IP Easy-pass: Edge Resource Access Control

Haining Wang† Abhijit Bose‡ Mohamed El-Gendy‡ Kang G. Shin‡

†Department of Computer Science
College of William and Mary
Williamsburg, VA 23187

‡EECS Department
The University of Michigan
Ann Arbor, MI 48109

Abstract— **Providing real-time communication services to multimedia applications and subscription-based Internet access often requires sufficient network resources to be reserved for real-time traffic. However, the reserved network resource is susceptible to resource theft and abuse. Without a resource access control mechanism that can efficiently differentiate legitimate real-time traffic from attacking packets, the traffic conditioning and policing enforced at ISP (Internet Service Provider) edge routers cannot protect the reserved network resource from embezzlement. On the contrary, the traffic policing at edge routers aggravates their vulnerability to flooding attacks by blindly dropping packets. In this paper, we propose a fast and light-weighted IP network-edge resource access control mechanism, called *IP Easy-pass*, to prevent unauthorized access to reserved network resources at edge devices. We attach a unique *pass* to each legitimate real-time packet so that an ISP edge router can validate the legitimacy of an incoming IP packet very quickly and simply by checking its pass. We present the generation of Easy-pass, its embedding, and verification procedures. We implement the IP Easy-pass mechanism in the Linux kernel, analyze its effectiveness against packet forgery and resource embezzlement attempts. Finally, we measure its overhead and performance.**

I. INTRODUCTION

The Internet is an open system, and hence, there is no strict access control upon network resources. An end-host with a valid IP address and connection to the Internet, can inject any number of IP packets in the Internet. As a best-effort model, the current Internet is vulnerable to DoS (Denial of Service) attacks [1], [2], in which an attacker exploits IP spoofing and floods bogus packets to the victim server. The flooding traffic cripples the Internet service either by swamping the victim server or by clogging the network link. To protect Internet servers and network resources, various packet filtering techniques have been proposed and used as defensive mechanisms [3–10]. The packet filters installed at Internet servers or their nearby firewalls [3], [7], [10] only prevent such IP packets from reaching the victim, but cannot protect network resources from theft and abuse. The intermediate-router-based packet filters [5], [6], [8], [9] can block further propagation of flooding traffic in the network, but cannot protect the upstream network resources. The ingress/egress filtering techniques [4] at ISP edge routers can prevent IP packets from being spoofed as an outsider, but cannot discriminate bogus IP packets with valid IP addresses within the same local ISP.

To provide real-time communication services to multimedia applications or "subscribed" Internet users, the proposed network QoS (Quality of Service) infrastructure like Diff-Serv (Differentiated Service) [11] reserves network resources for real-time traffic. Within the network QoS architecture, however, the reserved network resources will become a conspicuous target to potential adversaries, and will be more vulnerable to packet forgery and resource embezzlement. We call such attacks, targeting at reserved network resources and violating network QoS guarantees, as DQoS (*Denial of Quality of Service*) attacks. Basically, there are two distinct DQoS attacks: (1) control flow attacks, e.g., killer reservation in RSVP (Resource ReSerVation Protocol) [12], which directly attack the signaling/control protocol in the control plane for network resource reservation and connection setup; and, (2) data flow attacks (e.g., resource theft in the data plane), in which bogus data packets grab the reserved bandwidth from the "owners," or genuine real-time data flows. Previous research efforts have focused on providing secure communication in the control plane for control flows, such as in AR-QoS [13] and Authenticated QoS [14] projects. They proposed a secure RSVP, in which resources are reserved online using strong authentication, and subsequently, compliance with the reservation request parameters is verified.

However, little attention has been paid to defend against DQoS attacks in the data plane, and block the attacking traffic from consuming the reserved network resources. Currently, the usage of reserved network resource hinges on IP addresses and the setting of the ToS (Type of Service) field in the IP header, which can be easily spoofed. Even if we secure the QoS signaling procedure, an adversary within the same stub network or at a neighboring network that is connected with the same ISP, can passively monitor the ongoing traffic towards ISP edge routers. The adversary can then impersonate as legitimate sources by flooding the spoofed data packets that have the same IP header as valid data packets. The packet filters based on packet header information only, cannot screen out these spoofed packets. Moreover, to meet the SLA (Service Level Agreement) [15] between an ISP and its end-users, edge routers perform traffic shaping and policing according to the specified traffic profiles. Without a resource access control mechanism that can efficiently differentiate legitimate real-time traffic from spoofed packets, the traffic conditioning and policing conducted at ISP edge routers cannot protect the reserved network resource from embezzlement. On the contrary, the traffic policing at edge routers aggravates their vulnerability to flooding attacks by blindly dropping packets, since flooding bogus packets can easily cause traffic violation at the edge routers. Even a small amount of attacking traffic can disrupt the loss rate, delay and jitter guarantees, and seriously degrade the promised quality of service.

In this paper, we propose a fast and light-weighted mechanism for resource access control at an ISP edge router. It primarily protects real-time IP data flows, for which network resources are reserved, from DQoS attacks. Our resource access control mechanism is a checkpoint at edge routers, and is used for packet-level admission control. We call it an *IP Easy-pass*, because it is similar to the checkpoint at a toll road where only the cars with pre-paid stickers can go through the Easy-pass lane. Note that we apply the IP Easy-pass mechanism in the data plane, and assume the existence of a secure channel for QoS signaling in the control plane between a given end-host and the ISP edge router that connects the end-host to the Internet. Prior to data transfer, through the secure QoS signaling channel, the end-host and the ISP edge router must communicate shared secrets for generating Easy-passes.

Since IP is stateless, we attach a unique *Easy-pass* to every real-time data packet. Each IP Easy-pass is an encrypted order-sensitive information to warrant the legitimacy of the packet carrying it, and plays a role as an admission ticket which can be used only once and then becomes void. Stale *passes* are invalid. Even if an attacker can sniff the already-used *passes*, he cannot deceive the ISP edge router by simply copying these void passes into spoofed packets. Thus, the freshness of a *pass* is crucial to the admission of the data packet carrying the pass. A correct sequence of Easy-passes is pre-determined by both sides, and should be robust against cryptanalysis. It is extremely difficult, if not impossible, for the third party to decrypt the garbled *passes* and predict the correct sequence in a short time. This property ensures that an adversary cannot easily forge a valid unused IP Easy-pass. The rule for access control is simple: the ISP edge router knows the correct sequence of passes; it accepts the packet with a *new pass* that is in the right track; otherwise, any packet with a duplicated or out of track pass, is classified as forgery and simply dropped.[1]

Because the generation and verification of *passes* are done on a per-packet basis, it has to be very fast and light-weighted, incurring as little overhead as possible. To generate encrypted *passes*, several parameters, including a symmetric private key and the fundamental elements of producing plain *passes*, are shared between a given end-host and the ISP edge router that connects the end-host to the Internet. We attach an IP Easy-pass at the end of each IP packet as its trailer. The RC5 algorithm [16], a fast symmetric block cipher, is used for Easy-pass encryption and decryption. The plaintext and ciphertext of Easy-pass are both 64 bits long. We implement the IP Easy-pass mechanism in the Linux kernel as a kernel module, and analyze its effectiveness against packet forgery and resource embezzlement. Finally, we evaluate its overhead and performance.

[1]Considering possible packet losses, we admit normal out-of-sequence packets, which are still in the right track, as legal traffic.

The remainder of the paper is organized as follows. Section 2 discusses the related work. Section 3 reveals the vulnerability of the reserved network resource to DQoS attacks. Section 4 describes the generation of a plain Easy-pass, the encryption/decryption algorithm, the verification procedure and its embedding as an IP trailer. Section 5 describes the implementation of Easy-pass in the Linux kernel, and evaluates the effectiveness of the Easy-pass scheme in a network testbed. Finally, Section 6 concludes the paper with a summary of our work.

II. Related Work

The IPSec [17], [18] protocol is often used to provide end-to-end secure communications at the network layer. It protects each IP packet, supporting address authentication, data integrity and confidentiality. Besides serving secure communications between two end-hosts, IPSec can also be used between two gateways to achieve VPN (Virtual Private Network) services over the public Internet. In the VPN model, the end-hosts in a local area network are trusted entities. However, IPSec is heavy-weight, incurring non-negligible overhead in data transmission [19–21]. A 10% increase of end-to-end latency overhead introduced by IPSec may not be a serious problem to file transfers or Web sessions. However, such an overhead may violate the delay requirements for delivering real-time data streams. Furthermore, in a study of Voice over IPSec, the effective bandwidth was observed to be reduced up to 50% with respect to VoIP [19]. Overall, IPSec is inappropriate for protecting the reserved network resources at edges, especially in wireless LAN and DSL environments. In contrast, our scheme is intended mainly to protect the end-hosts that subscribe to the premium service, against the flooding attacks from malicious neighbors. For such DQoS problems, we need a light-weight network resource access control mechanism whose overhead does not cause violation of the stringent timing requirements of real-time traffic.

The hop integrity schemes proposed by Gouda *et al.* [22] support router authentication and hop-by-hop message integrity. The hop integrity protocols are executed at all routers in a network, and provide a minimum level of secure communications between two adjacent routers to defend against message modification and message replay. However, hop integrity does not keep track of individual data flows or sessions. It cannot prevent resource theft and message replay attacks between an end-host and its ISP edge router. Moreover, the accumulation of per-hop overhead may violate the end-to-end QoS requirements.

Session-level admission control in the network QoS infrastructure has been studied for a decade or so. A number of schemes have been proposed [12], [23–25], such as measured-based admission control [25], distributed admission control [24], and endpoint admission control [23], just to name a few. The fundamental goal of session-level admission control for real-time applications is to accommodate new session requests without compromising the ongoing sessions' QoS guarantees. However, IP Easy-pass is a packet-level resource authorization mechanism, which protects the reserved network resource from attacking traffic.

To protect a server's resources on the victim side, many schemes [26–28] have been proposed and implemented to counter DoS flooding attacks using sophisticated resource management schemes. These schemes provide more accurate resource accounting and fine-grained service isolation, for example, to shield interactive video traffic from bulk data transfers. However, these defensive mechanisms at victim servers cannot prevent the abuse of the reserved network-edge resources. In contrast, our scheme aims to protect the reserved network resources, instead of the server's resources.

III. Vulnerability of Reserved Network Resource

We will now show that the reserved network resource in the QoS infrastructure like DiffServ [11] is vulnerable to spoofed traffic, and the premium service [29] provided by an ISP is susceptible to flooding attacks. Two entities associated with the reserved network resource are a given end-host who has subscribed to the premium service (i.e., the customer), and the ISP edge router that connects the end-host to the outside world (i.e., the service provider). Before detailing the vulnerability of reserved network-edge resources, we first describe the DiffServ architecture briefly, in which DQoS attacks may happen. Note that the IP Easy-pass mechanism, which is independent of any network QoS architecture, is not tied with the DiffServ architecture; and can be applied to any other QoS infrastructures. We discuss Easy-pass in the context of DiffServ, just for the sake of presentation and experimentation.

A. *DiffServ Architecture*

To support network QoS, the DiffServ infrastructure has been proposed as a promising solution due mainly to its scalability and robustness. Within a DiffServ domain, packets entering a DiffServ-enabled network are marked with different DSCPs (DiffServ Code Points), and based on this marking, they are subject to classification and traffic conditioning (such as metering, shaping, and policing), leading to a small set of packet forwarding techniques called PHBs (Per-Hop Behaviors). The DiffServ architecture achieves scalability by applying traffic conditioning and per-flow management at edge routers only, and by applying PHBs to traffic aggregates at core routers. DiffServ provides three different services: premium, assured, and best-effort. Corresponding to these three different services, three types of PHBs are specified by DiffServ: EF (Expedited Forwarding), AF (Assured Forwarding), and BE (Best-Effort). EF is intended to support premium service for real-time applications that require strict guarantees on bandwidth, delay, jitter and packet loss. In this paper, the network resource reserved for EF traffic at ISP edge routers, including link bandwidth and router buffers, is the *target* of malicious attacks.

B. *Attacking Model and Assumptions*

DQoS attacks in the data plane are made by malicious insiders or unfriendly neighbors, who have not subscribed to the premium service. We assume that the adversary is located in the same stub network as the victim, or at a different stub network but is connected to the same ISP. The adversary can be a cracked machine, an unhappy employee, or a naughty freshman. As the joint that connects the stub networks to the rest of Internet, each ISP edge router performs traffic conditioning and policing for EF traffic: any violation of SLA will be punished by dropping packets. However, such traffic conditioning works only if no packets are spoofed; on the contrary, a blindly dropping policy makes the premium service much more susceptible to the flood of spoofed EF packets. The reason for this is that the network resource reserved for premium service is only a small portion of the link bandwidth and buffer space at ISP edge routers, due mainly to the following usage and financial factors:

- most of traffic will continue to be best-effort (BE), since BE service is free; and
- those users who have subscribed to the premium service will not waste their money by overbooking the resource.

Therefore, flooding even a small amount of spoofed premium service traffic can easily compromise its quality.

There may be several internal routers and switches between an end-host and its ISP edge router. For example, from the EECS Department at the University of Michigan, we use `traceroute` to track the route to `www.cnn.com`, and find that it takes 4 hops to reach `mich.net`, the regional ISP for the University of Michigan. The result is shown as follows. The topology of the campus network at the University of Michigan is shown in Figure 1. Within each segmental LAN environment, there is enough bandwidth — e.g., the link speed of CAEN (Computer-Aided Engineering Network) varies between OC-12 (622 Mb/s) and OC-48 (2.4 Gb/s) — to support real-time applications, such as VoIP (Voice over IP). It is also a common practice that no traffic conditioning is performed at internal routers. We further assume that these internal routers and switches are not sabotaged, so that in-flight packets cannot be intercepted and modified. However, the adversary can eavesdrop all the traffic between the end-host and the ISP edge router, and inject any packet at his will.

```
> 1  eecs2s-gw (141.213.10.1)
> 2  CAEN-EECS-GW (141.213.3.4)
> 3  141.213.101.4 (141.213.101.4)
> 4  141.213.127.14 (141.213.127.14)
> 5  atm3-0x1.michnet8.mich.net (192.122.183.13)
```

Suppose `mich.net` provides VoIP service, then between `mich.net` and users at the University of Michigan, there must be SLAs on the provision of network resource for supporting the voice traffic. Typically, the outgoing voice traffic will be policed at the edge router of `mich.net` before traversing its ISP network. Carrying voice traffic does not cost much bandwidth, its provision at the ISP edge router should be a small portion of its link capacity. Under such a scenario, a malicious user, who has not subscribed to the premium service, inside the campus network may monitor the ongoing voice traffic, and then flood duplicate or spoofed packets, simply launching a reply attack. The perceived quality of VoIP service

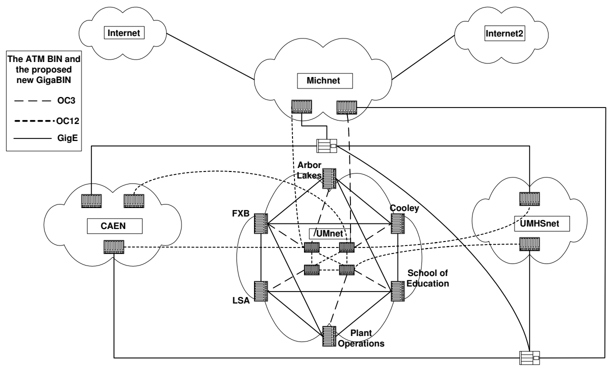

Fig. 1. The topology of UM campus networks

will be degraded significantly. Moreover, during certain peak times of the day (e.g., 10 am – 12 noon and 2 pm – 4 pm when most telephone calls are made), the individual callers may compete with each other for the reserved bandwidth. A rejected caller may disrupt the already-established calls by flooding spoofed packets, thereby degrading the ongoing phone conversations. Once several such conversations have been *forced* to abort, the resource left will allow the ISP edge router to admit such aggressive caller's request even at these peak times.

We prevent such DQoS attacks by deploying the IP Easy-pass mechanism at both the sending end-host and the ISP edge router that connects the end-host to the Internet. The sending end-host generates and encrypts an Easy-pass, then attaches it to each outgoing EF packet. The ISP edge router decrypts and verifies the received Easy-pass, then detaches it from the EF packet before forwarding the packet to the next-hop. Before EF data transmission, the end-host must have already set up a secure channel with the ISP edge router for resource reservation signaling. By utilizing this secure channel, the end-host and the ISP edge router share the secret information for constructing Easy-passes. Such an assumption is quite realistic. Besides the ARQoS [13] and Authenticated QoS [14] projects, the IETF NSIS Working Group [30] was recently formed to develop a set of standards for end-to-end resource reservation and QoS signaling between different administrative domains. Our Easy-pass scheme can be incorporated into the end-to-edge (host-to-network) signaling framework of NSIS so that the shared secrets can be exchanged between the end-host and the edge router once a trust relationship has been established between them.

Furthermore, the Easy-pass mechanism can be extended to validate the legitimacy of high-tiered traffic across two adjacent ISP edge routers that belong to two different ISPs. In the context of Mobile IP, we can also apply the Easy-pass mechanism to validate the IP packets originated from a legal mobile node for accessing the ISP resources subscribed by the foreign network. Since the mobile node needs to register with the foreign agent when it is connected to the foreign network, the mobile node and the foreign agent may set up a secure channel to negotiate on how to share secrets for Easy-pass construction during this registration period.

C. Flooding Attacks Disrupting Premium Service

To demonstrate the susceptibility of premium service to flooding attacks, we performed a series of experiments on our testbed. The topology of our testbed is shown in Figure 2. Our testbed consists of one Linux-based software router `rtcl11` and three end-hosts. One end-host `rtcl13` is used as the receiving host, and the other two as sending hosts — one `rtcl9` is a normal service subscriber and the other `rtcl10` is the adversary. Based on a set of traffic control (tc) APIs in the Linux kernel, we built a router configuration agent and placed it on the router of our testbed in order to configure the traffic control blocks inside the router. At the end-hosts, we built traffic generation agents, which are a modified version of Iperf [31], to generate both TCP and UDP traffic. To facilitate the experimental setup, a fast Ethernet switch is used to connect the end-hosts. Each machine in the testbed runs on a 600 MHz Prentium III processor with 256M RAM.

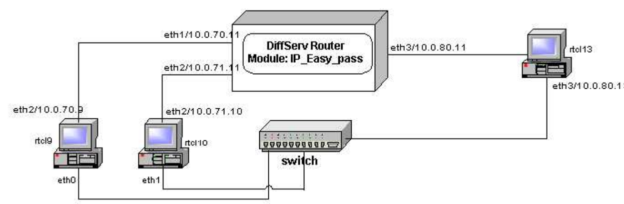

Fig. 2. The network topology of the DiffServ testbed

In order to support real-time traffic, we deployed the Diff-Serv EF (Expedite Forwarding) PHB (Per-Hop Behavior) [29] at the router. In our experiments, 500Kbps of the link bandwidth between the router `rtcl11` and the receiving host `rtcl13` is reserved for the EF traffic originating from the legitimate end-host `rtcl9`. Inside the router, the TBF (Token Bucket Filter) is used for EF traffic conditioning, and the packet scheduling policy is PQ (Priority Queueing). Due to the simple testbed setup, the measured end-to-end delays are not realistic, so we do not include them in our paper.

A well-behaved EF traffic carried by UDP is transmitted from `rtcl9` to `rtcl13`. The packet size is 1000 bytes. If no flooding attacks occur, the reserved network resource serves the EF traffic well, and achieves the goal of low loss rate, low delay and low jitter. However, under flooding attacks, even a small amount of flooding traffic can seriously degrade the quality of service received by EF traffic. As shown in Figure 3, for the flooding rate of 100 Kbps that is only one fifth of the reserved link bandwidth, the loss rate is increased from 0 to 19%. The measured end-to-end delay-jitter is plotted in Figure 4. Under the flooding rate of 100 Kbps, the jitter surges from 3μs to 5.95 ms, clearly showing that the flooding traffic increases the jitter by several orders of magnitude. Such serious degradation in packet loss and jitter will make any real-time application like VoIP or video-conferencing infeasible.

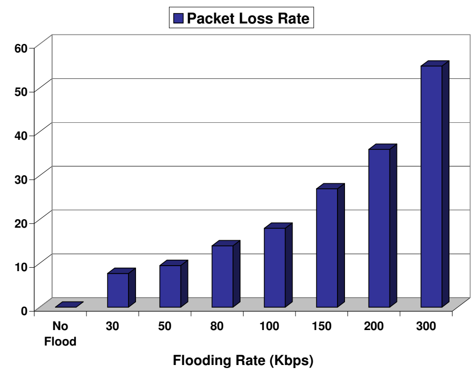

Fig. 3. Vulnerability of EF traffic to flooding attacks (I) (loss rate)

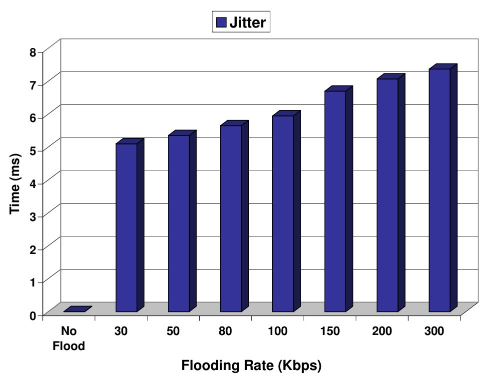

Fig. 4. Vulnerability of EF traffic to flooding attacks (II) (jitter)

IV. Description of IP Easy-pass

At an end-host subscribed to EF service, we create a unique *pass* and attach it to each outgoing EF packet. A valid *pass* authorizes its carrier packet to access the reserved network resource at downstream routers. In plain-text, a *pass* is just a random number. However, a sequence of *passes* for an EF data flow are generated according to certain rules. Both the sending host and the ISP edge router agree *a priori* on the generation rules. In this section, we will show how to generate a sequence of *passes* for an EF data flow, then discuss the choice of encryption/decryption algorithm, and finally, present the verification and embedding of Easy-passes.

A. Generation of Easy-Pass

The main parameters in generating a sequence of plain-text *passes* include a *nonce* Λ, a *gradient* Δ, and a *direction* γ. Of the tuple $\{\Lambda,\ \Delta,\ \gamma\}$, the first two elements are random numbers, and the last one is a boolean. The nonce is the starting point in generating a sequence of Easy-passes. The gradient is the span between two consecutive Easy-passes. The direction determines if the trend of the Easy-pass sequence is in an increasing or decreasing order, and its value is decided at run-time. The purpose of randomizing both the starting point and the span is to avoid Easy-pass collision[2] among the premium sessions that run on the same end-host, and share the same secret key with the ISP edge router.

Assuming that the number of bits in an Easy-pass is N, then the range space Ω of Easy-pass is 2^N. The chosen space for the initial nonce Λ is $[0, \Omega]$; and that for the gradient is $\{\ \Delta \mid 0 \leq \Delta \leq 2^k\ \cap\ (\Omega\ mod\ \Delta \neq 0),\quad k << N\}$. The growth direction of Easy-passes is dynamically determined by the chosen Λ. If $\Lambda > \Omega/2$, then the value of Easy-passes decreases; on the other hand, if $\Lambda < \Omega/2$, then the value of Easy-passes increases. Let $d(.)$ be the direction of Easy-passes for EF traffic transmitted between end-host m and the ISP edge router: '0' for increasing order and '1' for decreasing order.

$$d_m(\Lambda) = \gamma = \begin{cases} 0 & \text{if } \Lambda \leq \Omega/2; \\ 1 & \text{if } \Lambda > \Omega/2. \end{cases} \quad (1)$$

The plain Easy-pass is the sum of an initial random number Λ and the corresponding gradient Δ. Let $V(.)$ be the value of Easy-pass for the n-th data transmission. The construction of a plain Easy-pass is described as follows:

$$V(n) = \Lambda + (-1)^{\gamma} \times (n-1) \times \Delta. \quad (2)$$

where n is the transmission order of a data packet starting from 1. The algorithm for constructing an IP Easy-pass is illustrated in Table I. Step 0 in Table I shows the selection of two fundamental elements $\{\Lambda, \Delta\}$ during secure QoS signaling phase (e.g. via secure RSVP). The rest of steps in Table I present the construction algorithm of an Easy-pass. When the value of an Easy-pass reaches the lower_limit, 0, or the upper_limit, Ω, we need to wrap around the value to continue within the range space. The number of packets per round is Ω/Δ.

$$V(n) = \begin{cases} V(n) - \Omega & \text{if } V(n) > \Omega; \\ V(n) + \Omega & \text{if } V(n) < 0. \end{cases} \quad (3)$$

To prevent the wrap-around sequencer from coinciding with the previous values, in accordance to number theory, we require that the gradient Δ be a prime number.

To exemplify the working mechanism of IP Easy-pass, a sequence of Easy-passes in plaintext are shown in Figure 5. In this simple example, we assume that the sample space for nonce Λ is [0,120], and there are Easy-pass sequences with respect to different initial nonces. The randomly-selected nonce of the first and second sequences are 57 and 67, respectively. Both have the gradient $\Delta = 17$. Since the nonce of the first sequence, 57, is smaller than 60, the median of Ω, the first sequence is increasing. In contrast, the second sequence is decreasing, due to its nonce being larger than 60. Once the plaintext value of Easy-pass is computed, we encrypt it and attach the encrypted Easy-pass to the outgoing EF packet as an IP trailer. We discuss the choice of encryption/decryption algorithm in the next subsection.

[2]Two different data packets have the same value of Easy-pass.

TABLE I

PSEUDOCODE OF CONSTRUCTING AN EASY-PASS

0. Before data transmission:

    nonce $\leftarrow \Lambda$;   //select a random number for nonce
    gradient $\leftarrow \Delta$;   // select a prime number from $[0, 2^k]$

1. At the very beginning of data transmission:

    $n \leftarrow 1$;
    $W_0 \leftarrow \Lambda$;
    If $(\Lambda < \lfloor \Omega/2 \rfloor)$
        $\gamma \leftarrow 0$;
    Else
        $\gamma \leftarrow 1$;

2. On data transmission, build the n-th Easy-pass W_n as:

    Switch (γ) {
      Case 0 :
            $W_n \leftarrow W_{n-1} + \Delta$;
            If $(W_n > \Omega)$
                $W_n \leftarrow W_n - \Omega$;   // wrap around
      Case 1 :
            $W_n \leftarrow W_{n-1} - \Delta$;
            If $(W_n < 0)$
                $W_n \leftarrow W_n + \Omega$;   // wrap around
    }
    $n \leftarrow n + 1$;

B. Choice of Encryption/Decryption Algorithm

Since encryption/decryption is performed on each EF data packet, the overhead incurred by the encryption/decryption algorithm should be as low as possible without degrading its security. In the Easy-pass mechanism, we employ RC-5 [16] to encrypt Easy-passes at the end-host, and then decrypt it at the ISP edge router. This is mainly because (1) RC-5 is one of the fastest encryption/decryption algorithms available, and (2) RC-5 is fully parameterized, allowing flexible choices for its parameters. Briefly, RC-5 is a symmetric block cipher, in which the plaintext and ciphertext are fixed-length bit sequences. RC-5 is word-oriented, with a variable number of rounds and a variable-length cryptographic key. It is fast and has low memory requirement. Finally, RC-5 provides high-level security when parameter values are chosen properly.

The parameters in RC-5 that are adjustable include the word size in bits w, the number of rounds r, and the number of bytes in secret key b. Note that RC-5 uses an expanded key table, S, that is derived from the secret key. The size of table S also depends on the number of rounds, which is

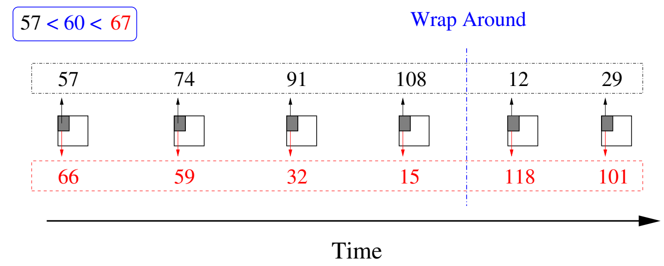

Fig. 5. The sequence of Easy-passes

equal to $2 \cdot (r + 1)$ words. RC-5 allows a range of parameter values so that one may choose a certain set of parameters that balance the requirement between security and performance. Moreover, applications can adjust these parameters when their own requirements change.

To test the efficiency of RC-5, we choose a secure option of RC5-32/10/16, where $w = 32$, $r = 10$, and $b = 16$, so that the secret key length is 128 bits. We conduct simple experiments on an off-the-shelf 550 MHz Pentium III PC with 256M RAM, running Linux kernel 2.4.7. The CPU time for encryption and decryption on the option of 32/10/16 is only $1\mu s$. Next, we vary the number of rounds and length of the secret key, to find out which one dominates the consumption of CPU cycles. As shown in Figure 6, the increase in the number of rounds from 10 to 80 linearly increases the CPU overhead, but increasing the secret key length does not affect the CPU overhead. Therefore, we choose 16 bytes (128 bits) for the secret key length, instead of 32 or 64 bits that are easier to break. Note that it took 1,757 days and 58,747,597,657 work units to crack a 64-bit RC5 key [32], thus it would take much longer to crack a 128-bit RC5 key. On the other hand, we choose the number of rounds to be 10 in order to reduce the resulting CPU overhead.

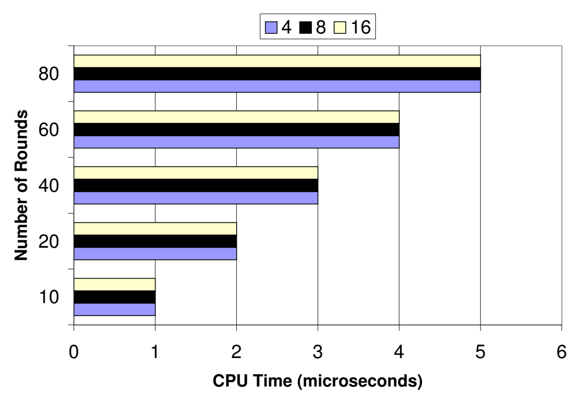

Fig. 6. CPU overhead for RC-5 encryption/decryption

The input (plaintext) and output (ciphertext) of RC-5 are two-word long. Since we choose the length of word as 32-bit, the length of Easy-pass is 64 bits. Then, the range space Ω is 2^{64}. Such a large space guarantees avoidance of Easy-pass collision, i.e., no two valid data packets within a reasonable time interval will have the same Easy-pass. For example, even if the real-time session reserves a 100 Mbps bandwidth, its average packet size is only 50 bytes, and its duration lasts for four weeks, the required space to avoid any Easy-pass collision is still less than 2^{40}.

C. Verification of Easy-passes

At the ISP edge router, after decrypting the encrypted Easy-pass in each received EF packet, we verify its legitimacy according to the generation rule of Easy-passes. The first step of the verification procedure is simply checking if $\frac{V_d-\Lambda}{\Delta}$ is an integer, where V_d is the value of the decrypted Easy-pass of the received EF packet. The next step is to make sure that the integer is fresh, i.e., it did not appear before.

If there is no out-of-order transmission between the end-host and its ISP edge router, after decrypting the Easy-pass from each valid EF packet, the ISP edge router will see a sequence of random numbers starting from Λ with an interval of Δ. Assume that the last *checking number*, $\frac{V_d-\Lambda}{\Delta}$, is an integer I, then the correct *checking number* for the one being validated should be $I+1$ if $I>0$ or $I-1$ if $I<0$. The correct sequence of checking numbers should be a series of $\{0,1,2,3,\cdots\}$ or $\{0,-1,-2,-3,\cdots\}$.

However, in case of congestion, packet losses or out-of-order packet arrivals may occur at the edge router. To account for possible holes in the sequence of checking numbers, we introduce a range-window. Given the maximum possible out-of-order value within the first-mile environment is m, we set the range-window size to $2m$. We also introduce two variables: one is *base* to record the *checking number* of the latest received in-order packet; the other is a $2m$-bit long variable, called *flag* as an bit-index-array to record the received out-of-order packets, whose default value is zero. In this paper, we set m as 16, so the range-window covers 32 packets. Then, $flag$ is a 32-bit word. Note that with the change of the real condition, the value of m is adjustable.

An out-of-order delivery or packet loss is detected, when the difference between the incoming packet's checking number and the *base* is larger than 1. As shown in Figure 7, at time t_1, the *base* is 32 but the received packet's checking number is 35. Since $35-32=3>1$, the out-of-order delivery or packet loss is detected. The value of *base* is not updated until the holes in the sequence are filled or the range-window reaches its limit later.

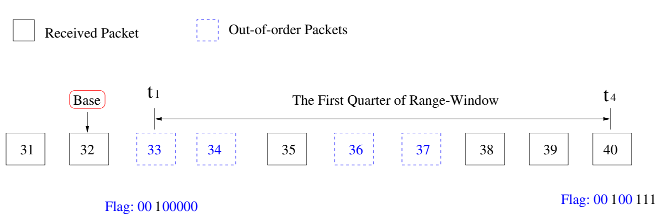

Fig. 7. Tracking the out-of-order delivery

In Figure 7, we only show one quarter of the range-window, and the first byte of *flag* is initially set to "00000000" as default. When the packet with the checking number 35 arrives, the third bit in the *flag* is set to "1", indicating that the packet has been received. Any later arrival of a packet with the same checking number will be treated as a duplicate and discarded. Following this rule, at time t_4, the first byte of flag becomes "00100111". The index of the received packet in the *flag* is its *checking number* deducted by *base*. In this manner, the verification module keeps track of the holes in the sequence of checking numbers, and updates the list of holes by changing the corresponding bit in *flag* from '0' to '1' when one of them is filled.

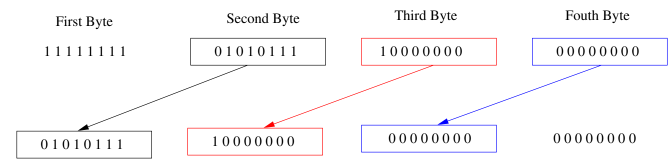

Fig. 8. Left shift the flag for 8 bits to update the out-of-order state

Once the first byte of *flag* becomes '0xFF' or the range-window reaches its right-most end, we shift the *flag* to left for 8 bits and reset the last byte as '0x00', as shown in Figure 8. Then, we increase the *base* by 8 correspondingly. This shift is very important to seamlessly keep track of the status of the out-of-order delivery using as little memory as possible. Note that in real-time communications, a data packet that arrived after its deadline would be useless. Thus, even if there exist holes in the first byte of the range-window, once its right limit is reached, the legitimacy of these holes in the first byte is deprived and the belated packets or retransmissions for these holes will be discarded. The pseudocode of verifying an Easy-pass is shown in Table II.

In summary, the filtering rule for weeding out attacking traffic at the ISP edge router is simple: if the checking number meets any one of the following conditions: (1) it is not an integer; (2) it is smaller than the value of *base*; and (3) its value is larger than the value of *base* but the corresponding bit in *flag* is already set to 1. Then, the packet carrying such a pass will be identified as spoofed and hence discarded without further processing.

D. Embedding of Easy-Pass

We embed an Easy-pass, which is 64 bits long, as the trailer to an IP packet. The attachment of an Easy-pass is shown in Figure 9. Although the occurrence of packet fragmentation between an end-host and its ISP edge routers is rare, the possible fragmentation will detach the *pass* from the fragments, except for the last one. A malicious attacker can exploit the fragmented packets to steal the reserved resource. To overcome the fragmentation problem, between an end-host and the ISP edge router, we require that an MTU discovery should be performed before data transmission, guaranteeing no fragmentation in the corresponding data flow.

The recent Internet measurement [33] has shown that less than 1% IP packets in the Internet are fragmented. More importantly, it confirmed that IP packet fragmentation is still

TABLE II

PSEUDOCODE OF VERIFYING AN EASY-PASS

```
Decrypt the Easy-pass to get V_d

 C_n = |(V_d − Λ)/Δ|;      // derive the absolute checking number
 If (C_n is not integer)
      discard the packet       // attacking packet
 Else
     i = C_n - B;   // get the index
     If (i < 0 )
       discard the packet    // duplicate packet
     Else
         If ( i > 1)    // out-of-order
            Switch (flag[i])  {     // check the bit in flag
              Case '1' :
                    discard the packet    // duplicate packet
              Case '0' :
                    accept the packet;
                     flag[i] = '1';
                 If (flag[1:8] == '0xFF' || flag[32] == '1')
                        flag << 8;    // left shift 8 bits
                        B = B + 8;    // update the base
                       // no holes in the first byte or
                       // reaches the highest-end
              }

         If ( i == '1')    // in the right track
            accept the packet
               B = C_n;     // update the base
```

"considered harmful." Therefore, enforcing no fragmentation not only fills the security holes, but also improves end-to-end performance.

One advantage of attaching the Easy-pass value as a trailer to each IP packet is that it is attached by the end-host and subsequently removed upon verification by the edge router before passing the packet on to the ISP network. Therefore, none of the downstream routers and the receiving end-hosts need to be modified to accommodate the proposed scheme. Another advantage is that like MPLS, it allows attachment of the Easy-pass value completely transparent to the transport protocol (TCP, UDP) used by the application-layer.

V. IMPLEMENTATION AND EVALUATION

The IP Easy-pass mechanism is implemented in the Linux kernel, and its effectiveness against flooding attacks is evaluated on the DiffServ testbed. In this section, we first describe the implementation of Easy-pass, and then perform a series of experiments to demonstrate the Easy-pass's protection on real-time traffic. Finally, we measure the overhead incurred by Easy-pass and discuss its impact on end-to-end performance.

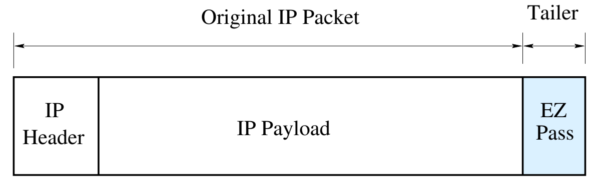

Fig. 9. Embedding the Easy-pass into an IP packet

A. Implementation of Easy-pass

We implemented the IP Easy-pass scheme as two loadable Linux kernel modules. One module, *HostModule*, is located at the end-host which calculates and attaches 64-bit Easy-passes to the packets; and the other module, *RouterModule* installed at the edge router, extracts the Easy-pass and verifies the packet before forwarding it to the next-hop.

The *HostModule* uses a hook that has been added to IP layer transmission function (`ip_build_xmit()`) of the Linux kernel version 2.4.7. After preparing the IP packet for transmission on the network, *HostModule* calculates the Easy-pass value, based on the current packet count from the designated data flow. The module then encrypts the field at run-time using RC5 and attaches it after the payload as the trailer of the packet. The whole packet is then placed on the outgoing interface.

On the edge-router side, *RouterModule* uses a hook added to the IP layer receiving function (`ip_rcv()`) to retrieve the encrypted Easy-pass value from the IP packet. The module then decrypts and verifies the pass. If the packet is found to have the correct Easy-pass value, it is allowed to proceed through the protocol stack. Otherwise, the packet is dropped.

Compared with inserting Easy-pass into IP header option fields, attaching the Easy-pass as a trailer has the advantage of being transparent to the protocol checking and processing. Neither header modification nor payload data shifting is required in this case. The only fields that have to be updated are the IP packet total length and the IP checksum, which are done inside our modules. This eases the implementation of our Easy-pass mechanism.

B. Effectiveness Against Resource Theft

To validate the effectiveness of Easy-pass against malicious attacks, we performed a series of experiments on our DiffServ testbed as described in Section III-C. Basically, our experiments can be divided into two groups: one is for protecting low-rate EF traffic, such as audio streams; and the other is for protecting high-rate EF traffic, such as video streams. Since multimedia traffic (real-time audio and Video) is usually transported by UDP [34], [35], all the EF traffic in our experiments are carried by UDP.

1) Protection of Low-rate (Audio) Traffic: As the maximum bandwidth required for supporting VoIP is 80 Kbps [36], we set the source sending rate and the reserved bandwidth at the router to 80 Kbps, respectively. Since the typical packet sizes

TABLE III

PACKET-LOSS RATE FOR LOW-RATE EF TRAFFIC

	w/o EZ-Pass		w/ EZ-Pass	
Flood Rate	254	502	254	502
20K	22%	21%	0	0
40K	38%	35%	0	0
60K	45%	40%	0	0
80K	54%	54%	0	0
120K	68%	71%	0	0
160K	80%	85%	0	0

TABLE IV

END-TO-END DELAY-JITTER FOR LOW-RATE EF TRAFFIC

	w/o EZ-Pass		w/ EZ-Pass	
Flood Rate	254	502	254	502
20K	6.8ms	9.3ms	5.2μs	5.8μs
40K	7.2ms	10.2ms	5.8μs	6.3μs
60K	7.4ms	10.5ms	6.1μs	6.7μs
80K	7.5ms	10.8ms	6.2μs	7.0μs
120K	7.9ms	11.2ms	6.5μs	7.3μs
160K	8.6ms	11.9ms	6.7μs	7.8μs

for real-time audio are 254 and 502 bytes [35] (depending on the encoding rate), we vary the packet size from 254 to 502 bytes in this set of experiments.

Table III illustrates the EF packet-loss rate for different flooding rates ranging from 20 Kbps — one fourth of reserved bandwidth — to 160 Kbps, twice the reserved bandwidth. Without the Easy-pass, the packet-loss rate ranges from 21% to 80%. Under the same flooding rate, the larger packet size incurs a slightly higher packet-loss rate, due to a slightly larger burst size. In contrast, with the Easy-pass, all the attacking packets are identified and discarded. That is, the reserved bandwidth is saved, and hence, no legitimate packets are dropped.

Table IV presents the corresponding results for end-to-end delay-jitter obtained from the same set of experiments. Without Easy-pass the jitter ranges from 6.8 to 11.9 ms. Also, for the same flooding rate, the larger packet size incurs a slightly higher jitter. In contrast, with Easy-pass, the jitter remains in the same order of magnitude as the one without any flooding attacks.

2) Protection of High-rate (Video) Traffic: Since MPEG-1, a popular video compression technique, has an encoding rate of 1.5 Mbps, in our second group of experiments, we set the source sending rate to 1.5 Mpbs, and reserve 1.5 Mbps bandwidth at the router. Since the typical packets for real-time video are 800 and 1000 bytes long [34], the EF packet size is varied from 800 to 1000 bytes in these experiments.

Table V presents the EF packet-loss rate for different flooding rates ranging from 300 Kbps, one fifth of reserved bandwidth, to 3 Mbps, twice the reserved bandwidth. Without

TABLE V

PACKET-LOSS RATE FOR HIGH-RATE EF TRAFFIC

	w/o EZ-Pass		w/ EZ-Pass	
Flooding rate	1000	800	1000	800
300K	22%	21%	0	0
500K	30%	27%	0	0
800K	37%	33%	0	0
1M	42%	38%	0	0
1.2M	48%	44%	0	0
1.5M	57%	53%	0	0
3M	67%	62%	0	0

TABLE VI

END-TO-END DELAY-JITTER FOR HIGH-RATE EF TRAFFIC

	w/o EZ-Pass		w/ EZ-Pass	
Flooding rate	1000	800	1000	800
300K	2.8ms	2.1ms	2.0μs	2.0μs
500K	3.6ms	2.4ms	2.3μs	2.2μs
800K	3.8ms	2.9ms	2.5μs	2.3μs
1M	4.2.ms	3.3ms	2.8μs	2.6μs
1.2M	4.5ms	3.6ms	3.4μs	3.1μs
1.5M	4.8ms	3.9ms	3.9μs	3.5μs
3M	5.2ms	4.3ms	4.3μs	3.9μs

Easy-pass, the packet loss rate ranges from 21% to 67%. As in the low-rate case, for the same flooding rate, the larger packet size incurs a slightly higher packet-loss rate. With Easy-pass, all the flooding traffic is identified and discarded, hence protecting the reserved bandwidth. None of the legitimate packets were dropped.

Table VI presents the results of end-to-end delay-jitter for the EF traffic. Unsurprisingly, without the Easy-pass, the jitter ranges from 2.1 to 4.3 ms, while the Easy-pass preserves the jitter at the same level of the one without any flooding attacks. In general, the experimental results of the high-rate case are similar to those of the low-rate case, demonstrating the effectiveness of Easy-pass against flooding attacks.

C. Overhead of Easy-pass

It is more important to characterize the overhead of Easy-pass introduced at the ISP edge router than that at an end-host, because the end-host can allocate more resources for each outgoing real-time packet than the edge router can. Moreover, the verification of Easy-pass incurs more CPU cycles than the Easy-pass embedding procedure at the end-host.

The per-packet overhead of Easy-pass, which is independent of packet size and the sending rate, is included in the processing overhead of an EF packet at the router. We measure the processing overhead for each EF packet at the router with and without the Easy-pass. In this set of experiments, we vary the reserved bandwidth from 100 Kbps to 2 Mbps, and the packet size from 100 to 1000 bytes. The network resources are well-

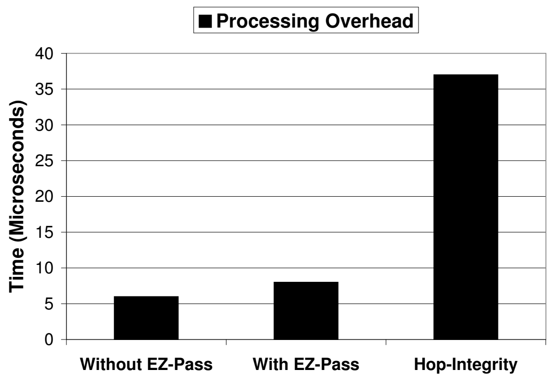

Fig. 10. Processing overhead with and without Easy-Pass

provisioned and the EF traffic is well-behaved, so there is no queueing delay for the EF traffic. The measured per-packet processing time in the case of an embedded Easy-pass is 8 μs, whereas the same without the Easy-pass is 6 μs. The results are shown in Figure 10. The overhead incurred by the verification of Easy-pass is 2 μs. For the purpose of comparison, the authentication overhead of the hop-integrity mechanism [22], which is 37 μs, is also shown in the Figure 10. This value was derived in a separate study [22] using a slightly faster machine — a Pentium III 730 MHz running Linux OS. More importantly, the verification of Easy-pass is performed only once at the ISP edge router, not at every downstream router. Since the average end-to-end delay between two Internet-hosts ranges from tens to hundreds of milliseconds, the processing overhead of Easy-pass is negligible.

Each Easy-pass incurs the space overhead of 8 bytes. In the context of video traffic, in which the typical packet size is 800 or 1000 bytes, the space overhead of Easy-pass is 1% or 0.8%. In the context of audio traffic, the space overhead is increased to 3.1% or 1.5%, with respect to the 254 or 500 bytes long packets. Such an increase of space overhead is acceptable to most users. To further reduce the space overhead, we can trade security for performance by decreasing the length of Easy-pass from 64 to 32 bits.

VI. Conclusion

In this paper, we proposed a fast and light-weighted IP network-edge resource access control mechanism, called *IP Easy-pass*, to protect reserved network resources at edge devices from theft and abuse. By conducting experiments on a DiffServ testbed, we demonstrated the vulnerability of the reserved network resource to flooding attacks, and the need for IP-layer resource access control. In our scheme, a unique, encrypted, *pass* is attached to each legitimate real-time packet at the end-host. The ISP edge router validates the legitimacy of each incoming real-time packet simply by checking its pass. We described the creation of Easy-pass at the end-host, and its verification procedure at the ISP edge router using the RC5 encryption/decryption algorithm. The Easy-pass mechanism has been implemented as loadable Linux kernel modules.

We conducted a series of experiments on the DiffServ testbed to evaluate the effectiveness of Easy-pass against flooding attacks. The experimental results have shown that the Easy-pass mechanism effectively shields the reserved network resources from spoofed packets — it is shown to protect the legitimate packets from either loss or increased end-to-end delay-jitters for all the flooding rates we considered. Moreover, we measured the computational overhead of Easy-pass, and found it to be a negligible fraction (a few microseconds) of the processing time of each packet at both the end-host as well as at the edge router. Since the verification of Easy-pass is done at the ISP edge router only, not every downstream router, the Easy-pass overhead added to the end-to-end delay of an application traffic stream is negligible when compared to typical delay values of tens of milliseconds. Overall, IP Easy-pass is a very light-weight and effective mechanism for providing network-edge resource access control.

References

[1] C. A. CA-2000.01, "Denial-of-service development," January 2000, available: http://www.cert.org/advisories/CA-2000-01.html.

[2] D. Moore, G. Voelker, and S. Savage, "Inferring internet denial of service activity," in *Proceedings of USENIX Security Symposium'2001*, Washington D.C., August 2001.

[3] C. Jin, H. Wang, and K. G. Shin, "Hop-count filtering: An effective defense against spoofed DDoS traffic," in *Proceedings of ACM CCS '2003*, Washington D.C, October 2003.

[4] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing," in *RFC 2267*, January 1998.

[5] J. Li, J. Mirkovic, M. Wang, P. Reiher, and L. Zhang, "SAVE: Source address validity enforcement protocol," in *Proceedings of IEEE INFOCOM '2002*, New York City, NY, June 2002.

[6] K. Park and H. Lee, "On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets," in *Proceedings of ACM SIGCOMM '2001*, San Diego, CA, August 2001.

[7] J. Reumann, H. Jamjoom, and K. G. Shin, "Adaptive packet filters," in *Proceedings of IEEE Globcom '2001*, San Antonio, TX, November 2001.

[8] M. Sung and J. Xu, "IP traceback-based intelligent packet filtering: A novel technique for defending against internet DDoS attacks," in *Proceedings of of IEEE ICNP '2002*, Paris, France, November 2002.

[9] H. Wang and K. G. Shin, "Transport-aware IP routers: A built-in protection mechanism to counter DDoS attacks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 9, September 2003.

[10] A. Yaar, A. Perrig, and D. Song, "Pi: A path identification mechanism to defend against DDoS attacks," in *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, CA, May 2003.

[11] S. Blake and *et al.*, "An architecture for differentiated services," in *RFC 2475*, December 1998.

[12] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource reservation protocol," *IEEE Network*, vol. 7, no. 5, September 1993.

[13] "The ARQoS project," available: http://arqos.csc.ncsu.edu/.

[14] "The authenticated QoS project," available: http://www.citi.umich.edu/projects/qos/.

[15] J. Martin and A. Nilsson, "On service level agreements for IP networks," in *Proceedings of IEEE INFOCOM '2002*, New York City, NY, June 2002.

[16] R. L. Rivest, "The RC5 encryption algorithm," *Lecture Notes in Computer Science*, vol. 1008, Springer-Verlag, 1995.

[17] S. Kent and R. Atkinson, "Security architecture for the internet prtocotl," in *RFC 2401*, November 1998.

[18] R. Thayer, N. Doraswamy, and R. Glenn, "IP security document roadmap," in *RFC 2411*, November 1998.

[19] R. Barbieri, D. Bruschi, and E. Rosti, "Voice over IPsec: Analysis and solutions," in *Proceedings of 18th Annual Computer Security Applications Conference*, December 2002.

[20] G. Hadjichristofi, N. D. IV, and C. Midkiff, "IPsec overhead in wireline and wireless networks for web and email applications," in *Proceedings of IEEE IPCCC '2003*, Phoenix, AZ, April 2003.

[21] S. Miltchev, S. Ioannidis, and A. D. Keromytis, "A study of the relative costs of network security protocols," in *Proceedings of the USENIX Annual Technical Conference'2002 Freenix Track*, Monterey, CA, June 2002.

[22] M. G. Gouda, E. N. Elnozahy, C.-T. Huang, and T. M. McGuire, "Hop integrity in computer networks," *IEEE/ACM Transactions on Networking*, vol. 10, no. 3, June 2002.

[23] L. Breslau, E. Knightly, S. Shenker, I. Stoica, and H. Zhang, "Endpoint admission control: Architectural issues and performance," in *Proceedings of ACM SIGCOMM'2000*, Stockholm, Sweden, August 2000.

[24] F. Kelly, P. Key, and S. Zachary, "Distributed admission control," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 12, December 2000.

[25] S. Jamin, P. Danzig, S. Shenker, and L. Zhang, "A measurement-based admission control algorithm for integrated services packet networks," *IEEE/ACM Transactions on Networking*, vol. 5, no. 1, February 1997.

[26] G. Banga, P. Druschel, and J. Mogul, "Resource containers: A new facility for resource management in server systems," in *Proceedings of USENIX OSDI'99*, New Orleans, LA, February 1999.

[27] X. Qie, R. Pang, and L. Peterson, "Defensive programming: Using an annotation toolkit to build dos-resistant software," in *Proceedings of USENIX OSDI'2002*, Boston, MA, December 2002.

[28] O. Spatscheck and L. Peterson, "Defending against denial of service attacks in Scout," in *Proceedings of USENIX OSDI'99*, New Orleans, LA, February 1999.

[29] B. Davie and *et al.*, "An expedited forwarding PHB (per-hop behavior)," in *RFC 3246*, March 2002.

[30] H. Tschofenig and D. Kroeselberg, "Security threats for NSIS," in *Internet Draft, draft-ietf-nsis-threats-01.txt*, January 2003.

[31] M. El-Gendy, A. Bose, H. Wang, and K. G. Shin, "Statistical characterization for per-hop QoS," in *Proceedings of IWQoS'2003*, Monterey, CA, June 2003.

[32] Distributed.net, "Rc5-64 project," July 2002, available: http://www.distributed.net/rc5/.

[33] C. Shannon, D. Moore, and K. C. Claffy, "Beyond folklore: observations on fragmented traffic," *IEEE/ACM Transactions on Networking*, vol. 10, no. 6, December 2002.

[34] M. Li, M. Claypool, and B. Kinicki, "MediaPlayer versus RealPlayer — a comparison of network turbulence," in *Proceedings of ACM Internet Measurement Workshop'2002*, Marseille, France, November 2002.

[35] A. Mena and J. Heidemann, "An empirical study of real audio traffic," in *Proceedings of IEEE INFOCOM '2000*, Tel Aviv, Israel, March 2000.

[36] "VoIP bandwidth, white paper," available: http://www.erlang.com/bandwidth.html/.