The Taming of The Shrew: Mitigating Low-Rate TCP-Targeted Attack

Chia-Wei Chang[†], Seungjoon Lee[‡], Bill Lin[†] and Jia Wang[‡]
[†]*University of California San Diego, La Jolla, CA 92093-0407*
[‡]*AT&T Labs-Research, Florham Park, NJ 07932-0971*
{*chc019, billlin*}*@ece.ucsd.edu,* {*slee, jiawang*}*@research.att.com*

Abstract

A Shrew attack, which uses a low-rate burst carefully designed to exploit TCP's retransmission timeout mechanism, can throttle the bandwidth of a TCP flow in a stealthy manner. While such an attack can significantly degrade the performance of all TCP-based protocols and services including Internet routing (e.g., BGP), no existing scheme clearly solves the problem in real network scenarios. In this paper, we propose a simple protection mechanism, called SAP (Shrew Attack Protection), for defending against a Shrew attack. Rather than attempting to track and isolate Shrew attackers, SAP identifies TCP victims by monitoring their drop rates and preferentially admits those packets from victims with high drop rates to the output queue. This is to ensure that well-behaved TCP sessions can retain their bandwidth shares. Our simulations indicate that under a Shrew attack, SAP can prevent TCP sessions from closing, and effectively enable TCP flows to maintain high throughput. SAP is a destination-port-based mechanism and requires only a small number of counters to find potential victims, which makes SAP readily implementable on top of existing router mechanisms.

1. Introduction

While a typical Denial-of-Service (DoS) attack [17] uses a large volume of traffic to disrupt the availability of network services (e.g., HTTP, routing, etc.), recent results show that a carefully-designed low-rate attack flow can throttle the bandwidth of a TCP flow in a stealthy manner [10]. Often referred to as a Shrew attack [10] or a RoQ (Reduction of Quality) attack [8], this type of attack exploits TCP's retransmission time-out (RTO) mechanism and uses attack bursts that are synchronized with the RTO value. Then, when a node retransmits a packet after retransmission timer expiration, the packet is likely to reach a router that is already inundated with the synchronized burst, which leads to repeated packet drops of the TCP flow. Zhang et al. [19] have shown that such a low-rate TCP-targeted attack can have severely negative impact on the Border Gateway Protocol (BGP), the de-facto standard inter-domain routing protocol in today's Internet. In particular, Zhang et al. [19] demonstrated that BGP routing sessions on current commercial routers are susceptible to such Shrew attacks launched remotely, leading to session resets and delayed routing convergence. This result implies that Shrew attacks can potentially disrupt routing stability and network reachability in the entire Internet.

Although the feasibility and potential impact of this attack have been known for some time, only a few approaches have been proposed to mitigate this type of low-rate Shrew attacks, none of which clearly solves this problem. Kuzmanovic and Knightly first investigated the use of an active queue management (AQM) scheme to mitigate Shrew attacks [10]. Although they experimented with a rather sophisticated scheme called RED-PD [13], which takes drop history of each flow into account, they found that it cannot satisfactorily mitigate the attack, which is also consistent with our experimental observations. They also explored a method that randomizes the TCP parameter minRTO to make a synchronized attack more difficult. Although this approach slightly changes the behavior of the flows under attack, it does not entirely solve the problem. Techniques based on sophisticated signal analysis [3], [12] (e.g., frequency or wavelet based) have also been proposed for Shrew attack detection. However, none of these detection schemes have been shown to be sufficiently accurate or scalable for deployment in real networks.

In this paper, we present a simple priority-tagging filtering mechanism, called SAP (Shrew Attack Protection), that protects well-behaved TCP flows against low-rate TCP-targeted Shrew attacks. In this scheme, a router maintains a simple set of counters and keeps track of the drop rate for each potential victim. When the monitored drop rates are low, all packets are treated as normal. However, if the drop rate for a certain victim becomes higher than some dynamically determined threshold (called fair drop rate), the router treats packets for this victim as high-priority, and these high-priority packets are preferentially admitted to the output queue. SAP keeps tagging victim packets as high priority until their drop rate is below the fair drop rate. By preferentially dropping normal packets to protect high-priority packets, SAP can prevent low-rate TCP-targeted Shrew attacks from causing a well-behaved TCP flow to lose multiple consecutive packets repeatedly. This simple strategy protects well-behaved TCP flows away from near zero throughput (due to slow start) under an attack. As SAP focuses on protecting TCP flows against Shrew attacks, we envision that SAP is used in conjunction with other systems that are more effective for different types of network attacks [14], [17]. In fact, SAP can help such systems by providing more information when some of the monitored applications experience unusually high packet drop rates.

Since keeping the information about per-flow state is typically prohibitive for a router to maintain, SAP aggregates flows and maintains statistics for each aggregate. While dif-

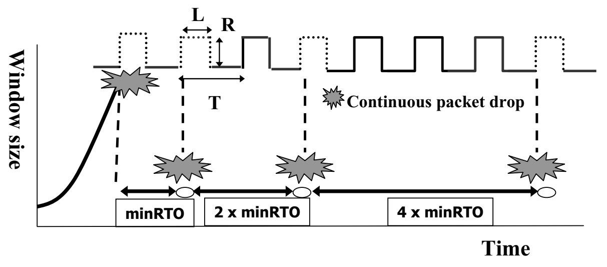


Figure 1. A Shrew attack uses a periodic on-off wave with period *T*, burst rate *R*, and burst length *L* to cause repeated packet drops for TCP flows.

ferent levels of aggregation obviously lead to different performance trade-off between accuracy and memory/computation requirements, in this paper, we use the application-level granularity to identify potential victims. Specifically, we identify the application of a packet based on the value of the destination port field in the TCP/IP header (regardless of the source and destination IP addresses) and maintain drop rate statistics for each port. Due to this aggregation, SAP may not be able to fully protect legitimate traffic from attack flows all the time. Furthermore, some attackers may try to evade SAP by using multiple destination ports or exploit SAP by manipulating the drop rate of particular ports. In this paper, we illustrate a number of attack scenarios when SAP is employed, and present experiment results where SAP performs well in these adversarial scenarios.

We conducted extensive evaluations using both an actual commercial Internet router testbed as well as ns-2 simulations. In our experiments, SAP can effectively protect victims from Shrew attacks whereas an AQM scheme alone (e.g., RED) cannot. In particular, in simulations involving a mix of normal TCP flows and a BGP session, we show that a Shrew attack can cause the BGP session to close and increase the drop rate of normal TCP traffic to near 100%, resulting in a degradation in the performance of normal TCP traffic to near zero throughput. When we employ SAP, the drop rate of normal TCP traffic only increased by 1.1%, allowing normal TCP traffic to retain most of their throughput, and we observe that the BGP session remained active with no loss in performance. We also consider a number of adversarial scenarios, and we demonstrate that SAP performs well when multiple destination ports have a drop rate higher than the fair drop rate.

The rest of the paper is organized as follows. Section 2 first reviews the key characteristics of Shrew attacks. Section 3 introduces our proposed SAP approach and the details of its central components. Section 4 presents the evaluation results based on simulation and testbed experiments. Section 5 reviews related work, and Section 6 concludes.

2. Background: Shrew Attack

Shrew attack [10] is a low-rate DoS attack that attempts to deny bandwidth to TCP flows while sending at sufficiently low average rate to elude detection by counter-DoS mechanisms. Fig. 1 illustrates a single source Shrew attack with a packet stream of a square waveform that has an attack period T, a burst length L, and a peak rate R. Kuzmanovic and Knightly [10] showed that such an attack can reduce the throughput of TCP flows to near zero throughput or cause session resets if the attack has the following characteristics: (1) R is large enough to induce victim's packet loss (i.e., R aggregated with existing traffic volume exceeds the link capacity); (2) L is long enough to induce timeout (e.g., typically no less than the round-trip time), but sufficiently short to elude detection; and (3) T is chosen such that when flows attempt to exit timeout, they will face continuous drop (i.e., T is scaled in accordance to the minRTO).

The rationale behind this form of attack is as follows. When the initial attack burst of a Shrew attack causes packet drops for a TCP flow, the TCP sender will wait for the retransmission timer to expire before it starts to retransmit. As such a retransmission timeout value is typically an integer multiple of the minRTO, subsequent retransmissions encounter another attack burst and are dropped repeatedly because the attack interval is synchronized with the retransmission timeout value. As a result, the TCP flow fails to exit the timeout phase and experiences near-zero end-to-end throughput or a session close. Moreover, most TCP implementations use among a small set of fixed minRTO values[1], which makes a single Shrew attack effective for a large set of TCP flows [10], [2], [19]. More specifically, Kuzmanovic and Knightly [10] show that the normalized TCP throughput under a Shrew attack is:

$$\mu_{norm}(T) = \begin{cases} \frac{T-\text{minRTO}}{T} & \text{if } T \geq \text{minRTO} \\ \frac{2T-\text{minRTO}}{T} & \text{if } T < \text{minRTO} \end{cases} \quad (1)$$

where T is the attack period and L the peak length.[2] This shows that a Shrew attack with sufficient peak rate and $T =$ minRTO can cause the TCP throughput to become zero.

3. Shrew Attack Protection

3.1. Overview

The main idea of SAP is to neutralize a Shrew attack by controlling the drop rates of TCP flows at the application-aggregate level via the use of differential packet prioritization. Fig. 2 depicts the high-level architecture of SAP. In this scheme, we monitor drop rates (Drop Rate Collector) of application-aggregates, based on which we identify potential victims. If all drop rates are under a certain dynamically-adjusted threshold (called fair drop rate), then all packets are considered *low* priority and equally compete to be admitted to the output queue, some of which may get dropped based on the AQM (Active Queue Management) policy when the output queue is (nearly) full. We also continuously adjust the fair drop rate based on the history of packet arrivals and drops over all packets (Fair Drop Rate Controller). If we detect that

1. Juniper routers use 1000ms as minRTO whereas Cisco routers use 300ms and 600ms depending on the models.

2. Eq. 1 assumes $L >$ RTT and minRTO $> (\text{SRTT} + (4 * \text{RTTVAR}))$ for all flows, where RTT, SRTT, and RTTVAR correspond to the round-trip time, the smooth round-trip time, and the round-trip time variation.

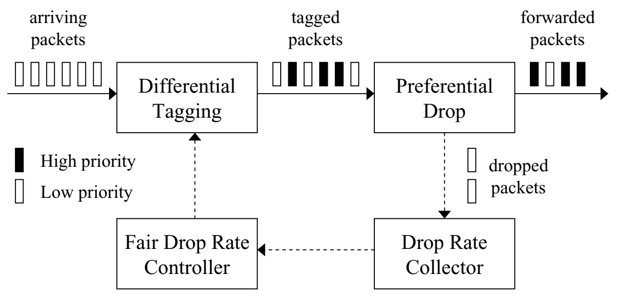


Figure 2. SAP Architecture

a drop rate for a particular aggregate grows higher than the fair drop rate, we start to protect the victim by tagging its TCP packets as *high* priority to lower the victim's drop rate (Differential Tagging). After packets are tagged, SAP passes all packets to the priority AQM module in the router, which implements preferential packet dropping.

Note that this preferential dropping mechanism is already available in modern routers [5]. As the AQM module prefers to admit high-priority packets over low-priority packets when a link is heavily congested, it is more likely to forward packets from victims, which allows the victim TCP flows to get their packets through, thus diminishing the drop rate and neutralizing the Shrew attack. While we can use SAP with any existing AQM scheme that implements preferential drops, in our experiments in Section 4, we use WRED [15] and our results demonstrate that SAP can indeed neutralize the impact of Shrew attacks on all legitimate TCP flows from Shrew attacks.

In SAP, we use application-level aggregates for drop rate monitoring and victim identification rather than flow-level drop rates, because using per-flow state information is typically impractical in practice. In this paper, for the simplicity of exposition, we use the destination port in the TCP/IP header of each packet to identify the application aggregate. As a result, if two different flows (even from distinct sources or to distinct destinations) use the same destination port, our scheme treats them as a single aggregate. Note that we can easily generalize it to other aggregation levels. Alternatively, as often used in modern routers, we can employ a hash of flow description fields in the packet [9]. While we also can consider using different fair drop rates for different types of applications (e.g., real-time applications vs. file transfer), in this paper, we use a single fair drop rate for simplicity.

For each aggregate (or destination port), we maintain two set of counters (for arrival and drop) and use them to identify victim applications. Since there are at most $2^{16} = 65536$ distinct destination ports, SAP can be easily implemented in hardware. In fact, the number of applications that need to be monitored are likely to be much smaller in practice (e.g., there are a few thousands ports that are commonly used on the Internet). Hence, SAP can be used to protect all legitimate TCP-based protocols, although our work was initially motivated in part by the BGP attack scenario [19]. We further elaborate on this aspect in Section 3.3.

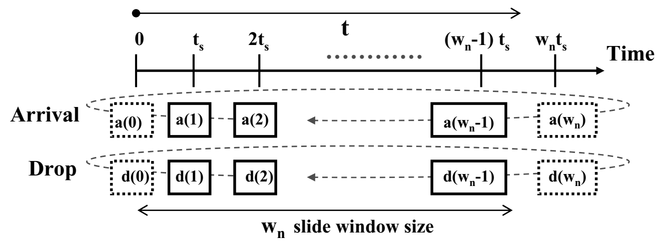


Figure 3. Drop Rate Collector by using a time-sliding window (TSW) mechanism.

3.2. SAP Components

Our proposed SAP architecture can be divided into a control plane and an execution plane. The drop rate collector and fair drop rate controller are on the control plane, whereas the differential tagging component and the preferential dropping component are on the execution plane.

3.2.1. Drop Rate Collector. The role of the drop rate collector component is to monitor drop rates. We maintain two sets of counters for each application port: one set for arrivals, and the other set for drops. The values of these counters are in terms of cumulative bytes, and we denote their respective values at time interval t by $a(t)$ and $d(t)$. In SAP, we use a time-sliding window (TSW) to provide a smooth estimate of the drop rate for each port, as depicted in Fig. 3. Each time interval is a fixed duration of t_s seconds. In particular, at the beginning of each time interval t, we initialize the arrival and drop counters for each port with $a(t) = a(t-1)$ and $d(t) = d(t-1)$. Then, during the time interval t, new byte arrivals or byte drops would increment $a(t)$ or $d(t)$ by the appropriate amount, respectively. To calculate the average drop rate for each port over a sliding window of the last w_n time intervals, we can compute it as follows:

$$p(t) = \frac{\Delta d(t)}{\Delta a(t)} = \frac{d(t) - d(t-w_n)}{a(t) - a(t-w_n)}$$

To compute average drop rates over a fixed sliding window of w_n time intervals, we simply need to maintain w_n pairs of counters for each port. By using TSW, we can recursively free and reuse counters using a circular modulo counter allocation, as depicted in Fig. 3. Therefore, the total number of counters needed per port is $2w_n$. The duration of t_s should be chosen small enough to catch the instant high drop rates while w_n should be large enough to consider the previous instant drop rates. In our experiments, we experiment with various values and focus on the results when we use $t_s = 0.1$ sec and $w_n = 10$, which results in a sliding window of $0.1 * 10 = 1$ sec.

3.2.2. Fair Drop Rate Controller. Given the historical drop rates of all ports collected, the role of the fair drop rate controller is to determine the fair drop rate threshold p_{fair} that it wants to limit. This fair drop rate threshold calculation actually depends on the average total drop rate p_{avg}. This average drop rate will be updated every t_s intervals using the same TSW algorithm:

$$p_{avg} = \frac{\sum_{i=1}^{i=N} \Delta d_i(t)}{\sum_{i=1}^{i=N} \Delta a_i(t)}$$

where $\Delta d_i(t)$ and $\Delta a_i(t)$ are the cumulative arrival and drop counts, respectively, for application port i over the last w_n intervals, and N is the number of application ports.

In addition to continuously updating the average total drop rate p_{avg}, the fair drop rate controller takes one more parameter, p_{min}. This parameter specifies a minimum drop rate threshold, under which SAP does not intervene. Specifically, if $p_{avg} \geq p_{min}$, then SAP uses the current average drop rate to serve as the fair drop rate threshold by setting $p_{fair} = p_{avg}$. If $p_{avg} < p_{min}$, then SAP sets $p_{fair} = p_{min}$. Since SAP is triggered to protect application ports with high drop rate, SAP effectively ignores any small fluctuation of error rates below p_{min}. Hence, p_{min} should be set large enough to tide over small fluctuations and low enough to trigger SAP quickly to protect victims against Shrew attacks. In our experiments, we use $p_{min} = 0.1\%$ to evaluate our scheme.

3.2.3. Differential Tagging & Preferential Dropping. Given the drop rate limits determined by the fair drop rate controller, the role of the differential tagging component is to perform the tagging of packets according to the determined fair drop rate. In particular, packets arriving are tagged as *high* priority if the instant drop rate of their application port is higher than the fair drop rate threshold set by the fair drop rate controller, which is updated every t_s intervals. Otherwise, they are tagged as *low* priority. These traffic management mechanisms for metering and tagging are commonly available in modern routers at line-speeds. Because SAP simply requires incrementing a counter in SRAM, it can easily support wirespeeds of 40 Gb/s and beyond[3].

With packets tagged on arrival, low priority packets can be dropped preferentially over high priority packets at the output queue whenever a sustained congestion occurs. Again, this preferential dropping mechanismis commonly available in modern routers at linespeeds, for example using WRED [15], or RIO [5] (e.g., Cisco 120000 Series Internet Router [4]). We can simply use these existing mechanisms to implement SAP. Under normal network conditions, in the absence of periodic bursty congestion attacks, packets will get forwarded in the same manner as without SAP.

Since we use instant drop rates for individual application ports, SAP enables each port to have some packets tagged as high priority after a relatively short sequence of packet losses. For example, suppose that p_{fair} is 5%, and port i has experienced nine successful transmissions and a packet drop since the beginning of the current window. Then, because the instant drop rate of port i is $p_i = 1/10 = 0.1 > p_{fair} = 0.05$, SAP tags next packets for port i as high priority until p_i becomes smaller than p_{fair}, which makes it more likely to transmit the subsequent packets for port i successfully. This is in contrast to the behavior of a typical drop-tail queue, where we typically experience bursty packet losses for the packets arriving after the output queue becomes full. This property of SAP helps each port avoid repeated bursty packet drops, which in turn neutralizes the Shrew Attack. In fact, this property also makes SAP effective against various attack scenarios, on which we further elaborate in Section 4.

3. At 40 Gb/s, a minimum size packet can arrive every 8 ns, which is more than enough time to increment an SRAM counter.

3.3. Discussion

a) Attack Flows Using Protected Ports: Since SAP gives high priority to packets for high-loss ports, an attacker can send bogus TCP packets to one of the protected ports in order to exploit the elevated access given to the packets for the port such that SAP cannot distinguish attack packets and legitimate packets. Although this type of attack is more effective than an attack using unprotected TCP packets or UDP packets, SAP still prevents legitimate TCP flows from session close but with low throughput. This is because SAP uses adaptive fair drop rate to serve as protection threshold. In this case, the fair drop rate is dominated by the attack flows. Therefore, the packets from attack flows are unlikely protected and stay with low priority. Instead, the incoming packets from legitimate TCP flows will be tagged as high priority and protected when they are going to stop (high drop rates). The detailed discussion and results are in Section 4.2.3.

b) Detecting Misbehaving Flows: While a DoS attack cannot exploit SAP, an attacker may attempt to increase their throughput for a particular port by causing packet drops for the port and triggering SAP protection. Since SAP aggregates all the flows using the same port, it is difficult for SAP to differentiate such misbehaving traffic. We envision SAP operates with other systems that detect such anomalous TCP flows [14]. In fact, SAP can inform those external systems as to which ports are suddenly experiencing high drop rates, so that these systems can narrow down their investigation and focus on a reduced set of potential misbehaving flows.

c) Fairness: In the backbone network, each port typically has a large number of legitimate TCP flows, while each TCP flow may have different parameters such as round-trip time (RTT), bottleneck link throughput, etc. Therefore, it is important to verify that SAP does not change the dynamics of TCP fairness, especially because flows with fewer packets to send and a shorter round-trip time might be more vulnerable to Shrew attacks. We present our experiment results in Section 4.2, which illustrate that SAP does not change the TCP fairness behavior, despite the existence of Shrew attacks.

d) State Requirements: Because SAP tracks drop rates at the application-aggregate levels, the amount of state that it needs to maintain is limited by the number of TCP application ports. Suppose we monitor all $2^{16} = 65536$ application ports, and we use a sliding window of $w_n = 10$. Then, we need to maintain $2 * 10 * 65536 \approx 1.3$ million counters. Using 32-bit counters (4 bytes), then these 1.3 million counters can be stored with approximately 5 MB of SRAM, which is well within SRAM technology today. However, the number of ports that are used by legitimate applications are an order of magnitude smaller on the Internet. Then the number of counters and the corresponding SRAM requirements can be reduced to about 0.5MB. With counters implemented in fast SRAMs, SAP can be readily implemented at Internet backbone wirespeeds.

4. Evaluation

4.1. Experimental Setup

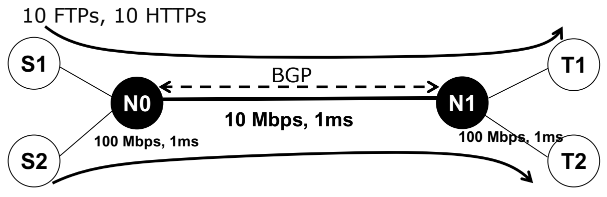
In this section, we describe experiments setup for evaluating SAP. Since we cannot easily implement our scheme into a real router, we primarily report simulation results using ns-2 simulations. However, we do perform a set of controlled experiments on a real Juniper router testbed to verify that our simulation setting is realistic.

In our experiments, we focus on three TCP applications: BGP, FTP, and HTTP. We choose them because they represent low-volume long TCP application sessions, high-volume long TCP application sessions, and high-volume short TCP application sessions. We use the FTP and HTTP modules that are available in the ns-2 package. For BGP, we incorporat the BGP module developed by Feng et al. [7].

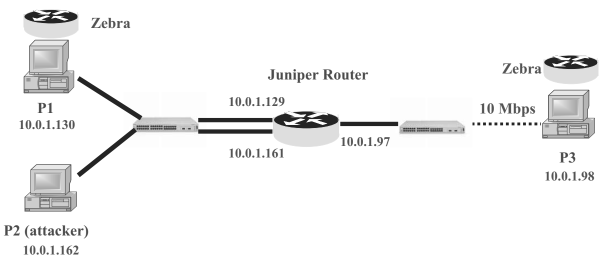
The network topology used in our simulations is shown in Fig. 4(a). We create two routers ($N0$ and $N1$) that are interconnected via a 10Mbps link. Each router is connected to two servers via 100Mbps links, labeled as ($S1$, $S2$) and ($T1$, $T2$), respectively. We set the propagation delay of each link to be 1ms and the queue size of each router to be 600 packets. A BGP session is configured between $N0$ and $N1$ that exchange routing updates according to a real BGP trace collected from a router in a Tier-1 ISP backbone network. We also set up 10 FTP and 10 HTTP sessions between $S1$ and $T1$. A Shrew attack is launched from $S2$ sending attack traffic destined to $T2$. Unless stated otherwise, we use a Shrew attack with parameters (R, L, T) = (15Mbps, 300ms, 1000ms).

We implement SAP mechanism in the ns-2 simulation code. When a packet arrives at a router, the arrival counter for the corresponding port is incremented. Similarly, whenever an output queue drops a packet, the corresponding drop counter is incremented. In our simulation, unless otherwise specified, we use WRED as the queue management scheme, which is available in ns-2. When SAP is not employed (i.e., when no differential tagging is used), WRED is the same as RED since there is only one priority class. While we have experimented with different parameters, we here present results using $t_s = 0.1$ sec, $w_n = 10$, and minRTO $= 1000$ ms. We also set $p_{min} = 0.1\%$ based on the average drop rate in our simulations when there is no attack. In our experiments, each simulation run lasts one hour.

4.1.1. Validation Using Juniper Router Testbed. Before presenting our simulation results, we compare experimental results using a real router testbed with simulation results. The goal here is to evaluate whether the results from our simulation setup are similar to those from realistic environments. As illustrated in Fig. 4(b), our testbed consists of a Juniper router, two ethernet switches, and three PCs. The default minRTO value for Juniper router is 1000ms. All links are 100Mbps



(a) ns-2 testbed



(b) Juniper router testbed

Figure 4. Network topology for experiments

Table 1. Drop rate comparison between Juniper testbed and ns-2 simulation experiments.

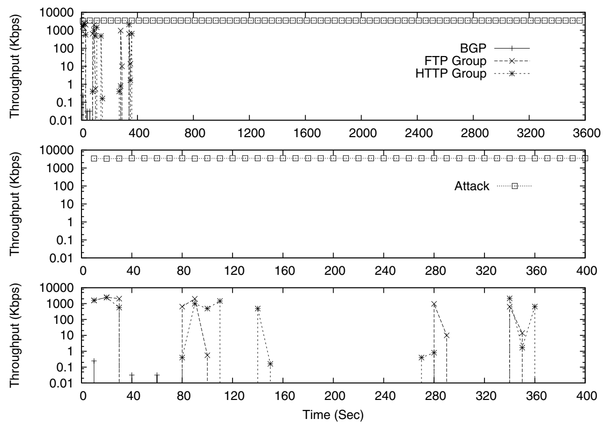
	Juniper testbed		ns-2 simulation	
Peak rate	BGP	Attack flow	BGP	Attack flow
15Mbps	17.4%	33.1%	18.1%	35.0%
18Mbps	28.1%	45.2%	28.3%	44.8%
20Mbps	28.2%	50.3%	29.0%	49.8%

except the 10Mbps bottleneck link connected to $P3$. Drop-tail queues are used in the testbed experiments. In this validation experiment, we focused on BGP session performance (i.e., there is no FTP or HTTP traffic being sent). We run the Zebra open-source routing software [1] on $P3$ and configure a BGP session between the Juniper router and $P3$. We set up a Shrew attacker at $P2$ that sends attack packets to $P3$. We fix the burst length L to 300ms and vary the peak rate R from the attacker. We also perform simulation experiment using the same settings.

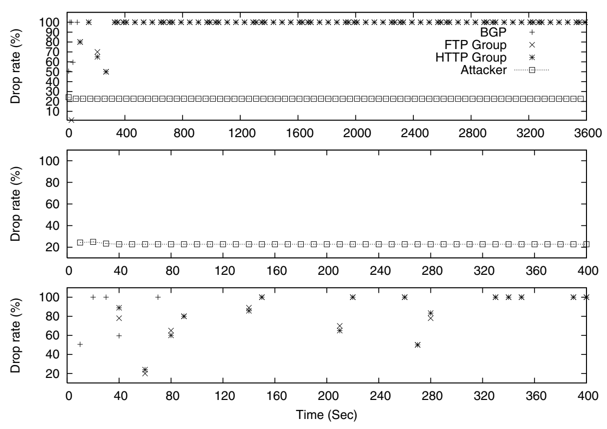
Table 1 compares the drop ratio results between the real Juniper router testbed and the ns-2 simulation. We observe that the testbed results are indeed close to the simulation results. For example, when $R = 18$Mbps, the difference in the BGP drop ratio is less than 1% (28.1% vs. 28.3%), and that in the attack traffic drop ratio is similar as well (45.2% vs. 44.8%). We also observe that when $R \geq 18$Mbps, the BGP session always closes except that the closing time differs among experiments. These results indicate that our simulation environment is indeed close to real-world scenarios. We next evaluate how SAP can effectively protect application performance under Shrew attacks using simulation experiments.

4.2. Evaluation Results

In this section, we present our evaluation results based on ns-2 simulations. We focus on two application performance metrics: *end-to-end throughput* and *drop rate*. We first show that SAP can effectively neutralize a Shrew attack if the attack uses a port which is not monitored and protected by SAP.



(a) Throughput under attack



(b) Drop rate under attack

Figure 5. Throughput and drop rate of TCP applications under Shrew attack.

Next, we show that SAP becomes less effective if the attacker uses one of the SAP protected port to launch the Shrew attack.

4.2.1. Impact of a Shrew Attack on TCP Applications. Table 2 presents the average throughputs and drop rates of flows with no attack and under attack. We observe that when there is no attack, the the BGP session throughput is around 5Kbps. The FTP sessions and HTTP sessions split the bandwidth, with each group obtaining about 5Mbps. It also shows that the drop rates of those FTP and HTTP sessions are low (around 0.2%), but the drop rate of BGP session is relatively high (around 6%). This relatively high drop rate for BGP session is because BGP trace is usually of low volume, but can be bursty.

Fig. 5 shows the performance of TCP applications under a Shrew attack of (R, L, T) = (15Mbps, 300ms, 1000ms). Due to repeated packet drops, the BGP session closes at around 70 seconds. The average drop rates of the FTP and HTTP flows gradually increase and reach 100% before 400 seconds (Fig. 5(b)), and their throughput is close to zero after that, despite the attack flow being idle during non-peak periods and consuming only around 35% of the bandwidth (i.e., 3.5Mbps of 10Mbps link bandwidth). These results show that legitimate TCP application flows can suffer a high instant drop rate and low throughput (or even session close) under a Shrew attack. These results are consistent with observations in [10].

4.2.2. Protecting Application Performance Using SAP. We next illustrate how SAP protects TCP application flows against Shrew attack by monitoring application drop rates. Table 2 shows the TCP applications performance when SAP is used to mitigate the Shrew attack. In this experiment, the Shrew attack uses TCP ports which are not monitored and protected by SAP. (The results when a Shrew attack uses SAP protected ports are shown later in this section.) We observe that the BGP flow stays alive and exchanges routing information even under an attack. In addition, all FTP/HTTP flows achieve significantly better performance than the scenario shown in Fig. 5. Specifically, the total throughput of FTP sessions and HTTP sessions are over 80% of those when there is no Shrew attack. The BGP session also achieve a similar level of throughput as when there is no attack. Compared with Fig. 5(b), we observe that SAP also significantly lowers drop rates of BGP, FTP, and HTTP flows under the Shrew attack. This is because SAP will protect legitimate TCP application flows once their instant drop rateis above the fair drop rate. These results clearly demonstrate that the simple strategy of counter-based priority-tagging used in SAP can effectively prevent TCP application flows from losing multiple consecutive packets and experiencing multiple timeouts when there is a Shrew attack.

One important parameter in SAP is the fair drop rate. Ideally, if the network administrator can measure the fair drop rate of different TCP applications, then SAP can directly use the realtime measured value as the threshold. However, obtaining such realtime measurement may not always be feasible. Instead of using the dynamic fair drop rate controller algorithm described in Section 3, we now examine how SAP performs using fixed fair drop rate. As we have observed before when there is no Shrew attack, the approximate average drop rate is 5% for the BGP session and 0.1% for the HTTP and FTP flows. In this set of experiments, we run SAP with a fixed fair drop rate of 5% and 0.1%, which we call SAP-5 and SAP-0.1, respectively, as reported in Table 2.

Note that SAP-0.1 is more aggressive in the sense that it starts preferential tagging even with a small number of packet drops. Although SAP-0.1 helps TCP flows achieve high throughput, it may unnecessarily penalize unprotected flows (e.g., UDP flows) even in the case of small flash crowds. In contrast, SAP-5 is more conservative than SAP-0.1 and does not start preferential tagging until the average drop rate reaches 5%, which results in slightly lower TCP throughput, compared to SAP-0.1. We observe that SAP performs reasonably well with fixed fair drop rates. A lower threshold usually yields better protection of TCP applications against Shrew attacks, potentially at the cost of the performance degradation of non-protected applications. For example, FTP flows achieve an aggregate throughput of 4.9Mbps with SAP-0.1, and 3.1Mbps with SAP-5.

Table 2. Performance results of using SAP with different fixed fair drop rates.

		Throughput (in Kbps)				Drop rate (in %)			
		FTP	HTTP	BGP	Attack	FTP	HTTP	BGP	Attack
Without attack		4996	4995	4.5	-	0.2	0.2	5.8	-
Under Attack	RED	≈0	≈0	≈0	3462	≈100.0	≈100.0	(close)	22.7
	SAP	3975	3870	5.4	1784	3.0	3.0	6.1	57.0
	SAP-5	3180	3165	5.1	3198	1.2	1.1	4.2	28.5
	SAP-0.1	4950	4930	6.6	110	0.2	0.2	1.1	86.0

Table 3. Impact of SAP on performance when there are 1 HTTP and 10 FTP flows.

		Throughput (in Kbps)				Drop Rate (in %)			
		FTP	HTTP	BGP	Attack	FTP	HTTP	BGP	Attack
Without attack		9069	922	5.68	-	0.07	0.07	2.9	-
Under Attack	RED	≈0	≈0	≈0	3462	≈100.0	≈100.0	(close)	22.7
	SAP	6471	674	6.2	2800	2.8	2.8	5.4	37.8
	SAP-5	4838	450	5.0	3229	0.9	0.9	5.0	28.3
	SAP-0.1	8712	858	6.9	422	0.1	0.1	1.1	90.0

We also evaluate the impact of the number of application flows on how SAP performs. In our experiments, we vary the number of HTTP flows from 10 to 1. We observe that, even though SAP is a port-based scheme, each application flow is treated equally. For example, Table 3 shows that the single HTTP flow has approximately the same throughput as the average throughput of individual FTP flows. This result illustrates that SAP allows individual flow to share the link bandwidth, even though it does not use flow-level information. Although we do not present here due to the space constraint, we also experimented with various other scenarios (e.g., using different parameters such as RTTs), and SAP works well in these settings as well.

4.2.3. Protection Against Shrew Attacks Using Protected Ports. So far our experiments have been focused on attacks using ports that are unprotected by SAP. However, a Shrew attacker can also launch attack packets using one or more ports that are monitored and protected by SAP. Here, we compare the protection capability of SAP against Shrew attacks using (1) Unprotected-Port (UP) packets, (2) Protected-Port (PP) packets with randomly chosen ports, and (3) Protected-Port packets with HTTP port (PP-HTTP). We launch 100 synchronized attack flows at *100sec* in each simulation. Each attack flow has period of *1.0sec*, burst rate of *150Kbps*, and burst length of *0.3sec*. Therefore, the total burst rate is 15Mbps lasting for *0.3sec* for each period.

The results are shown in Table 4. We observe that, when SAP is not used, all TCP application sessions close regardless of which port attack packets use. SAP can protect TCP-application flows from session close when attack packets use protected ports. Unlike Drop-tail and RED where all incoming packets are dropped if the queue is full, SAP will tag the potential victim packets high if its instant drop-rate is higher than the fair drop rate (i.e., average drop rate), and start dropping low priority packets that are already in the queue. As a result, normal TCP flows have more chances to survive during the attack burst period and be able to send packets during the attack idle period. However, the protection provided by SAP is limited in the sense that all normal TCP application flows will suffer lower throughput under Shrew attacks using protected TCP ports. We also note that SAP keeps TCP sessions alive, which is important for certain TCP applications such as BGP [19].

We also experimented with the attack scenario where the attack flow and legitimate flows share a port. The result is presented at the last row of Table 4. We observe that HTTP flows get higher throughput than other TCP application flows. This is because SAP notices a large number of consecutive packet drops for the HTTP port during the attack burst period and tags legitimate HTTP packets when the attack is idle. Compared to the case where legitimate and attack flows use separate ports (e.g., the PP case), the throughput of HTTP flows is significantly higher. Again, SAP keeps all the flows alive in this attack scenario. Although we do not present the detailed results, we also performed experiments where we remove non-HTTP flows, such that all flows (both legitimate and attack) use the same HTTP port. SAP still can keep the legitimate flow alive in this case. This is because as we mentioned earlier, SAP considers instant packet drop rate and prevents consecutive packet drops, which gives normal TCP flows more chances to survive during the attack burst period.

In summary, our extensive simulation experiments illustrate that SAP can protect TCP flows when there is an active Shrew attack.While SAP can always prevent legitimate TCP flows from closing or getting near-zero throughput, the degree of performance degradation due to an attack varies depending on the types and details of the attack.

5. Related Work

There have been a number of mechanisms proposed to protect against Shrew attack. Kuzmanovic and Knightly first investigated the use of an active queue management (AQM) scheme to mitigate Shrew attacks [10]. Although they experimented with a rather sophisticated scheme called RED-PD [13], which takes the drop history of each flow into account, they found that it cannot satisfactorily mitigate the attack, which also agrees with our own results. Another method that they explored was to randomize the fixed minRTO

Table 4. Shrew attacks using different ports.

		Throughput (in Kbps)				Drop Rate (in %)			
		FTP	HTTP	BGP	Attack	FTP	HTTP	BGP	Attack
Without attack		4996	4995	4.5	-	0.2	0.2	5.8	-
Attack	RED	≈0	≈0	≈0	3462	≈100.0	≈100.0	≈100.0	22.7
Attack (using UP)	SAP	3975	3870	5.4	1784	3.0	3.0	6.1	57.0
Attack (using PP)	SAP	83	76	1.8	3410	8.9	9.1	22	23
Attack (using PP-HTTP)	SAP	75	1760	1.7	3281	9.0	1.1	22	28

value in the TCP implementation in an attempt to mitigate the synchronized attack, which is not sufficient to fully mitigate the attack. Sun et al. [18] proposed a router-based detection approach against Shrew attack, where multiple routers along the path collectively detect the attach based on the auto-correlation analysis among attack packets. However, their detection scheme requires multiple data manipulation steps (e.g., noise filtering, feature extraction, etc.), which is often prohibitive for an on-line mechanism due to a huge number of packets going through high-speed network links. Shevtekar et al. [16] proposed an approach to detect these attack flows at edge routers by monitoring any periodic traffic pattern that is synchronized with minRTO and RTTs observed for other connections. Kwok et al. [11] proposed a scheme called HAWK, where they record attack flows into a small table and drop packets from those flows to halt the attack. In practice, it is difficult to identify and isolate the attack flows. In contrast, SAP does not attempt to identify the attack flows; it simply controls the drop rates of victim flows.

In recent years, researchers have also explored the application of signal analysis techniques to Shrew attack detection [3], [12]. Chen et al. [3] used frequency domain spectrum analysis to identify attacker flows. Luo et al. [12] proposed a wavelet-based approach to study the characteristics of low-rate TCP-targeted DoS attacks. In general, these detection algorithms are based on complicated signal analysis, which can be prohibitively expensive to realize at wirespeeds for high-speed networks. We are unaware of any such solutions that can effectively mitigate Shrew attacks in real network environments. In contrast, our proposed SAP requires a small number of counters, and we expect the hardware implementation can handle the huge number of packets in today's high-speed networks.

6. Concluding Remarks

In this paper, we proposed a simple Shrew attack protection mechanism called SAP. SAP provides network operators with a broad first line of proactive defense against Shrew attacks, significantly neutralizing their impact. By monitoring the drop rates of potential victims, SAP prevents consecutive packet drops for a victim, which we observe for well-behaved TCP flows under a Shrew attack. SAP achieves this through differentiated tagging of victims' packets and preferential admission to the output queue. Unlike other existing mechanisms, SAP focuses on protecting victims without explicitly identifying attackers. SAP is a port-based victim-detection scheme and readily deployable on top of existing router mechanisms, as SAP does not rely on any proprietary packet header information or sophisticated signal analysis techniques. Our results show that SAP is able to stop the crippling BGP attack scenario identified in [19]. More broadly, our results show that SAP is also effective in allowing TCP flows in general to recover their throughput under a Shrew attack.

References

[1] GNU Zebra-routing software. http://www.zebra.org.

[2] M. Allman and V. Paxson. On estimating end-to-end network path properties. In *Proc. ACM SIGCOMM*, 1999.

[3] Y. Chen, Y.-K. Kwok, and K. Hwang. Filtering Shrew DDoS Attacks Using A New Frequency-Domain Approach. In *Proc. IEEE LCN Workshop on Network Security*, 2005.

[4] Cisco Systems. WRED and MDRR on the Cisco 12000 Series Internet Router with a Mix of Unicast, Multicast, and Voice Traffic Configuration Example.

[5] D. Clark and W. Fang. Explicit allocation of best-effort packet delivery service. In *IEEE/ACM Trans. on Networking*, 6(4), 1998.

[6] M. A. El-Gendy, A. Bose, and K. G. Shin. Evolution of the Internet QoS and support for soft real-time applications. In *Proceedings of IEEE*, 2003.

[7] T. D. Feng, R. Ballantyne, and L. Trajkovic. Implementation of BGP in a network simulator. In *Applied Telecommunication Symp.*, 2004.

[8] M. Guirguis, A. Bestavros, I. Matta, and Y. Zhang. Reduction of Quality (RoQ) Attacks on Internet End Systems. In *Proc. IEEE INFOCOM*, 2005.

[9] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992 (Informational), November 2000.

[10] A. Kuzmanovic and E. W. Knightly. Low-Rate TCP-Targeted Denial of Service Attacks (The Shrew vs. the Mice and Elephants). In *Proc. ACM SIGCOMM*, 2003.

[11] Y. K. Kwok, R. Tripathi, Y. Chen, and K. Hwang. HAWK: Halting Anomalies with Weighted Choking to Rescue Well-Behaved TCP Sessions from Shrew DoS Attacks. In *International Conferences on Computer Networks and Mobile Computing*, 2005.

[12] X. Luo and R. K. C. Chang. On a New Class of Pulsing Denial-of-Service Attacks and the Defense. In *Proc. Network and Distributed System Security Symposium*, 2005.

[13] R. Mahajan, S. Floyd, and D. Wetherall. Controlling High-Bandwidth Flows at the Congested Router. In *Proc. International Conference on Network Protocols*, 2001.

[14] M. Roesch. Snort - lightweight intrusion detection for networks. In *LISA '99: Proceedings of the 13th USENIX conference on System administration*, pages 229–238, Berkeley, CA, USA, 1999.

[15] M. Rupinder, L. Ioannis, H. S. Jamal, S. Nabil, N. Biswajit, and B. Jozef. Empirical Study of Buffer Management Scheme for Diffserv Assured Forwarding PHB. In *ICCCN*, 2000.

[16] A. Shevtekar, K. Anantharam, and N. Ansari. Low Rate TCP Denial-of-Service Attack Detection at Edge Routers. *IEEE Communications Letters*, April 2005.

[17] S. M. Specht and R. B. Lee. Distributed Denial of Service: Taxonomies of Attacks, Tools, and Countermeasures. In *Proc. of Int'l Conf. on Parallel and Distributed Computing Systems*, 2004.

[18] H. Sun, J. C. Lui, and D. K. Yau. Defending Against Low-rate TCP Attacks: Dynamic Detection and Protection. In *Proc. International Conference on Network Protocols*, 2004.

[19] Y. Zhang, Z. M. Mao, and J. Wang. Low-Rate TCP-Targeted DoS Attacks Disrupts Internet Routing. In *Proceedings of 14th Annual Network & Distributed System Security Symposium (NDSS)*, 2007.