P takes the value of (a) $K_P = 0.05$ ($T_s = 13.4s, J = 92.0Mb$), (b) $K_P = 0.1$ ($T_s = 8.6s, J = 50.6Mb$) and (c) $K_P = 0.2$ ($T_s = 13.5s, J = 71.0Mb$). (d) Estimated number of effective throttles using the *est()* algorithm for the case $K_P = 0.1$.

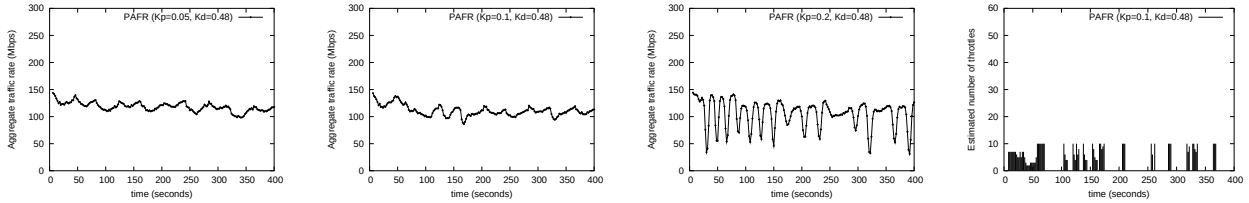

Fig. 14. Performance of the PAFR throttle algorithm with complete information that $|R(k)|_{\max} = 10$, $K_D = 0.48$ as K_P takes the value of (a) $K_P = 0.05$ ($T_s = 35.4s, J = 286.8Mb$), (b) $K_P = 0.1$ ($T_s = 19.5s, J = 145.0Mb$) and (c) $K_P = 0.2$ ($T_s = 60.0s, J = 790.1$). (d) Estimated number of effective throttles using the *est()* algorithm for the case $K_P = 0.1$.

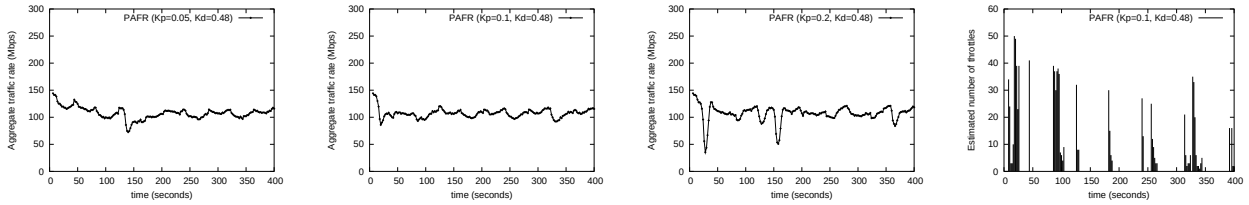

Fig. 15. Performance of the PAFR throttle algorithm using low pass filter time constant $\psi = 0.95$ for *est()* with an estimate of upper bound $|R(k)|_{\max} = 50$ for (a) $K_P = 0.05$ ($T_s = 16.1s, J = 84.6Mb$), (b) $K_P = 0.1$ ($T_s = 12.9s, J = 73.7Mb$) and (c) $K_P = 0.2$ ($T_s = 18.6s, J = 180.0Mb$). (d) Estimated number of effective router for the case of $K_P = 0.1$.

A. **Discussion**

There are three major observations from the experiments above: (1) the proposed throttle algorithms are very effective in reducing the fluctuation when there are many unknown parameters during high volume DDoS attacks, e.g., the number of effective throttles. The PAFR throttle algorithm is more robust than PAC, binary search and AIMD throttle algorithms in minimizing the load variation under all window size we used for traffic intensity measurement. This is very important because it ensures that statistical filtering can be performed with a higher degree of confidence in the measured traffic intensity between consecutive window measurements.

VI. **Related Work**

Handling high bandwidth aggregates using router throttling can be viewed as a congestion control problem. A large body of work on congestion control assumes the feedback is in the form of a binary signal. In our context, we have the luxury of sending an explicit throttling rate to the traffic regulators (routers in this case). So it is simpler for us to achieve maxmin fairness without using AIMD-style distributed algorithms. In this respect, our approach is similar to XCP [14], in being able to separate the problem of fair allocation with the problem of controlling the total aggregate

traffic.

In searching for the simplest control (for total aggregate rate), we consider binary search which is inspired by the analysis in [15]. But this approach quickly becomes more complex and less effective when considering dynamic traffic and delay.

Our analysis is similar to those efforts in analyzing Internet and ATM congestion control with delay based on fluid models, for example, the rate-based control strategy with delayed feedback in [9], the ATM Available Bit Rate (ABR) model in [3], [17] and the TCP model in [11], [18]. The departure is in the detailed model where we use explicit rate feedback and introduce a hysteresis for the desired capacity. Also, unlike congestion control in the Internet, an effective estimator that estimates the number of effective throttles and the role of the window size is crucial for distinguishing legitimate users and attackers.

Defending against DDoS flooding based attack or flash crowds using different rate throttling mechanism at upstream routers can be found in [20], [22]. In [20], rate throttling is achieved using the Indirection Infrastructure, and filtering only at the last hop routers. In [22], a *pushback* max-min fairness mechanism used for controlling large bandwidth aggregates is proposed, which is initiated at the congested source and applied recursively to the upstream routers. However, this may cause upstream routers with low traffic demand to be excessively penalized.

Other router-assisted network congestion control schemes that handle high bandwidth aggregates in a fair manner include detection and dropping using Active Queue Management (AQM) mechanism in congested routers such as balanced RED [1], RED with Preferential Dropping [23], Stochastic Fair Blue [10] and CHOKe [25]. In addition, edge routers that use mechanism such as the edge-based utility mapping in [4] that manipulates utility functions of un-cooperative flows, and the Network Border Patrol (NBP) with Enhanced Core-stateless Fair Queueing (ECSFQ) framework [2] are proposed to control flow aggregates in the core network. However, some of these schemes require either full or partial state information and some schemes do not guarantee strict fairness in the shortest possible time. Besides, these schemes are often much more complex than rate throttling which has low runtime overhead.

VII. **Conclusion**

We analyze how a class of distributed server-centric router throttling algorithms can be used to achieve max-min fairness in a number of routers that are k hops away when the server is overloaded by large bandwidth aggregates that may occur under DDoS attack or flash crowd scenarios. Our class of adaptive throttle algorithms, which are based on a control-theoretic approach, are shown to be highly adaptive to dynamic traffic situations. The stability and convergence issues of these algorithms are also studied. In particular, we show that the PAFR fair throttle algorithm is robust against a wide range of settings – not only does it guarantee convergence, but also it has much smaller over/under-shoot for the aggregate rates to the server S. Level-k max-min fairness is also guaranteed for all the throttling routers in $R(k)$.

Designing throttle algorithms to counter DDoS attacks with limited information is challenging, thus it is not surprising that many issues remain open in this topic. First, it will be desirable to have a quantitative analysis of AIMD, binary search, PAC and PAFR throttle algorithms in terms of convergence rate and transient performance. Second, we can investigate other cost functions for optimal setting, which may lead to other optimal fair throttle algorithms. The effectiveness of various throttle algorithms can thus be examined under a variety of cost functions that correspond to different network scenarios under DDoS attacks. Third, we have taken delay into account in our analysis of our throttle algorithms. The next step would be to test our throttle algorithms in network scenarios under which the deployment depth varies. This has implication on the practical deployment of rate throttle algorithms in a real network as well as providing different lines of defense against DDoS attacks. Related issues on the impact of deployment depth on throttle algorithms can be found in [28].

Future directions of our work include testing our algorithms in networks with diverse delays. We also plan to implement our proposed throttle algorithms in OPERA, which is an open-source programmable router platform that provides flexibility for researchers to experiment security services [24]. Particularly, OPERA allows dynamically loadable software modules to be installed on the fly without shutting down the routing functionalities in the router, which facilitates testing different throttle and statistical filtering algorithms on a security testbed. In practice, information provided by rate throttles such as traffic burstiness may also be incorporated in our throttle algorithms for rate estimation or used in statistical filtering.

Acknowledgement: We thank the editor for coordinating the review process and the anonymous reviewers for their insightful comments and constructive suggestions. We also like to thank Wang Yue for his help in the ns-2 simulation.

APPENDIX

A. **Proof of Theorem 1**

The change in the aggregate input rate is given by $\mathrm{d}y(t)/\mathrm{d}t = K_P(C - y(t-D))$. Let $x(t) = y(t) - C$. Hence, we have $\mathrm{d}x(t)/dt = -K_P x(t-D)$. Theorem 1 can be proved using Nyquist criterion ([19], pp. 330) to yield a sufficient condition. A necessary and sufficient proof that follows the method in [29] is given as follows. For stability, the roots of any characteristic equation must lie on the left half complex plane (See [19], pp. 321). We wish to determine K_P such that the characteristic equation $s + K_P e^{-sD} = 0$ has stable poles with delay from 0 up to D_0, and this set of K_P is denoted as S_R (following the convention in [29]). First, by the Routh-Hurwitz criterion (See [19], pp. 322), the set of K_P's that stabilize the delay-free plant (given as S_0 in [29]) is $S_0 = \{K_P > 0\}$. Second, there exists K_P such that the characteristic equation is stable if $D > 0$. Third, we compute the set $S_L = \{K_P | K_P \notin S_N \text{ and } \exists\ D \in [0, D_0], w \in R,\ s.t.\ jw + K_P e^{-jwD} = 0\}$, which is given by $S_L = \{K_P \geq \pi/(2D_0)\}$. Hence, $S_R = S_0 \backslash S_L$ and thus $S_R = \{0 < K_P < \pi/(2D_0)\}$. But K_P must necessarily be not more than 1 for convergence since the dynamics of the system is a search for a value between the current server load and the capacity C. Hence, $0 < K_P < \min(\pi/2D_0, 1)$. Denote the open loop transfer function as $L(s, K_P, D) = K_P e^{-sD}/s$. For a fixed w and for any $D < D_0$, $\angle L(jw, K_P, D) > \angle L(jw, K_P, D_0)$, i.e., we always have positive phase margins. Hence, the PAC algorithm can stabilize the system for any $D \leq D_0$. To prove convergence of $x(t)$ to zero, note that $|K_P e^{-jsD}/s| \to \infty$ as $s \to 0$. Hence, zero steady state error is ensured in the face of a step disturbance. The tradeoff between system response and delay tolerance can be verified by noting that the closed loop system bandwidth w_B satisfies $K_P < w_B < 2K_P$, and w_B is roughly inversely proportional to the response time of the system in response to a step disturbance.

B. **Proof of Theorem 2**

Note that for the integral J to exist, $x(t)$ must converge to zero in the limit as $t \to \infty$. From Theorem 1, this is true if and only if $0 < K_P < \min(\pi/2D_0, 1)$. Hence, we first assume that any optimal K_P, if it exists, must satisfy Theorem 1. Next, we use Parseval's theorem to compute the integral J in the complex frequency domain (See [19], pp. 583). Hence,

$$J = \frac{1}{j2\pi} \int_{-j\infty}^{j\infty} X(s)X(-s)\mathrm{d}s$$

where $X(s)$ denotes the Laplace transform of the mismatch $x(t)$. Ignoring initial conditions, we let $X(s) = 1/(s + K_P e^{-sD})$ and $X(s) = 1/(s + (K_P + K_D s)e^{-sD})$ for the PAC and PAFR algorithm respectively. For the PAC algorithm, we obtain

$$J = \frac{\tan K_P D + \sec K_P D}{2K_P} = \frac{\tan(K_P D/2 + \pi/4)}{2K_P}.$$

To yield the optimal K_P, we require that $\mathrm{d}J/\mathrm{d}K_P = 0$. Solving for the stationary point, we have $\cos K_P D = K_P D$. Using Taylor's series approximation, we have $1 - (K_P D)^2/2 \approx K_P D$. Finally, we have $K_{P,opt} = K_P = (\sqrt{3}-1)/D$. Since $\mathrm{d}^2J/\mathrm{d}K_P^2 = (1+\sqrt{1-(K_{P,opt}D)^2})/(2K_{P,opt}^2 D) > 0$, J is indeed the minimum. Moreover, the set of $K_{P,opt}$ for all D lies within the set of K_P given by Theorem 1, thus we have a necessary and sufficient result that $K_{P,opt}$ stabilizes the system for all delay $D < D_0$.

C. **Proof of Theorem 3**

For the PAFR algorithm, the integral J exists since $|(K_P + K_D s)e^{-jsD}/s| \to \infty$ as $s \to 0$. The set of controller parameters K_P and K_D must satisfy the sufficient and necessary condition for asymptotic stability which can be obtained by the approach in [29] and the Nyquist criterion to yield $S_{PD} = \{0 < K_P < 1, \text{ and } K_P D/\sqrt{1-K_D^2} < \arccos(-K_D)\}$. Hence the optimal set of control parameters $K_{P,opt}$ and $K_{D,opt}$ must lie in the set, S_{PD}. For the PAFR algorithm, we have

$$J = \frac{1}{2K_P\sqrt{1-K_D^2}} \frac{K_P \cos\kappa D + K_D \kappa \sin\kappa D}{\kappa - K_P \sin\kappa D + \kappa K_D \cos\kappa D}$$

where $\kappa = K_P/\sqrt{1-K_D^2}$, which can be simplified as

$$J = \frac{D}{2\Psi \sin^2 \upsilon} \tan(\frac{\upsilon + \Psi}{2})$$

where $\Psi = K_P D/\sqrt{1-K_D^2}$ and $\upsilon = \arccos K_D$. To yield the optimal K_P and K_D, we have $\partial J/\partial \upsilon = 0$ and $\partial J/\partial \Psi = 0$ for $0 < \upsilon < \pi$ and $0 < \Psi < \pi - \upsilon$. Finally, we have $\Psi = \sin(\upsilon + \Psi)$ and $\Psi = 0.5 \tan \upsilon$ which is a pair of transcendental set of equations. The numerical solution for the optimal Ψ_{opt} is $\Psi = 0.92$, and since $\Psi_{opt} = 0.5 \tan \upsilon$, we have

$$K_P = \frac{2\Psi_{opt}^2}{D\sqrt{1+4\Psi_{opt}^2}} \quad \text{and} \quad K_D = \frac{1}{\sqrt{1+4\Psi_{opt}^2}}.$$

Hence, we obtain $K_P \approx 0.8/D$ and $K_D \approx 0.48$. We invoke the same steps as before by fixing w and for all $D \leq D_0$, we always have a loop transfer function with positive phase margin. Hence, the theorem is proved.

D. Proof of Corollary 1

For a fixed D, from Theorem 2, the cost function for the PAC algorithm is

$$J_P = (1 + \sqrt{1-(K_{P,opt}D)^2})/2K_{P,opt}^2 D$$

which is approximately $1.6D$, and, for the PAFR algorithm, from Theorem 3, $\kappa = K_{P,opt}/\sqrt{1-K_{D,opt}^2} \approx 0.91/D$, hence the cost function, $J_{PD} \approx 1.1D$. Thus, for all $D > 0$, $J_P > J_{PD}$.

REFERENCES

[1] F. M. Anjum and L. Tassiulas, "Fair bandwidth sharing among adaptive and non-adaptive flows in the internet", *Proc. of IEEE INFOCOM'99*, New York, March 1999.

[2] C. Albuquerque, B. J. Vickers and T. Suda, "Network border patrol: preventing congestion collapse and promoting fairness in the internet", *IEEE/ACM Transactions on Networking*, vol. 12, No. 1, pp. 173-186, February 2004.

[3] L. Benmohamed and S. M. Meerkov, "Feedback control of congestion in packet switching networks: The case of a single congested node", *IEEE/ACM Transactions on Networking*, vol. 1, No. 6, pp. 693-708, December 1993.

[4] K. Chandrayana and S. Kalyanaraman, "Un-cooperative congestion control", *Proc. of ACM SIGMETRICS*, June 2004.

[5] D. M. Chiu and R. Jain, "Analysis of the increase/decrease algorithms for congestion avoidance in computer networks", *Journal of Computer Networks and ISDN*, 17(1), pp. 1-14, June 1989.

[6] T. H. Cormen, C. E. Leiserson and R. L. Rivest, "Introduction to Algorithms", *Second Edition*, MIT Press, 2001.

[7] L. Garber, "Denial-of-Service attacks rip the internet", *IEEE Computer*, 3(4), pp 12-17, April 2000.

[8] L. Feinstein, D. Schnackenberg, R. Balupari and D. Kindred, "Statistical approaches to DDoS attack detection and response", *Proc. of DARPA Information Survivability Conference and Exposition*, pp. 303-314, April 2003.

[9] K. W. Fendick, M. A. Rodrigues, and A. Weiss, "Analysis of a rate-based control strategy with delayed feedback", *Proc. of ACM SIGCOMM*, pp. 136-148, September 1992.

[10] W. Feng, K. Shin, D. Kandlur and D. Saha, "Stochastic Fair Blue: A queue management algorithm for enforcing fairness", *Proc. of IEEE INFOCOM'01*, New York, April 2001.

[11] C. V. Hollot, V. Misra, D. Towsley and W. Gong, "Analysis and design of controllers for AQM routers supporting TCP flows", *IEEE Transactions on Automatic Control*, 47(6), pp. 945-959, June 2002.

[12] K. Y. K. Hui, J. C. S. Lui and D. K. Y. Yau, "Small-world overlay P2P networks", *Proc. of the IEEE International Workshop on Quality of Service (IWQoS)*, Montreal, June 2004.

[13] A. Hussain , J. Heidemann , C. Papadopoulos, "A framework for classifying denial of service attacks", *Proc. of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, Karlsruhe, Germany, August 2003.

[14] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks", *Proc. of ACM SIGCOMM*, pp. 89-102, August 2002.

[15] R. Karp, E. Koutsoupias, C. Papadimitriou and S. Shenker, "Optimization problems in congestion control", *Proc. of the 41 st Annual IEEE Computer Society Conference on Foundations of Computer Science*, California, November 2000.

[16] Y. Kim, W. C. Lau, M. C. Chuah and J. H. Chao, "PacketScore: Statistics-based overload control against distributed denial-of-service attacks", *Proc. of IEEE INFOCOM'04*, Hong Kong, March 2004.

[17] A. Kolarov and G. Ramamurthy, "A control-theoretic approach to the design of an explicit rate controller for ABR service", *IEEE/ACM Transactions on Networking*, 7(5), pp. 741-753, October 1999.

[18] S. Kunniyur and R. Srikant, "Analysis and design of an adaptive virtual queue algorithm for active queue management", *Proc. of ACM SIGCOMM*, pp. 123-134, August 2001.

[19] B. C. Kuo, "Automatic Control Systems", Prentice Hall, 1975.

[20] K. Lakshminarayanan, D. Adkins, A. Perrig and I. Stoica, "Taming IP packet flooding attacks", *Proc. of the Workshop on Hot Topics in Networks*, Cambridge, MA, November 2003.

[21] Q. Li, E. Chang and M. C. Chan , "On the effectiveness of DDoS Attacks on statistical filtering", *Proc. of IEEE INFOCOM'05*, Miami, FL, March 2005.

[22] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson and S. Schenker, "Controlling high bandwidth aggregates in the network", *ACM SIGCOMM Computer Communication Review*, Vol. 32, No. 3, pp. 62-73, July 2003.

[23] R. Mahajan, S. Floyd and D. Wetherall, "Controlling high-bandwidth flows at the congested router", *Proc. of the 9th IEEE International Conference on Network Protocols (ICNP)*, Riverside, C. A., November 2001.

[24] OPERA: an open-source router platform, available at http://www.cse.cuhk.edu.hk/∼cslui/ANSRlab/software/opera/.

[25] R. Pan, B. Prabhakar and K. Psounis, "CHOKe: a stateless AQM scheme for approximating fair bandwidth allocation", *Proc. of IEEE INFOCOM'00*, Tel Aviv, Israel, March 2000.

[26] Network Simulator 2.0, available at http://www-nrg.ee.lbl.gov/ns/.

[27] H. Wang, D. Zhang and K. G. Shin, "Detecting SYN flooding attacks", *Proc. of IEEE INFOCOM'02*, New York, NY, June 2002.

[28] D. K. Y. Yau, J. C. S. Lui, and F. Liang, "Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles", *Proc. of IEEE IWQoS*, Miami, FL, USA, May 2002.

[29] H. Xu, A. Datta and S. P. Bhattacharyya, "PID stabilization of LTI plants with time-delay" *Proc. of the 42nd IEEE Conference on Decision and Control*, Hawaii, December 2003.