CERIAS Tech Report 2001-39

On Defending Against Distributed Denial-of-Serivce Attacks with Server-centric RouterThrottles-

David K.Y. Yau[1], Feng Liang[1], John C.S. Lui[2]
Center for Education and Research in
Information Assurance and Security
&
[1]Department of Computer Sciences, Purdue University
West Lafayette, IN 47907-1398
[2]Department of Computer Science,
Chinese University of Hong Kong

On Defending Against Distributed Denial-of-service Attacks with Server-centric Router Throttles[*]

CERIAS TR-2001-39 and Purdue CS TR-01-008 May, 2001

David K. Y. Yau Feng Liang
CERIAS and Dept of Computer Science
Purdue University
West Lafayette, IN 47907
{yau,liangf}@cs.purdue.edu

John C. S. Lui
Department of Computer Science
Chinese University of Hong Kong
Shatin, Hong Kong
cslui@cse.cuhk.edu.hk

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 is 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 forstalling an impending attack. In allocating the server capacity among the routers, we propose a notion of *level-k max-min fairness.* We present simulation results using a realistic global network topology, and various models of good user and attacker distributions and behaviors. 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.

1. 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. To accomodate bursty traffic, a throttle should be implemented as a leaky bucket with the desired rate limit and some bucket size s (in bits) to absorb the burstiness. 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 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 r 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 $L_S < U_S$, then the throttle rate is increased (i.e., relaxed). If

[*] Research supported in part by CERIAS, and in part by the National Science Foundation under grant numbers EIA-9806741 and CCR-9875742 (CAREER)

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

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.

Obviously, we cannot ask the routers to maintain state about every Internet server, as this will cause an explosion of the state information needed. 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.

1.1 Our contributions

Our contributions in this paper are:

- We contribute to the fundamental understanding of router throttling as a mechanism against DDoS attacks.
- 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.

1.2 Paper organization

The balance of the paper is organized as follows. in Section 2, we introduce our system model. In Section 3, we formally specify the algorithm for computing throttle rates and discuss an optimization technique of throttle pruning that relieves portions of the network not under attack of deployment costs. After defining our solution approach, we compare it with related work in the literature, in Section 4. Section 5 discusses performance metrics to evaluate the effectiveness of the proposed solution. Diverse simulation results using a realistic network topology are reported in Section 6. In Section 7, we discuss several issues about the practical deployment of our solution. Section 8 concludes.

2. 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 1. (For ease of presentation, we assume that an overloaded server is still capable of initiating the defense actions. However, as discussed in Section 7, 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.

3. THROTTLE ALGORITHM

We formally specify in Fig. 2 the algorithm by which S determines the throttle rate to be installed in $R(k)$. In the specification, r_S is the current throttle rate to be used by S. It is initialized to $U_S/f(k)$, 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 throttle algorithm is to be invoked whenever 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. 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 interatively adjusts r_S to an appropriate value. Notice

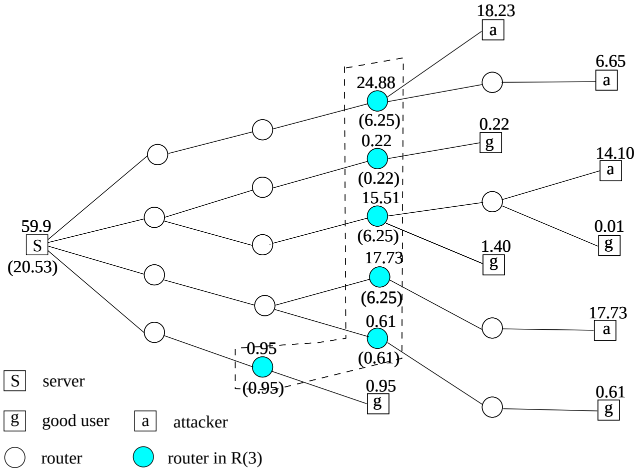

Figure 1: Network topology illustrating $R(3)$ deployment points of router throttle, and offered and throttled rates. The target range of server load is set to be $[18, 22]$, and the actual achieved load in this example is 20.53.

Algorithm throttle

```
$\alpha_{\text{last}} := -\infty$;
**while** (1)
        multicast current rate-$r_S$ throttle to $R(k)$;
        monitor traffic arrival rate $\alpha$ for time window $w$;
        **if** ($\alpha > U_S$) /* throttle not strong enough */
                /* further restrict throttle rate */
                $r_S := r_S/2$;
        **elif** ($\alpha < L_S$) /* throttle too strong */
                **if** ($\alpha - \alpha_{\text{last}} < \epsilon$)
                    remove rate throttle from $R(k)$;
                    **break**;
                **else**
                    /* try relaxing throttle by additive step */
                    $\alpha_{\text{last}} := \alpha$;
                    $r_S := r_S + \delta$;
                **fi**;
        **else**
                **break**;
        **fi**;
**end while**;
```

Figure 2: Throttle algorithm specification.

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. Otherwise, if the server load is below L_S and the next increase in the throttle rate increases 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. Also, let $L_S = 18$ and $U_S = 22$. The total offered load to S exceeds U_S, and hence the throttle algorithm will be invoked at S. When the algorithm terminates, S determines the throttle rate to be 6.25, and installs this rate at each of the router in $R(3)$. In the figure, the number above a router indicates the arrival rate of traffic destined for S, and the number in parenthesis below the router indicates the throttled rate at which the traffic is being forwarded. As a result of the throttling, the load at S will be limited at 20.53, which is the sum of the throttled rates. Notice that the throttled rate at a router in $R(3)$ is the router's max-min fair share of the achieved server load of 20.53.

Notice that similar to TCP congestion control, each throttle rate will take in the worst case (depending on k) one network round trip time to take effect. Hence, the throttle algorithm can take multiple round trips to terminate. Because of this, it can be difficult to achieve exact max-min fairness in a highly dynamic network. The result will be some degree of under-utilization of the server capacity. We believe that since throttling is to be deployed under extreme conditions (e.g., in the face of a DDoS attack), it is acceptable for the defense mechanism to restore the availability of a large fraction – if not the entirety – of the server capacity.

3.1 Throttle pruning

With the basic throttle algorithm described thus far, $R(k)$ can increase quickly with k, resulting in unnecessary deployment costs if most of these routers are not on an attack path. The situation can be improved without affecting system performance if routers located between S and $R(k)$ can *monitor* the arrival rates of packets destined for S from different upstream links. When under stress, S sends a rate-r throttle to the directly connected routers. On receiving the throttle, a router does not immediately forward the throttle upstream as in the basic algorithm. Instead, it starts monitoring the forwarding rates for upstream traffic destined for S. If the rate from an incoming link is less than r, the throttle message can be pruned for that link. If the rate from a link is higher than r, then the throttle message is forwarded to the upstream router connected to the link. The upstream router similarly performs the rate monitoring for its upstream links and makes the decision to either prune or forward a received throttle.

Figure 3 illustrates throttle pruning for the example network topology previously given in Fig. 1. In the figure, notice that throttles are avoided in three of the routers in $R(3)$, because they are not on the path of any attacker. The figure also shows the routers that have to perform monitoring in order to support the pruning decision. Although the

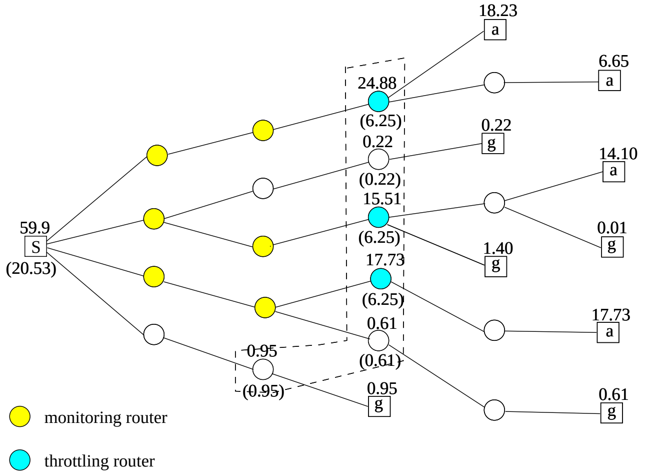

Figure 3: Introducing monitor routers between S and $R(3)$ allows throttle requests to be pruned closer to S, and avoids throttle deployment in all of the $R(3)$ routers.

total number of monitoring and throttling routers in Fig. 3 exceeds the number of throttling routers in Fig. 1, for a large k in a large-scale network where attacks are localized in a few regions, pruning will significantly reduce the scope of router participation needed.

4. 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 6). 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.

5. 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. The throttle implementation requires the overhead of classifying packets to throttled flows and enforcing their leaky bucket rates. While the classification is solely based on IP destination address, and hence has similar cost as traditional IP forwarding, the overall mechanism requires additional state and accounting overheads. More importantly, since 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.

- Algorithm stability in response to changing user demands. In a real network, both good users and attackers may change their behaviors over time. It is important to evaluate how our control algorithm responds to such dynamics, in terms of (i) how well server load can be kept within $[L_S, U_S]$ during an attack, and (ii) how well good users are protected from attackers.

6. EXPERIMENTAL 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 traffic to S at a rate 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 4 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 traffic that passes the router throttles and arrives at the server. Figures 5 and 6 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.

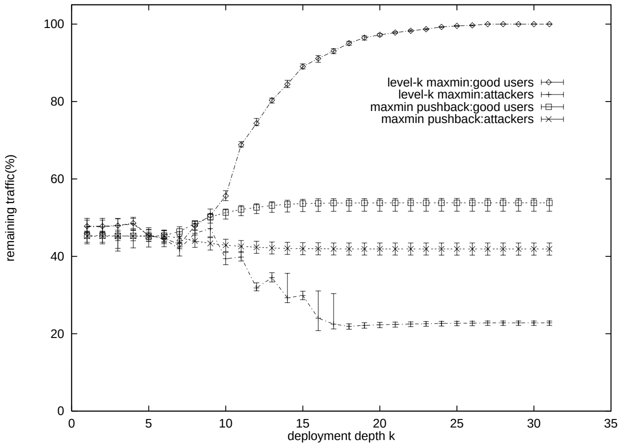

Figure 4: Protection for good users under 20% evenly distributed aggressive attackers: mean attacker rate 10 times mean good user rate.

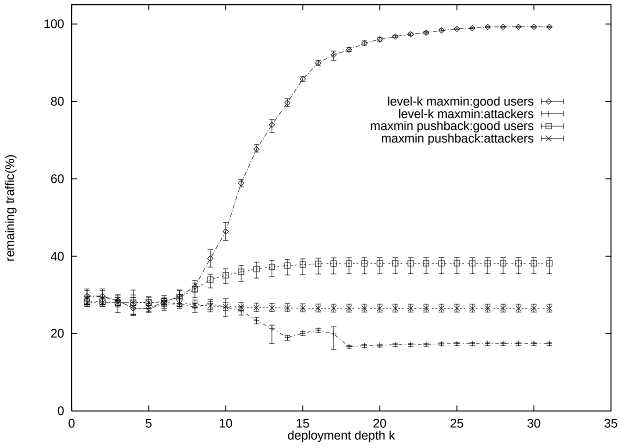

Figure 5: Protection for good users under 40% evenly distributed aggressive attackers: mean attacker rate 10 times mean good user rate.

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. 7 as 1–5. The five subtrees have properties as shown in Table

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

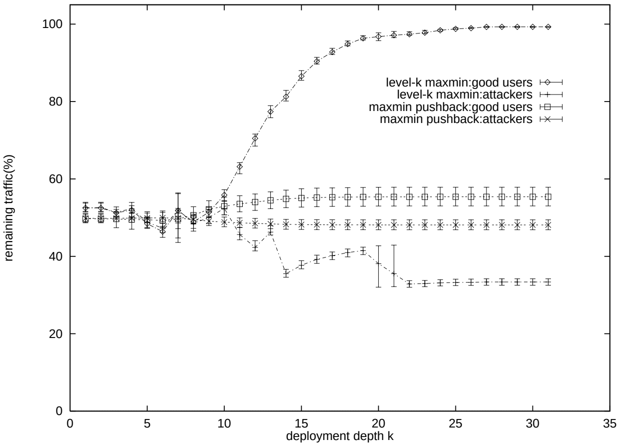

Figure 6: Protection for good users under 40% evenly distributed moderately aggressive attackers: mean attacker rate 5 times mean good user rate.

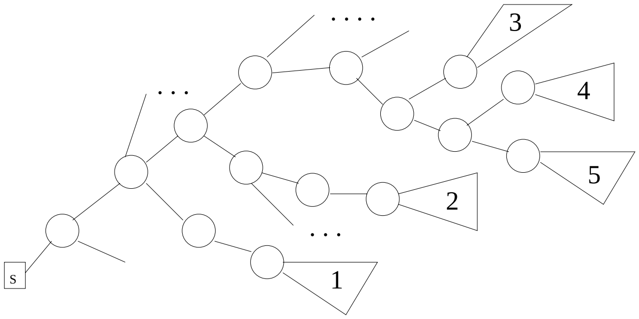

Figure 7: Subtrees 1–5 used in attacker concentration experiments.

1. We then define four concentration configurations, 0–3, for the attackers, as shown in Table 2. 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. 8 shows the percentage of remaining good traffic for the four concentrations, using level-k max-min fairness. Fig. 9 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.

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 1: 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 2: Configured concentrations of attackers.

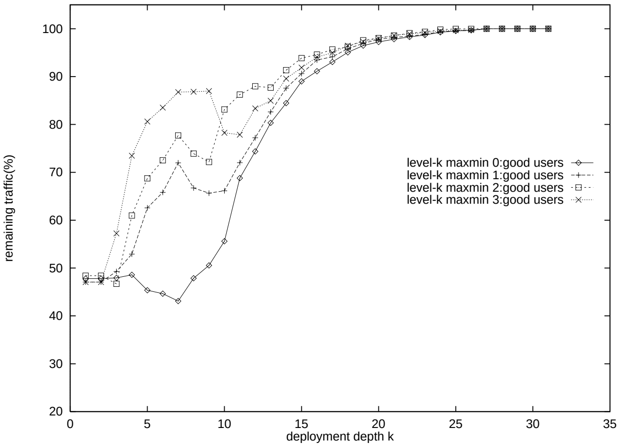

Figure 8: Protection for good users, under four different attacker concentrations, using level-k max-min fairness.

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. 8 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 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 10 more

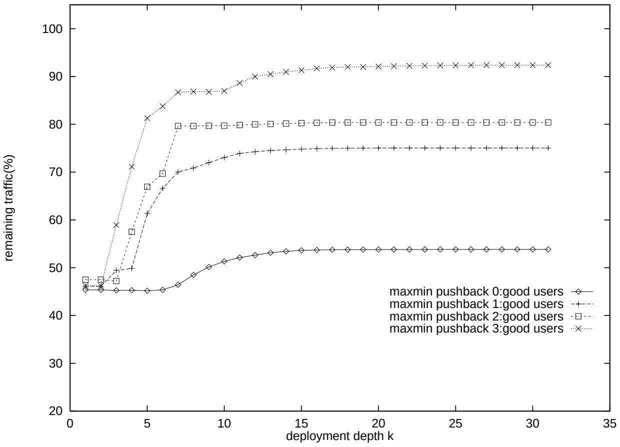

Figure 9: Protection for good users, under four different attacker concentrations, using pushback max-min fairness.

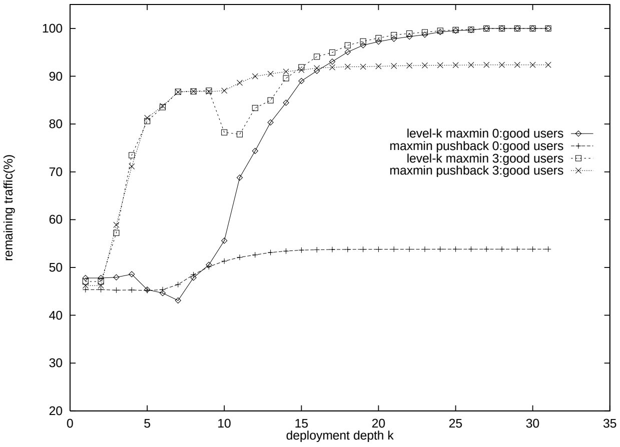

Figure 10: Comparions of good-user protection between level-k and pushback max-min fairness – for configurations 0 and 3 only.

clearly compares the two approaches by plotting their results together, for configurations 0 and 3.

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, 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.

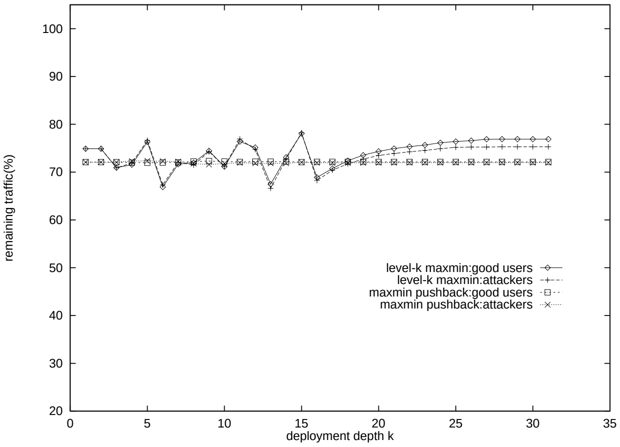

Figure 11: Protection for good user traffic under evenly-distributed “meek” attackers, for both level-k and pushback max-min fairness.

The percentages of remaining good user and attacker traffic that arrives at S are shown in Figure 11, 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.

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 12 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 use of a throttle; hence, the number of deployment points is not increased.

Dynamic users

Next, we investigate the effects of user dynamics (for both good users and attackers) on our control algorithm. For

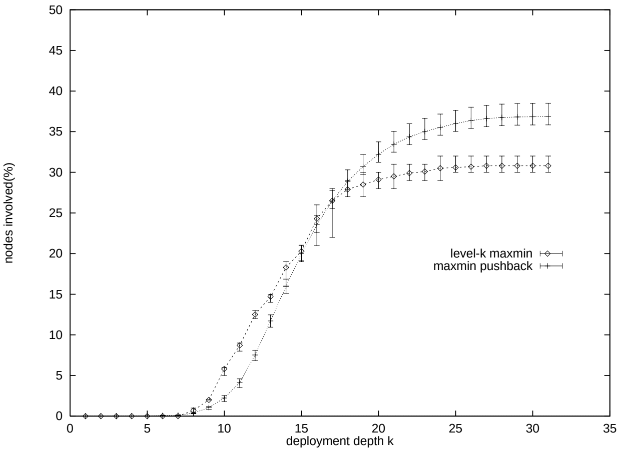

Figure 12: Number of participating routers for level-k and pushback max-min fairness, as a function of the deployment depth.

this experiment, we use $k = 15$, $L_S = 4700$ and $U_S = 5300$. Twenty percent of the hosts are chosen to be attackers, and the rest are good users. The attackers are evenly distributed over the experimental network. We measure time in units of maximum round trip delay between S and a router in $R(15)$.

As attackers and good users vary their sending rates, we capture the dynamics of the control algorithm. The results are shown in Fig. 13. In the figure, we plot over time (i) the aggregate rate at which traffic is sent by all the hosts, (ii) the rate at which all traffic arrives at S, (iii) the aggregate rate at which traffic is sent by all the good users, and (iv) the rate at which good user traffic arrives at S. By comparing (ii) against L_S and U_S – which are also shown in the figure – we can evaluate the effectiveness of our algorithm in protecting S from resource overload. By comparing (iii) and (iv), we can get an idea of how well good user traffic is protected from attacker traffic. Fig. 14 shows how the throttle rate r_S evolves over time.

We now explain the network dynamics that produce Fig. 13 and Fig. 14. At $t = 0$, each good user sends at a rate randomly chosen from $[0, 2]$, and no throttle is in effect. Immediately afterwards, the attack starts with each attacker sending at a rate randomly chosen from $[0, 20]$. The throttle algorithm is invoked with r_S initialized to 128. At $t = 1$, the throttle takes effect but fails to bring down the server load to below L_S. The throttle rate is then halved to 64, and the throttle algorithm continues. At $t = 7$, after five throttle rate reductions and one increase, the throttle algorithm terminates with rate 4729.

During time interval $[7, 10]$, the attacker and good user traffic rates keep changing, but are kept in the original ranges of $[0, 20]$ and $[0, 2]$, respectively. At $t = 9$, the server load increases slightly above U_S, which causes the throttle algorithm to run with a single iteration and return 4710 as the throttle rate.

At $t = 10$, half of the attackers stop sending any traffic.

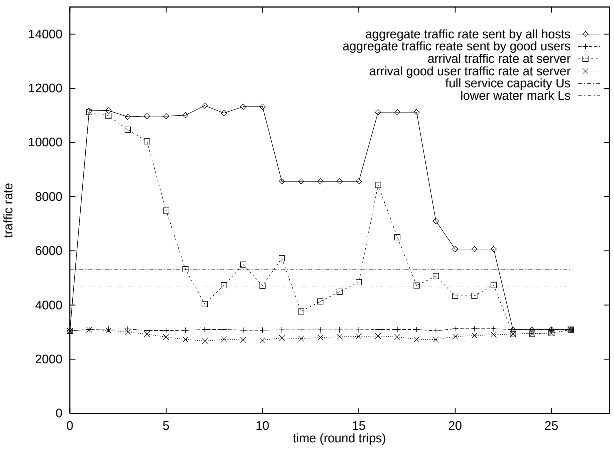

Figure 13: Algorithm dynamics in response to changing attacker and good user behaviors over time.

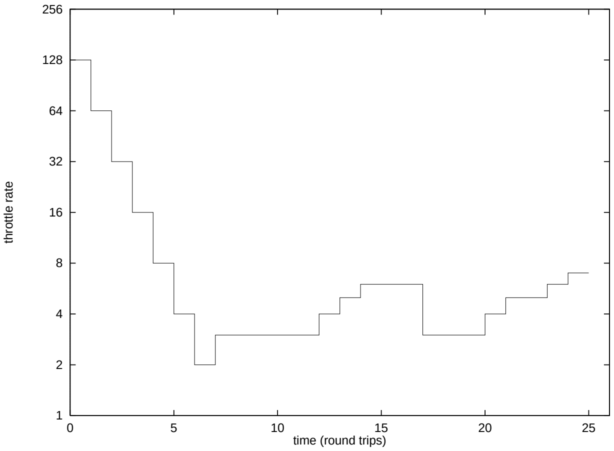

Figure 14: Evolution of throttle rate r_S over time.

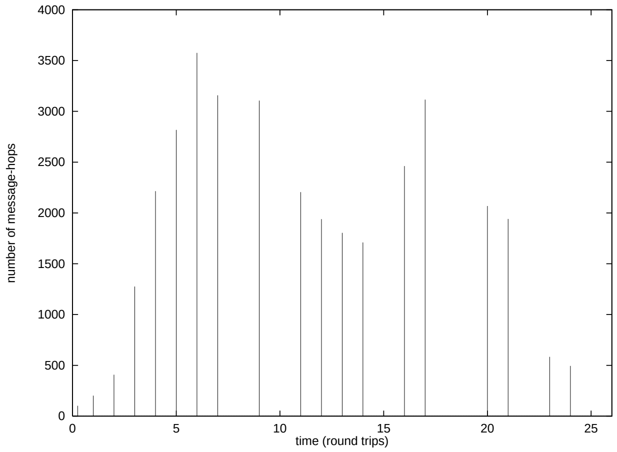

Figure 15: Number of message-hops required for each throttle action taken by S. Notice that from an individual link's perspective, the overhead of each throttle is at most *one* control message. Hence, the per-link additional load due to throttling is very small.

This causes the server load to drop below L_S at $t = 11$. The throttle algorithm ramps up the throttle rate, and at $t = 14$, returns with $r_S = 4843$. (During time $[10, 15)$, the good users and attackers keep changing their rates, though using the same distributions of rate.) At $t = 15$, the half of the attackers that stopped resume sending, each with rate chosen from $[0, 20]$. The throttle algorithm then runs with two iterations, and determines r_S to be 4718. At $t = 18$, the attackers reduce their traffic rate by 50%, with each sending at a rate chosen from $[0, 10]$. The resulting server load at $t = 19$ remains in $[L_S, U_S]$, and so no control action is taken. Meanwhile, at $t = 19$, half of the remaining attackers stop sending. This causes the server load to drop below L_S at $t = 20$, and the throttle algorithm to run with two iterations, and returns $r_S = 4736$.

At $t = 22$, the DDoS attack stops completely. The throttle algorithm increases the throttle rate at $t = 23$ and $t = 24$. At $t = 25$, the throttle is removed because the last throttle rate increase did not result in significant increase in the server load.

Each throttle message is multicast from S to $R(S)$. Hence, each throttle will produce a load of only one packet on each network link, which is a very small practical overhead. It is also interesting, however, to measure the number of *message-hops* required by each throttle (i.e., if a message traverses x links, the number of message-hops is x). This performance measure is shown in Fig. 15. Notice that in general, a large throttle rate requires a small number of message hops, because the message can be pruned early in the distribution tree.

7. 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. Also, they 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 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.

8. 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.

9. 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.