# Authenticated Access to Reserved Network Resources

Kenneth L. Calvert, Srinivasan Venkatraman, and James N. Griffioen (Corresponding author: Kenneth L. Calvert)

Laboratory for Advanced Networking, University of Kentucky Lexington, Kentucky, USA. (Email: {calvert, griff}@netlab.uky.edu)

(Received Aug. 26, 2005; revised and accepted Oct. 1, 2005)

# Abstract

Enhanced network services often involve preferential allocation of resources such as transmission capacity ("bandwidth") and buffer space to packets belonging to certain flows or traffic classes. Such services are vulnerable to denial-of-service attacks if access to those resources is granted based on information that can be forged, such as source and destination addresses and port numbers. Traditional message authentication codes (MACs) are not designed to solve this problem and have high per-packet processing costs. In this paper we propose a packet authentication algorithm specifically designed to solve the problem of protecting access to reserved network resources. We present measurements from a prototype implementation, and argue that our approach is a better solution for this problem than traditional MACs.

Keywords: Denial-of-service, integrated services, packet authentication, RSVP

# 1 Introduction

network Enhanced services typically involve infrastructure-based mechanisms that reserve resourcesbandwidth and/or buffer space—for the exclusive use of packets belonging to particular classes or flows. These quality-of-service (QoS) mechanisms rely on information carried in the packets themselves to distinguish those that are allowed to consume the reserved resources from those that are not. Routers in Internet Protocol (IP)-based networks generally *classify* packets based on the contents of their IP and higher-level headers; packets assigned to the same class (in particular, packets with the same values of source and destination IP address and transport ports) receive the same treatment with respect to scheduling, buffer space, etc.

Because these header fields are controlled entirely by the sending host, this classification mechanism is not secure. In particular, it is easy for an attacker to create and send packets with header information that will give access to some other flow's reserved resources. While this may not directly benefit the attacker, it can prevent the legitimate packets of the class from receiving the level service for which, presumably, someone has paid.

To counter such attacks, some form of authentication check must be applied to packets before granting them access to reserved resources. This can be done by requiring that packets belonging to the legitimate flow have some property that is hard for attackers to duplicate, easy for legitimate senders to create, and inexpensive to verify. Traditional data origin authentication mechanisms (such as keyed-hashes [15] or digitial signatures [17]) are designed to solve a different problem, namely end-toend verification of the authenticity and integrity of data. They are relatively expensive to verify, which—in this context, where the main consideration is the upper bound on processing cost per packet—opens the possibility of denial-of-service by simply overwhelming the verification system. Moreover, they provide a level of security that is poorly matched to this problem. Consider that a false acceptance rate of  $10^{-20}$  would be considered ridiculously high for a traditional message authenticity check (MAC) mechanism, while a reduction of  $10^{-5}$  in the amount of bandwidth available to an end-to-end flow would be unlikely to make a noticeable difference to the end user in many applications that could benefit from enhanced services. In other words, when deciding whether to grant access to router resources, an acceptable rate of access control failure is any one that is not noticeable by the end user. Because enhanced QoS services are generally specified with only limited precision, "acceptable" here could easily be many orders of magnitude higher than what is provided by traditional message authentication codes.

In the next section, we discuss the threat model in more detail. In Section 3 of this paper, we propose a set of design requirements and consider related work, including alternative solutions and their shortcomings. Section 4 introduces FPAC, a fast packet authentication check designed specifically to solve this problem, along with a possible extension to RSVP to support setup of the required state. In Section 5, we present measurements from a prototype implementation, and compare the computational cost of various forms of FPAC to conventional MAC codes such as HMAC. Section 7 summarizes and concludes the paper.

# 2 Denial of Enhanced Services

In this section we first describe our high-level model of how routers process packets to provide both enhanced and best-effort forwarding services. Then we describe the threat model under consideration. Throughout this paper we use the term *class* to refer generically to a set of packets that are treated similarly by routers with respect to forwarding behavior—in particular, access to reserved resources. A class may correspond to a single instance of an application, for example a real-time multimedia flow between a particular source and destination, or it may be an aggregate of flows from a variety of sources travelling to multiple destinations. Each packet in a class follows a *path* through the network. Each path is assumed (for now) to be a linear sequence of routers that is the same for every packet in the class.

# 2.1 Router Processing Model

In general, packets arrive on the input channels or *ports* of a router and undergo some input processing, then are switched through an interconnect to the appropriate output port. There they undergo more processing and eventually are transmitted on the output channel (refer to Figure 1). Because many packets destined for the same port may arrive at more or less the same time, while only one packet may be transmitted at a time, arriving packets may have to queue, and a scheduling mechanism decides which is transmitted next. The queue has finite capacity; when it is full, any packet arrival results in a packet being dropped. Which packet is selected from the queue for transmission (or selected to be dropped when the queue is full) is determined by the scheduling policy (respectively *drop policy*). Enhanced services can be provided by taking into account the class of a packet in making these selections. In general, per-class state information must be maintained by the scheduling mechanism in order to implement such policies. (An example would be information related to the number of bits from this class transmitted in the recent past.) Some means of setting up the necessary classifier and scheduling state is required; the Resource Reservation Protocol RSVP [6] has been designed for that purpose.

# 2.2 Threats

The adversary's goal is to *degrade the service provided* to some class, so that a significant fraction of packets in that class fail to receive the expected (or guaranteed)

quality of service. For example, if the enhanced service provides for a class to experience a very low loss rate, provided its packets are transmitted at or below some maximum rate, an attacker may try to increase the loss rate experienced by that class. If a class' packets are supposed to experience a low delay variance (jitter), the attacker may attempt to increase the variance of the delay experienced by packets of that class. Depending on the service agreement, these disruptions may result in lost revenue for the provider. Note that this is a special form of the more general denial of service problem [9].

The finite-capacity queue is the basis for the denial-ofaccess attack: if an attacker can cause arriving packets to find the queue full (i.e. filled with bogus packets), they will be delayed or dropped, and the class will not receive the enhanced quality of service to which it is entitled. Thus, to mount a denial-of-service attack on a class X, an attacker needs to arrange for bogus packets to be placed in the queue associated with X at a rate that exceeds the limits for X.<sup>1</sup> Note that it may not require an especially high rate of bogus packet insertion to accomplish this. On the other hand, a nonzero but very low rate of bogus packet insertion may be acceptable: enhanced service specifications generally only specify parameters to within some tolerance.

We assume the adversary has the following abilities:

- The ability to cause a potentially large number of end systems ("zombies"), distributed around the network, to transmit many packets containing arbitrary source and destination address information.
- The ability to eavesdrop or *snoop* packets traveling along a path, and to *inject* packets at some point along the path. Either of these can be accomplished by breaking into an end system (non-router) connected to a shared-medium network anywhere along the path—say, a management station in a service provider's network, or a host in the originating domain.

In designing our solution we also assume certain limits on the adversary. We assume the attacker does not have the ability to take full control of a router along the path, nor to mount any kind of "cut and splice" attack (i.e. delay, prevent, or modify packets in flight) on a channel along the way. Given any such capability it is possible to disrupt service to *all* flows along the path, notwithstanding any data-plane mechanism we might propose. We assume that the attacker cannot snoop a legitimate packet, modify and transmit it, and have the modified packet arrive anywhere along the path *before* the original. Although it is conceivable that this assumption might not hold, any violation would require that the attacker have access to routes with *lower delays* than those used to provide enhanced services

 $<sup>^{1}</sup>$ We assume that the enhanced service is correctly implemented, so that other avenues of attack, e.g. modifying the per-class state to make the scheduler think more bits have been transmitted than is actually the case, are not open.



Figure 1: Router processing model

to the attacked class, and that seems unlikely in view of before the scheduling step. Packets that fail the auour other assumptions. Finally, we assume that end systems authorized to emit legitimate packets belonging to the class are not compromised; otherwise such systems might themselves consume all reserved resources.

### 3 Solution Space

Certain techniques have been proposed (and used) as countermeasures against denial-of-service attacks of all kinds, including attacks on end systems. Ingress filtering is a technique in which packets that arrive at a border router via an interface that is not on the route to the packet's source address are assumed to be spoofed and are discarded. These checks have several drawbacks. First, they are not effective under certain conditions that are not especially unusual, including asymmetric routes. Second, they must be applied without exception on the perimeter of the network (i.e. the boundary between domains containing end systems and the core of the network) to be effective. If any border router fails to apply reversepath checks, the attacker may be able to inject spoofed packets into the path. Finally, reverse path checks do not stop an adversary that controls one or more end systems co-located with the actual packet source.

Traceback has been proposed as a method of determining the origin of denial-of-service attacks, in order to identify and punish the attackers. Recently various methods have been proposed for modifying the forwarding path to enable reconstruction of the paths followed by packets [20, 23]. Although traceback may indeed be useful as a deterrent, we do not consider it a solution because it is fundamentally reactive rather than preventive. Traceback does not prevent an attack from disrupting the enhanced services for which customers have paid a premium. (For that matter, its utility as a reactive mechanism is questionable if an attacker controls hundreds or even thousands of zombie systems all over the world.) Before enhanced services can be commercially successful, providers need to be confident they can be robustly implemented, and are not subject to the whims of attackers.

Our proposed approach involves adding a step called authentication in the router processing loop somewhere thentication check are denied access to any reserved resources. Although this involves modification of the packet-processing path in routers, we believe the modifications required are consistent with the processing required for enhanced services, both in scope and in kind. In the next subsection we present desiderata for such an authentication mechanism, considering both security and performance.

### 3.1**Design Requirements**

The authentication mechanism must have certain security and performance properties in order to be considered a solution to our problem. These include:

- Cryptographic strength. It must be infeasible for an attacker to consistently forge packets without breaking some believed-to-be-strong cryptographic primitive(s). Brute-force methods (i.e. random guessing) must yield a sufficiently low probability of success that the service delivered to legitimate packets is unaffected.
- Replay prevention. The mechanism must include a means to detect re-use of header information. This implies that per-class state must be updated each time a legitimate packet is verified. Given a fixed field size in the packet, it also implies that the number of packets that can be verified is bounded; however, that number should be large, so that resynchronization (rekeying) is required infrequently, even for high-data-rate classes.
- Robustness. Because we are dealing with enhanced services, losses and packet reorderings should be rare. Nevertheless, the mechanism must tolerate a certain (limited) degree of loss and reordering.
- **Performance.** It must be possible to perform the authentication check fast enough to keep up with the maximum possible arrival rate of packets at the authentication check. Otherwise, an attacker can deny service by overwhelming the mechanism. We assume that authentication can be pipelined with other operations, and consider only requirements that follow

simplicity, all incoming and outgoing links are assumed to operate at the same rate,  $\rho$  bytes/s. Let  $\alpha$ be the "degree", i.e. the number of incoming interfaces of the router. Assume for the moment that all packets are S bytes in length; then  $S/\rho$  is the time between packet arrivals on each link. Thus it must be possible to authenticate at least  $\alpha$  packets every  $S/\rho$ seconds. Note that this constraint is strongest when S is minimized, i.e. when all packets are minimumsized.

We call this the *minimum-sized packet constraint*.

• Implementability. The authentication mechanism must be amenable to implementation in hardware, so that it can be placed close to the scheduler (i.e., on port cards in routers).

#### 3.2**Alternative Authentication Solutions**

Here we consider why traditional origin-authentication solutions are not suitable as solutions for this problem. Public-key signatures are the canonical method of verifying authenticity. They have the advantage of not requiring any secret information to be distributed to or stored at the routers—only the public key associated with the class is required for verification. Unfortunately at the present state of the art signature verification is so computationally expensive that it cannot possibly meet the performance requirements outlined above. We therefore do not consider it further.

Hop-by-hop authentication, where each router "vouches" for the legitimacy of the packets it forwards to its neighbors, has the advantage that it does not require distribution of any shared secret to routers along a path. However, as with ingress filtering, it is only effective when applied throughout the network, to every packet (not just those belonging to the target class) passing through every router along the targeted path. This raises the cost for *every* user, not just those who want reserved resources. Moreover, hop-by-hop authentication requires that the authentication code be re-created for each packet at each hop.

Conventional end-to-end MACs, such as HMAC [15] or UMAC [3] provide high security and protection against tampering, but do not protect against replay, and more importantly have relatively high costs for minimum-sized packets, and therefore open the possibility of an attack based on saturating the verification mechanism itself.

In the next section we propose a solution based on block ciphers.

### 4 Fixed-cost Packet Authentication

We now introduce FPAC, an authentication code intended for use in controlling access to reserved network resources,

directly from the basic architecture of the router. For i.e. bandwidth and buffer space. FPAC enables a verifier to conclude that, with reasonably high probability, the code carried in a packet was constructed with knowledge of a secret. The security of FPAC rests upon standard cryptographic primitives, primarily a block cipher. FPAC is not a conventional message authentication code (MAC), in that it does not protect against tampering of data in the packet. We believe this is appropriate because, as discussed in the previous section, FPAC is intended to solve a different problem. To emphasize this distinction, we refer to conventional MAC algorithms like HMAC [15] as "end-to-end MACs" in the rest of this paper. Applications concerned about tampering and forgery need to use an end-to-end MAC in addition to FPAC. FPAC can also be conveniently parameterized, allowing users to select a level of security appropriate for their needs. In the following description, the *sender* is the originator of the packet that is authorized to receive enhanced services; the ver*ifier* is the authentication mechanism in the router that needs to determine a packet's legitimacy.

### 4.1Overview

FPAC requires two fields in the header of a packet that is to be authenticated for access to reserved resources, nonce and cookie. Sender and verifier are assumed to share three secrets,  $k_0$ ,  $k_1$  and  $s_0$ . (Note that every router providing enhanced services must be a verifier and must know these secrets; we discuss possible solutions for distributing them later in this section.)

FPAC is based on the standard technique of constructing a bitstring to have a particular property, and then encrypting that string with a symmetric cipher using a key known to both the sender and the verifier. The algorithm is shown in Figure 2. The sender constructs the cookie C from a random nonce X, the shared secrets, and a sequence number s. (The function *choosenonce*() selects a fresh bit string in an unpredictable way; the function nextrandseq(x) chooses the next sequence number semirandomly as described in the next subsection.)

First the sender maps the nonce to another value using the random function  $f_{k_0}$ , which is known only to the sender and verifier and need not be invertible. One possibility is for  $f_{k_0}$  to be chosen (via  $k_0$ ) from a family of universal hash functions [3]. These functions are ideal for construction of V because they are very inexpensive to compute and they produce outputs of about the right size. Next the result V is concatenated with a sequence number (the computation of which is described below) and the whole block is encrypted under a block cipher  $E_k[\cdot]$ using secret  $k_1$  as key. The resulting cookie is placed in the packet along with the nonce.

The verifier knows the three secrets, and also keeps track of sequence numbers contained in previously verified packets. It extracts the nonce and the cookie, computes V, and verifies that the result of decrypting the cookie begins with V. If so, it then verifies that the set of already-seen sequence numbers does not contain s. If

Sender:	Verifier: secret $k_0, k_1, s_0;$		
secret $k_0, k_1, s_0$ ;	set of sequence# seen;		
set of sequence# sent;	extract $(X, C)$ from packet		
s := nextrandseq(sent);	$V := f_{k_0}(X);$		
$sent := sent \cup \{s\};$	$P s := D_{k_1}[C];$		
X := choosenonce();	if $(P = V \land s \notin \text{seen})$		
$V := f_{k_0}(X);$	seen := seen $\cup \{s\};$ accept packet;		
$C := E_{k_1}[V s];$	else		
place $(X, C)$ in packet;	discard packet;		

Figure 2:	Sender	and	Verifier	algorithms	for	FPAC

that is the case, it records the fact that s was seen and forwards the packet; otherwise the packet is dropped. Thus the cost of verifying FPAC is one block decryption, plus the cost of computing V and comparing it to part of the decryption result.

The security of FPAC rests on the block cipher, which prevents an attacker from forging a valid cookie without knowing  $k_1$ . The purpose of the random function f is to prevent known plaintext attacks (since the nonce Xis included with the packet). Given a strong cipher and no information about  $k_1$ , an attacker should not be able to do better than brute force, i.e. trying arbitrary bit strings in the hope that one decrypts to a valid authenticator. In that case the likelihood of a successful forgery depends on the size of V. At the same time, sequence numbers cannot be re-used, so the size of s determines the number of packets that can be sent before resynchronization/rekeying is necessary. Thus, for a given size of block cipher, there is a tradeoff in FPAC between the likelihood of forgery and the frequency of resynchronization. A larger V makes forgery less likely, but the correspondingly smaller s implies that fewer packets can be sent before resynchronization.

The number of bits allotted to V and to s should be a configurable parameter of the protocol; implementations should allow selection from a set of supported splits. In our implementations with 64-bit block ciphers, we have used two different splits: a 24-bit V and 40-bit s, and an equal split of 32 bits each. For most applications, we believe a 24-bit V provides a sufficiently low likelihood of forgery (under the assumptions above), while allowing enough packets to be sent between resynchronizations. New block ciphers, such as AES, provide an even larger block size of 128 bits, while still achieving high performance using hardware acceleration. With a 128-bit block size, a rather large range of splits is possible.

## 4.2 Protection Against Replay

Protection against replayed packets is handled through the use of sequence numbers, as in the Encapsulating Security Payload (ESP) of IPSec [14]: Each packet is assigned a sequence number by its originator, and the verifier keeps track of which sequence numbers have been seen; sequence numbers are not reused. There are several differences between our approach and ESP, however. In ESP the sequence number is transmitted in cleartext, outside the encrypted payload. In FPAC, the sequence number is tied to the authentication check by encrypting both in the same block; this provides a check against modification of the FPAC field, and in particular prevents the authentication code from being re-used with a different sequence number. Second, sequence numbers are not assigned in strictly monotonic order by the sender. This has two purposes: it increases the entropy of the anti-replay field in the encrypted block, to make linear or differential cryptanalysis more difficult; and it reduces synchronization requirements when packets are originated by more than one sender (see discussion below). Third, the sequence begins at a random value (the ESP sequence number begins at 1). Again, this is intended to reduce the attacker's knowledge of the plaintext to make cryptanalysis harder.

We now describe the construction of s. In this description all arithmetic is understood to be modulo  $2^w$ , where w is the size of the sequence number in bits.

The sender maintains a send window of sequence numbers in the range  $L_s$  to  $L_s + T - 1$  (inclusive); the initial value of  $L_s$ , or initial sequence number (ISN) is chosen randomly and is part of the secret information shared with the verifiers (i.e.  $s_0$ ). Each verifier maintains a window of acceptable sequence numbers, defined by the range  $L_r$  to  $L_r + W - 1$  (inclusive), where T < W and initially  $L_r$  is set to the ISN. In addition, each verifier knows the value of T. For each packet, the sender randomly chooses an unused sequence number in the send window, and records that it has been used. After verifying the authenticity of a packet, each router checks whether the sequence number of the packet is in its window and whether it has been seen before; if not, it records the use of that sequence number (e.g. by setting a bit in a bitmap).

It remains to specify how the sender and verifiers advance their respective windows, and the details of the construction of s. The sender can advance  $L_s$  by an amount j > 0 whenever sequence numbers  $L_s$  through  $L_s + j - 1$ have been used. Similarly, the verifier can advance  $L_r$  by an amount j > 0 whenever all sequence numbers in the range  $L_r$  to  $L_r + j - 1$  have been seen. (Note the winISN  $(s_0)$  to be in the window; this indicates that the sequence number space has wrapped. Sender should watch for the send window to move within some fixed distance of ISN, and initiate rekey/resynchronization operations when that occurs; otherwise the class becomes vulnerable to replay.)

The above policy for advancing the window at the verifier is not adequate if losses can occur—the window would never advance past the sequence number carried by any lost packet. Therefore a verifier must also be prepared to advance its window when it sees evidence that the sender's window has advanced. When the verifier sees a sequence number j in the range  $L_r + T \leq j < L_r + W$ , it can infer that it is safe to advance  $L_r$  to j - T + 1, which maintains  $L_r \leq L_s$  because  $j < L_s + T$ . However, this inference is only valid if no reordering and no forgery have occurred. Because of the requirements to limit the damage done by successful forgery and also to tolerate some amount of packet reordering, the verifier may want to wait for stronger evidence before advancing the window past unused sequence numbers—say, seeing more than some threshold number of packets in the range  $L_r + T$  to  $L_r + W$ .

The exact choices of T, W, and the amount of evidence required to advance the window past an unused sequence number are policy decisions which embody tradeoffs. Wneeds to be enough bigger than T to accomodate a case where the sender advances the window in a large jump;  $W \geq 2T$  should suffice. A large value of T is desirable so the sender can maximize the entropy of the sequence number (see below); however, a larger T (and thus W) requires more state at the verifier. Entropy can also be kept higher by advancing the send window frequently.

The policy for advancing the verifier's window past unseen packets should be set based on the probabilities of loss, reordering and forgery. Because we are dealing with enhanced services, loss and reordering are expected to be rare events. A reasonable policy might be to require that three (valid) packets with numbers beyond  $L_r + T$  be seen for every unseen sequence number skipped.

Finally, consider the advancement of s. If we simply let s be the arithmetically next sequence number, the highorder bits of s will contain very little entropy. On the other hand, if the sender chooses sequence numbers carefully the low-order bits of s will contain a good deal of entropy. Assume that T is large enough that the low-order byte of the sequence number in any packet is essentially random. We want to "spread" this entropy over s a little bit.

Let the actual sequence number be N, and divide Ninto *d*-bit chunks. Assuming a 32-bit sequence number and 8-bit chunks (bytes), let  $n_i$  be the *i*th byte of N, with  $n_0$  the low-order and  $n_3$  the high-order byte. The sender does the following: Let  $n'_0 = n_0$ , and for i = 1, 2, 3set  $n'_i := n_i \oplus n'_{i-1}$ . Then set  $s := n'_3 | \dots | n'_0$ . The verifier similarly divides s into  $n'_0, \ldots, n'_3$ , and for i = 3, 2, 1 sets  $n_i := n'_i \oplus n'_{i-1}$ , and  $n_0 := n'_0$ . The verifier uses N :=

dow must not be advanced if doing so would cause the  $n_3|n_2|n_1|n_0$  as the sequence number, and applies the tests described above. The effect of these two steps together is to make it more difficult for an attacker to determine whether the block cipher key has been guessed.

> When multiple sources send packets to multiple destinations using the same class, it presents problems for the anti-replay mechanism. Some advance coordination is required so that sources use different subsets of the sequence number space. One possible solution is to partition the anti-replay sequence number space among sourcedestination (or ingress-egress) pairs, using some bits of sas a tag to designate the pair-path to which the packet belongs. For example, using 8 bits suffices to designate all combinations of 16 sources and 16 destinations. Each source needs to know the tag for each destination with which it communicates. Each verifier would need to maintain a separate, independent window of acceptable sequence numbers for each tag that it sees. The same  $s_0$ can be used for all tags.

> The main drawbacks of this approach are a reduction in sequence space available for each path, and additional state required at each verifier. The reduced sequence space can be ameliorated by using a cipher with a larger block size, such as AES. Also, routers would see only tags for paths of which they are a part; most routers would need to keep state for only a few additional paths. On the whole, however, the best solution may be to simply establish a separate class for each sender.

### 4.3Key Distribution

Each verifier along the path needs to acquire the secrets  $k_0, k_1$  and  $s_0$  from the sender in a secure fashion. This is a nontrivial problem; group key distribution protocols are an active research topic, especially for group services [1, 11]. Here we sketch the design of a solution that fits fairly well within the general framework of RSVP.

The basic idea is to use extend RSVP so that the secrets needed to establish the FPAC state are carried in an encrypted object in the RSVP PATH message [6]. To accomplish this, a *separate* security association—i.e. a ciphersuite, keys, and other parameters needed to establish a secure channel—must exist between each pair of adjacent RSVP-capable routers along the path. RSVP already needs such hop-by-hop security associations to verify that RSVP messages have not been tampered with, using the INTEGRITY object [6]. In addition, a new FPACS object would be defined for inclusion in the PATH message. The FPACS object would contain a security association index along with the three secrets, encrypted per the given security association. At each intermediate hop, the FPACS object would be decrypted and re-encrypted under the appropriate security association for the next hop. (This is the same approach proposed for authentication with the RSVP INTEGRITY object.)

The challenge is that the RSVP sending or forwarding a PATH message may not be aware of what the next hop will be, and therefore may not know which security as-



Figure 3: Message exchange for FPAC setup

sociation to use to encrypt the FPACS object. Therefore an approach similar to that used to establish the initial sequence numbers for RSVP INTEGRITY objects can be used. (Refer to Figure 3.) The PATH message is initially sent with an "empty" FPACS object, e.g. with the SA index set to zero.

When a node receives such a message, it first determines whether a security association already exists with the sender of the message, and if not, establishes one using a standard key-management protocol. It then sends an FPACACK message (a new RSVP message, similar in function to the CHALLENGE message used for sequence number initialization in RSVP [2]) back to the sender, to inform it of the identity of the next hop for this class. The sender then transmits subsequent PATH messages with complete FPACS objects, including keys encrypted according to the security association. Note that only the FPACS portion of the PATH message is encrypted; the rest is transmitted normally. Also, note that an intermediate node may need to transmit PATH messages with FPACS objects encrypted under multiple different security associations (e.g. in the case of a multicast destination). This is, however, similar to what must already be done for the INTEGRITY object.

Routers in a path that belong to a single administrative domain could easily be configured with a common security association for FPACS objects. For the case of border routers, it seems likely that such routers will need security associations for other purposes, especially if they cooperate in providing premium or enhanced services to customers.

### 4.4 Security

The security of FPAC rests on the security of the underlying block cipher. The use of the block cipher in FPAC has been designed to to minimize known plaintext, and to make it difficult for an attacker to determine whether the key has been found. We believe that this design makes it infeasible to systematically forge packets. Random FPAC values will successfully verify with a probability that decreases exponentially with the length of the authenticator V. When the anti-replay check is also considered, the probability of a random FPAC code being accepted is at most  $2^{-B}W$ , where B is the size of the block cipher. So for a 64-bit block cipher and a window of 2048 acceptable sequence numbers, the probability of random forgery is no more than  $2^{-53}$ . This is sufficiently low to ensure that forged packets do not steal a significant amount of must be noted that as freely available code, the perfor-

Table 1: Average Verification Costs for 1024-byte packets

Algorithm	Cycles/Packet
HMAC-MD5	20951
FPAC (Khufu-8)	999
FPAC (Blowfish-16)	2056
FPAC (Khufu-8) with anti-replay	1521
FPAC (Blowfish-16) with anti-replay	2520

reserved resources from a class.

### Implementation and Evaluation $\mathbf{5}$

We added authentication to the *traffic control* facility of the Linux kernel (ver. 2.2.14). Traffic control provides queueing disciplines, classes within queueing disciplines, classification filters, and policing. Each network device has an associated queueing discipline, which determines how packets enqueued for that device are treated. When a packet is forwarded by the kernel, after the outgoing interface and next hop address are determined, control gets the packet, classifies it, and decides if it should be queued, dropped, or delayed.

In our implementation, authentication can be enabled for a particular class. The necessary information, including secrets, is distributed out-of-band in our implementation; anti-replay is a configurable option. The authentication method (and required secrets/initialization information) are specified by the sender using a socket option on the sending socket. At intermediate routers, this information is specified for the class via the tc program, which provides an administrative interface to the traffic control facility. At the source node, the FPAC code is added in the kernel as part of the IP protocol processing. It is carried in an IPv4 option whose length depends on the particular authentication algorithm selected. At intermediate nodes, packets are authenticated during classification.

We implemented several authentication algorithms and versions of FPAC, including HMAC-MD5, FPAC, in two versions based on 16-round Blowfish [21] and 8-round Khufu [21] as block ciphers, respectively. (Note that the latter offers increased performance at the expense of reduced security.) Publicly-available versions of Blowfish, Khufu, and MD5, with minor hand-tuning, were used. It



Figure 4: Cycle count per packet

Figure 5: Verification cost vs. packet size

UM-U

1400

1600

mance of these implementations should not be interpreted as any kind of limit or indication of what can be achieved with these algorithms. Rather, our purpose here is to illustrate the general *differences* between approaches and costs for the different parts of the algorithm.

First we measured the number of CPU cycles required for each packet verification, with and without anti-replay, for the different authentication algorithms. We instrumented our implementation to count the cycles required to verify a packet on the forwarding path, using the cycle counter (TSC register) on a 500 MHz Pentium III. For each algorithm, we measured the cycles required for each of 1000 packets of 1024 bytes each; the numbers presented in Table 1 are the resulting averages. The interval measured was the time from when the authentication routine was called (as part of traffic control processing) until it returned. Note that these measurements are from packets that arrived on one 100Mbps Ethernet interface and were forwarded to another, on an unloaded machine.

Figure 4 shows details of the individual per-packet measurements for four versions of FPAC using two different block ciphers, with and without anti-replay. Variations in the individual cycle counts are believed to be due to the varying state of the cache upon packet arrival. The difference between the top and bottom curves illustrates the performance cost of increasing security, in going from an 8-round block cipher without anti-replay to a 16-round cipher with anti-replay—about 1300 cycles per packet. Also, based on these figures we can place the cost of the anti-replay check at about 500 cycles per packet. Finally, we note that cost of HMAC-MD5 (which does not include anti-replay) is about one decimal order of magnitude higher than the slowest version of FPAC, due to its need to "touch" every bit in the packet.

Figure 5 shows that the per-packet cost of FPAC is C, and the attacker D via three different interfaces. At independent of the packet size. We measured the cycle count for packets of sizes from 100 bytes to 1400 bytes packets with destination address C. The sender at A and in intervals of 200 packets. Each point is the average of the DoS attacker at D can both send UDP packets to C.

the measured value of 1000 packets of that size; again, each measurement was taken on a forwarded packet, with no cache warming. Again, all the FPAC implementations exhibit constant time, and are faster than conventional MACs by a significant margin, even for the smallest packets.

This graph also includes data for UMAC, an end-toend MAC designed specifically for extreme speed in software implementations [3]. UMAC belongs to the family of MACs based on universal hash functions, and the version measured provides a probability of forgery of about  $2^{-60}$ . The numbers shown were obtained using a publicly available user-space implementation. The code was run on the same machine used for the other measurements, and the number shown was obtained by taking the "cycles per byte" result output by the UMAC implementation and multiplying it by the packet size. Note well that the code from which the UMAC measurements were obtained is extensively tuned. Most importantly, it primes the cache before measurement, by executing the code once on the input data before starting the clock. This makes a difference of about a factor of two in the total per-packet time. Nevertheless, our fastest FPAC implementation is still a factor of three faster than UMAC, even for smaller packet sizes. These results show that eliminating unneeded security properties pays off with significant performance improvement: even a straightforward implementation of the FPAC algorithm significantly outperforms conventional MACs implemented in software.

Finally, we verified that our authentication mechanism does what it is supposed to do, by measuring performance of a class with reserved bandwidth during a simulated denial-of-service attack. The test setup is shown in Figure 6. The router B is connected to the sender A, receiver C, and the attacker D via three different interfaces. At router B, bandwidth of 500,000 bits/sec was reserved for packets with destination address C. The sender at A and the DoS attacker at D can both send UDP packets to C.



Figure 6: Test setup for measurements



Figure 7: Packet arrivals vs. time

We recorded (at C) the arrival times of packets sent from A, with and without the DoS interference from D, and with and without FPAC; the results are shown in Figure 7. Three of the four curves ("non-FPAC sender with no attack", "FPAC sender with no attack", and "FPAC sender with attack") coincide: they show data arriving at C at exactly the reserved rate. The other curve ("non-FPAC sender with attack") shows that when the DoS stream is present and FPAC is not used, the rate of arrival of legitimate packets at C is reduced by about a factor of two. We emphasize that the point of these graphs is mainly qualitative and illustrative, rather than quantitative. The use of block ciphers such as AES [18] offers the potential for both higher performance (due to the availability of commercial hardware accelerators that can reduce the cost of encryption/decryption to a few *dozen* cycles per byte) and better security (due to AES's larger block size, which gives a substantially lower probability of succeeding by random guessing).

# 6 Related Work

Denial-of-service for regular best-effort service has received a good deal of attention among researchers; the problem problem of preventing denial of access attacks against reserved resources has received substantially less. The proposed standards for Integrated Services in the Internet [5, 22, 26, 27] mention the problem, but do not specify any particular solution. RSVP, the signaling protocol specified for use with Integrated Services, contains provisions for authentication of signaling messages, but they are irrelevant to the problem of authentication of data packets. Some researchers have considered the problem of DoS attacks using the signaling and reservation mechanisms as specified [19]. The active networking community has considered the general problem of controlling access to all kinds of node resources, but proposed solutions rely on conventional authentication mechanisms [8]. Shalunuv and Teitelbaum argue that the very notion of "Quality of Service" should already imply resistance to DoS [24], and suggest (as do we) that providers' inability to guarantee quality in the face of attacks has limited demand for enhanced services. Handley and Greenhalgh have outlined a novel architecture [10] that includes features designed to make denial of service more difficult for all traffic, not just traffic for which resources are reserved.

Braun [7] proposed a method that allows providers of Differentiated Services to distinguish valid packets from malicious packets, by having the source "sign" each packet and QoS routers verify each packet. Braun's method, however, does not use cryptographic primitives, and therefore its security properties are difficult to quantify.

The high computational cost of conventional end-toend MAC methods bashed on cryptographic hash functions is a well-known problem [25]. Additional evidence of this is provided by the competition to design the fastest hash function. UMAC [3] is claimed to be one of the fastest current MAC functions for implementation in software. UMAC is claimed to cost less than one machine cycle per byte for messages larger than about 2Kbytes, however its per-byte cost for small messages is significantly larger. Also, the cost of UMAC will always exceed the minimum cost of an HMAC-SHA1 computation, because UMAC requires computing HMAC-SHA1 over a small block.

Both the security and performance of FPAC rest on the same properties of an underlying block cipher. Schneier [21] presents a comprehensive overview of block ciphers, including a performance comparison of several algorithms. Schneier claims that his Blowfish algorithm can be implemented for 26 cycles per byte, or about 210 cycles for a block encryption. We have so far been unable to reproduce those numbers, but we have not attempted to optimize our code significantly. Given a block algorithm of such speed, the per-packet times of FPAC would be reduced by a factor of about three from those given in the previous section.

### 7 Conclusions

We have carefully examined the problem of protecting the resources required to implement enhanced network services. We concluded that the most plausible attack scenario involves an adversary whose capabilities are sufficiently limited that it is reasonable to forego some of the security guarantees provided by conventional message authentication checks, in order to reduce the cost of perpacket authentication. This cost must be low enough to preclude denial-of-service attacks based on exceeding the capacity of the verification machinery.

Accordingly, we have proposed a new approach that provides reduced security at significantly reduced perpacket cost. The approach, which we call FPAC, is not a substitute for more powerful algorithms when it comes to end-to-end security and integrity. However, it is a good match for the problem of ensuring that reserved bandwidth and buffering are always available for those users who have paid for premium service. It is not clear whether the general lack of enthusiasm for premium services in the present Internet is due to the inability of service providers to preclude attacks like those considered here. If that is indeed the case, FPAC—or other codes designed for the same purpose—may enable deployment of such services to grow in the future.

We have implemented FPAC, and presented data on its performance. However, comparing performance of software-based implementations is somewhat beside the point. The goal is to design a mechanism suitable for implementation in hardware, FPGA or network processor that can run at many multiples of line speed. We believe FPAC achieves that goal.

# References

- [1] R. Atkinson and S. Floyd, IAB Concerns and Recommendations Regarding Internet Research and Evolution, Internet Architecture Board, RFC 3869, Aug. 2004.
- [2] F. Baker, B. Lindell, and M. Talwar, RSVP Cryptographic Authentication, RFC 2747, Jan. 2000.
- [3] J. Black, S. Halevi, H. Krawcyk, T. Krovets, and P. Rogaway, "UMAC: Fast and secure message authentication," in CRYPTO '99, LNCS 1666, pp. 216-233, Sringer-Verlag, 1999.
- [4] S. Blake et al., An Architecture for Differentiated Service, RFC 2475, Dec. 1998.
- [5] R. Braden, D. Clark, and S. Shenker, Integrated [24] S. Shalunov and B. Teitelbaum, "Quality of service Services in the Internet Architecture: An Overview, RFC 1663, June 1994.
- [6] R. Braden, Resource Reservation Protocol (RSVP) - Functional Specification (ver. 1), RFC 2205, Sept. 1997.
- [7] M. J. Braun, Mitigating Denial of Service AttacksonDifferentiated Services Networks, Naval Postgraduate School, A960704, Sept. 2002, http://www.stormingmedia.us/96/9607/A960704.html.

- [8] R. Campbell, D. Mickunas et al, Building a Dynamic Interoperable Security Architecture for Active Networks, Quarterly Project Report of the Seraphim Project, Jan. 2001.
- [9] CERT Coordination Center, Denial of Service Attacks, Carnegie Mellon University, Oct. 2005, http://www.cert.org/tech\_tips/denial\_of\_service.html.
- [10]M. Handly and A. Greenhalgh, "Steps towards a DoS-resistant Internet architecture," in ACM Workshop on Future Directions in Network Architecture, pp. 49-56, Portland, Oregon, Aug. 2004.
- [11] T. Hardjono and B. Weis, The Multicast Group Security Architecture, Internet Engineering Task Force, RFC 3740, Mar. 2004.
- [12] S. Kent and R. Atkinson, Security Architecture for the Internet Protocol, RFC 2401, Nov. 1998.
- [13] S. Kent and R. Atkinson, IP Authentication Header, RFC 2402, Nov. 1998.
- [14] S. Kent and R. Atkinson, *IP Encapsulating Security* Payload (ESP), RFC 2406, Nov. 1998.
- [15] H. Krawcyk, M. Bellare, and R. Canetti, HMAC: Keyed-hashing for Message Authentication, RFC 2104, Feb. 1997.
- [16]M. Moyer, J. Rao, and P. Rohatgi, "A survey of security issues in multicast communications," IEEE Network Magazine, vol. 13, no. 6, pp. 12-23, Nov./Dec. 1999.
- [17] National Institute of Standards and Technology, Digital Signature Standard, Federal Information Processing Standards Publication 186, 1994.
- [18] National Institute of Standards and Technology, Advanced Encryption Standard, Federal Information Processing Standards Publication 197, Nov. 2001.
- [19] D. Reeves, F. Wu, P. Wurman, D. Stevenson, and X. Wu, Protecting Network Quality of Service Against Denial of Service Attacks, North Carolina State University, A938024, Dec. 2003, http://www.stormingmedia.us/93/9380/A938024.html.
- [20]S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Practical network support for IP traceback," in ACM SIGCOMM 2000, pp. 295-306, Aug. 2000.
- [21]B. Schneier, Applied Cryptography, Wiley, 1996.
- [22]S. Shenker, C. Partridge, and R. Guerin, Specification of Guaranteed Quality of Service, RFC 2212, Sept. 1997.
- [23]A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, S. Kent, and W. Strayer, "Hash-based IP traceback," in ACM SIGCOMM 2001, pp. 3-14, Aug. 2001.
- and denial of service," in ACM SIGCOMM 2003, pp. 137-140, Karlsruhe, Germany, Aug. 2003,
- J. Touch, "Performance analysis of MD5," in ACM [25]SIGCOMM '95, pp. 77-86, Cambridge, USA, Sept. 1995.
- [26] J. Wroclawski, The Use of RSVP with IETF Integrated Services, RFC 2210, Sept. 1997.
- [27]J. Wroclawski, Specification of the Controlled-Load Network Element Service, RFC 2211, Sept. 1997.



Kenneth L. Calvert is Associate Professor in the Department of Computer Science at the University of Kentucky. His research deals with the design and implementation of advanced network protocols and services. Current projects include

network programmability, generalized routing, network security, and models of Internet topology. He holds a PhD in Computer Science from the University of Texas at Austin.

Srinivasan Venkatraman holds a Bachelor of Engineering in Computer Science and a Master of Science in Mathematics, both from the Birla Institute of Technology and Science, Pilai, India. He also holds a Master of Science in Computer Science from the University of Kentucky, Lexington, Kentucky, USA. He has also worked as a Senior Software Engineer at Wipro Infotech in Bangalore, India.

