Using Adaptive Router Throttles Against Distributed Denial-of-Service Attacks[*]

LIANG Feng[1], David Yau[2]

[1](*Zhejiang Provincial Key Laboratory of Fiber Optical Communication Technology, Zhejiang University of Technology, Hangzhou* 310014, *China*);

[2](*Department of Computer Science, Purdue University, IN* 47907, *USA*)

E-mail: liangf@zjut.edu.cn

http://www.zjut.edu.cn

Received December 7, 2001; accepted April 29, 2002

Abstract: In this paper, an adaptive router throttle algorithm is presented to defend a server against distributed denial-of-service (DDoS) attacks. The key point of the algorithm is that the server asks selected upstream routers k hops away to install *throttles* on traffic flows destined for it so that the server's service capacity can be allocated among all flows with a max-min like fairness. The algorithm effectiveness is evaluated by using a realistic Internet topology and various models for attacker and good user distributions and behaviors. The results indicate that this server-centric router throttling is a promising approach to countering DDoS attacks.

Key words: network security; DDoS; router; Internet; computer network

In a distributed denial-of-service (DDoS) attack[1,2], thousands of malicious or compromised hosts coordinate to send a large volume of aggregate traffic to a victim. Network nodes near the victim will progressively become more vulnerable to resource overruns, as a node that is closer to the server most likely has less service capacity while delivering a larger fraction of the attacking traffic. In particular, the victim itself is most vulnerable.

Former works on against DDoS attacks either drop or reroute the attacking packets before they enter the victim [3][4][5]. For this kind of approach, the key problem is that the DDoS attacking packets can be no different from normal packets, and as the packets' source IP addresses are usually forged, it's also difficult to distinguish the attacking flows from normal ones by traffic rates. Meanwhile, the protecting system itself and the routers on transmission networks can also be incapacitated.

IP traceback[6,7] utilizes routers' spare resources to trace back the paths from attackers to the server. The algorithm itself doesn't provide anything to cease the attack directly. As a fully deployment of IP traceback on every router of Internet is difficult, most probably, the traceback paths can't reach the attackers but routers several hops from the attackers, which leaves an open problem.

To actively defend against attacks, analysis of routing information can enable a router to drop certain packets with spoofed source IP address [8][9]. This approach requires sophisticated and potentially expensive routing table

* Supported by the Natural Science Foundation of Zhejiang Province of China under Grant No.697053 (浙江省自然科学基金); CERIAS, the National Science Foundation of US under Grant No.CCR-9875742 (CAREER)

LIANG Feng was born in 1967. He is an associate professor and master supervisor of the Telecommunication Department, Zhejiang University of Technology. His research interests are network security and multimedia communication. **David Yau** was born in 1967. He is an assistant professor and doctoral supervisor of the Computer Science Department, Purdue University. His current research areas is network, operating system architectures, algorithms for QoS provisioning, multimedia communication, software-programmable router technologies, database and information system.

analysis on a per-packet basis. Also, DDoS attackers can still launch an attack with real IP source addresses.

Mahajan *et al.*[10] 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. To protect good user traffic from attacker traffic, they study recursive *pushback* of max-min fair rate limits starting from the victim server to upstream routers, and define a global, cross-router notion of max-min fairness. However, the pushback mechanism always starts the resource sharing decision at the server, where good user traffic may aggregate to a large volume and thus can be severely punished (see Section 5). Such aggregation of good user traffic has been observed to occur in practice [11].

The use of network authentication mechanisms also helps defending against DDoS attacks, e.g. IPsec[12]. Gouda *et al.*[13] propose a framework for providing *hop integrity* in computer networks. Efficient and cheap algorithms for authentication and key exchanges are important research questions in this class of solutions.

In this paper, we prohibit DDoS attacks by resource management: The server's service capacity is allocated among all incoming traffic flows (including attackers') with a max-min like fairness, which provides that the attackers can not gain more resource capacity than normal users by sending more traffic. To forestall the aggressive packets converging to overwhelm the victim and nearby intermediate routers, a proactive approach is adopted: The victim asks selected upstream routers *k* hops away to install *throttles* which limit the forwarding rate of packets destined for it. The throttle is implemented as a leaky bucket to absorb the burst of traffic. Traffic that exceeds the rate limit will be dropped. The *appropriate* throttle rate is negotiated dynamically between the victim and the throttle routers, such that all users can share the service capacity of *S* with fairness and the throttle can be adaptive to the change of demand distributions.

1 Network Model of DDoS Attack

The entire network is represented as a connected graph $G=(V,E)$, where V is the set of nodes and E is the set of links. We have $V=H\cup R$, where H is the set of hosts (leaf nodes) and R the set of routers. The victim is a host $S\in H$ with a capacity U_S. The set of attackers is $H_a\subset H$, and the set of good users is $H_g\subset H$. Notice that H_a and H_g are dynamic, but they are relatively static in a short time period. Assume during a certain short time period, there's m good users, $H_g=\{g_1,g_2,..,g_m\}$, and n attackers, $H_a=\{a_1,a_2,..,a_n\}$. The traffic rate from g_i to S is r_{gi}, and from a_i to S is r_{ai}. Assume the traffic rate of good user or attacker is relatively static during this period, then the aggregate traffic rates of attackers and good users are $T_g=\sum_{i=1}^{m} r_{gi}=m\cdot\overline{r_g}$ and $T_a=\sum_{i=1}^{n} r_{ai}=n\cdot\overline{r_a}$, where $\overline{r_a}$ and $\overline{r_g}$ are the average rates of traffic from one good user or one attacker to S respectively.

If the total arrival traffic rate of S, $T_S = T_g + T_a$ U_S, the services for good users are not influenced. However, if $T_S>U_S$, T_S-U_S of the traffic will be dropped*, thus denial of service (DoS) occurs. We define the *degree of DoS state* on S, $\boldsymbol{h}$, as the percentage of good user traffic being dropped by S.

If S uniformly drops the overload traffic, the traffic from each user (either a good user or an attacker) is dropped with same percentage, and we have $\boldsymbol{h}=\frac{T_S-U_S}{T_S}=1-\frac{U_S}{T_S}$.

Assume S is designed to serve a maximum of M users, $U_S\approx M\cdot\overline{r_g}$, then

$$\boldsymbol{h}=1-\frac{U_S}{T_S}=1-\frac{M\cdot\overline{r_g}}{m\cdot\overline{r_g}+n\cdot\overline{r_a}}\Rightarrow\frac{\overline{r_a}}{\overline{r_g}}=\frac{1}{n}\left(\frac{M}{1-\boldsymbol{h}}-m\right) \tag{1}$$

For a big scale server, M is a significantly large number. Assume the attacker's capability of compromising a large

* Assume each link in E has infinite bandwidth. This assumption can actually be relaxed for our throttle algorithm, as the routers can also be protected from overload.

number of hosts is limited, so that $M >> n$. With these assumptions, Fig.1 shows that to reach a significant degree of DoS, $\overline{r_a}$ must be significantly higher than $\overline{r_g}$. This is the foundation of our throttle algorithm.

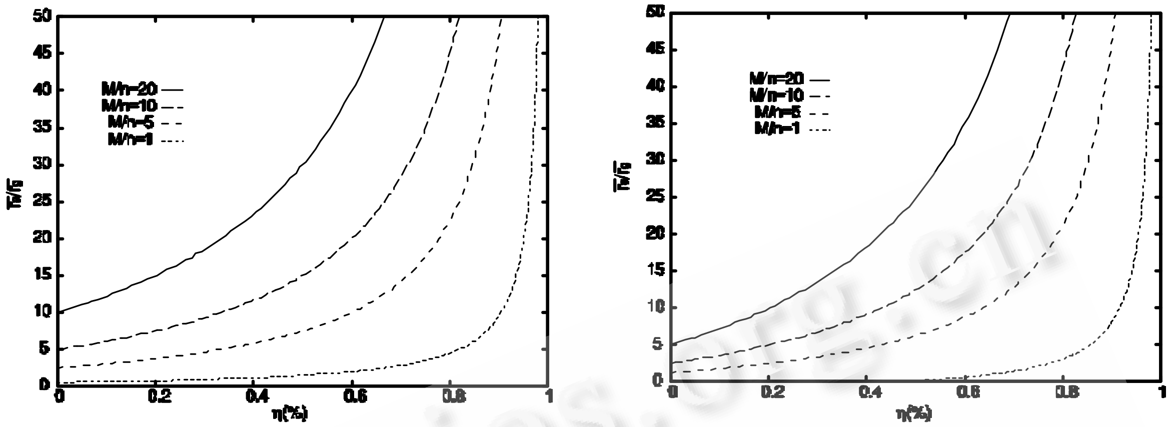


Fig.1 For m/M=0.5 (left) and m/M=0.75 (right), the ratio of $\overline{r_a}$ to $\overline{r_g}$ over h for M/n=20,10,5,1

2 Router Throttle Algorithm

2.1 Level-*k* max-min fairness

When S is under attack, it will initiate a throttle defense mechanism on a subset of the victim's upstream routers, $R(k)$, which contains all the routers k hops away from S and all the edge routers, which are directly connected to hosts, less than k hops away from S. For example, in the network topology of Fig.2, the $R(3)$ routers are shaded.

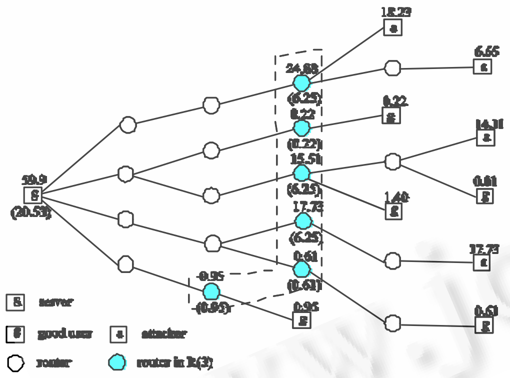


Fig.2 Network topology illustrating $R(3)$ deployment points of router throttle, and offered throttled rates

The throttle rate is determined by allocate S's capacity among $R(k)$ with max-min fairness, which is called *level-k* max-min fairness. In the example shown in Fig.2, the number above a node denotes the rate at which the node delivers traffic to S before throttling, and the numbers in parenthesis below $R(3)$ routers indicate the throttled rates. As a result of the throttling, the load at S will be limited at 20.53 (we assume U_s=20), which is the sum of the throttled rates. Notice that the throttled rate at a $R(3)$ router is the router's max-min fair share of the achieved server load of 20.53.

2.2 Throttle negotiation algorithm

Fig.3 specifies the throttle negotiation algorithm. When a DDoS attack on a server S begins, S's load a increases and finally exceeds capacity U_S. At this moment, S begins negotiating the throttle rate r_S with $R(k)$ routers. At first, r_S 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 routers typically needed in $R(k)$. After throttles of rate r_S are installed at $R(k)$ routers and take effect, if a is still bigger than U_S, then r_S is reduced to half of its current value. On the other hand, if a falls below a low-water-mark $L_S<U_S$, then the throttle rate is increased by a constant additive step d. The negotiation process stopped when a is within $[L_S,U_S]$, and will be resumed if a is out of this range again. If a is below L_S, and several successive increases of r_S do not increase a significantly, which means no traffic is dropped by the throttles, and then the throttles can be removed.

Notice that similar to TCP congestion control, the throttle needs a time delay to take effect. So a is monitored after r_S is sent for somewhat larger than the maximum round trip time between S and a router in $R(k)$. Also, 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. At last, the throttles can be set only on those routers with traffic rate destined for S bigger than the throttle rate for a lower cost.

```
Algorithm throttle
$a_{\text{last}}$ :=
while (1)
    multicast current rate-$r_S$ throttle to $R(k)$;
    monitor traffic arrival rate $a$ for time window $w$;
    if ($a > U_S$)    /* throttle not strong enough */
        $r_S := r_S / 2$;    /* further restrict throttle rate */
    elif ($a < L_S$)    /* throttle too strong */
        if ($a - a_{\text{last}}) < \varepsilon$)    /*no drop by throttle */
            remove rate throttle from $R(k)$;
            break;
        else\
            /* try relaxing throttle by additive step */
            $a_{\text{last}} := a$;
            $r_S := r_S + d$;
        fi;
    else
        break;
    fi;
endwhile
```

Fig.3 Throttle algorithm specification

2.3 Convergence of algorithm

For each router $R_k^i \in R(k)$, we denote the rate of arrival traffic destined to S at time t as $r_k^i(t)$. For simplification, we assume the traffic rates from all hosts keep static during one negotiation process, (the effects of dynamic traffics on our algorithm will be investigated later in Section 6), so that we can denote $_k^i(t)$ as r_k^i during one negotiation process.

Definition 2. r_U is the level-k maxmin-fairness throttle rate for capacity U_S, if $U_S=\sum_i^{R_k^i\in R(k)}\min(r_U,r_k^i)\cdot$

Definition 3. r_L is the level-k maxmin-fairness throttle rate for low-water-mark L_S, if $L_S=\sum_i^{R_k^i\in R(k)}\min(r_L,r_k^i)\cdot$

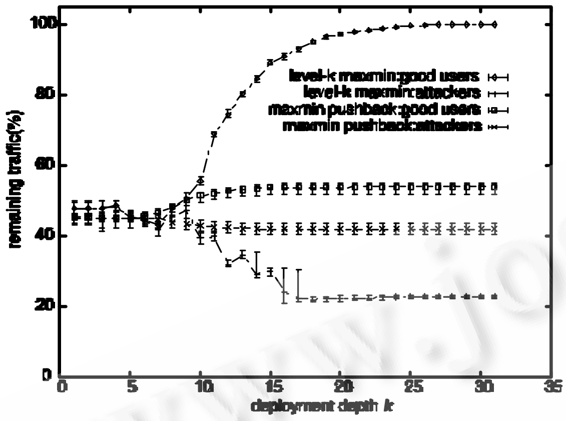
Theorem 1. If $d \le r_U - r_L$, the throttle negotiation algorithm will converge after a multiplicative decrease process and an additive increase process. The time for multiplicative decrease process, $j\in\left[\log_2\frac{r(0)}{r_U},\log_2\frac{r(0)}{r_U}+1\right)$. The time for additive increase process, $l\in\left(\frac{2^j r_L-r(0)}{2^j d},\frac{2^j r_L-r(0)}{2^j d}+1\right]$.

Proof. Omitted.

3 Performance Evaluation and Experimental Results

We conducted simulations using a network topology reconstructed from real traceroute data (from the Internet mapping project at AT&T, http://cm.bell-labs.com/who/ches/map/dbs/index.html), which 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 the server S, and randomly select a subgraph G' from the original data set as the graph of all paths delivering traffic to S. G' has 135 821 nodes, of which 3 879 are hosts (either an attacker or a good user). We set L_S=4700 and U_S=5300. Hosts are modeled to send traffic to S at constant rates, which are randomly and uniformly drawn from the range [0,2] for a good user, and from the range $[0,r_a]$ for an attacker. The performance of algorithms is shown by remaining percentage of attacking and good user traffic over the throttling level k. We plot the average results over ten independent experimental runs, (the attackers, good users and their rates are rechoosed for each run), and show the standard deviation as an error bar around the average. For a comparison, in each simulation experiment, we also provides results for a fully pushback max-min fairness as described in Ref.[10], which deploys to the same depth of k.

Figure 4 compares the performance of two algorithms for evenly distributed attackers, where each host in the network is independently chosen to be an attacker with probability p, and a good user with probability $1-p$.



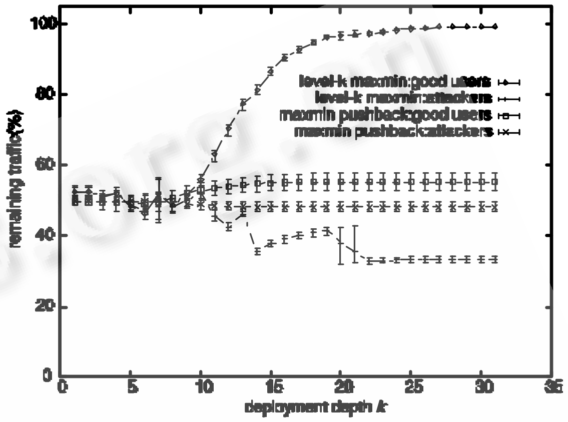
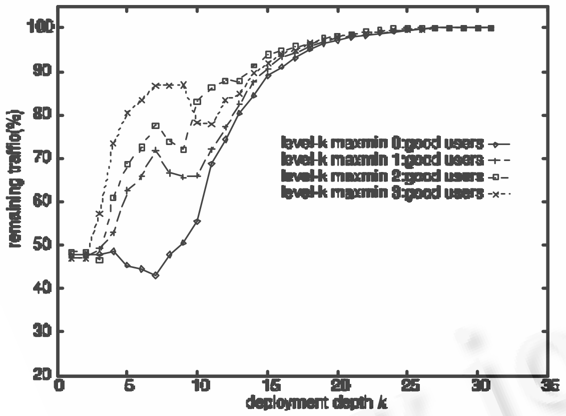


Fig.4 Performance comparing for evenly distributed attackers: r_a=20 and p=0.2 (left) and r_a=10 and p=0.4 (right)



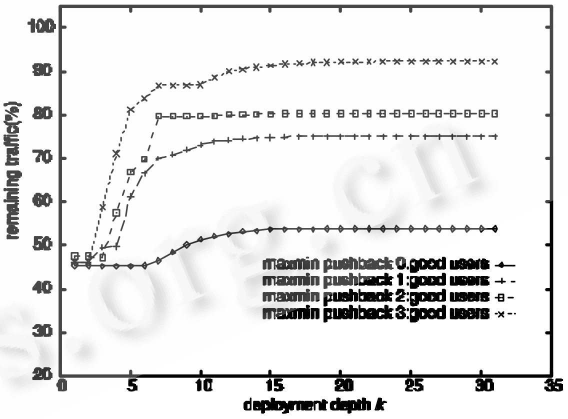


Fig.5 Performance comparing for unevenly distributed attackers: our algorithm (left) and pushback (right)

Figure 5 compares the performance of two algorithms for unevenly distributed attackers. The attackers' distributions have different *concentration* properties. Specifically, we pick five disjoint subtrees of G', of which properties are shown in Table 1. We then define four concentration configurations, 0–3, for the attackers (see Table 2). The intention is for attacker concentration to increase as we go from configurations 0 to 3.

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. For example, we model evenly distributed attacker by r_a=2, p=30%, and U_S=2800. Results in Fig.6 shows both algorithms fail to distinguish between the good users and the attackers, and punish both classes of hosts equally.

To compare the cost of two algorithms, Fig.7 plots the percentage of routers involved in throttling over k.

Table 1 Properties of subtrees 1~5

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 2 Configured concentrations of attackers

Configuration	Attackers uniformly chosen from
0	G'
1	all the five subtrees
2	subtrees 1 & 3
3	subtrees 4 & 5

(For the level-k approach, we count both throttling routers and the routers between S and $R(k)$ routers which deliver throttle messages.) Notice that the two approaches basically require a comparable number of deployment points.

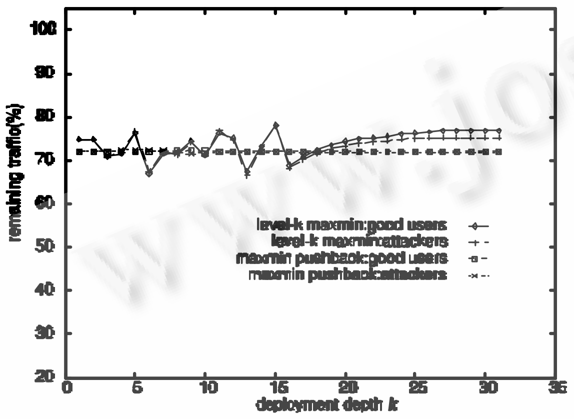


Fig.6 Performance comparing for evenly distributed attackers of r_a=2, p=30%

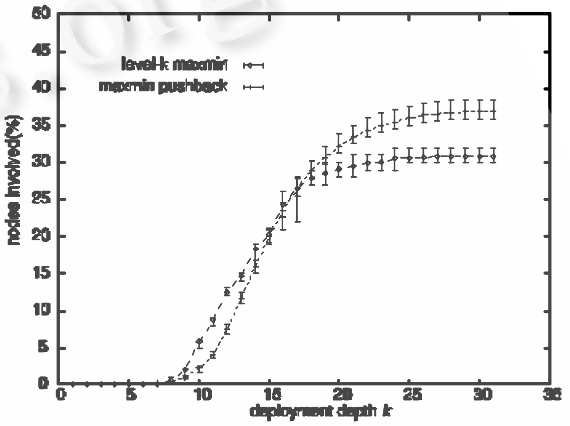


Fig.7 Number of participating routers as a function of the deployment depth

Figure 8 investigates the effects of user dynamics (for both good users and attackers) on level-15 throttle algorithm. 20% of the hosts in G' are chosen to be attackers, and the rest are good users. The attackers are evenly distributed over G'. 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 notice that good user traffic is still protected from attacker traffic. Fig.9 shows how the throttle rate r_S evolves over time.

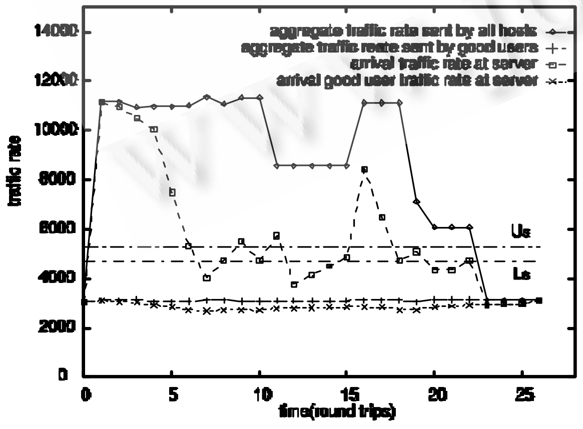


Fig.8 Algorithm response to attacker and good user dynamics

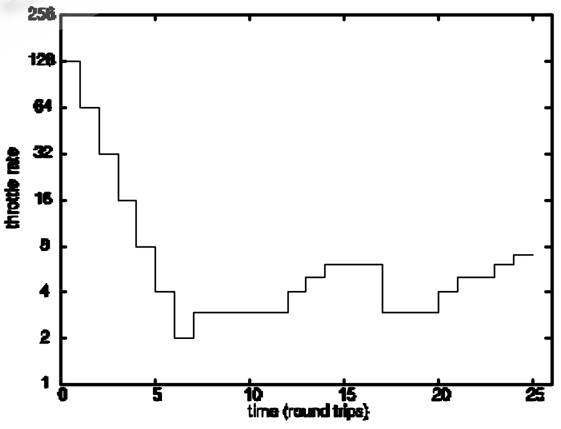


Fig.9 Evolution of throttle rate r_S over time

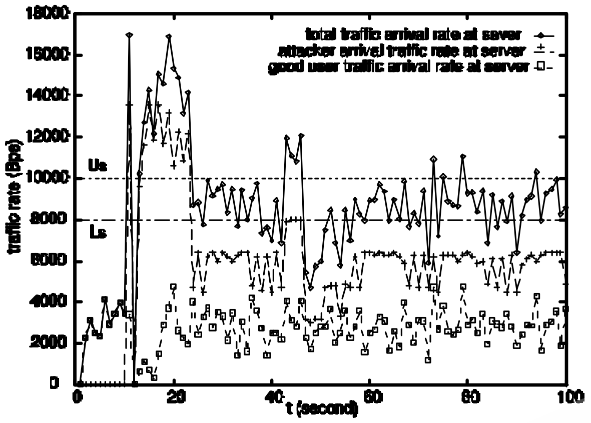


Fig.10 NS-2 simulation results for level-10 throttling

To evaluate our scheme on protecting a web server under DDoS attack, we use the Network Simulator NS-2 developed at Lawrence Berkeley Laboratory (LBL) and UC Berkeley. The simulated network topology is also from the AT&T traceroute, however because of the limit of computation ability, we only chose a small graph with 85 hosts, 17 of which are attackers. Every good user is simulated by an http traffic generator (http://www.tomh.org/software/httptrafficgen.tar), which connects S with HTTP 1.0 over TCP Reno/IP. The attackers generate UDP traffic at 6k bps to S. We model U_S=10 kBps, and make L_S=8 kBps. The throttles are set in level-10 routers. Figure 14 shows the experiment result. The attack starts at time t=10s. Notice that the throttle negotiation algorithm effectively keeps the actual server load between L_S and U_S, and the traffic dropping is most on the attackers' traffic.

4 Discussions

Several observations are in order about the practical deployment of our defense mechanism. First, to ensure reliability in installing router throttles, throttle messages must be authenticated before an edge router (assumed to be trusted) admits them into the network, and must be efficiently and reliably delivered from source to destination. Second, to ensure that the throttle mechanism remains operational when the server transiently experience resource overload, we can deploy a helper machine to monitor the traffic and initiate defense actions. Third, the throttle mechanism may not be universally supported in a network. Our solution remains applicable provided that most $R(k)$ routers on attacking paths support the mechanism. Fourth, 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. Fifth, while the sever itself can be protected by our approach, the intermediate routers between the server and $R(k)$ routers are also protected as the excess traffic is dropped by $R(k)$ routers. Sixth, 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.

References:

[1] CERT Advisory CA-1996-21 TCP SYN flooding and IP spoofing attacks. http://www.cert.org/ advisories/CA-1996-21.html.

[2] CERT Advisory CA-1998-01 Smurf IP denial-of-service attacks. http://www.cert.org/ advisories/CA-1998-01.html.

[3] Banga, G., Drusched, P., Mogul, J. Resource containers: a new facility for resource management in server systems. In: OSDI, ed. Proceedings of the 1999 USENIX/ACM Symposium on Operating System Design and Implementation (OSDI' 99). New Orleans, LA: OSDI, 1999. 45~58.

[4] Spatscheck, O., Peterson, L. Defending against denial of service attacks in scout. In: OSDI, ed., Proceedings of the 1999 USENIX/ACM Symposium on Operating System Design and Implementation (OSDI' 99). New Orleans, LA: OSDI, 1999. 59~72.

[5] Meadows, C. A formal framework and evaluation method for network denial of service. In: PCSFW, ed., Proceedings of the 1999 IEEE Computer Security Foundations Workshop. Mordano: IEEE Computer Society Press, 1999. 4~13.

[6] Savage, S., Wetherall, D., Karlin, A., *et al*. Practical network support for IP traceback. In: ACM, ed., Proceedings of the ACM SIGCOMM2000. Sweden: ACM, 2000. 295~300.

[7] Song, D., Perrig, A. Advanced and authenticated techniques for IP traceback. In: INFOCOM ed., Proceedings of the IEEE INFOCOM2001, Anchorage, Alaska: INFOCOM, 2001.
[8] Park, K., Lee, H. On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack. In: INFOCOM, ed. Proceedings of the IEEE INFOCOM' 2001. Anchorage, Alaska: INFOCOM, 2001.
[9] Ferguson, P., Senie, D. RFC2827: network ingress filtering: defeating denial of service attacks which employ IP source address spoofing. 2000. http://www.ietf.org/rfc/rfc2827.txt.
[10] Mahajan, R., Bellovin, S., Floyd, S., *et al*. Controlling high bandwidth aggregates in the network. Technical Report, ACIRI and AT&T Labs Research, 2001. http://www.icir.org/pushback/pushback-Jul01.pdf.
[11] Fang, W., Peterson, L. Inter-AS traffic patterns and their implications. In: IEEE, ed. Proceedings of the IEEE Global Internet Symposium. Rio: IEEE, 1999.
[12] Kent, S., Arkinson, R. RFC 2401: security architecture for the Internet protocol. 1998. http://www.ietf.org/rfc/rfc2401.txt.
Gouda, M.G., Elnozahy, E.N., Huang, C.T., *et al*. Hop integrity in computer networks. In: IEEEICNP, ed. Proceedings of the 8th IEEE International Conference on Network Protocols. Osaka: IEEE, 2000. 3~11.

利用路由器自适应限流防御分布拒绝服务攻击

梁 丰[1], David Yau[2]

[1](浙江工业大学 浙江省光纤通信技术重点实验室,浙江 杭州 310014);
[2](普度大学 计算机科学系, IN47907,美国)

摘要: 提出一种自适应路由器限流算法防御分布拒绝服务攻击的机制.该算法的关键是由被攻击者要求经挑选的相距 k 跳(hop)的上游路由器对目的为被攻击者的数据流进行限流,从而将被攻击者的服务支援在各数据流之间达到一种类最大-最小公平的流量分配.还在一个实际的因特网拓扑上针对攻击数据流和合法数据流的不同分布和流量模型考察了算法的效果.结果表明这种以服务器为中心的路由器限流是对抗分布拒绝服务攻击的一种很有前途的方法.

关键词: 网络安全;分布拒绝服务;路由器;因特网;计算机网络
中图法分类号: TP393 **文献标识码**: A