Defending Against Distributed Denial-of-service Attacks with Max-min Fair Server-centric Router Throttles

Abstract

We present a network architecture and accompanying algorithms for countering distributed denial-of-service (DDoS) attacks directed at an Internet server. The basic mechanism is for a server under stress to install a *router throttle* at selected upstream routers. The throttle can be the leaky-bucket rate at which a router can forward packets destined for the server. Hence, before aggressive packets can converge to overwhelm the server, participating routers *proactively* regulate the contributing packet rates to more moderate levels, thus forestalling an impending attack. In allocating the server capacity among the routers, we propose a notion of *level-k max-min fairness*. We present a control-theoretic model to evaluate algorithm convergence under a varitey of system parameters. In addition, we present packet network simulation results using a realistic global network topology, and various models of good user and attacker distributions and behaviors. Using a generator model of web requests parameterized by empirical data, we also evaluate the impact of throttling in protecting user access to a web server. First, for *aggressive* attackers, the throttle mechanism is highly effective in preferentially dropping attacker traffic over good user traffic. In particular, level-k max-min fairness gives better good-user protection than recursive pushback of max-min fair rate limits proposed in the literature. Second, throttling can regulate the experienced server load to below its design limit – in the presence of user dynamics – so that the server can remain operational during a DDoS attack.

I. Introduction

In a distributed denial-of-service (DDoS) attack (e.g., [1], [2]), a cohort of malicious or compromised hosts (the "zombies") coordinate to send a large volume of aggregate traffic to a victim server. In such an episode, it is likely that network nodes near the edge will progressively become more vulnerable to resource overruns as their distance from the server decreases. There are two reasons. First, a node that is closer to the server will likely have less service capacity because it is closer to the network edge, and is designed to handle fewer users. Second, such a node will generally see a larger fraction of the attack traffic, which has gone through more aggregation inside the network. In particular, the server system itself is highly vulnerable, and can become totally incapacitated under extreme overload conditions.

We view DDoS attacks as a resource management problem. Our goal in this paper is to protect a server system from having to deal with excessive service request arrivals over a global network. (However, the approach can be easily generalized to protecting an intermediate routing point under overload.) To do so, we adopt a *proactive* approach: Before aggressive packets can converge to overwhelm a server, we ask routers along forwarding paths to regulate the contributing packet rates to more moderate levels, thus forestalling an impending attack. The basic mechanism is for a server under stress, say S, to install a *router throttle* at an upstream router several hops away. The throttle limits the rate at which packets destined for S will be forwarded by the router. Traffic that exceeds the rate limit can either be dropped or rerouted to an alternate server, although we will focus exclusively on the dropping solution in this paper.

A key element in the proposed defense system is to install appropriate throttling rates at the distributed routing points, such that, globally, S exports its full service capacity U_S to the network, but no more. The "appropriate" throttles should depend on the current demand distributions, and so must be negotiated dynamically between server

Research supported in part by the National Science Foundation under grant number CCR-9875742 (CAREER)

and network. Our 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 and activating a rate throttle at a subset of its upstream routers. After that, if the current throttle fails to bring down the load at S to below U_S, then the throttle rate is reduced[1]. 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 (i.e., relaxed). If an increase does not cause the load to significantly increase over some observation period, then the throttle is removed. The goal of the control algorithm is to keep the server load within $[L_S, U_S]$ whenever a throttle is in effect.

Since throttling requires only looking up the IP destination address of a packet, it has essentially the same processing complexity as standard IP forwarding, and adds little computational overhead at a deployment router. The amount of state information a router has to keep per throttle is a few bytes, for storing the destination IP address and the throttle value. However, since the total amount of state information needed at a router is linear in the number of installed throttles, we cannot expect the routers to maintain state about every Internet server. However, the approach can be feasible as an on-demand and selective protection mechanism. The premise is that DDoS attacks are the exception rather than the norm. At any given time, we expect at most only a minor portion of the network to be under attack, while the majority remaining portion to be operating in "good health". Moreover, rogue attackers usually target "premium sites" with heavy customer utilization, presumably to cause maximal user disruptions and to generate the most publicity. These selected sites may then elect to protect themselves in the proposed architecture, possibly by paying for the offered services.

A. Our contributions

Our contributions in this paper are:

- We contribute to the fundamental understanding of router throttling as a mechanism against DDoS attacks. In particular, we advance a control-theoretic model useful for understanding system behavior under a variety of parameters and operating conditions.
- We present an adaptive throttle algorithm that can effectively protect a server from resource overload, and increase the ability of good user traffic to arrive at the intended server.
- We show how max-min fairness can be achieved across a potentially large number of flows, and the implication of a notion of *level-k max-min fairness* on DDoS attacks.
- We study how throttling may impact real application performance. Specifically, we demonstrate via simulations the performance impact on an HTTP web server.

B. Paper organization

The balance of the paper is organized as follows. In Section II, we introduce our system model. In Section 3, we formally specify a baseline and a fair algorithm for computing throttle rates. After defining our solution approach, we compare it with related work in the literature, in Section IV. In Section V, we present a control-theoretic mathematical model for understanding system performance under a variety of parameters and operating conditions. To further examine system performance under detailed packet network models, Section VI presents diverse ns2 simulation results using a

[1] Notice that *reducing* the throttle rate *increases* the extent of throttling, because a router will more restrict traffic destined for S.

realistic network topology. In Section VII, we discuss several issues about the practical deployment of our solution. Section VIII concludes.

II. System Model

We begin by stating Convention 1 that simplifies our presentation throughout the rest of the paper. Then, we go on to describe our system model.

Convention 1: **All traffic rate and server load quantities stated in this paper are in units of kb/s, unless otherwise stated.**

We model a network as a connected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. All leaf nodes are hosts and thus can be a traffic source. An internal node is a router; a router cannot generate traffic, but can forward traffic received from its connected hosts or peer routers. We denote by R the set of internal routing nodes. All routers are assumed to be trusted. The set of hosts, $H = V - R$, is partitioned into the set of ordinary "good" users, H_g, and the set of attackers H_a. E models the network links, which are assumed to be bi-directional. Since our goal is to investigate control against *server* resource overload, each link is assumed to have infinite bandwidth. The assumption can be relaxed if the control algorithm is also deployed to protect routers from overload.

In our study, we designate a leaf node in V as the target server S. A good user sends packets to S at some rate chosen from the range $[0, r_g]$. An attacker sends packets to S at some rate chosen from the range $[0, r_a]$. In principle, while r_g can usually be set to a reasonable level according to how users normally access the service at S (and we assume $r_g \ll U_S$), it is hard to prescribe constraints on the choice of r_a. In practice, it is reasonable to assume that r_a is significantly higher than r_g. This is because if every attacker sends at a rate comparable to a good user, then an attacker must recruit or compromise a large number of hosts to launch an attack with sufficient traffic volume.

When S is under attack, it initiates the throttle defense mechanism outlined in Section I. (For ease of presentation, we assume that an overloaded server is still capable of initiating the defense actions. However, as discussed in Section VII, the assumption can be relaxed in practice.) The throttle does not have to be deployed at every router in the network. Instead, the deployment points are parameterized by a positive integer k and are given by $R(k) \subseteq R$. Specifically, $R(k)$ contains all the routers that are either k hops away from S or less than k hops away from S but are directly connected to a host.

Fig. 1 shows an example network topology. In the figure, a square node represents a host, while a round node represents a router. The host on the far left is the target server S. The routers in $R(3)$ are shaded in the figure. 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.

Given our system model, an important research problem is how to achieve *fair* rate allocation of the server capacity among the routers in $R(k)$. To that end, we define the following notion of *level-k max-min fairness*:

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 $L_S \leq r \leq U_S$.

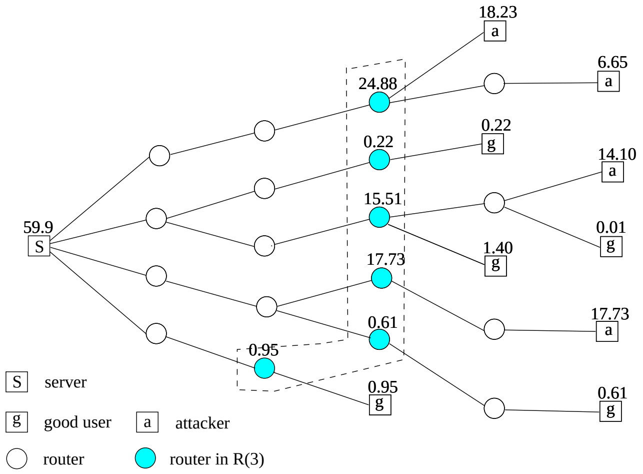


Fig. 1. Network topology illustrating $R(3)$ deployment points of router throttle, and offered traffic rates.

III. Throttle Algorithms

A. Baseline algorithm

We present a baseline algorithm in which each router throttles traffic for S by forwarding only a fraction f $(0 \leq f \leq 1)$ of the traffic. The fraction f is taken to be one when no throttle for S is in effect. In adjusting f according to current server congestion, the algorithm mimics TCP congestion control. Specifically, f is reduced by a multiplicative factor α when S is congested and sends the router a *rate reduction signal*. It is increased by an additive constant β – subject to the condition that $f \leq 1$ – when S has extra capacity and sends the router a *rate increase signal*.

The baseline algorithm that S runs is specified in Fig. 2. It is to be invoked when either (i) the current server load (measured as traffic arrival rate to S) crosses U_S, or (ii) a throttle is in effect and the current server load drops below L_S. In case (i), S multicasts a rate reduction signal to $R(k)$; in case (ii), it multicasts a rate increase signal. The algorithm can take multiple rounds until a server load within $[L_S, U_S]$ is achieved. Also, if the server load is below L_S, and the next rate increase signal raises the server load by an insignificant amount (i.e., by less than ϵ), we remove the throttle. The monitoring window w should be set to be somewhat larger than the maximum round trip time between S and a router in $R(k)$.

In the example network shown in Fig. 1, let the number above each host (except S) denote the current rate at which the host sends traffic to S. The number above each router denotes the offered rate of traffic at the router, destined for S. Also, let $L_S = 18$, $U_S = 22$, $\alpha = 1/2$, and $\beta = 0.05$. Initially, the total offered load to S exceeds U_S, and hence the baseline throttle algorithm is invoked at S. A rate reduction signal causes each router to drop half of the traffic for S, resulting in a server load of 29.95, still higher than U_S. The next rate reduction signal causes the server load to drop below L_S, at 14.975 and a rate increase signal to be sent, raising the server load to 17.97. Finally, another rate increase signal raises the server to 20.965, which is within $[L_S, U_S]$.

Table III-A shows how f and the server load changes at each round of the algorithm. When the algorithm terminates, the forwarding rates at the deployment routers (from top to bottom of the figure) are 8.708, 0.077, 5.4285, 6.2055, 0.2135 and 0.3325, respectively. The algorithm achieves a server load within the target range. However, it does *not* achieve level-k max-min fairness, since some router is given a higher rate than another router, even though the latter has unmet demands.

```
Algorithm baseline_throttle
ρ_last := −∞;
while (1)
    monitor traffic arrival rate ρ for time window w;
    if (ρ > U_S) /* throttle not strong enough */
        /* further restrict throttle rate */
        multicast reduction signal to R(k)
    elif (ρ < L_S) /* throttle too strong */
        if (ρ − ρ_last < ε)
          remove rate throttle from R(k);
          break;
        else
          /* try relaxing throttle at the routers */
          multicast increase signal to R(k);
        fi;
    else
        break;
    fi;
end while;
```

Fig. 2. Baseline throttle algorithm specification.

Round	f	server load
1	1.0	59.9
2	0.5	29.95
3	0.75	14.975
4	0.7	17.97
5	0.65	20.965

TABLE I

Trace of the throttle fraction f and server load for the baseline algorithm.

Algorithm fair_throttle

```
ρ_last := −∞;
while (1)
    multicast 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 − ρ_last < ε)
          remove rate throttle from R(k);
          break;
        else
          /* try relaxing throttle by additive step */
          ρ_last := ρ;
          r_S := r_S + δ;
        fi;
    else
        break;
    fi;
end while;
```

Fig. 3. Fair throttle algorithm specification.

B. Fair throttle algorithm

The baseline algorithm is not fair because it penalizes all routers equally, irrespective of whether they are greedy or well behaving. We now present a fair throttle algorithm that installs at each router in $R(k)$, a uniform leaky bucket rate (i.e. the throttle rate) at which the router can forward traffic for S. Fig. 3 specifies the algorithm by which S determines the throttle rate to be installed. In the specification, 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 typically needed in $R(k)$. We use a constant additive step, δ, to ramp up r_S if a throttle is in effect and the current server load is below L_S.

The fair throttle algorithm is to be invoked as with the baseline algorithm. Each time it is called, it multicasts a rate-r_S throttle to $R(k)$. This will cause a router in $R(k)$ to regulate traffic destined for S to a leaky bucket with rate r_S. The algorithm may then continue in the while loop that iteratively adjusts r_S to an appropriate value. Notice that the additive increase/multiplicative decrease iterative process aims to keep the server load in $[L_S, U_S]$ whenever a throttle is in effect. The termination conditions and choice of w in the fair algorithm are the same as in the baseline algorithm.

We apply the fair throttle algorithm to the previous example scenario in Fig. 1. We initialize r_S to $(L_S+U_S)/2 = 10$,

Round	r_S	server load
1	10	31.78
2	5	16.78
3	6	19.78

TABLE II

Trace of throttle rate and achieved server load for the fair algorithm.

and use an additive step of one. Table III-B shows how r_S and the aggregate server load evolve. 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 figure) 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).

Since the fair throttle algorithm achieves our fairness objective whereas the baseline algorithm does not, we will focus only on the fair algorithm for the rest of the paper.

IV. Related Work

Probabilistic IP marking is advanced by Savage et al [11] to identify attackers originating a denial-of-service attack, in spite of source address spoofing. The analysis in [10] confirms the remark in [11] that their form of IP traceback may not be highly effective for *distributed* DoS attacks. Subsequently, Song and Perrig [12] improves upon the information convergence rate that allows to reconstruct the attack graph (by eliminating false positives when markers can be fragmented across packets), and reduces the time overhead in the reconstruction process itself, for DDoS attacks. These algorithms expose the true attackers, which supposedly facilitates defense actions that can then be taken to curtail an attack. However, the required defense mechanisms are external to IP trackeback, which in and of itself offers no active *protection* for a victim server.

To actively defend against attacks, analysis of routing information can enable a router to drop certain packets with spoofed source address, when such a packet arrives from an upstream router inconsistent with the routing information. The approach requires sophisticated and potentially expensive routing table analysis on a per-packet basis. Also, it is not necessary for attackers to spoof addresses in order to launch an attack. The latter observation also limits the effectiveness of ingress filtering approaches [5].

A defense approach most similar to ours is proposed by Mahajan et al [9]. They describe a general framework for identifying and controlling high bandwidth *aggregates* in a network. As an example solution against DDoS attacks, an aggregate can be defined based on destination IP address, as in our proposal. To protect good user traffic from attacker traffic destined for the same victim server, they study recursive *pushback* of max-min fair rate limits starting from the victim server to upstream routers. Similar to level-k max-min fairness, pushback defines a *global*, cross-router notion of max-min fairness. Unlike level-k max-min fairness, the pushback mechanism always starts the resource sharing decision at the server, where good user traffic may have aggregated to a large volume and thus can be severely punished (see

Section VI-B). Such aggregation of normal user traffic has been observed to occur in practice [4].

Architecturally, our control algorithm is more of an end-to-end approach initiated by the server, whereas the proposal in Mahajan et al [9] is more of a hop-by-hop approach in which routers participate more heavily in the control decisions. Hence, our routers have simplified responsibilities, when compared with [9] – they do not need to compute server-centric max-min fair allocations, and are not required to generate and send back *status messages* to the server.

The use of authentication mechanisms inside the network will also help defend against DDoS attacks, e.g. IPsec [7]. Recently, Gouda et al [6] propose a framework for providing *hop integrity* in computer networks. Efficient alogrithms for authentication and key exchanges are important research questions in this class of solutions.

Lastly, our solution aims to achieve max-min fairness across a potentially large number of flows. Scalable max-min fair allocation in such a situation is studied in [3], where the optimal sharing objective is relaxed to achieve substantial reductions in overhead.

V. General Mathematical Model

Router throttling is a feedback control strategy. To better understand its stability and convergence behavior, we formulate its control-theoretic model. Using the model, we explore how different system parameters, including feedback delays, the hysteresis control limits, and the number and heterogeneity of traffic sources, can impact system performance. We point out that our mathematical model can also provide a general framework for studying various multi-source flow control problems.

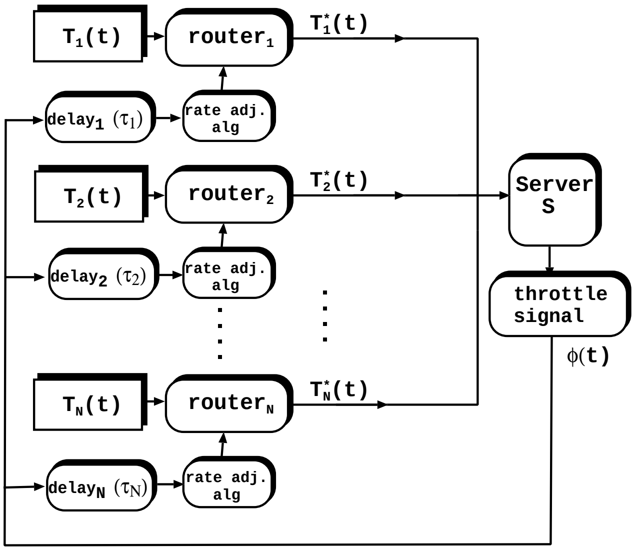


Fig. 4. High-level description of mathematical model for router throttling.

Figure 4 gives a high-level description of our mathematical model for router throttling. We model each deployment router as a source of traffic for S, where S is the server to be protected. Let there be N sources and $T_i(t)$ be the instantaneous *offered* traffic rate i has for S at time t. Given a throttle algorithm and a *throttle signal* $\phi(t)$ from S, i forwards traffic for S at an instantaneous rate $T_i^*(t)$. The instantaneous *forwarding* rate $T_i^*(t)$ is a function of the offered traffic rate, $T_i(t)$, and a throttle rate $r_i(t)$ computed by a *rate adjustment module* deployed at i, according to the throttle algorithm used.

Given $T_i^*(t)$ from each deployment router (i.e., each router in $R(k)$), S receives an aggregate traffic rate of $\sum_{i=1}^{N} T_i^*(t)$.

Based on the aggregate rate, S computes and sends the throttle signal $\phi(t)$ to all the routers in $R(k)$. Notice that the throttle signal may arrive at different routers at *different* times, because we model *heterogeneous delays* from S to each router. Specifically, we let $\tau_i \geq 0$ denote the network delay from S to i. We use a set of coupled differential equations to model the dynamics of how the throttle signal $\phi(t)$, the throttle rate $r_i(t)$, and the forwarding traffic rates $T_i^*(t)$, for $i = 1, \ldots, N$, change over time.

A. Model for baseline algorithm

We first consider the baseline algorithm. The server generates the throttle signal $\phi(t)$ as a function of the aggregate traffic workload and the hysteresis control limits L_S and U_S. We have:

$$\phi(t) = \begin{cases} -1 & \text{if } \sum_{i=1}^{N} T_i^*(t) \geq U_S \\ 1 & \text{if } \sum_{i=1}^{N} T_i^*(t) \leq L_S. \end{cases} \tag{1}$$

Note that when the aggregated traffic rate is within $[L_S, U_S]$, the throttle signal will be off (e.g, $\phi(t) = 0$). Upon receiving the throttle signal $\phi(t)$, each router will adjust its throttle rate $r_i(t)$. The differential equation for $r_i(t)$, where $i \in \{1, 2, \ldots, N\}$, is:

$$\frac{dr_i(t)}{dt} = \beta_i \mathbf{1}_{(\phi(t-\tau_i)==1)} - \frac{r_i(t)}{2} \mathbf{1}_{(\phi(t-\tau_i)==-1)} \tag{2}$$

where $r_i(0) = 1$ and $\beta_i > 0$ is the incremental step size for router i. Note that if router i receives a throttle signal $\phi(t - \tau_i)$ of 1, it implies that the aggregate traffic rate at the server is below L_S, and therefore router i increases the fraction of forwarded traffic by β_i. On the other hand, if the received throttle signal $\phi(t-\tau_i)$ is equal to -1, this implies that the aggregate traffic rate at the server is above U_S, and therefore, router i reduces the fraction of forwarded traffic by half. The forwarding traffic rate at router i is given by:

$$T_i^*(t) = \begin{cases} T_i(t) r_i(t) & \text{if } r_i(t) < 1, \\ T_i(t) & \text{if } r_i(t) \geq 1. \end{cases} \tag{3}$$

It is important to note that router i can potentially forward the full received traffic rate $T_i(t)$ from upstream hosts to S.

B. Model for fair throttle algorithm

Let us now consider the fair throttle algorithm. In this case, the server generates a throttle signal $\phi(t)$ as the throttle rate $r_S(t)$, which is a function of the aggregate server workload, the hysteresis control limits L_S and U_S, and the additive step size $\delta > 0$. The differential equation expressing the change in the throttle rate is:

$$\frac{dr_S(t)}{dt} = \delta \mathbf{1}_{\left(\sum_{i=1}^{N} T_i^*(t) \leq L_S\right)} - \frac{r_S(t)}{2} \mathbf{1}_{\left(\sum_{i=1}^{N} T_i^*(t) \geq U_S\right)} \tag{4}$$

Essentially, when the server discovers that the aggregate traffic is below L_S, it will increase the throttle rate $r_S(t)$ by δ. Otherwise, if the aggregate traffic is above U_S, it will reduce the throttle rate $r_S(t)$ by half. The objective is to achieve an aggregate server load within $[L_S, U_S]$.

Upon receiving the throttle signal $\phi(t-\tau_i)$, router i adjusts its forwarding rate, $T_i^*(t)$, of traffic for S. The differential equation expressing the change in $T_i^*(t)$ can be given as:

$$\frac{dT_i^*(t)}{dt} = \min\{r_S(t-\tau_i), T_i(t)\} - T_i^*(t) \tag{5}$$

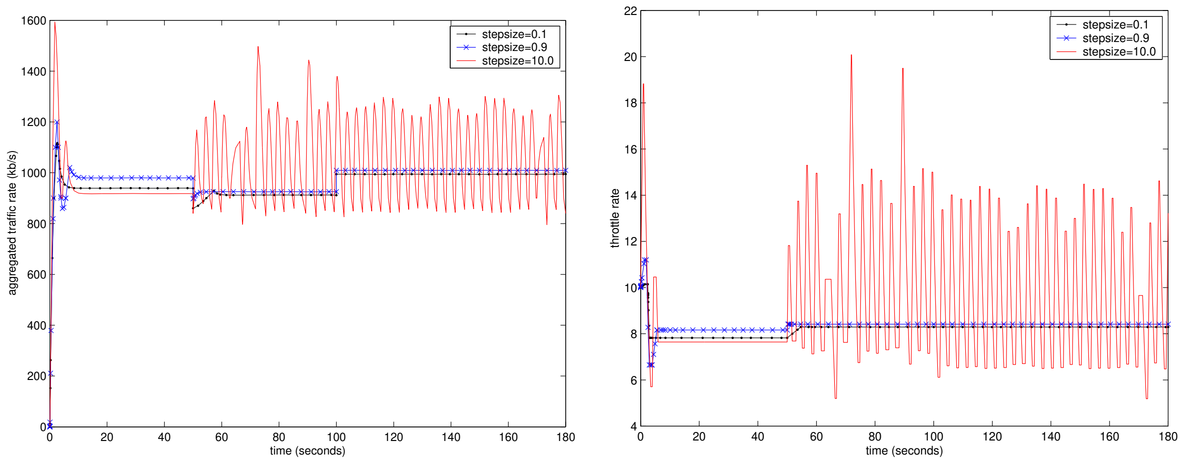
for $i=1,\ldots,N$ and $T_i^*(0)=0$. Note that the rate of change of the forwarding traffic rate $T_i^*(t)$ is a function of the throttle rate $r_S(t-\tau_i)$ and the offered traffic rate $T_i(t)$. If the throttle rate $r_S(t-\tau_i)$ is larger than the offered traffic rate, then there is no need to throttle and the change is simply $T_i(t)-T_i^*(t)$. On the other hand, if $r_S(t-\tau_i)$ is smaller than $T_i(t)$, then we throttle and the change in the forwarding traffic rate is $r_S(t-\tau_i)-T_i^*(t)$.

C. Mathematical Analysis

We now study the *"stability"* and *"convergence"* properties of router throttling. Because of the lack of space, we only present results for the fair algorithm. In our experiments, we consider 100 *heterogeneous* sources. The first eighty are constant sources wherein $T_i(t)=30$ for $i=1,\ldots,80$. In each experiment, ten of these constant sources are switched off at $t=50$ and are activated again at $t=100$. The network delay between S and each of the constant sources is 100 ms. The next ten sources are sinusoidal sources wherein $T_i(t)=10\sin(0.4t)+30$ for $i=81,\ldots,90$. The network delay for each of these sinusoidal sources is 50 ms. The last ten sources are square-pulse sources wherein

$$T_i(t)=\begin{cases}50 & \text{for } 20n\le t<10(2n+1)\\ 10 & \text{for } 10(2n+1)\le t<20(n+1)\end{cases}$$

for $i=91,\ldots,100$ and $n\in\{0,1,2,\ldots\}$. The network delay for each of these square-pulse sources is 50 ms.



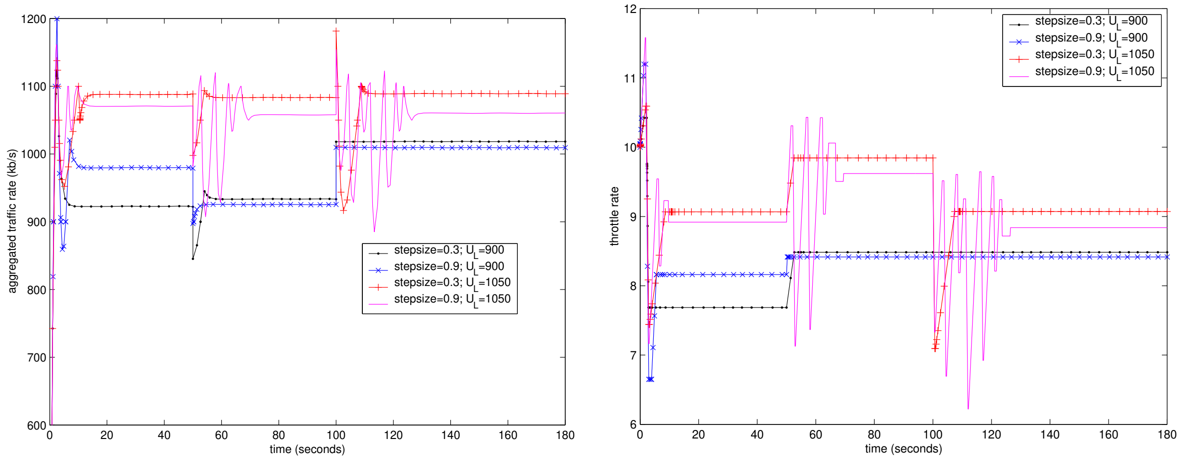
(a) Aggregate workload at server S

(b) Throttle rate $r_S(t)$

Fig. 5. System performance for $U_S=1100, L_S=900$, and various δ step sizes.

Experiment 1: *Handling of heterogeneous sources and system stability.* Figure 5 illustrates the results for the first experiment where $L_S=900$ and $U_S=1100$. We consider three different step sizes, namely $\delta=0.1,0.9,10.0$. We make two important observations about the results: (1) The proposed fair algorithm is effective in keeping the server load within the target limits, under *heterogeneous sources* and *heterogeneous network delays*, and (2) the additive step size δ can affect system stability. As shown, system performance is not stable for the large step size of $\delta=10$. Hence, we recommend a small step size, relative to U_S-L_S, so that the system can operate in a stable region.

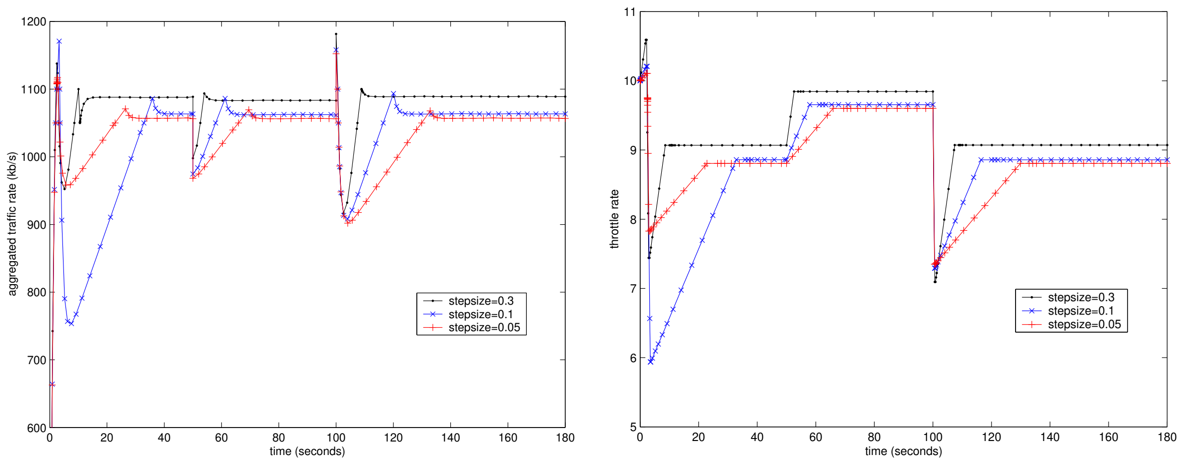
Experiment 2: *Determination of step size δ for a stable system.* Figure 6 illustrates the results of our second experiment where $U_S=1100$ and L_S can be 900 or 1050. We observe that when U_S-L_S is large, the system is stable with $\delta<1$, and the achieved server workload at convergence is slightly above 1000. On the other hand, when S advertises a smaller target load region, with $L_S=1050$ and $U_S=1100$, we need a smaller step size (e.g, $\delta\le0.3$) to have stable performance,



(a) Aggregate workload at server S (b) Throttle rate $r_S(t)$

Fig. 6. System performance for $U_S = 1100$ and $L_S = 900$ or 1050, and various δ step sizes.

and the achieved server workload at convergence is closer to U_S. After experimenting with a large number of different step sizes and many different system configurations, we recommend a step size of $\delta < 0.5$.



(a) Aggregate workload at server S (b) Throttle rate $r_S(t)$

Fig. 7. System performance for U_S=1100 and $L_S = 1050$, and various δ step sizes.

Experiment 3: *Effect of δ on the convergence rate.* Figure 7 illustrates the results of our third experiment in which we consider how δ can affect the convergence speed. In the experiment, $L_S = 1050$ and $U_S = 1100$. We experiment with three different step sizes, namely $\delta = 0.3, 0.1, 0.05$. Although the system is stable for all the three step sizes, we observe that if a step size is too small, it takes *longer* for the system to converge. For example, when ten constant sources are activated at $t = 100$, the system converges around $t = 110$ for $\delta = 0.3$. On the other hand, if we use $\delta = 0.05$, the system converges around $t = 137$. Another important point is that if δ is smaller, the achieved server workload at convergence is also smaller. Therefore, in order to have a stable system and, at the same time, achieve a high server workload, we recommend δ to be between 0.1 and 0.3.

VI. Packet Network Simulation Results

Our general, high-level control-theoretic results provide basic understanding about algorithm stability and convergence. To further examine system performance, under detailed packet network models (including both unreliable UDP and reliable TCP communication), we conduct experiments using the ns2 simulator.

A. Performance metrics

One basic performance measure is how well router throttles installed by S can floor attackers in their attempt to deny good users of the ability to obtain service from S. It is clear that the defense mechanism cannot completely neutralize the effects of malicious traffic – in part because attackers are themselves entitled to a share of U_S in our model. Hence, good users must see a degraded level of performance, but hopefully are much less prone to *aggressive* attack flows than without network protection.

Apart from the basic performance measure, it is necessary to evaluate the deployment costs of the proposed defense mechanism. Therefore, the following are important evaluation criteria that we adopt:

- The percentage of good user traffic that makes it to the server. Since the control algorithm ensures that the server operates under its maximum designed load, the good user requests that arrive should be adequately served.
- The number of routers involved in protecting S. Because throttling clips forwarding rate to some preset ceiling, it is less tolerant to traffic variabilities than best-effort transmissions. For example, normal traffic that occasionally exceeds the ceiling and cannot be absorbed by the token bucket will get clipped, instead of being served by opportunistic resource availabilites. We measure the number of routers at which traffic is actually dropped due to the throttle rate limit.

B. Packet network results

To evaluate how the proposed throttle mechanism would perform over a real network, we conducted simulations using a global network topology reconstructed from real traceroute data. The traceroute data set is obtained from the Internet mapping project at AT&T[2]. It contains 709,310 distinct traceroute paths from a single source to 103,402 different destinations widely distributed over the entire Internet. We use the single source as our target server S, and randomly select 5000 traceroute paths from the original data set for use in our simulations. The resulting graph has a total of 135,821 nodes, of which 3879 are hosts. We assume, therefore, that out of all the hosts in the total global network, these 3879 hosts access S, either as an attacker or a good user.

Evenly distributed aggressive attackers

In our first set of experiments, we model *aggressive* attackers, whose average individual sending rate is several times higher than that of normal users. Specifically, each good user is chosen to send fixed size UDP packets to S, where the packet interarrival times are Poisson and the average traffic rate is randomly and uniformly drawn from the range $[0, 2]$. Each attacker is chosen to send traffic at a rate randomly and uniformly drawn from the range $[0, r_a]$, where r_a is either 10 or 20 according to the particular experiment. Furthermore, we select attackers and good users to be evenly distributed in the network topolgy: each host in the network is independently chosen to be an attacker with probability p, and a good user with probability $1 - p$.

Figure 8 compares the performance of our algorithm (labeled "level-k max-min fairness") with that of the pushback max-min fairness approach in [9], for $r_a = 20$ and $p = 0.2$. We show the percentage of remaining good user and attacker

[2] http://cm.bell-labs.com/who/ches/map/dbs/index.html

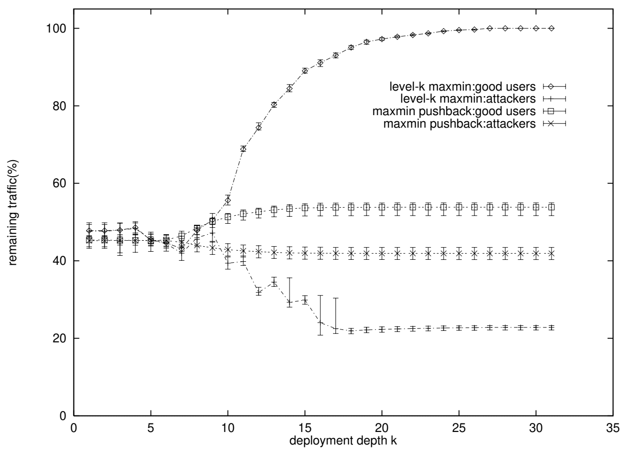


Fig. 8. Protection for good users under 20% evenly distributed aggressive attackers: mean attacker rate 10 times mean good user rate.

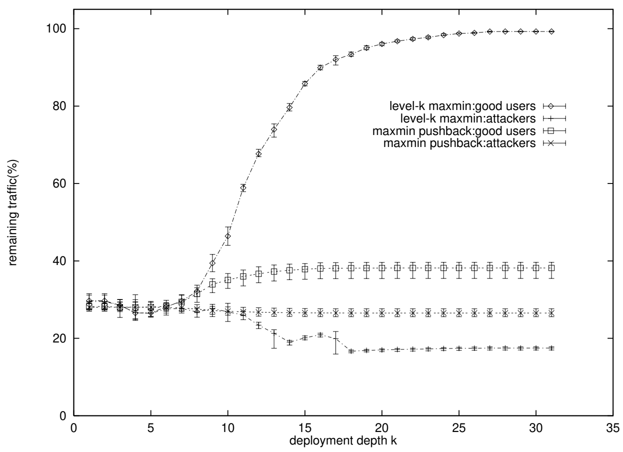


Fig. 9. Protection for good users under 40% evenly distributed aggressive attackers: mean attacker rate 10 times mean good user rate.

traffic that passes the router throttles and arrives at the server. Figures 9 and 10 show the corresponding results when $r_a = 20$ and $p = 0.4$, and $r_a = 10$ and $p = 0.4$, respectively. We plot the average results over ten independent experimental runs, and show the standard deviation as an error bar around the average.

Notice from the figures that generally, level-k max-min fairness gives signifcantly better protection for good user traffic than pushback max-min fairness. The performance advantage of level-k max-min fairness increases as k increases, until it levels off at k roughly equal to 20. This is because good traffic can aggregate to a significant level near S (the increase rate can be exponential), making it hard to distinguish from the attacker traffic at that location. Since pushback always originates control at S, it can severely punish good traffic. By initiating control further away from S (specifically, about k hops away), level-k max-min fairness achieves better good user protection.

Unevenly distributed aggressive attackers

In this set of experiments, each good user traffic rate is chosen randomly and uniformly from the range $[0, 2]$, while each attacker rate is similarly chosen from the range $[0, 20]$. In each experiment, about 20% of the hosts are chosen to be attackers, and the remaining hosts to be good users.

In these experiments, we select the attackers to have different *concentration* properties. Specifically, we pick five disjoint subtrees from the network topology, labeled in Fig. 11 as 1–5. The five subtrees have properties as shown in

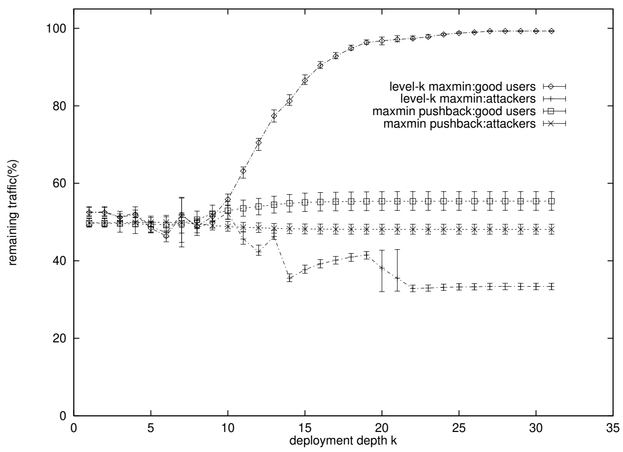


Fig. 10. Protection for good users under 40% evenly distributed moderately aggressive attackers: mean attacker rate 5 times mean good user rate.

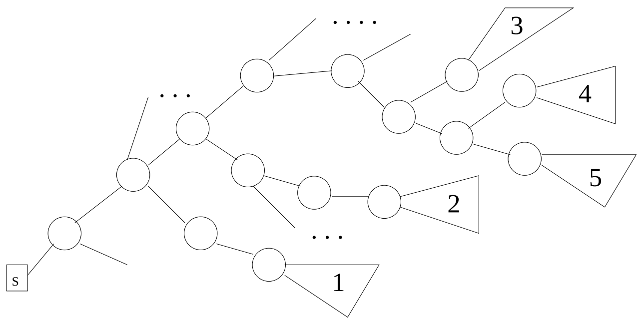


Fig. 11. Subtrees 1–5 used in attacker concentration experiments.

Table III. We then define four concentration configurations, 0–3, for the attackers, as shown in Table IV. The intention is for attacker concentration to increase as we go from configurations 0 to 3. (Notice that the roots of subtrees 4 and 5 in configuration 3 share a common parent, and so attacker traffic converges more quickly than the subtrees 1 and 3 in configuration 2.)

Fig. 12 shows the percentage of remaining good traffic for the four concentrations, using level-k max-min fairness. Fig. 13 shows the corresponding results for pushback max-min fairness. Notice that as k increases, level-k max-min fairness achieves good protection for the good users in all four configurations. For configurations 1–3, however, notice a "dip" in the achieved protection over k values between about 6 to 11. For example, the percentage of remaining good traffic for configuration 3 decreases from $k = 9$ to $k = 11$, and rises again afterwards.

To explain the dip, consider the case when all attackers are contained in *one* subgraph, say G', whose root is m hops away from S. For the traffic seen at $R(k)$, as k *decreases* from m to 1, there will be more and more aggregation of good user traffic *but no further aggregation of attack traffic*. This will cause a larger fraction of good user traffic to be dropped (its volume is more comparable to attack traffic) as throttling is performed with a smaller k, for $k \in [1, m]$. This explains the initial rising curves in Fig. 12 before the dip. For k a few hops larger than m, the aggregation situation for both good user and attack traffic is similar to the case of evenly distributed attackers. Hence, we observe increased protection for good user traffic as k increases from $m + c$ onwards, where c is a small constant. This explains the rising curves shortly after the dip. At the point when k *just* increases past the root of G', however, there is progressively less aggregation of attack traffic. This may cause reduced dropping rate for the attack traffic (since its volume at the

Subtree	No. of nodes	No. of hosts	Root's distance from S (hops)
1	1712	459	4
2	1126	476	6
3	1455	448	7
4	1723	490	8
5	1533	422	8

TABLE III

Properties of subtrees 1–5.

Configuration	Attackers uniformly chosen from
0	entire graph
1	all the five subtrees
2	subtrees 1 and 3
3	subtrees 4 and 5

TABLE IV

Configured concentrations of attackers.

control points is smaller and more comparable to good user traffic), when compared with control after full attack traffic aggregation has occurred at the root of G'. This explains the dip itself.

Despite the above "anomaly", level-k max-min fairness consistently and signifcantly outperforms pushback max-min fairness for $k > 15$. The performance advantage decreases from 0–3, because pushback max-min fairness becomes more effective as attackers get more concentrated. Figure 14 more clearly compares the two approaches by plotting their results together, for configurations 0 and 3.

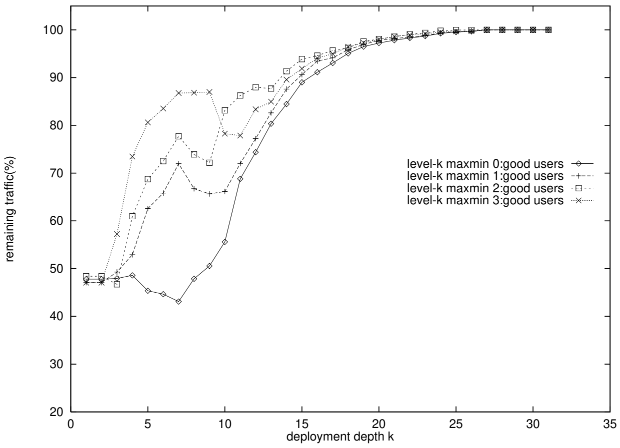


Fig. 12. Protection for good users, under four different attacker concentrations, using level-k max-min fairness.

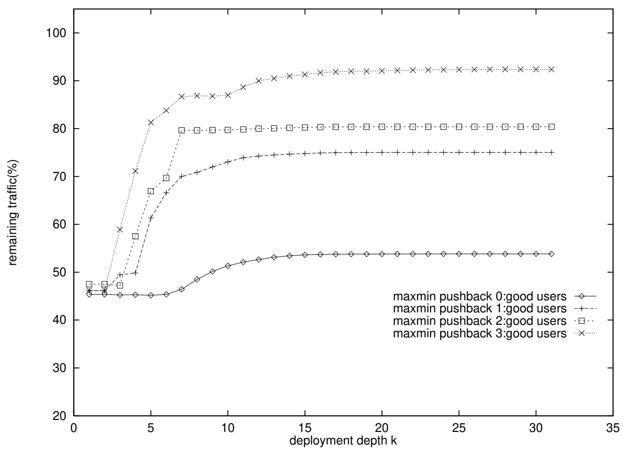


Fig. 13. Protection for good users, under four different attacker concentrations, using pushback max-min fairness.

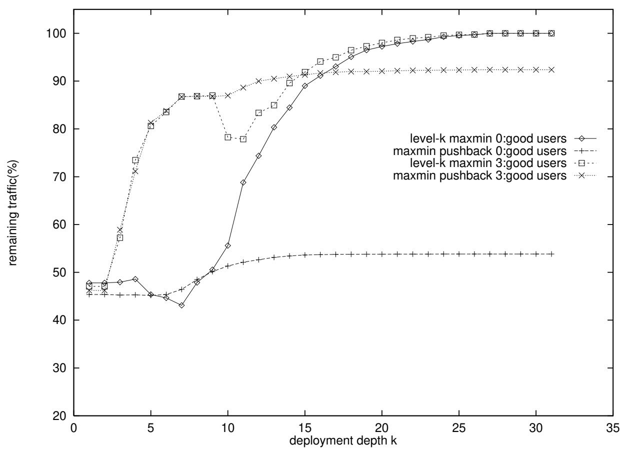


Fig. 14. Comparions of good-user protection between level-k and pushback max-min fairness – for configurations 0 and 3 only.

Evenly distributed "meek" attackers

Our results so far assume that attackers are significantly more aggressive than good users. This may be a reasonable assumption in practice. However, should a malicious entity be able to recruit or compromise *many* hosts to launch an attack (which clearly requires a *much more powerful* attacking entity), then each of these hosts behaving like a normal user can still together bring about denial of service.

It is inherently more difficult to defend against such an attack. In an experiment, we model both attackers and good users to send traffic to S at a rate randomly and uniformly drawn from $[0, 2]$. We randomly pick about 30% or 1169 of the hosts to be attackers, which are evenly distributed over the network. The remaining hosts are taken as good users. This produces an aggregate traffic rate of 3885, which is about 39% higher than the server capacity of 2800 that we model.

The percentages of remaining good user and attacker traffic that arrives at S are shown in Figure 15, for both level-k and pushback max-min fairness. As shown in the figure, both approaches essentially fail to distinguish between the good users and the attackers, and punish both classes of hosts equally. *However, the throttling mechanism, whether it employs level-k or pushback max-min fairness, can still be useful because it does protect the server from overload.* Hence, the 70% of good user requests that do make it to S may still be able to obtain service from S, whereas the same may not be true of a server that is simply overwhelmed with excessive packet arrivals.

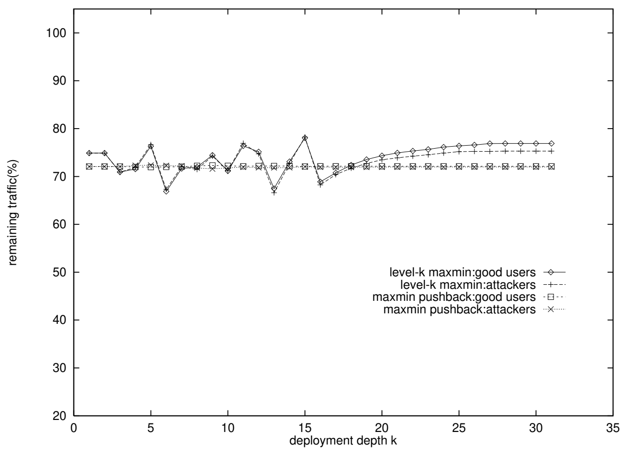


Fig. 15. Protection for good user traffic under evenly-distributed "meek" attackers, for both level-k and pushback max-min fairness.

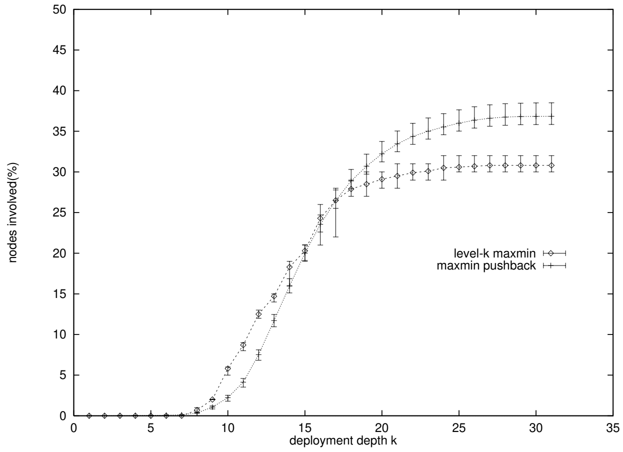


Fig. 16. Number of participating routers for level-k and pushback max-min fairness, as a function of the deployment depth.

Deployment extent

The previous two sets of experiments suggest that, for aggressive attackers, the effectiveness of level-k max-min fairness increases with k. At the same time, however, the cost of deployment may also increase, as the number of routers in $R(k)$ becomes larger.

Figure 16 plots the percentage of routers involved in throttling as a function of k, for both level-k and pushback max-min fairness. (For the level-k approach, we count both monitoring and throttling routers.) Notice that the two approaches basically require a comparable number of deployment points, although for k equal to 4–9, pushback max-min fairness is somewhat more efficient, and for larger k, level-k max-min fairness is somewhat more efficient. Also, the percentage of deployment points levels off as k rises above 20 for both approaches. This is because as k increases, a throttling node will likely see a progressively smaller rate of traffic destined for S. If the rate is small enough, both algorithms avoid the actual use of a throttle.

Web server performance

To evaluate the impact of throttling on real user applications, we simulate the performance of a web server under DDoS attack. The simulations are performed using ns2, and clients access the web server via HTTP 1.0 over TCP Reno/IP. (TCP is interesting because the achieved throughput by a client also depends on the rate at which acks are

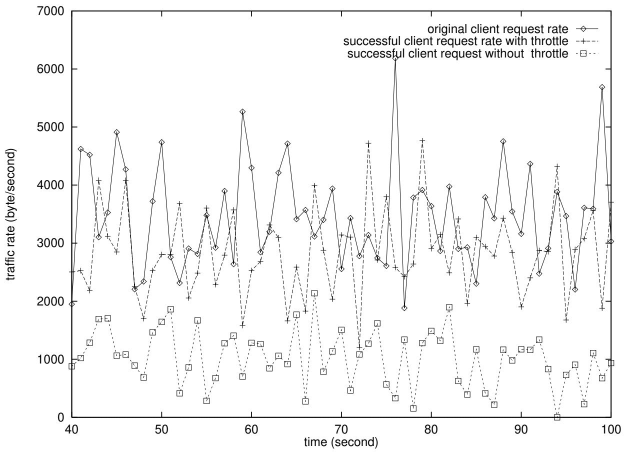


Fig. 17. Plot of (i) original client request rate, (ii) rate of successfully processed client requests with level-10 router throttling, and (iii) rate of successfully processed client requests without throttling, over time.

returned from the server to the client.) The simulated network is a subset of the AT&T traceroute topology described above. It consists of 85 hosts, of which 20% (i.e., 17 out of 85) are chosen as attackers. The maximum and average numbers of hops between a client and the server is 30 and 15, respectively.

Attackers generate UDP traffic destined for the server, at a constant rate of 6000 bits/s. Web clients make requests for documents to the server, where the document sizes and times between requests are probabilistically generated according to collected empirical distributions.[3] If a request arrives at the server successfully, the server will return the requested document after a random processing time, also chosen according to collected empirical distributions.

We model the web server to have $L_S = 8$ kbytes/s and $U_S = 10$ kbytes/s. We report two experiments with $k = 10$ and $k = 9$, respectively. To compare web server performance with and without throttling, we plot the rates of client requests that are *successfully processed* by the server in both cases, over time. The aggregate rate at which the clients originally make requests is also shown for baseline comparison. Each experiment runs for 100 seconds of simulated time, and an attack starts at time 10 seconds.

Fig. 17 shows the results for $k = 10$. Notice that with throttling, the rate of client requests that are successfully processed is *much closer to the original client request rate*, than without throttling (the averages are 3.8, 2.5 and 0.9 kbytes/s, respectively). Fig. 18 shows the corresponding results for $k = 9$, and supports the same conclusions. Fig. 19 shows the web client, attacker, and total traffic arrival rate at the server, for $k = 10$. Notice that our throttle negotiation algorithm is effective in keeping the actual server load between L_S and U_S.

VII. Discussions

Several observations are in order about the practical deployment of our defense mechanism. First, we must achieve reliability in installing router throttles. Otherwise, the throttle mechanism can itself be a point for attack. To ensure reliability, throttle messages must be authenticated before an edge router (assumed to be trusted) admits them into the network. Notice that *only* edge routers that have made arrangement with the (relatively few) server sites that desire protection have to do authentication. Other edge routers can just drop throttle requests unconditionally. Also, throttle requests must be efficiently and reliably delivered from source to destination, which can be achieved by high network priority for throttle messages and retransmissions in case of loss. Since throttle messages are infrequent and low in

[3]Please see http://http.cs.berkeley.edu/ tomh/wwwtraffic.html for further details.

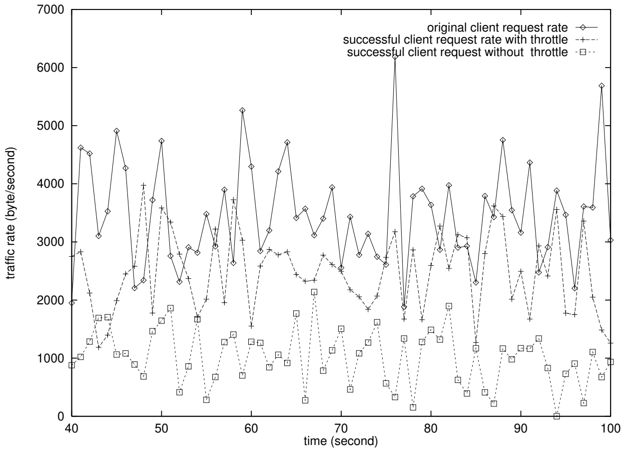


Fig. 18. Plot of (i) original client request rate, (ii) rate of successfully processed client requests with level-9 router throttling, and (iii) rate of successfully processed client requests without throttling, over time.

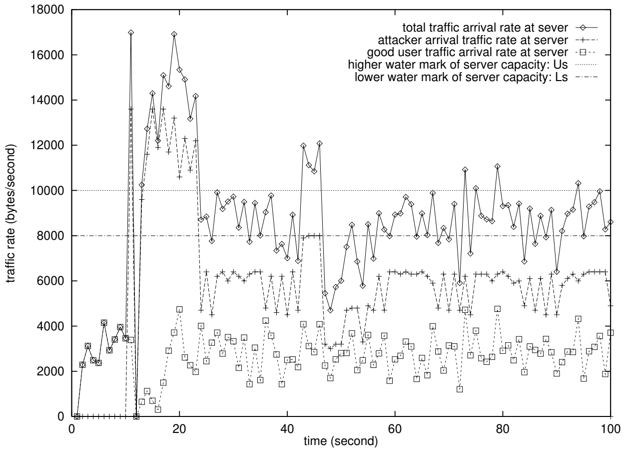


Fig. 19. Plot of total, attacker, and web client traffic arrival rates at server, over time, for level-10 throttling.

volume, the cost of authentication and priority transmissions should be acceptable (notice that edge authentication will prevent the network from seeing a high load of phony throttle messages).

Second, because of the feedback nature of the control strategy, it is possible that the server will transiently experience resource overload. To ensure that the throttle mechanism remains operational during these times, we can either use a coprocessor on the server machine that is not concerned with receive-side network processing, or deploy a helper machine, whose job is to periodically ping the server, and initiate defense actions when the server is not responsive.

Third, the throttle mechanism may not be universally supported in a network. Our solution remains applicable provided at least one router supports the mechanism on a network path that sees substantial attacker traffic. Depending on the position of such a router, the feasible range of k may be more restricted.

Fourth, we have adopted a generic notion of max-min fairness in our study, which makes it easy to manage and deploy. As observed in [9], however, it is also possible to have a policy-based definition of max-min fairness in practice. The policy can refer to different conditions in different network regions, in terms of tariff payments, network size, susceptibility to security loopholes, etc.

VIII. Conclusion

We presented a server-centric approach to protecting a server system under DDoS attacks. The approach limits the rate at which an upstream router can forward packets to the server, so that the server exposes no more than its designed capacity to the global network. In allocating the server capacity among the upstream routers, we studied a notion of level-k max-min fairness, which is policy-free and hence easy to deploy and manage.

We evaluated algorithm effectiveness using a realistic global network topology, and various models for attacker and good user distributions and behaviors. Our results indicate that the proposed approach can offer significant relief to a server that is being flooded with malicious attacker traffic. First, for aggressive attackers, the throttle mechanism can preferentially drop attacker traffic over good user traffic, so that a larger fraction of good user traffic can make it to the server as compared with no network protection. In particular, level-k max-min fairness performs better than recursive pushback of max-min fair rate limits previously proposed in the literature [9]. This is especically the case when attackers are evenly distributed over the network. Second, for both aggressive *and* "meek" attackers, throttling can regulate the experienced server load to below its design limit, so that the server can remain operational during a DDoS attack.

Our results indicate that server-centric router throttling is a promising approach to countering DDoS attacks. However, modeling the behaviors of attackers is inherently difficult, and modeling the behaviors of good users needs to be service and environment specific. Hence, more study is needed to evaluate the robustness of the approach in more diverse deployment scenarios. Also, our focus has been on DDoS attacks in which attackers try to overwhelm a victim server by directing an excessive volume of traffic to the server. Other forms of attacks are possible that do not depend on the sheer volume of attack traffic [8]. However, more sophisticated attack analysis (e.g., intrusion detection) is usually feasible to deal with these other forms of attacks.

References

[1] TCP SYN flooding and IP spoofing attacks. CERT Advisory CA-96.21. available at http://www.cert.org/.

[2] Smurf IP denial-of-service attacks. CERT Advisory CA-1998-01, January 1998. avaliable at www.cert.org/advisories/CA-98.01.html.

[3] B. Awerbuch and Y. Shavitt. Converging to approximated max-min flow fairness in logarithmic time. In *Proc. IEEE Infocom*, San Francisco, CA, March 1998.

[4] W. Fang and L. Peterson. Inter-AS traffic patterns and their implications. In *Proc. IEEE Global Internet Symposium*, Rio, Brazil, December 1999.

[5] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. RFC 2827, May 2000.

[6] M. G. Gouda, E. N. Elnozahy, C. T. Huang, and T. M. McGuire. Hop integrity in computer networks. In *Proc. IEEE ICNP*, Osaka, Japan, November 2000.

[7] S. Kent and R. Atkinson. Security architecture for the Internel Protocol. RFC 2401, November 1998.

[8] G. de Vivo M. de Vivo and G. Isern. Internet security attacks at the basic levels. In *ACM Operating Systems Review*, volume 32, April 1998.

[9] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. Technical report, ACIRI and AT&T Labs Research, February 2001. Draft paper; available from http://www.aciri.org/floyd/papers.html.

[10] K. Park and H. Lee. On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack. In *Proc. IEEE Infocom*, Anchorage, Alaska 2001.

[11] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. In *Proc. ACM SIGCOMM*, Stockholm, Sweden, August 2000.

[12] D. Song and A. Perrig. Advanced and authenticated techniques for IP traceback. In *Proc. IEEE Infocom*, Anchorage, Alaska, 2001.