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

Chia-Wei Chang, Seungjoon Lee, Bill Lin, Jia Wang

***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 the victims with high drop rates to the output queue. This is to ensure that well-behaved TCP sessions can retain their bandwidth shares. Our simulation results 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.**

***Index Terms*—Shrew attack, differential tagging, fair drop rate.**

## I. Introduction

WHILE a typical Denial-of-Service (DoS) attack [22] 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 [15]. Often referred to as a Shrew attack [15] or a RoQ (Reduction of Quality) attack [13], 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. [24] 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. [24] 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 [15]. Although they experimented with a rather sophisticated scheme called RED-PD [18], which takes drop history of each flow into account, they found that it cannot satisfactorily mitigate the attack, which is also consistent with the observation from our experiments. Another method that they explored was to randomize 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 [4], [17] (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 to identify all of the possible attack patterns (e.g., more possible Shrew attack models are discussed in our previous work in Section 5 of [24]) or scalable for deployment in real networks (e.g., details are in Section V).

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. If the monitored drop rates are low, all packets are treated as normal (e.g., low-priority) and equally compete to be admitted to the output queue and only dropped based on the AQM (Active Queue Management) policy when the output queue is (nearly) full. 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

Manuscript received March 20, 2009; revised August 10, 2009 and September 23, 2009. The associate editor coordinating the review of this paper and approving it for publication was M. Sloman.

C. W. Chang and B. Lin are with Electrical & Computer Engineering, UCSD Atkinson Hall 9500 Gilman Drive #0409 La Jolla, CA 92093-0409 (e-mail: chc019@ucsd.edu).

S. Lee and J. Wang are with AT&T Labs-Research.

This paper is an extended journal version of a conference paper at the ICDCS Conference [3].

Digital Object Identifier 10.1109/TNSM.2010.I8P0308

1932-4537/10/$25.00 © 2010 IEEE

other systems that are more effective for different types of network attacks [19], [22]. In fact, SAP can help such systems by providing more information when some of the monitored applications experience unusual 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 different 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 experimental 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. In addition, we evaluate performance of SAP when regular DDoS attacks use ports that SAP wants to protect. In this scenario, SAP protects legitimate TCP flows from getting zero throughput or session closing.

The rest of the paper is organized as follows. Section II-A first reviews the key characteristics of Shrew attacks. Section III introduces a high-level overview of our proposed SAP approach and the details of its central components. Section IV presents the evaluation results based on simulation and testbed experiments. Section V reviews related work, and Section VI concludes.

## II. Problem Statement

In this section, we first present the background of Shrew attack. The research challenge is to devise a simple and efficient scheme to protect the well-behaved TCP applications. Previous approaches have focused on identifying such Shrew attack flows by searching some well-defined “syndromes" in every passing flow, which are very expensive to implement and typically impractical in practice. Instead, we try to solve this problem by identifying potential victims. After analyzing the common behaviors of the victims under Shrew attack, we find out that they all share one feature, successive high drop rates, prior to session close. Therefore we proposed a protection scheme to prioritize packets from these victims by managing their drop rates.

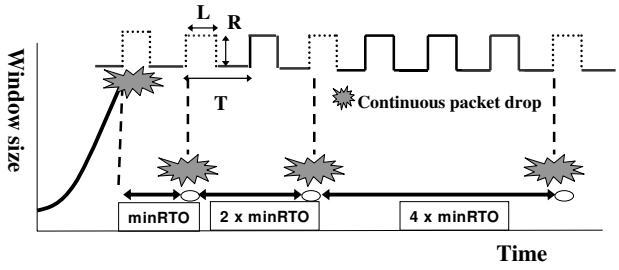



Fig. 1. A Shrew attack uses a periodic on-off wave with period *T*, burst rate *R*, and burst length *L* to cause repeated packet drops for TCP flows.

### A. Background: Shrew Attack

Shrew attack [15] 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 [15] 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 low-rate attack is to let TCP mistake that the link is highly congested. 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 [2], [15], [24]. More specifically, Kuzmanovic and Knightly [15] show that

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

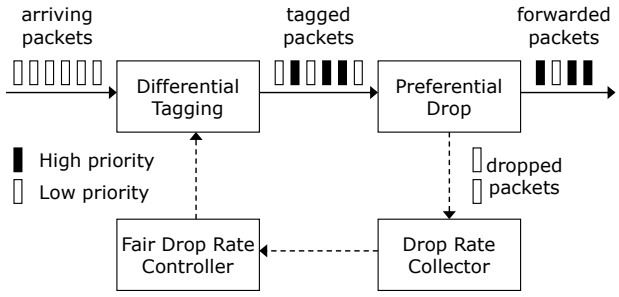



Fig. 2. SAP Architecture

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.

### B. Managing Drop Rates of TCP

In principle, TCP uses packet drops as an indication of congestion and reacts to a packet drop by reducing the rate of the corresponding flow. Assume a TCP flow under a steady-state packet drop rate of $p$, the average incoming rate can be bounded with respect to a round-trip time of $r$ seconds[3] and a average packet (segment) size of $b$ bytes. Let $A(p, r, b)$ denote the average arrival rate of this deterministic model of a TCP flow, and $\mu(p, r, b)$ as the corresponding average throughput, both in units of bytes/s. In particular, the upper-bounds for $A(p, r, b)$ and $\mu(p, r, b)$ can be simplified as

$$A(p, r, b) \leq \frac{b\sqrt{\frac{3}{2}}}{r\sqrt{p}} \quad (2)$$

$$\mu(p, r, b) \leq \frac{b\sqrt{\frac{3}{2}}}{r\sqrt{p}}(1-p). \quad (3)$$

The details could be found in [12], [18]. With respect to $r$ and $b$, the throughput of a TCP flow in Equation 3 is proportional to $\frac{1}{\sqrt{p}} - \sqrt{p}$ in the steady-state model.

As shown in Fig. 1, when a Shrew attack occurs, the TCP throughput will degrade to near zero (e.g., $\mu(p, r, b) \approx 0$) because of the bursty high drop rates (e.g., $p \approx 100\%$). Therefore, our main idea is to protect these well-behaved TCP flows by managing their drop rates. However, doing so for every single flow is typically prohibitive. In the next section, we show how well-behaved TCP flows can be protected in a practical manner by managing drop rates at an aggregate level.

[2]Eq. 1 assumes $L >$ RTT and minRTO $>$ (SRTT $+$ (4 $*$ 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.

[3]The steady-state model assumes a non-zero, non-bursty average packet drop rate of $p$, where an individual TCP connection has at most one packet drop in a time window of data.

## III. Shrew Attack Protection

### A. 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. 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[4]. Fig. 2 depicts the high-level architecture of SAP. The drop rates of application-aggregates, based on which SAP identifies potential victims are monitored by Drop Rate Collector (e.g., details are in Section III-B1). Note that SAP can easily generalize it to other aggregation levels. Alternatively, as often used in modern routers, SAP can employ a hash of flow description fields in the packet [14]. While SAP also can consider using different fair drop rates for different types of applications (e.g., real-time applications vs. file transfer), in this paper, SAP uses a single fair drop rate for simplicity which is dynamically adjusted by Fair Drop Rate Controller (e.g., details are in Section III-B2).

After the fair drop rate is determined, SAP starts to protect the victims by tagging their TCP packets as *high* priority to lower the victims' drop rate (e.g., controlled by Differential Tagging module) if their drop rates grow higher than the fair drop rate (e.g., See details in Section III-B3). Otherwise they will be tagged as normal (e.g., *low* priority). All tagged packets will be passed to the priority AQM module in the router, which implements preferential packet dropping mechanism [9], [10], [20]. In our experiments in Section IV, we use SAP with WRED [6], [20], and our results demonstrate that SAP can indeed neutralize the impact of Shrew attacks on all legitimate TCP flows from Shrew attacks. Note that SAP could be treated as a form of traffic management mechanism that aims to ensure all application flows experience similar drop rates when going through the same network link by using multiple classes/tagging on flow level.

For each aggregate (e.g., 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 [24]. We further elaborate on this aspect in Section III-C.

### B. 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.

[4]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.

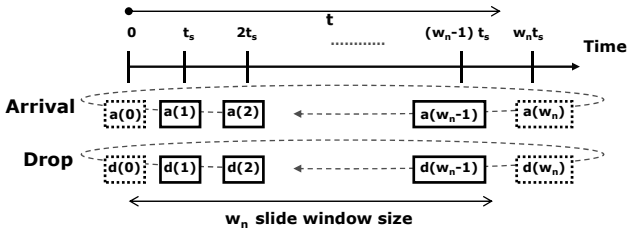



Fig. 3. Drop Rate Collector by using a time-sliding window (TSW) mechanism.

*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.

*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 $p_{avg}$ will be updated every $t_s$ intervals and will consider the previous values by 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 parameters, $p_{min}$. The parameter $p_{min}$ specifies a minimum drop rate threshold, under which SAP does not intervene. More 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 protection scheme.

*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, arrival packets 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 linespeeds. Because SAP simply requires incrementing a counter at SRAM, it can easily support wirespeeds of 40 Gb/s and beyond[5].

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 mechanism [6] is commonly available in modern routers at linespeeds, for example using WRED [20], or RIO [9] (e.g., Cisco 120000 Serious Internet Router [8]). 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 effectively against various attack scenarios, on which we further elaborate in Section IV.

Note that there are other legitimate reasons to cause the drop rates of TCP flows increase. For example on the edge, there can be lots of packet drops due to service level agreement (SLA) enforcement where it is a negotiated agreement between two parties (e.g., customer and service provider).

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

Therefore it might be interesting to see if SAP is triggered by other legitimate packet drops or interferes with other router's mechanisms where we defer it as our future work.

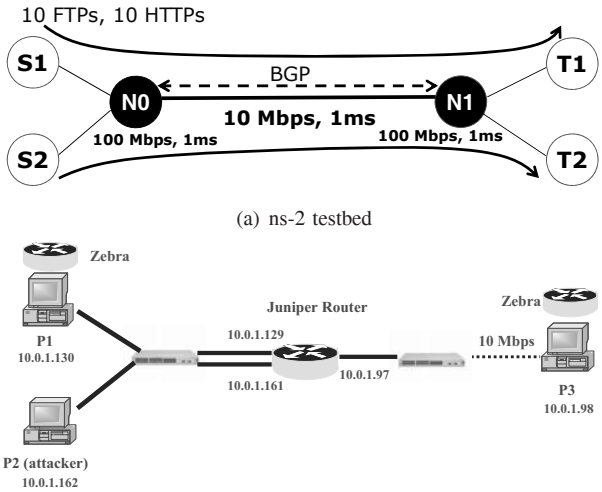
### C. Discussion

*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 IV-B3.

*b) Detecting Misbehaving Flows:* While a DoS attack cannot exploit SAP since it uses dynamically-adjusted fair drop rate of overall traffic ports, an attacker may attempt to increase their throughput for a particular port by causing packet drops for that 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 [19]. 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 see if 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 IV-B, which illustrate that SAP does not change the TCP fairness behavior, regardless of the existence of Shrew Attack.

*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 where the factor 2 stands for one for arrivals, $a(t)$, and the other for drops, $d(t)$. Although SAP maintains total $\approx$ 1.3 million counters, there is only one counter to be updated per packet arrival/drop. The oldest 2*65536 counters (e.g., $a_i(0)$ and $d_i(0)$, $i$=1 to 65536) will be recycled for the newest cumulative bytes calculation in every $t_s$= 0.1 seconds while only $2 * 2 * 65536$ counters are used (e.g., $a_i(0)$, $d_i(0)$, $a_i(w_n - 1)$, $d_i(w_n - 1)$, $i$=1 to 65536) instead of 1.3 million counters (e.g., details are in Section III.B.(1)(2)) to calculate the fair drop rate, $p_{fair}$, and average drop rate, $p_{avg}$, which will be updated in every $t_s * w_n$= 0.1*10=1.0 seconds.




(a) ns-2 testbed

(b) Juniper router testbed

Fig. 4. Network topology for experiments

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 (e.g., see IANA list in [7]). Then the number of counters and the corresponding SRAM requirements can be reduced to about 0.5MB. With counters implemented in SRAM with fast access times, SAP can be readily implemented at Internet backbone wirespeeds.

## IV. Evaluation

### A. Experiment 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. Alternatively, we can also use existing public infrastructure such as DeterLab (http://www.deterlab.net) for validation purpose, which we defer to a future work.

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 incorporate the BGP module developed by Feng et al. [11]. We also use heterogeneous RTTs in FTP and HTTP modules to create a mix of high, average and low legitimate traffic rate test-bed (e.g., details are in IV-B2).

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 ($T1$,

TABLE I
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% |

$T2$), 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 routers $N0$ 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 traffic 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.

*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 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 I compares the drop rate 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 rate is less than 1% (28.1% vs. 28.3%), and that in the attack traffic drop rate 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.

### B. 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. We also examine a number of factors that could impact the behavior of SAP in various operating environments. Next, we show that SAP becomes less effective if the Shrew attackers use one of the SAP protected port to send bogus TCP traffic. Finally, we show that SAP helps even under other attack scenarios (e.g., DDoS attacks). Note that, without attack, SAP performs just like RED because all TCP flows have similar drop rates[6]. Therefore SAP won't change the end-to-end fairness of TCP applications.

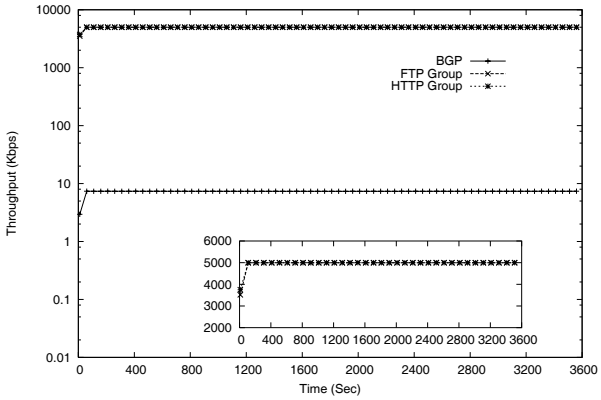
*1) Impact of a Shrew Attack on TCP Applications:* Fig. 5 shows the results of throughputs and drop rates using average value for 10-second periods with no attack and under attack. The insets of Fig. 5 present the same results of TCP applications in finer scale. We observe that when there is no attack (see Fig. 5(a)), the BGP session throughput is around 5Kbps. The FTP sessions and HTTP sessions split the bandwidth, with each group obtaining about 5Mbps. Fig. 5(b) presents 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(c) and 5(d) show the performance of TCP applications results under a Shrew attack of $(R, L, T) =$ (15Mbps, 300ms, 1000ms) without any protection mechanism. 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(d)), 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[7] 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 [15].

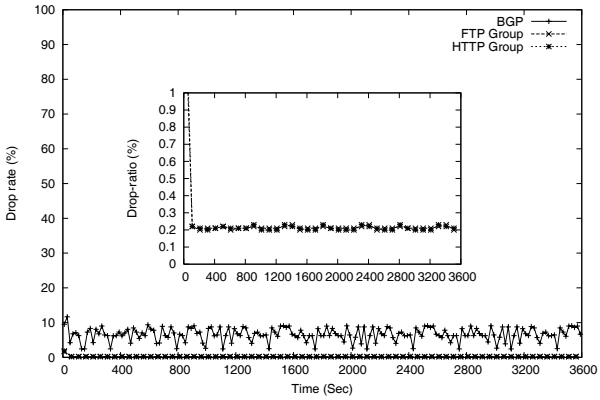
*2) Protecting Application Performance Using SAP:* We next illustrate how SAP protects TCP application flows against Shrew attack by monitoring application drop rates. Fig. 6 shows the performance of TCP applications when SAP is used to mitigate the Shrew attack and the inset presents the same results in finer scale. In this experiment, the Shrew attack uses

[6]Due to the space limitation, we omit these comparisons in most of the cases but only show a slight difference if application flows have heterogeneous RTTs in Fig III.

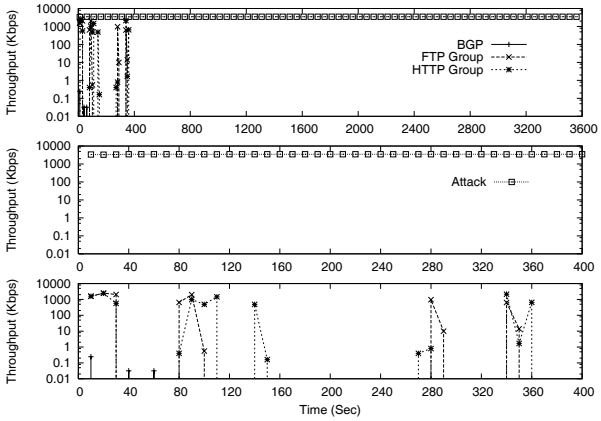
[7]Since the aggregated attack rate is 15Mbps with bursty cycles (i.e., 0.3s of 1s) and the link capacity is 10Mbps, the average bandwidth usage can be calculated as nearly 30% (0.3s/1s). Due to the effect of queue size in the router (e.g., 600 packets), the real average bandwidth usage of attack is around 35% (3462Kbps/10Mbps in Table II).



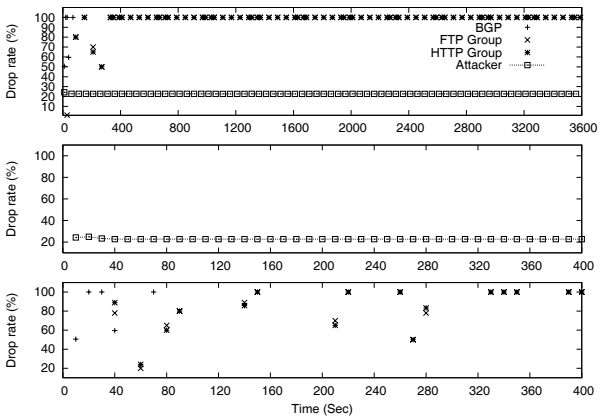

(a) Throughput without attack




(b) Drop rate without attack

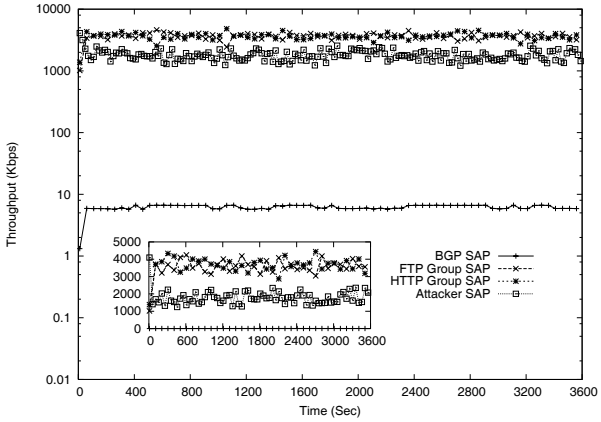



(c) Throughput under attack

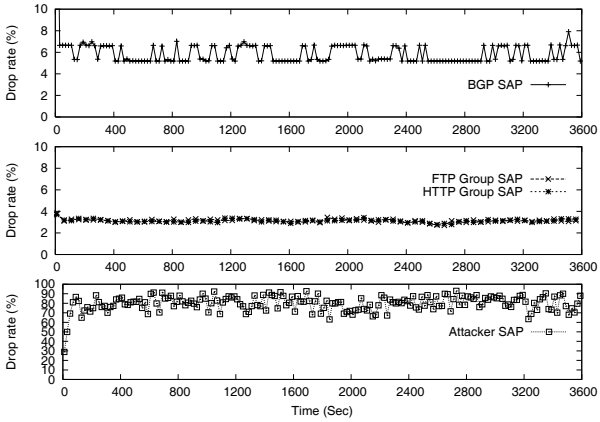



(d) Drop rate under attack

Fig. 5. Throughput and drop rate of TCP applications with/without Shrew attack.




(a) Throughput under attack (with SAP)




(b) Drop rate under attack (with SAP)

Fig. 6. Throughput and drop rate of TCP applications when SAP is used.

TABLE II
PERFORMANCE RESULTS OF USING SAP WITH DIFFERENT 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 |

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(c). Specifically,the total throughput of FTP sessions and HTTP sessions are over 80% of those when there is no Shrew attack (i.e., 4.0Mbps of 4.9Mbps throughput). The BGP session also achieve a similar level of throughput as when there is no attack. Compared with Fig. 5(d), 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 rate is 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.

Next, we explore a number of factors that could impact the behavior of SAP in various operating environments: (i) choice of fair drop rate; (ii) heterogeneity in RTT of application flows; (iii) heterogeneity in number of application flows using each port.

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 III, 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. The results are provided in Fig. 7.

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. Table II compares average throughputs and drop rates of SAP, SAP-5, and SAP-0.1. We observe that SAP performs reasonably well with fixed fair drop rates. Intuitively, a lower threshold (e.g., $P_{fair}$) usually yields better protection of TCP applications against Shrew attacks and gives more link-bandwidth to protected legitimate TCP flows , (e.g., FTP flows achieve an aggregate throughput of 4.9Mbps with SAP-0.1, and 3.1Mbps with SAP-5.), potentially at the cost of the performance degradation of non-protected applications (e.g., attack flows and normal UDP flows). Therefore the $P_{fair}$ in SAP needs to be dynamically adjusted in order not to over-protect the legitimate TCP traffic and over-punish the normal UDP traffic.

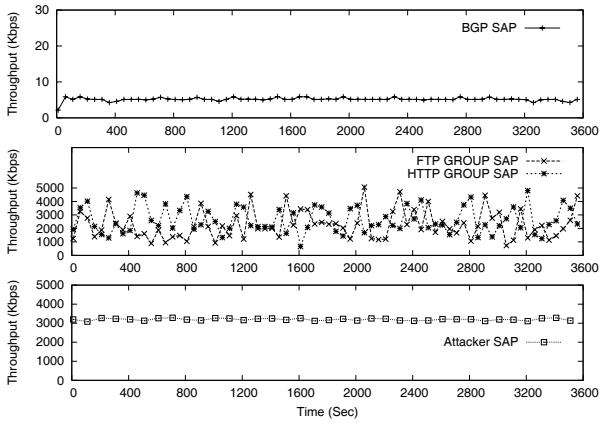
Another important factor that affect TCP application performance is its RTT(Round-Trip Time). We evaluate the impact of SAP on the performance of TCP flows of heterogeneous RTTs. In our experiments, we split each of HTTP and FTP groups to 2 small groups, where we fix the RTT of one group at 4ms and vary the RTT of the other group between 4ms, 12ms, and 22ms. Table III compares the performance of TCP flows of 22ms RTT with those of 4ms RTT. The results of application flows of 12ms RTT are similar and are not included in the paper due to space limitation. Without a Shrew attack, the drop rates of FTP and HTTP flows when SAP is used are almost the same as the ones when RED is used. When SAP is used, although flows with shorter RTT achieve higher throughput than those with longer RTT, we observe that the difference is smaller than the case of Drop-tail or RED. However, when there is a Shrew attack, SAP actually protects TCP-application flows from session close and ensures these flows have similar throughputs to those in no-attack scenarios. It is also worth noting that SAP can be used in combination with any advanced AQM algorithm which supports more than 2 classifications. In addition, different levels of fair drop rates can be used in SAP to provide more customized protection abilities. We defer these to our future work.

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 IV 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 flows to share the link bandwidth, even though it does not use flow-level information.

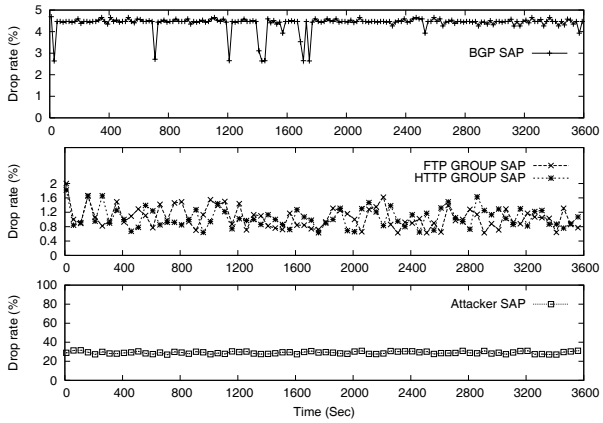
*3) Protection Against Shrew Attacks Using Protected Ports:* So far our experiments have been focused on attacks using ports that are unprotected by SAP. However, a Shrew attacker can also launch attack packets using one or more ports that are monitored and protected by SAP. Here, we compare the protection 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) to send bogus TCP packets. 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 V. 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 even when attackers use protected ports to send bogus TCP packets. First we performed experiments where the attacks use the same port as legitimate TCP flows (e.g., all flows (both legitimate and attack) use the same HTTP port). Therefore the calculation of drop rate on port 21 in SAP includes the packets sent by legitimate TCP flows and the shrew attacks.

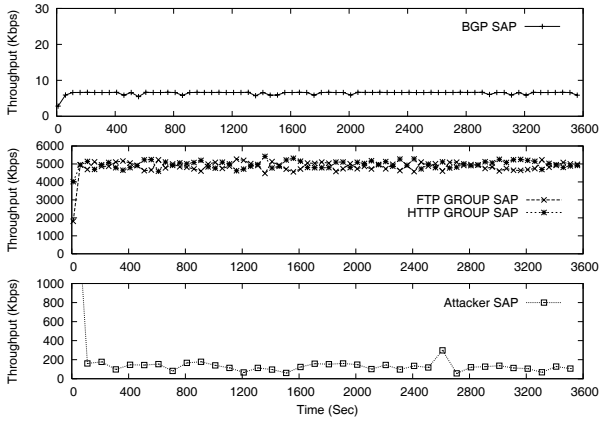
In this case, there are 10 legitimate TCP flows with 100 attack flows which all use the same port 80. We launch 100 synchronized attack flows at 100sec in simulation (total is 2000sec). Each attack flow has period of 1.0sec, burst rate of 150Kbps, and burst length of 0.3sec. Therefore, the total burst



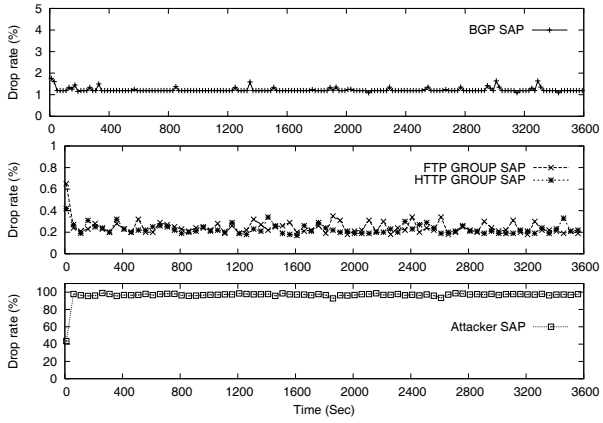

(a) Throughput under attack (with SAP-5)




(b) Drop rate under attack (with SAP-5)




(c) Throughput under attack (with SAP-0.1)




(d) Drop rate under attack (with SAP-0.1)

Fig. 7. Throughput and drop rate of TCP applications when SAP is used with different fixed fair drop rates.

TABLE III
IMPACT OF SAP ON PERFORMANCE OF APPLICATION FLOWS WITH HETEROGENEOUS RTTS.

| | | Throughput (in Kbps) | | | | | | Drop Rate (in %) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | FTP | HTTP | FTP | HTTP | BGP | Attack | FTP | HTTP | FTP | HTTP | BGP | Attack |
| Link Delay | | 22ms | 22ms | 4ms | 4ms | 2ms | 4ms | 22ms | 22ms | 4ms | 4ms | 2ms | 4ms |
| Without Attack | Drop-tail | 568 | 564 | 4421 | 4415 | 3.7 | - | 0.3 | 0.3 | 0.2 | 0.2 | 6.8 | - |
| | RED | 1861 | 1832 | 3171 | 3145 | 4.6 | - | 0.3 | 0.3 | 0.3 | 0.3 | 5.1 | - |
| | SAP | 2205 | 2201 | 2985 | 2978 | 7.1 | - | 0.3 | 0.3 | 0.3 | 0.3 | 1.1 | - |
| Under Attack | DT/RED | ≈0 | ≈0 | ≈0 | ≈0 | ≈0 | 3462 | ≈100.0 | ≈100.0 | ≈100.0 | ≈100.0 | (close) | 22.7 |
| | SAP | 1885 | 1853 | 2523 | 2518 | 6.4 | 1360 | 3.1 | 3.1 | 3.1 | 3.1 | 6.4 | 66.8 |
| | SAP-5 | 1086 | 1049 | 1857 | 1844 | 5.2 | 2998 | 2.2 | 2.2 | 1.8 | 1.8 | 4.5 | 33.4 |
| | SAP-0.1 | 2123 | 2105 | 2836 | 2814 | 6.8 | 351 | 0.2 | 0.2 | 0.2 | 0.2 | 1.1 | 92.0 |

TABLE IV
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 |

rate is 15Mbps lasting for 0.3sec for each period where the link bandwidth is 10Mbps. It means the average drop rate for attack flows is 33%. In Fig. 8, the total legitimate TCP flows from the same port under SAP protection scheme can occupy 3.8*106*8/10s = 3.0Mbps instead of session close (0kbps) (e.g., see Fig. 8(a)) where the average throughput for overall attack flows is (5.8-1.8)*106*8/10s = 3.2Mbps and the link capacity is 10Mbps. It is because even through the calculation of drop rate in SAP includes the packets sent by legitimate TCP flows and the shrew attacks (e.g., SAP cannot segregate

TABLE V
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 |

the malicious packets from legitimate packets), SAP actually protects legitimate flows at least to have average drop rate less than $P_{fair}$ (e.g., 33% in this case) by using priority-tagging. Therefore these legitimate flows can have chance to grow up during the attack idle period with frequency 0.7sec over 1.0sec (e.g., see Fig. 8(b)). Compared to the case without SAP, 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 and grow up during the attack idle period.

We next experimented with the same attack scenario but included non-HTTP legitimate TCP flows. We observe that HTTP flows get higher throughput than other TCP application flows. The result is presented in the bottom row of Table V. We also record the accumulated incoming bytes and dropping bytes of flows in different time scale granularity (i.e., 10s and 0.1s) in Fig. 9. Compared to the Fig. 8, the throughput and the drop rate of the attack remain the same while the HTTP sessions are over-protected compared to the FTP sessions and therefore they have higher probabilities to send more packets. This is because SAP notices a large number of consecutive packet drops for the HTTP port during the attack burst period and keep tagging legitimate HTTP packets high-priority when the attack is idle even their instant drop rates are higher than $P_{fair}$ since the overall average drop rate and $P_{fair}$ are dominated by Shrew attacks. Compared to the case where legitimate and attack flows both use the same ports, the throughput of FTP flows is significantly lower. Note that SAP keeps all the legitimate flows alive in this attack scenario.

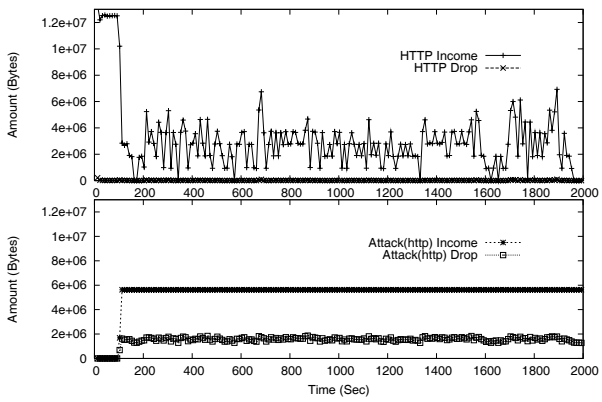
We also performed the same experiments where the attacks use one of the monitored and protected ports which is different than legitimate TCP flows. Fig. 10 records the accumulated incoming bytes and dropping bytes of flows in different time scale granularity (i.e., 10s and 0.1s). Before attack starts (100s), the FTP and HTTP sessions have nearly 5Mbps incoming rate (i.e., $6.2*10^6*8/10s$) but down to the 100Kbps after attack shows (i.e., Fig. 10(a)). The incoming rate of aggregated attackers is nearly 15Mbps (i.e., $1.8*10^5*8/0.1s$) with bursty cycles (i.e., 0.3s of 1s). We can see that with SAP, the FTP and HTTP flows can still have irregular growing opportunities to send packets during the attack idle periods (i.e., Fig. 10(b)). 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 adaptive 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 but the throughput degradation could be seen in Table V. Again, SAP keeps all TCP sessions alive.

In summary, while the throughput of legitimate flows may degrade depending on what port the attack flow uses, SAP can maintain all legitimate flows alive in various scenarios of Shrew attack, which is important for certain TCP applications such as BGP [24].

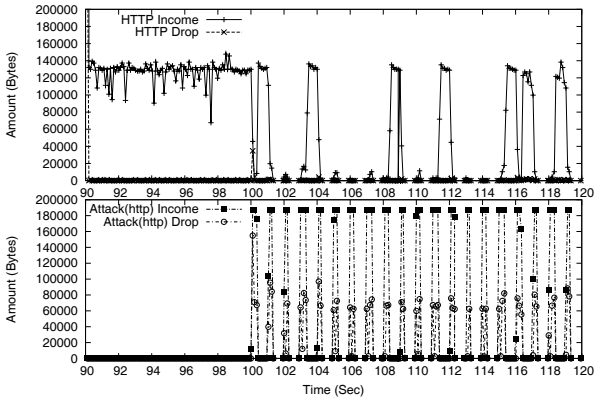
### C. Protection Against Different Types of Attack

In this subsection, we evaluate how SAP handles traditional Distributed Denial of Service (DDoS) attacks and Reduction of Quality (RoQ) attacks. The difference between them is the aggregated attack rate. DDos attacks rely on overwhelming the victim with load that constantly exceeds its link capacity; RoQ attacks, on the other hand, optimize the attack traffic to produce the maximum damage, while keeping a low profile to avoid detection. RoQ attacks do not necessarily result in a complete denial of service and usually has a lower attack rate. In the first set of experiments, we use 100 DDoS attack flows, each sending a constant packet stream at the rate of 150Kbps and each simulation run lasts one hour with all DDoS attack starting at 100 sec. Therefore, the total traffic demand of attack flows is 15Mbps. Since the link bandwidth is 10Mbps, the link is always congested in this attack scenario, which is different from the previous experiments using Shrew Attack. We also consider scenarios where attack flows use different ports as before. From Table VI, when the attack flows use a non-protected port, SAP allows legitimate flows to achieve reasonably high throughput, although the values are lower than those of the Shrew attack case shown in Table V. This is because in the DDoS attack scenario, attack has no idle period, and legitimate packets always enter congested queues. When the attack flows use a protected port or share a port with legitimate flows, the throughput of legitimate flows becomes significantly low. We again observe that all legitimate flows stay alive in this case.

We simulate RoQ attack by reducing the sending rate of each attack flows from 150Kbps to 80Kbps. Therefore, the total traffic demand for all 100 RoQ attack flows is 8Mbps which is less than the link bandwidth, 10Mbps. Table VII compares results under scenarios where attack flows use different ports as before. As expected, the throughput of TCP flows overall is higher than the case with 15Mbps attack as shown in Table VI. We observe that when the attack flows use protected port to attack, SAP yields a similar performance to RED. In this case, SAP can also alert other counter-attack systems based on the high drop rate for the particular port used by the attack. Note that SAP aims to protect TCP applications against Shrew Attack which sends at sufficiently low average rate to elude detection by counter-DDoS mechanisms. Therefore it cannot totally replace the functionality of existed counter-DoS mechanisms.

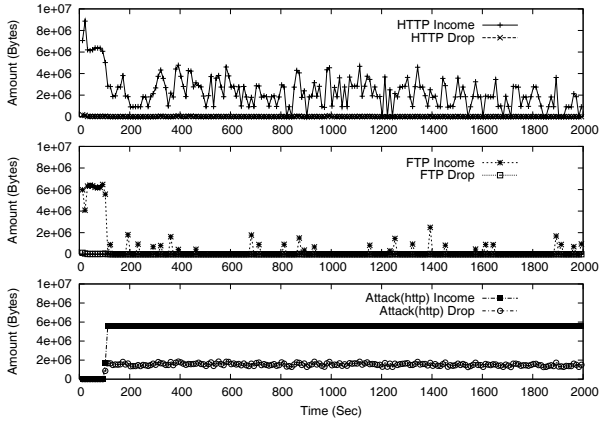



(a) Result plotted in 10s time-scale granularity

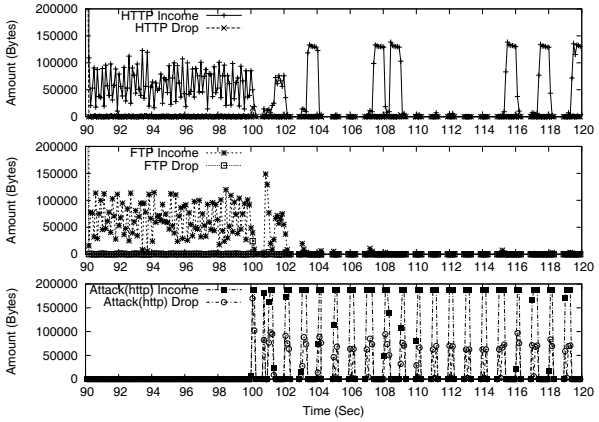



(b) Zoomed-in result plotted in 0.1s time-scale granularity

Fig. 8. TCP performance when all flows (both legitimate and Shrew attack) uses HTTP port.

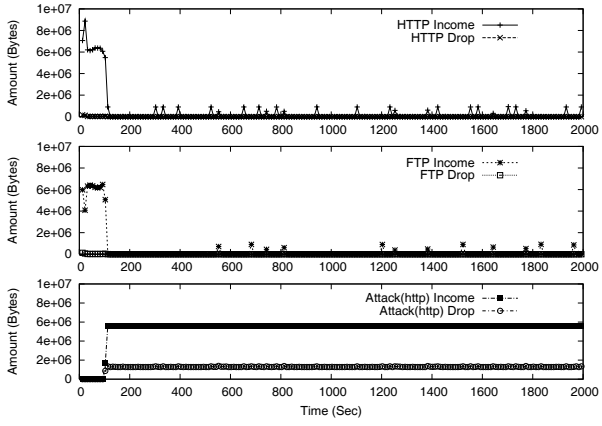



(a) Result plotted in 10s time-scale granularity

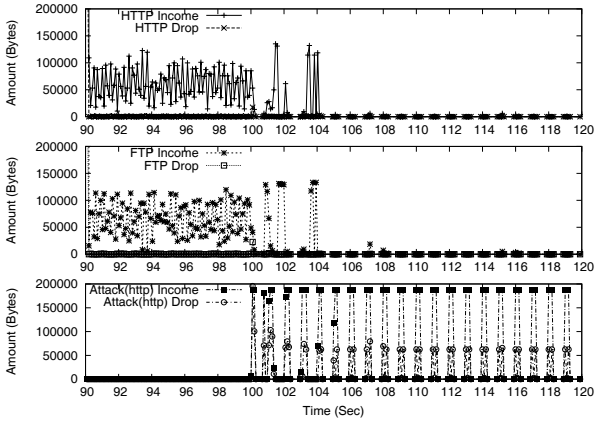



(b) Zoomed-in result plotted in 0.1s time-scale granularity

Fig. 9. TCP performance when Shrew attack uses the same TCP port monitored by SAP (HTTP).




(a) Result plotted in 10s time-scale granularity




(b) Zoomed-in result plotted in 0.1s time-scale granularity

Fig. 10. TCP performance when Shrew attack uses other TCP port monitored by SAP.

In summary, our extensive simulation experiments illustrate that SAP can protect TCP flows when there is an active Shrew attack or even a DDoS 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.

## V. 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 [15]. Although they experimented with a rather sophisticated scheme called RED-PD [18], which takes the drop history of each flow into

TABLE VI
APPLICATION PERFORMANCE UNDER TRADITIONAL DDOS ATTACK: 15MBPS

| | | Throughput (in Kbps) | | | Drop rate (in %) | | |
|---|---|---|---|---|---|---|---|
| | | FTP | HTTP | Attack | FTP | HTTP | Attack |
| Without attack | | 4996 | 4995 | - | 0.2 | 0.2 | - |
| Attack | Drop-tail/RED | ≈0 | ≈0 | 9964 | ≈100.0 | ≈100.0 | 33.6 |
| Attack (using UP) | SAP | 3191 | 3163 | 3487 | 3.2 | 3.2 | 77.9 |
| Attack (using PP) | SAP | 7.9 | 7.5 | 9961 | 21 | 22 | 33.8 |
| Attack (using PP-HTTP) | SAP | 7.4 | 34.9 | 9948 | 23.1 | 4.9 | 34.2 |

TABLE VII
APPLICATION PERFORMANCE UNDER TRADITIONAL ROQ ATTACK: 8MBPS

| | | Throughput (in Kbps) | | | Drop rate (in %) | | |
|---|---|---|---|---|---|---|---|
| | | FTP | HTTP | Attack | FTP | HTTP | Attack |
| Without attack | | 4996 | 4995 | - | 0.2 | 0.2 | - |
| Attack | Drop-tail | 1286 | 1272 | 7379 | 16 | 15 | 8 |
| Attack | RED | 1430 | 1418 | 7185 | 10 | 10 | 11 |
| Attack (using UP) | SAP | 4817 | 4815 | 246 | 1.1 | 1.1 | 95.7 |
| Attack (using PP) | SAP | 1428 | 1415 | 7190 | 10 | 10 | 11 |
| Attack (using PP-HTTP) | SAP | 584 | 2407 | 7021 | 11.2 | 1.1 | 12.3 |

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 value in the TCP implementation in an attempt to mitigate the synchronized attack, which is not sufficient to fully mitigate the attack. However, randomizing the fixed minRTO can degrade the TCP connection performance in the absence of an attack [2], because the value is selected to eliminate unnecessary TCP timeouts. Also, the minRTO value is inherently limited in its range, so the randomization strategy alone is not enough to fully mitigate the attack. Sun et al. [23] proposed a router-based detection approach against Shrew attack, which uses auto-correlation among attack packets. Multiple routers in a network need to implement their scheme, and each router along the path from attacker to destination must extract periodic signatures of the attack flow. Then, the routers collectively detect the existence of attack using a dynamic time warping method. They also proposed a fair resource allocation mechanism, which minimizes the number of affected TCP flows and provides sufficient resource protection in the presence of attack. 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.

There are other schemes that have proposed to exploit the periodic behavior of attack traffic for the detection of attack flows. Shevtekar et al. [21] proposed an approach to detect these attack flows at edge routers. Based on the flow history stored in their light-weight data structure, each edge router marks a flow as malicious if the flow shows a periodic pattern with the period equal to minRTO and the burst length is larger than or equal to the RTTs of other connections. However, normal TCP flows often exhibit a periodicity behavior as well, which makes it difficult to distinguish attack flows from well-behaved TCP flows. Kwok et al. [16] proposed a scheme called HAWK (Halting Anomaly with Weighted choKing), where they record attack flows into a small table and drop packets from those flows to halt the attack. A general drawback of this type of approaches is that they have difficulty identifying a set of distributed Shrew attack flows where each flow occupies a small bandwidth share, while the aggregate bandwidth is large enough. In particular, although HAWK can possibly tell whether a Shrew traffic exists among legitimate flows, it cannot precisely isolate malicious 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 [4], [17]. Chen et al. [4] used frequency domain spectrum analysis to identify attacker flows. Since packet arrivals for a TCP flow typically exhibit a periodicity on its PSD (Power Spectrum Density) in the frequency domain, the lack of periodicity can mean that a DoS attack is under way [5]. In addition, the PSD of multi-source DoS attacks are distinct from normal TCP flows in that they are mostly distributed in lower frequency band. Luo et al. [17] proposed a wavelet-based approach to study the characteristics of low-rate TCP-targeted DoS attacks. They proposed a two-stage algorithm that detects the existence of pulse streams by observing anomalous fluctuation in incoming traffic rates and decreases in outgoing TCP throughputs. They also validated their approach using ns-2 simulations and experiments on a NIST network testbed. Unfortunately, while the wavelet detection outcomes are largely dependent on the choice of detection parameters, it is unclear how to find an optimal set of parameters that are sensitive enough to detect low-rate distributed attacks while maintaining an acceptable false positive rate. 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. By 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.

## VI. 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 deployed 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 [24]. 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. [Online]. Available: http://www.zebra.org.

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

[3] C. W. Chang, S. Lee, B. Lin, and J. Wang, "The taming of the shrew: Mitigating low-rate TCP-targeted attack," in *Proc. IEEE ICDCS*, 2009.

[4] Y. Chen, Y.-K. Kwok, and K. Hwang, "Filtering shrew DDoS attacks using a new frequency-domain approach," in *Proc. IEEE LCN Workshop Netw. Security*, 2005.

[5] C.-M. Cheng, H. Kung, and K.-S. Tan, "Use of spectral analysis in defense against DoS attacks," in *Proc. IEEE GLOBECOM*, 2002.

[6] Cisco Systems, "Distributed Weighted Random Early Detection." [Online]. Available: http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/wred.pdf.

[7] "Official port number defined by IANA (Internet Assigned Numbers Authority)." [Online]. Available: http://www.iana.org/assignments/port-numbers.

[8] Cisco Systems, "WRED and MDRR on the Cisco 12000 Series Internet Router with a Mix of Unicast, Multicast, and Voice Traffic Configuration Example." [Online]. Available: http://www.ciscosystems.com/application/pdf/paws/22561/WRED_config_22561.pdf.

[9] D. Clark and W. Fang, "Explicit allocation of best-effort packet delivery service," *IEEE/ACM Trans. Networking*, vol. 6, no. 4, 1998.

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

[11] T. D. Feng, R. Ballantyne, and L. Trajkovic, "Implementation of BGP in a network simulator," in *Applied Telecommun. Symp.*, 2004.

[12] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the internet," *IEEE/ACM Trans. Networking*, 1999.

[13] M. Guirguis, A. Bestavros, I. Matta, and Y. Zhang, "Reduction of quality (RoQ) attacks on Internet end systems," in *Proc. IEEE INFOCOM*, 2005.

[14] C. Hopps, "Analysis of an equal-cost multi-path algorithm," RFC 2992 (Informational), Nov. 2000.

[15] 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.

[16] 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 Conf. Computer Netw. Mobile Computing*, 2005.

[17] X. Luo and R. K. C. Chang, "On a new class of pulsing denial-of-service attacks and the defense," in *Proc. Netw. Distributed Syst. Security Symp.*, 2005.

[18] R. Mahajan, S. Floyd, and D. Wetherall, "Controlling high-bandwidth flows at the congested router," in *Proc. International Conf. Netw. Protocols*, 2001.

[19] M. Roesch, "Snort—lightweight intrusion detection for networks," in *Proc. LISA '99: 13th USENIX Conf. Syst. Administration*, pp. 229-238, Berkeley, CA, USA, 1999. USENIX Assoc.

[20] 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.

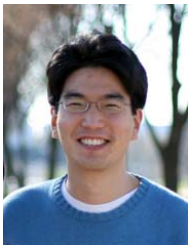
[21] A. Shevtekar, K. Anantharam, and N. Ansari, "Low rate TCP denial-of-service attack detection at edge routers." *IEEE Commun. Lett.*, Apr. 2005.

[22] S. M. Specht and R. B. Lee, "Distributed denial of service: taxonomies of attacks, tools, and countermeasures," in *Proc. International Conf. Parallel Distributed Computing Syst.*, 2004.

[23] H. Sun, J. C. Lui, and D. K. Yau, "Defending against low-rate TCP attacks: dynamic detection and protection," in *Proc. International Conf. Netw. Protocols*, 2004.

[24] Y. Zhang, Z. M. Mao, and J. Wang, "Low-rate TCP-targeted DoS attacks disrupts Internet routing," in *Proc. 14th Annual Netw. Distributed Syst. Security Symp. (NDSS)*, 2007.


**Chia-Wei Chang** received the B.S. and M.S. degrees in communication engineering from the National Chiao-Tung University, Hsinchu, Taiwan, in 2002 and 2004, respectively. He currently pursues his Ph.D. degree in electrical, computer engineering of UCSD from 2005. Mr. Chang received the 2004 Prize of the Graduate Student thesis Contest from CIEE, Taiwan and Calit2 Fellowship Award 2005 from UCSD. His research interests generally lie in information and coding theory, multimedia application, active queue management, security and anomaly detection.



**Seungjoon Lee** received his Bachelor's and Master's degrees in Computer Science from Seoul National University, Seoul, Korea, in 1996 and 2000 and his Ph. D. in Computer Science from the University of Maryland in 2006. Currently, he is a senior member of technical staff in AT&T Labs, Research. His research interests include wireless networks, mobile computing, peer-to-peer systems, and multicasting.

**Bill Lin** holds a BS, a MS, and a Ph.D. degree in Electrical Engineering and Computer Sciences from the University of California, Berkeley. He is currently on the faculty of Electrical and Computer Engineering at the University of California, San Diego, where he is actively involved with the Center for Wireless Communications (CWC), the Center for Networked Systems (CNS), and the California Institute for Telecommunications and Information Technology (CAL-IT$^2$) in industry-sponsored research efforts. His research has led to over 100 journal and conference publications. He also holds two awarded patents.

**Jia Wang** is a principal member of the Networking Research department in the Internet and Networking Systems Research Center at AT&T Labs Research. Her research focuses on Internet routing, network measurement and management, and network security. She received her MS and PhD degrees in Computer Science from Cornell University in May 1999 and January 2001, respectively. She is currently a senior member of IEEE and a member of ACM.