IP Easy-pass: A Light-weight Network-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
hnw@cs.wm.edu

‡EECS Department
The University of Michigan
Ann Arbor, MI 48109
{*abose,mgendy,kgshin*}*@eecs.umich.edu*

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, and measure its overhead in our testbed. Finally, we demonstrate its effectiveness against packet forgery and resource embezzlement attempts by conducting a series of experiments.**

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 [7, 24], in which an attacker either knocks off a victim server by sending a few crafted packets, or blocks legitimate users' access to its Internet service by flooding bogus packets. Due to readily available tools and its simple nature, flooding packets is the most common DoS attack. 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 [11,15,18,25,29,33,36,37]. The packet filters installed at Internet servers or their nearby firewalls [15, 29, 33, 37] only prevent malicious IP packets from reaching the victim, but cannot protect network resources from theft and abuse. The intermediate-router-based packet filters [18, 25, 33, 36] can block further propagation of flooding traffic in the network, but cannot protect the upstream network resources. The ingress/egress filtering techniques [11] 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 area network.

To provide real-time communication services to multimedia applications or "subscribed" Internet users, the proposed network QoS (Quality of Service) infrastructure like DiffServ (Differentiated Service) [5] reserves network resources for real-time traffic. Within the network QoS architecture, the reserved network resources should be allocated to the subscribers only who purchase value-added services from ISPs via SLAs (Service-Level Agreements) [**?**, 40]. 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 [39]), 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. Without efficient countermeasures, DQoS attacks will be rampant once the network QoS has been widely deployed.

Previous research efforts have focused on providing secure communication in the control plane for control flows, such as in ARQoS [1] and Authenticated QoS [2] projects. They proposed a secure RSVP, in which resources are reserved online using strong authentication, and subsequently, compliance with the reservation re-

quest 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) [20] 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 secret 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 [30], 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 loadable kernel modules, and evaluate its overhead and performance. Finally, we analyze its effectiveness against packet forgery and resource embezzlement.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 reveals the vulnerability of the reserved network resource to DQoS attacks. Section 4 presents the architecture of Easy-pass mechanism. Section 5 describes the generation of a plain Easy-pass, the encryption/decryption algorithm, the verification procedure and its embedding as an IP trailer. Section 6 details the implementation of Easy-pass in the Linux kernel, measures its overhead and evaluates the effectiveness of the Easy-pass scheme in our DiffServ testbed. Finally, Section 7 concludes the paper with a summary of our work.

II. Related Work

The IPsec [17, 34] 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. IPsec defends against reply attacks by including a 32-bit sequence number with each IPsec packet header. Besides

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

serving secure communications between two end-hosts, IPsec can also be used as edge-to-edge or end-to-edge models. The edge-to-edge model has been widely used to achieve VPN (Virtual Private Network) services over the public Internet between two gateways. In the VPN model, the end-hosts in a local area network are trusted entities. Although rarely used, the end-to-edge model fits in our scenario of network-edge resource access control, where adversaries may reside in the same local area network as victims.

However, IPsec is heavy-weight, incurring non-negligible overhead in data transmission [4, 13, 22]. 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 streams. Furthermore, in a study of Voice over IPsec, the effective bandwidth was observed to be reduced up to 50% with respect to VoIP [4]. We compare the overhead of IP Easy-pass with that of the lightest version of IPsec that supports end-to-edge model (IPsec AH tunnel mode) in Section VI-B.4. Overall, IPsec is inappropriate for protecting the reserved network resources at edges, especially in wireless LAN and DSL environments.

The hop integrity schemes proposed by Gouda *et al.* [12] 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. In contrast, our scheme is intended mainly to protect the end-hosts that subscribe to 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.

TESLA [26, 27] is an efficient packet-level source authentication protocol for broadcast and multicast. Through loosely-synchronized time clocks and delayed key disclosure, TEASLA achieves asymmetric cryptographic properties without using public-key based digital signatures. TESLA incurs low communication and computation overheads, is scalable to a large number of receivers and can tolerate packet losses. However, two aspects of TESLA make it inappropriate to authenticate real-time traffic at an ISP router for accessing reserved network resources. First, the authentication for a received packet is deferred until the corresponding key is disclosed. The addition of this key disclosure delay may violate the end-to-end latency requirement. Second, a receiver has to buffer these recently-arrived packets whose keys have not yet been disclosed. Since multiple premium sessions could run concurrently between different end-hosts and the ISP edge router, buffering all these real-time traffic from different sessions imposes a very high memory overhead at the ISP edge router.

Session-level admission control in the network QoS infrastructure has been studied for a decade or so. A number of schemes have been proposed [6,14,16,39], such as measured-based admission control [14], distributed admission control [16], and endpoint admission control [6], 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 [3,28,32] 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 [5] is vulnerable to spoofed traffic, and the premium service [8] 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 the IP 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, delay-jitter[2] and packet loss. Because DiffServ routers differentiate EF packets from other types of packets purely based on the value of six-bit DSCP field in the IP header, the network resource reserved for EF traffic at ISP edge routers, including link bandwidth and router buffers, is vulnerable to spoofed EF flood and becomes the *target* of malicious attacks.

B. Attacking Model and Assumptions

DQoS attacks in the data plane are launched 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; otherwise, a blindly dropping policy makes the premium service much more vulnerable 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: (1) most of traffic will continue to be best-effort (BE), since BE service is free; and (2) 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 EF traffic can easily compromise the quality of premium service.

[2]Delay-jitter is the variation in inter-arrival times between packets delivered to final output device; the delay jitter bound for a packet stream is defined to be the bound of the maximum difference between delays experience by any two packets [38].

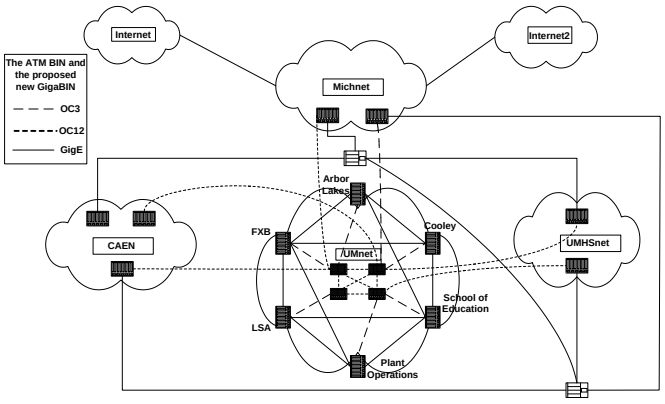

Fig. 1. The topology of UM campus networks

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 four 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 resources 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 inside the campus network, who has not subscribed

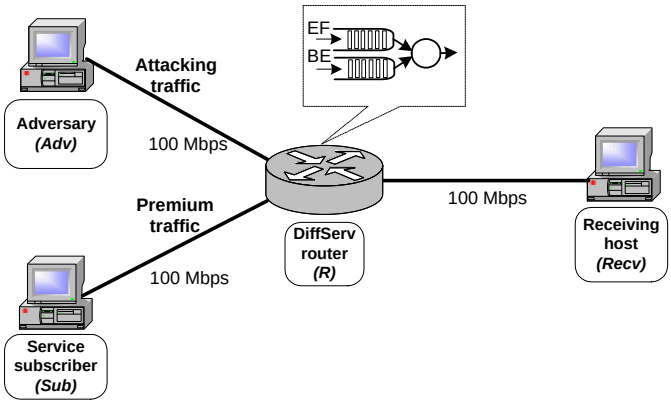

Fig. 2. The network topology of the experiment testbed

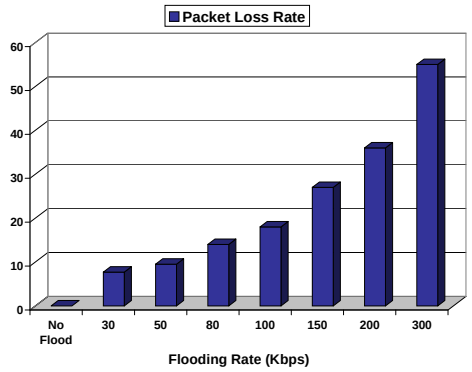

Fig. 3. Vulnerability of EF traffic to flooding attacks (I) (loss rate)

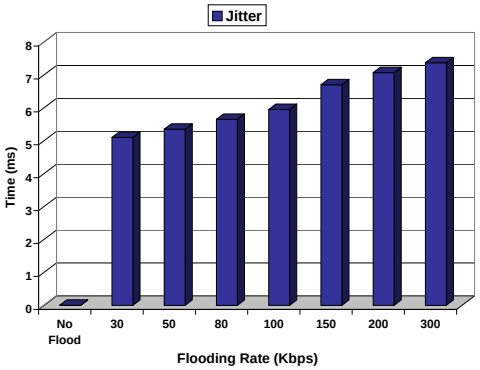

Fig. 4. Vulnerability of EF traffic to flooding attacks (II) (jitter)

to the premium service, may monitor the ongoing voice traffic, and then flood duplicate or spoofed packets, simply launching a reply attack. The perceived quality of VoIP service 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 also assume that QoS control plane is secured. Before transmitting EF data, the end-host must have already set up a secure channel with the ISP edge router for resource reservation signaling. Otherwise, attackers can deceive the ISP edge router during RSVP signaling, and easily abuse the reserved resource without flooding spoofed EF traffic in QoS data plane. Besides the AR-QoS [1] and Authenticated QoS [2] projects, the IETF NSIS Working Group [35] has recently been formed to develop a set of standards for end-to-end resource reservation and QoS signaling between different administrative domains.

C. Flooding Attacks Disrupting Premium Service

To demonstrate the susceptibility of premium service to flooding attacks, we performed a series of experiments in our DiffServ testbed. The topology of our testbed is shown in Figure 2. Our testbed consists of one Linux-based software router, `R`, and three end-hosts. One end-host, `Recv` is used as the receiving host, and the other two as sending hosts — one, `Sub`, is a normal service subscriber and the other, `Adv`, 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. On the end-hosts, we built traffic generation agents, which are a modified version of Iperf [10], to generate both TCP and UDP traffic. All the machines in the testbed have a 600 MHz Pentium III processor with 256MB RAM each.

In order to support real-time traffic, we deployed the DiffServ EF (Expedite Forwarding) PHB (Per-Hop Behavior) [8] at the router. In our experiments, 500Kbps of the link bandwidth between the router `R` and the receiving host `Recv` is reserved for the EF traffic originating from the legitimate end-host `Sub`. 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 did not include them here.

A well-behaved EF traffic carried by UDP is transmitted from `Sub` to `Recv`. The packet size is 1000 bytes. If no flooding attacks occur, the reserved network resource serves the EF traffic well, and achieves the goal

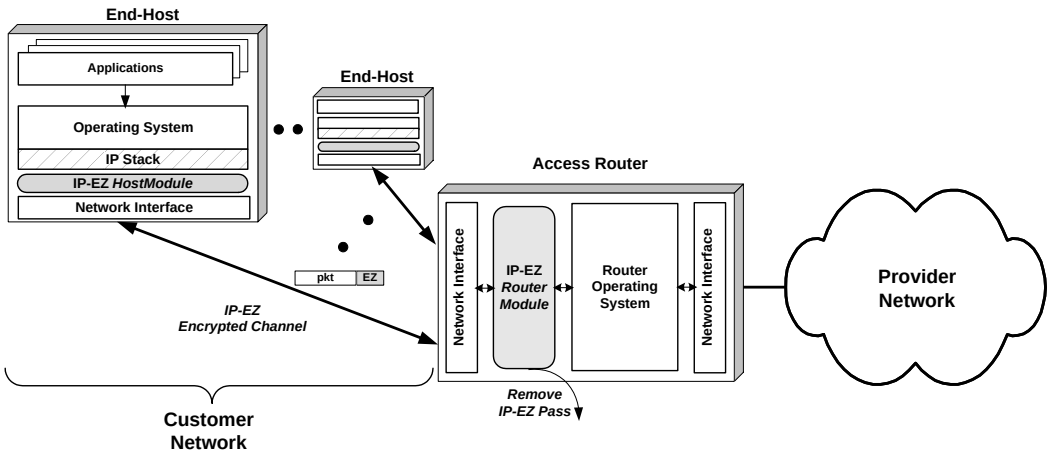

Fig. 5. Architecture of the IP Easy-pass framework

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 (J)[3] 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 many real-time applications like VoIP or video-conferencing infeasible.

IV. IP Easy-pass Architecture

We aim to prevent DQoS attacks in the data plane by deploying the IP Easy-pass mechanism at both a sending end-host that subscribes the premium service and the ISP edge router that connects end-hosts to the Internet. A valid *pass* authorizes the packet that carries the pass, to access the reserved network resource at ISP edge routers. At the end-host, as the *last*-step of IP processing, we generate and encrypt a unique *pass*, and then attach it to each outgoing EF packet. At the ISP edge router, as the *first*-step of IP processing, we decrypt and verify the *pass*, then detach it from the received EF packet. If the verification succeeds, the packet will be forwarded to the routing processing; otherwise, it will be simply discarded. The architecture of Easy-pass is shown in Figure 5. Real-time or premium-class applications running on the sending end-hosts are transparently protected by the IP Easy-pass mechanism, which does not require any modifications on applications. In the rest of this section, we address why we need encryption, and discuss the scalability of Easy-pass and its extension to inter-domain and Mobile IP scenarios.

[3]Calculated as a low-pass filter of the delay variation between two successive packets: $J = J + (|D(i-1,i)| - J)/16$, where $D(i-1,i)$ is the delay variation between packets i and $(i-1)$ and $\frac{1}{16}$ is the value of the low-pass filter.

A. Why Encryption?

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. Had we only used Pseudo-random Number Generators (PRNGs), such as the popular linear congruential algorithm for constructing passes, without encryption, adversaries would have determined the parameters of the PRNG algorithm with the exposure of a small part of the sequence of random numbers, and would then have learned the whole sequence of passes. Therefore, we employ a symmetric cipher for Easy-pass encryption and decryption. Both the sending host and the ISP edge router agree *a priori* on the generation rules and the shared secret key.

As mentioned before, during a secure QoS signaling we should have established a security association between the end-host who subscribes the premium service and its ISP edge router in the control plane. By utilizing the secure channel in QoS signaling, the end-host and the ISP edge router can share the secret information for constructing Easy-passes. Even without an already-established secure QoS signaling channel, in the control plane we can utilize Diffie-Hellman and public-key based schemes [**?**] to derive a shared secret key between the end-host and the ISP edge router for Easy-pass construction, encryption and decryption.

B. Scalability of Easy-pass

To verify the Easy-passes attached to outgoing EF packets, the ISP edge router must maintain a 16-byte key for each end-host (see Section V.B) that subscribes the premium service, and consume a few runtime variables for each on-line premium user (see Section V.C). One may raise a question about the scalability of Easy-pass at the ISP edge router.

Note that in the DiffServ architecture, the **edge** routers still maintain per-flow states and perform flow-based traffic conditioning for EF and AF traffic. In general, the number of flows should be no less than the number of end-hosts from which the flows originate. Also, the Easy-pass mechanism incurs only a low per-host state overhead. At edge routers, the 16-byte key can be stored as a new component of the traffic profile, which already includes average rate, peak rate, burst size, etc., for each EF flow. Every time when the edge router retrieves the traffic profile to perform traffic conditioning on the EF traffic, the key will be automatically fetched into the router memory and maintained with other traditional parameters of the traffic profile. The runtime variables per on-line premium user are volatile, and there is no need to retrieve or store them. The per-host Easy-pass verification should, therefore, not impose any scalability problem upon the ISP edge router.

Moreover, it is possible for a group of end-hosts that subscribe the premium service to share the same secret key in order to reduce the key maintenance overhead at the ISP edge router. The members of the same group must trust each other, and the size of a group should be properly bounded so that the probability of Easy-pass collision[4] among their sessions is very low. However, the key-management is done in the control plane and also off-line before the data transfer begins. How to agree on a key, how to re-key and how to distribute a key are beyond the scope of this paper. (One can use one of many existing approaches for this purpose.)

C. Extension of Easy-pass

The Easy-pass mechanism can be easily extended to validate the legitimacy of high-tiered traffic between two adjacent ISP edge routers that belong to two different ISPs. The inter-domain case is even simpler than that of intra-domain, because we deal with a single EF aggregate (instead of individual EF flows) between the two adjacent ISP edge routers in the inter-domain case. At the egress interface of one edge router, we attach Easy-passes for outgoing EF packets that are destined for the other domain. Then, the passes are validated at the ingress interface of the corresponding adjacent edge router to differentiate them from spoofed crossing EF traffic. Only one secret key is needed at these two adjacent ISP edge routers for validating the EF aggregate between them.

Note that the entities of Easy-pass in the inter-domain case are two adjacent edge routers. In the inter-domain scenario, the number of symmetric keys maintained at an ISP edge router is a linear function of the number of its egress interfaces (i.e., the number of neighboring edge routers in other domains it is connected to). Since the number of neighboring edge routers in other domains is much less than that of end-hosts it connected to the Internet, there is no scalability problem in maintaining secret keys in the inter-domain case.

In the context of Mobile IP, we can also apply the Easy-pass mechanism to validate the IP packets that originated from a legal mobile node for accessing the ISP resources subscribed by a 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.

V. Description of IP Easy-pass Algorithms

We now show how to generate a sequence of *passes* for an EF data flow, then investigate the choice of encryption/decryption algorithms, 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/decreasing order, and its value is determined at run-time. The purpose of dynamically setting the direction of a sequence and randomizing both the starting point and the span is to avoid Easy-pass collision among the premium sessions, which run from the same end-host (or the same group of end-hosts) and share the same secret key with the ISP edge router.

If 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.

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

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.

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

TABLE I
PSEUDOCODE OF CONSTRUCTING AN EASY-PASS

```
0. Before data transmission:

   nonce ← Λ;      //select a random number for nonce
   gradient ← Δ;   // select a prime number from [0, 2^k]

1. At the very beginning of data transmission:

   n ← 1;
   W_0 ← Λ;
   If (Λ < ⌊Ω/2⌋)
       γ ← 0;
   Else
       γ ← 1;

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

   Switch (γ) {
     Case 0 :
               W_n ← W_{n-1} + Δ;
               If (W_n > Ω)
                     W_n ← W_n − Ω;     // wrap around
     Case 1 :
               W_n ← W_{n-1} − Δ;
               If (W_n < 0)
                     W_n ← W_n + Ω;     // wrap around
   }
   n ← n + 1;
```

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 the 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 $\lfloor \Omega/\Delta \rfloor$.

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

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 6. In this simple example, we assume that the sample space for nonce Λ is [0,120], and there are two Easy-pass sequences with respect to different initial nonces.

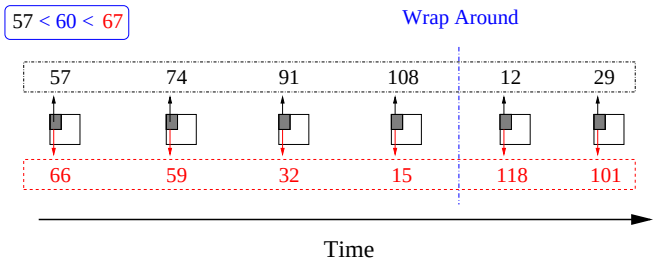

Fig. 6. The sequence of Easy-passes

The randomly-selected nonces of the first and second sequences are 57 and 67, respectively. Both sequences have the same gradient Δ as 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 following.

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 [30] 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 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 requirements 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

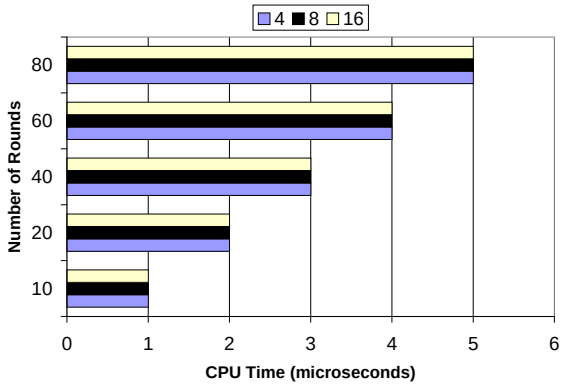

Fig. 7. CPU overhead for RC-5 encryption/decryption

Pentium III PC with 256MB 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 7, 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 [9], 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.

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 a 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 $I+1$. The correct sequence of checking numbers should be a series of $\{0, 1, 2, 3, \cdots\}$.

However, in case of data corruption and 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 real condition, the value of m is adjustable.

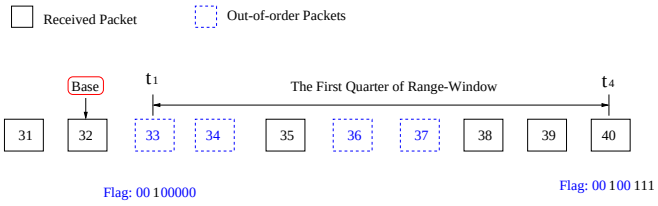

Fig. 8. Tracking the out-of-order delivery

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 8, 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.

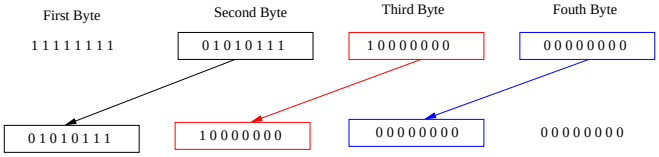

Fig. 9. Left shift the flag for 8 bits to update the out-of-order state

In Figure 8, 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

TABLE II
PSEUDOCODE OF VERIFYING AN EASY-PASS

```
Decrypt the Easy-pass to get V_d

C_n = |(V_d - A)/Δ|;     // derive the 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
```

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.

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 9. 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.

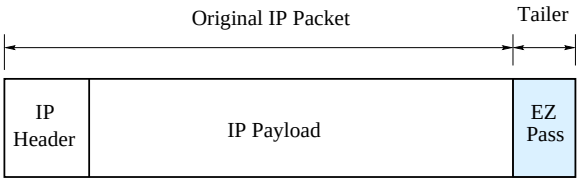

Fig. 10. Embedding the Easy-pass into an IP packet

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 10. 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-layer protocol (TCP or UDP) and the various application-layer protocols above.

If an end-host that subscribes the premium service is only one-hop away from the ISP edge router, no IP fragementation happens. However, if the end-host is a few hops away from the ISP edge router, it is possible that an IP packet is fragemented during the transmission from the end-host to the ISP edge router. The recent Internet measurement [31] has shown that less than 1% IP packets in the Internet are fragmented. Although the occurrence of packet fragmentation between an end-host and its ISP edge routers is even rarer, 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. Moreover, IP packet fragmentation is still "considered harmful" to end-to-end performance [31]. Therefore, enforcing no fragmentation not only fills the security holes, but also improves end-to-end performance.

To prevent the possible fragmentation between an end-host and the ISP edge router, we require that a path MTU discovery [23] should be performed before EF data transmission, guaranteeing no fragmentation in the corresponding EF data flow. Currently, most Unix-variant OSes support the path MTU discovery mech-

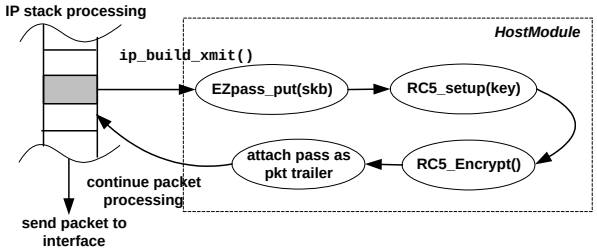

Fig. 11. *HostModule* sequence of operation

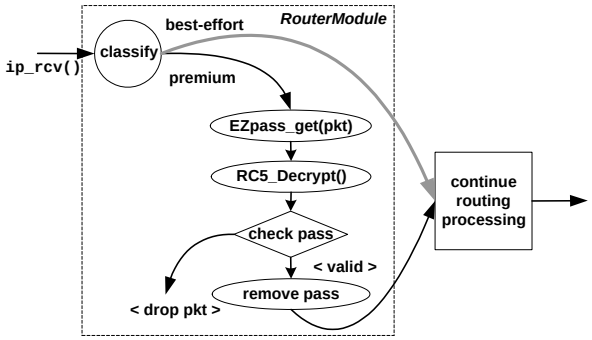

Fig. 12. *RouterModule* sequence of operation

anism. We can enable the path MTU discovery by appropriately setting the corresponding system control parameter, e.g., `ip_path_mtu_discovery` in Solaris and `ip_no_pmtu_disc` in Linux. Note that the attachment of easy-pass is the last-step of IP processing at the sending host. For each fragment of an IP packet, we attach a unique easy-pass as its trailer before forwarding it to the data-link layer. At the other end, as the first-step of IP processing at the ISP edge router, the easy-pass is detached from the fragment before packet re-assembly.

VI. 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 as described in Section III-C. In this section, we first describe the implementation of Easy-pass, measure the overhead incurred by Easy-pass and discuss its impact on end-to-end performance. Finally, we perform a series of experiments to demonstrate the Easy-pass's protection of real-time traffic.

A. Implementation of Easy-pass

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

HostModule uses a hook that has been added to the IP layer transmission function (`ip_build_xmit()`) of the Linux kernel version 2.4.7. After preparation of an IP packet for transmission over the network, the packet is sent to *HostModule* via `EZpass_put()`, which calculates the Easy-pass value, based on the current packet count from the designated data flow (via `RC5_setup()`). The module then encrypts the field at runtime using RC5 (via `RC5_Encrypt()`) and attaches it to the payload as the trailer of the packet. The entire packet is then placed on the outgoing interface. The sequence of *HostModule* operations is illurstrated in Figure 11. In an alternative implementation, we use Linux Netfilter hooks (specifically the `NF_IP_POST_ROUTING` hook) to edit the packet and add the calculated RC5 Easy-pass before placing it on the wire.

At the edge-router side, after packet classificatioin *RouterModule* uses a hook added to the IP-layer receiving function (`ip_rcv()`) to retrieve the encrypted Easy-pass value from the premium packet IP header (via `EZpass_get()`). The module then decrypts (via `RC5_Decypt()`) 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. The detailed operational sequence of *RouterModule* is illustrated in Figure 12.

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. Overhead of Easy-pass

We evaluate the overhead of Easy-pass in terms of bandwidth and CPU consumption, discuss the impact of Easy-pass processing overhead upon the EF throughput, and finally, compare the overhead of Easy-pass with that of IPsec.

B.1 Bandwidth Overhead

Each Easy-pass introduces additional 8 bytes. In the context of video traffic, in which the typical packet size is 800 or 1000 bytes [19], the bandwidth overhead of Easy-pass is 1% or 0.8%. In the context of streaming audio traffic, the bandwidth overhead is increased to 3.1% or 1.5%, with respect to the 254- or 500-byte packets [21]. Such an increase of bandwidth overhead is acceptable to most users.

However, in the context of VoIP, the packet payload ranges from 20 to 150 bytes with an RTP/UDP/IP

header of 40 bytes (IP=20 bytes; UDP=12 bytes; RTP=8 bytes), then the bandwidth overhead of Easy-pass becomes 4.2% up to 13.3%. To further reduce the bandwidth overhead, we can trade security for performance by decreasing the length of Easy-pass from 64 to 32 bits. Most importantly, the Easy-pass is detached from its data packet by the ISP edge router before forwarding the packet to its next hop. Therefore, the Easy-pass mechanism induces no overhead on the reserved link bandwidth of the ISP, and costs nothing along the downstream links and routers.

B.2 Computational Overhead

The computational overhead of Easy-pass per packet is included in the processing overhead of an EF packet at the sending host and the ISP edge router, respectively. We first measure the processing overhead of Easy-pass per packet at the sending host in our testbed as a relative *footprint*, since it depends on the CPU power and the workload of the end-host being tested. Our measured result is 1.29 μs on average, and its dominating factor is the RC-5 encryption. We have found it more important to characterize the overhead of Easy-pass induced at the ISP edge router than that at the end-host, since the end-host can allocate more resources for each outgoing real-time packet than the edge router can. Moreover, the verification of Easy-pass requires more CPU cycles than the Easy-pass embedding procedure at the end-host.

At the ISP edge router, we measure the processing overhead for each EF packet 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-provisioned and the EF traffic is well-behaved, so there is no queueing delay for the EF traffic. Our measured results are constant, irrespective of packet size and traffic rate. 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 13. The overhead incurred by the verification of Easy-pass is 2 μs per packet. For the purpose of comparison, the authentication overhead of the hop-integrity mechanism [12], which is 37 μs per packet, is also shown in the Figure 13. This value was derived in a separate study [12] using a slightly faster machine — a Pentium III 730 MHz running Linux OS. Moreover, 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 per packet is negligible.

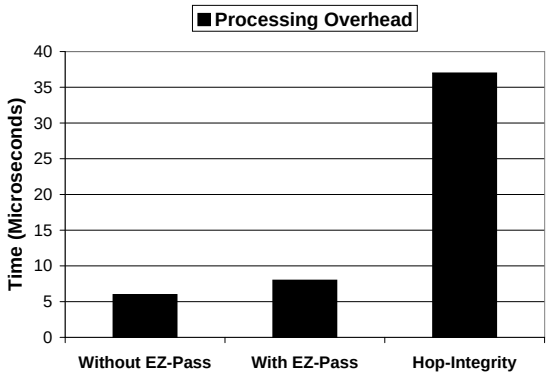

Fig. 13. Processing overhead with and without Easy-Pass

B.3 Impact upon EF Throughput

The per-packet processing overhead of Easy-pass at edge routers not only increases the end-to-end delay, but also decreases the packet-forwarding rate. Although the per-packet processing overhead of Easy-pass is independent of packet size and traffic rate, the cumulative Easy-pass processing overhead is proportional to the number of EF packets being forwarded per unit of time. Thus, we need to consider the impact of the cumulative Easy-pass processing overhead on EF throughput, and measure the potentially largest degradation of EF throughput caused by Easy-pass. So, we ran a set of experiments on our DiffServ testbed. In order to generate an extremely large of number of EF packets, we set the EF packet size as small as 64 bytes while setting the reserved EF bandwidth as large as 10 Mbps. As shown in Figure 14, the EF throughput degradation caused by Easy-pass is negligible. The largest degradation of EF throughput is only 0.036% in this extreme experimental setup that is unfavorable to Easy-pass. Therefore, the decrease of packet-forwarding rate due to the addition of Easy-pass is negligible.

B.4 Comparison with IPsec

Although IPsec can achieve end-to-edge authentication by using tunnel mode, its security model is an overkill for protecting the reserved network resources at edges. We compare the Easy-pass with IPsec AH (Authentication Header) in tunnel mode in three different aspects.

Bandwidth overhead: the length of Easy-pass is 8 bytes, whereas the header overhead of the IPsec AH tunnel mode is 44 bytes, which is 5.5 times as much as that of Easy-pass.

CPU overhead: we measure the Easy-pass CPU overhead on an off-the-shelf 600 MHz Pentium III PC with 256MB RAM as shown above, which is 2μs; in contrast, in a similar platform, the CPU overhead of IPsec

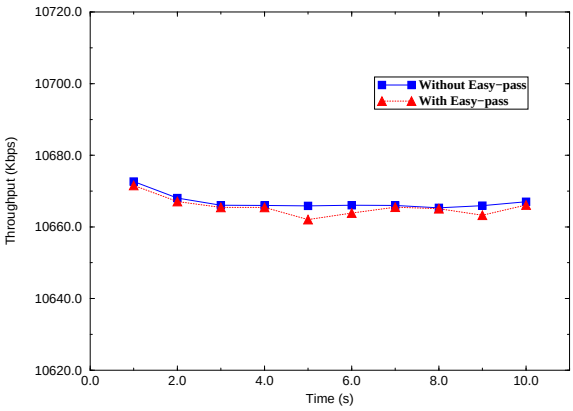

Fig. 14. Impact of processing overhead upon EF throughput

authentication (HMAC-MD5) is 15 to 30 times more than that of Easy-pass with different packet sizes [26], since IPsec authentication covers the entire IP packet. Note that the addition of IPsec encapsulation overhead will further widen the gap of CPU cycle consumption between IPsec and Easy-pass.

Memory overhead at edges: to combat a reply-attack, IPsec maintains a window of 64 sequence numbers for each IP flow. Since the sequence number is 32 bits long, the memory required by anti-reply in IPsec is 4096 bits for each IP flow, whereas Easy-pass only introduces 32-bit flag and 8-bit base variables to achieve the same goal. Therefore, at an edge router, IPsec consumes more than 100 times the memory space of Easy-pass.

C. Effectiveness Against Resource Theft

To validate the effectiveness of Easy-pass against malicious attacks, we performed a series of experiments on our DiffServ testbed. 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 [19, 21], all the EF traffic in our experiments are carried by UDP.

C.1 Protection of Low-rate (Audio) Traffic

As an example of bandwidth requirement for supporting real-time audio applications, we set the source sending rate and the reserved bandwidth at the router to 80 Kbps, respectively. Since the typical packet sizes for real-time audio are 254 and 502 bytes [21] (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 85%. 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 III
PACKET-LOSS RATE FOR LOW-RATE EF TRAFFIC

	w/o EZ-Pass		w/ EZ-Pass	
Flood Rate	502	254	502	254
20K	22%	21%	0	0
40K	38%	35%	0	0
60K	45%	40%	0	0
80K	54%	54%	0	0
100K	63%	61%	0	0
120K	71%	68%	0	0
160K	85%	80%	0	0

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

	w/o EZ-Pass		w/ EZ-Pass	
Flood Rate	502	254	502	254
20K	9.3ms	6.8ms	5.8μs	5.2μs
40K	10.2ms	7.2ms	6.3μs	5.8μs
60K	10.5ms	7.4ms	6.7μs	6.1μs
80K	10.8ms	7.5ms	7.0μs	6.2μs
100K	11.1ms	7.7ms	7.1μs	6.3μs
120K	11.2ms	7.9ms	7.3μs	6.5μs
160K	11.9ms	8.6ms	7.8μs	6.7μs

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.

C.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 [19], 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 band-

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

width. Without 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 5.2 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.

VII. 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 presented the architecture of Easy-pass and discussed its scalability. Then, we described how to create an Easy-pass at the end-host, and how to verify it at the ISP edge router using the RC5 encryption/decryption algorithm. The Easy-pass mechanism has been implemented as loadable Linux kernel modules.

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 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. Moreover, in the experimental setup that is unfavorable to Easy-pass, we found that the throughput degradation due to the addition of Easy-pass is negligible, and hence concluded that the decrease of packet-forwarding rate caused by Easy-pass is negligible as well.

In order to evaluate the effectiveness of Easy-pass against flooding attacks, we conducted a series of experiments on our DiffServ testbed. 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. Furthermore, the Easy-pass mechanism can be easily extended to the inter-domain scenario without any scalability problem, in which two adjacent ISP edge routers validate a higher-tiered traffic aggregate between them. In summary, IP Easy-pass is a very light-weight and effective mechanism for providing network-edge resource access control.

REFERENCES

[1] The ARQoS project. Available: http://arqos.csc.ncsu.edu/.

[2] The authenticated QoS project. Available: http://www.citi.umich.edu/projects/qos/.

[3] 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.

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

[5] S. Blake and *et al.* An architecture for differentiated services. In *RFC 2475*, December 1998.

[6] 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.

[7] CERT Advisory CA-2000.01. Denial-of-service development, January 2000. Available: http://www.cert.org/advisories/CA-2000-01.html.

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

[9] Distributed.net. Rc5-64 project, July 2002. Available: http://www.distributed.net/rc5/.

[10] 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.

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

[12] M. G. Gouda, E. N. Elnozahy, C.-T. Huang, and T. M. McGuire. Hop integrity in computer networks. *IEEE/ACM Transactions on Networking*, 10(3), June 2002.

[13] G. Hadjichristofi, N. Davis 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.

[14] 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*, 5(1), February 1997.

[15] 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.

[16] F. Kelly, P. Key, and S. Zachary. Distributed admission control. *IEEE Journal on Selected Areas in Communications*, 18(12), December 2000.

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

[18] 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.

[19] 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.

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

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

[22] 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.

[23] J.C. Mogul and S.E. Deering. Path mtu discovery. In *RFC 1191*, Novemeber 1990.

[24] 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.

[25] 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.

[26] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *Proceedings of IEEE Symposium on Security and Privacy'2000*, May 2000.

[27] A. Perrig, R. Canetti, J. D. Tygar, and D. Song. The TESLA broadcast authentication protocol. In *RSA Cryptobytes*, Summer 2002.

[28] 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.

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

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

[31] C. Shannon, D. Moore, and K. C. Claffy. Beyond folklore: observations on fragmented traffic. *IEEE/ACM Transactions on Networking*, 10(6), December 2002.

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

[33] 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.

[34] R. Thayer, N. Doraswamy, and R. Glenn. IP security document roadmap. In *RFC 2411*, November 1998.

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

[36] 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*, 14(9), September 2003.

[37] 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.

[38] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. *Proceedings of the IEEE*, 83(10), October 1995.

[39] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: A new resource reservation protocol. *IEEE Network*, 7(5), September 1993.

[40] Z. Zhang, Z. Duan, L. Gao, and Y. T. Hou. Decoupling qos control from core routers: A novel bandwidth broker architecture for scalable support of guaranteed services. In *Proceedings of ACM SIGCOMM '2000*, Stockholm, Sweden, August 2000.