

FPAC: Fast, Fixed-Cost Authentication for Access to Reserved Resources

Kenneth L. Calvert, Srinivasan Venkatraman, and James N. Griffioen

Laboratory for Advanced Networking

University of Kentucky

Lexington, Kentucky

{calvert,srini,griff}@netlab.uky.edu

Abstract—Enhanced network services often involve allocating resources (bandwidth/buffer space) preferentially to packets belonging to certain flows or traffic classes. Such services are vulnerable to denial-of-service attacks if packet classification is based on information that can be forged, such as source and destination addresses and port numbers. Traditional message authentication codes (MACs), often considered the only solution to this problem, are really not designed to solve it. In particular, their per-packet costs are so high that they enable another form of denial-of-service attack based on overwhelming the verification mechanism.

We describe the problem of denial of access to reserved resources and the inadequacies of conventional solutions. We then observe that it is reasonable to trade some of the strong security guarantees provided by conventional MACs for a lower per-packet cost. We propose a new packet authentication algorithm, designed to solve the problem of protecting reserved resources, with a very low, fixed per-packet cost. While it cannot replace conventional MACs for end-to-end authentication, we argue that it is a better solution for the problem considered here. We present measurements from a prototype implementation that can verify a packet of arbitrary size in as few as 1000 machine cycles on an Intel architecture machine.

I. INTRODUCTION

Enhanced network services typically involve router-based mechanisms that reserve resources (bandwidth and/or buffer space) for exclusive use by packets belonging to particular flows or classes. These mechanisms rely on information carried in the packets themselves to distinguish those allowed access to 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 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, an attacker who knows the classification algorithm can send packets with header information that will give them access to some other flow’s reserved resources. Although this generally does not directly benefit the attacker, it can prevent the legitimate packets of the class from receiving the services which, presumably, someone has paid for.

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 be-

longing to the legitimate flow have some property that is hard for attackers to duplicate, but easy for legitimate senders to create. Conventional cryptographic approaches to implementing such a *message authentication check* (MAC) for packets include digital signatures or including in the message a hash of its contents with a secret shared among the flow participants and routers (e.g. HMAC [?] or UMAC [?]).

Such traditional authentication mechanisms are a poor match for the problem of controlling access to reserved resources, for two reasons. First, they are computationally expensive, and their cost is proportional to the length of the packet. Because computation is a notoriously scarce resource in routers, this opens the possibility that an attacker could saturate the authentication-checking capacity with bogus packets at some point along a flow’s path, effectively denying service to the legitimate anyway.

Second, traditional mechanisms provide a level of security that is very high; they ensure that the probability of an attacker generating a packet that passes for legitimate is astronomically low. While such strong security is critical for many applications (especially to protect against tampering on an end-to-end basis), it is overkill in the context of guaranteeing low delay and loss *probabilities* to particular classes of packets. When deciding whether to grant access to router resources, it is sufficient to limit the probability of accepting a bogus packet to some acceptable level. 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 this paper we develop a model of router processing for enhanced services, and consider several attack scenarios in the context of that model. We then consider the drawbacks of existing countermeasures for such attacks. We then present FPAC: a fast packet authentication code, designed to solve this specific problem. FPAC has very low per-packet computational cost that is independent of the packet size.

The rest of this paper is organized as follows. In the next section we describe the threat model. We then discuss alternative solutions and their shortcomings. Two variants of the FPAC mechanism are described in detail in Section IV. In Section V, we compare the computational cost of various forms of FPAC to conventional MAC codes such as HMAC. Section VII summarizes and concludes the paper.

Work supported by Defense Advanced Research Projects Agency and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-99-1-0514; and by the National Science Foundation under grant number EPS-9874764. Views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, the Air Force Research Laboratory, or the U.S. Government.

II. DENIAL OF ENHANCED SERVICES

We first set context via a general description of the packet-forwarding process in modern routers, and the constraints implied by it. Then we consider various attack scenarios, based on the capabilities of the adversary, and identify one class of scenarios as suitable for solution via per-packet authentication checks. Finally, we derive a set of requirements for a solution.

Throughout this paper, we use the term *class* to refer to a set of packets that are treated identically with respect to forwarding behavior and access to resources. A class may correspond to a single instance of an application, e.g. a real-time multimedia flow between a particular source and destination, or it may be an aggregate of flows from a variety of sources traveling to multiple destinations.

A. Router Processing Model

In general, packets arrive on each interface of a router, undergo some processing, and are routed through an interconnect to an output interface,¹ where they await transmission on the outgoing link. In a router that supports enhanced services, the steps that need to be carried out include:

- Validation. The packet is checked for well-formedness, header checksum, nonzero TTL, etc.
- Classification. The packet is assigned to a class, usually based on the fields in its headers. Packets that receive plain old “best effort” service go into a default class.
- Forwarding lookup. The outgoing interface and next hop forwarding information is determined. If the packet belongs to a class for which routing is fixed (because resources have been reserved along a particular route), the forwarding information could be retrieved from the state; otherwise it is determined from the packet’s headers.
- Switching. The packet is transported to the outgoing interface port, typically via a switch fabric.
- State retrieval. If the packet belongs to a class that receives enhanced services, state information may be kept regarding resource usage.
- Policing. If the packet belongs to a class that receives enhanced services, the relevant state is checked to ensure that the class stays within its allotted usage specification.
- Scheduling. At the output interface, packets are selected for transmission from each class according to the scheduling algorithm. Again, this generally involves access to the per-class state.
- Transmission.

Some of these operations may occur in various orders, or in parallel. Figure 1 shows one possible arrangement of the operations. Here, header validation and forwarding lookup occur on the input side of the router (before switching), while classification, state retrieval, policing and scheduling occur on the output side. Each output interface stores state information for only those classes (flows) that use it.

Another possibility, shown in Figure 2 is to do classification on the input side. For classes that follow more or less fixed routes, the destination lookup is then replaced by retrieval of the next-hop/outputgoing interface information associated with that

class. This requires that per-class information be distributed across both input and output interfaces, but the information for each class only needs to be stored at the particular interface(s) on which packets of that class arrive.

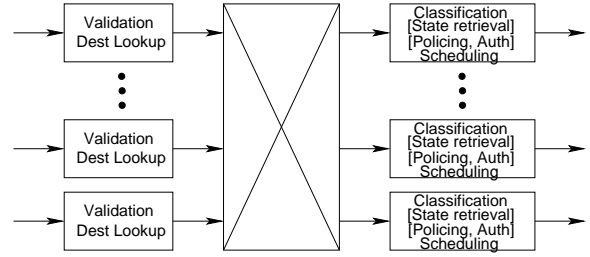


Fig. 1. Router processing context

Some means of establishing the necessary classifier and scheduling state must be provided for each class. This might be handled dynamically, via a signaling protocol such as RSVP [?] initiated by the end systems. Alternatively, it could be accomplished via an administrative interface accessible only to a “bandwidth broker” or administrator

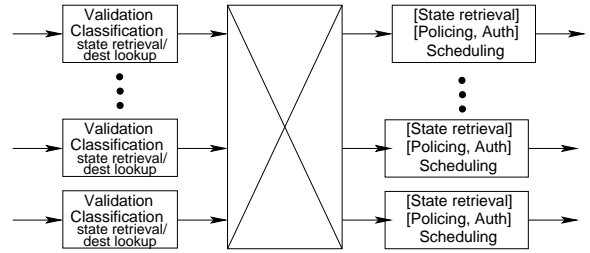


Fig. 2. Input-side classification

Output-side processing is shown in more detail in Figure 3. The “policing” function is applied only to packets belonging to classes that receive enhanced services; it ensures that the class does not exceed the limits of its allocation of resources, i.e. buffer space and transmission bandwidth. The state information for the class records its resource usage; if an arriving packet would cause the class to go over its limits, it is dropped by the policing function.

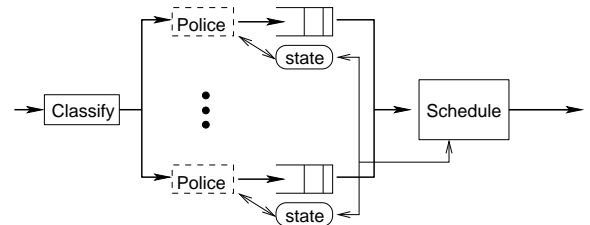


Fig. 3. Output processing

Note that an overlimit packet might simply be delayed (“shaped”) instead of being dropped, but if a class exceeds its limits by too much, packets *will* be dropped at some point. Note also that for some enhanced services—e.g. differentiated services [?]¹—the policing/shaping function is not necessarily ap-

¹Multicast packets are duplicated and sent to one or more output interfaces.

plied at every hop; instead, it is applied at “ingress” (and possibly egress) nodes of a domain. For simplicity, we assume the above model applies at every node along a particular path through the network.

We assume that the input-output interconnect is not a bottleneck, i.e. packets can be switched from input to output ports as fast as they arrive. If this assumption does not hold, denial of service is possible by saturating the switch fabric, causing packets to be dropped on the input side. We further assume that the other operations described above can be performed in essentially constant per-packet time,² and that output-side processing (except transmission) can be completed at a fast enough rate keep up with the maximum arrival rate of packets to that output port. If this requirement is not met, denial of service is possible by saturating the output port processing mechanisms.

Every traffic class has a maximum transmission rate, which is either explicit in the service definition or implicit in the outgoing link speed along the path. When packets arrive faster than they can be transmitted, they wait in the queue for the class. Each queue has limited capacity, either as specified in the service definition for the class, or as implied by the total amount of buffer space available in the router. When a packet arrives and there is no space in the queue for its class, it must be discarded. This is the basis for the denial-of-access threat: if an attacker can keep the queue mostly filled with bogus packets, legitimate packets will have to be discarded, and thus not receive the paid-for enhanced service.

The packets in a class follow a *path* through the network. In what follows, this path is assumed to be a linear sequence of routers that is the same for every packet in the class. (We discuss possible solutions when this assumption does not hold in Section IV-E.1.)

B. 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. Depending on the service agreement, this may result in lost revenue to the provider in one form or another.

Avenues available to the attacker depend to some extent on the security of the routers and channels that make up the path followed by packets through the network. For example, if the attacker can compromise a single router along the path, she can modify the service parameters of the class or remove its state information completely. It seems clear that prevention is the only recourse against this type of attack; it is difficult, if not impossible, to protect a service when the mechanisms used to implement it are under full control of the attacker. Because such a compromise could also be used to *enhance* the service received by the attacker’s packets (i.e. without paying for the enhancements), it seems reasonable to assume that service providers take strong measures to ensure that their routers are secure before offering premium services. Along similar lines, an adversary who can take complete control of some *channel* in the path can mount a “man-in-the-middle” attack, and again deny service directly

by dropping, corrupting, or merely delaying legitimate packets. Therefore it seems that protection against such cut-and-splice attacks must also be assumed.

However, there are other forms of compromise short of full control of some part of the path—most importantly, the ability to eavesdrop or *snoop* packets traveling along the path, and the ability to *inject* packets at some point along the path. Both of these can be accomplished by breaking into some trusted end system (non-router) connected to a shared-medium network somewhere along the path—say, an administrative system in a service provider’s network, or a host in the originating domain.

Techniques for flooding a particular link, router, or host are well known; they are the basis for denial-of-service attacks targeting end systems. By breaking into and taking control of a number of end systems (“zombies”) distributed around the network, and then causing them to transmit bursts of packets addressed to the target simultaneously, an attacker can cause packets to arrive at almost any router’s output port at a rate exceeding the capacity of the output channel. Indeed, most of the DoS attacks observed in the Internet in recent years have been of this type [?]. Thus it may not even be necessary to compromise any portion of the path in order to insert packets into it.

Our assumptions about the attacker’s capabilities can be summarized as follows:

- The attacker can insert bogus packets into the path, possibly from many different sources.
- The attacker can eavesdrop at some point along the path.
- The attacker *cannot* prevent, delay, or otherwise interfere with the transmission of *legitimate* traffic along the path. In particular, the attacker cannot transmit a modified copy of a legitimate packet and have it arrive anywhere on the path *before* the original.

Although it is conceivable that the last assumption above might be violated, any violation would require that the attacker have access to routes with lower delays than those used to provide enhanced services to the attacked class, which seems rather unlikely.

To mount a denial-of-service attack, an attacker needs to arrange for bogus packets, which can pass for legitimate members of the class, to arrive at the output port of some router on the path at a sufficiently high rate that the class exceeds the limits of its reserved resources, thereby degrading the service received by legitimate packets of the class. Note that it may not require an especially high rate of bogus packet insertion to accomplish this. On the other hand, a low, but nonzero, rate of bogus packet insertion may be acceptable: service specifications generally include a certain amount of “slack”; in addition, the legitimate traffic may underutilize the service.

III. AUTHENTICATED PACKET CLASSIFICATION

We propose to augment the packet classification step with an authentication check: packets are admitted to a class that receives enhanced services only after their authenticity is verified based on information carried in the packet plus information stored at the router. Packets that fail the authentication check are discarded. In this section we consider the architectural implications of such a mechanism, including where the check might be applied in router processing and how the verification code is car-

²While classification and state retrieval may require time proportional to the number of classes supported, that number changes relatively slowly, so for our purposes it is reasonable to assume those operations are constant time.

ried in a packet. Then we discuss requirements for security and performance. Finally, we consider some alternative solutions.

A. Architectural Considerations

Adding authentication makes packet classification a two-step, sequential process: first, the packet is assigned to a class in the usual way (viz., header matching). Based on that assignment, the verification information for the class is retrieved, and is used together with the code from the packet to verify authenticity. Authentication thus requires the addition of some new information to the per-class state. The form of this information depends on the authentication method, but in some cases it must be kept secret.

If classification is performed on the output side (Figure 1), the verification information can be stored with the scheduling state, and there is no additional cost to retrieve it, beyond that of the check itself. Input-side classification, on the other hand, requires that every input interface store the verification information for *every* class; this is an additional cost versus unauthenticated classification as discussed above (Figure 2), where each input interface only needs the information for classes that enter via that interface. (On the other hand, this is attractive from a performance standpoint, as discussed below.)

The authentication information needs to be placed in the packet where it can be located easily by any router that enforces resource reservations. For example, it might be included as an IPv4 option. It might also be carried in a “shim header”, as with IPSEC protocols.

In general, the authentication information will be added by the originator of a packet. In the case of more than one source for packets in a class, it may be more convenient to add the information at a “bump in the wire,” i.e. a common intermediary through which all packets of the traffic class pass before entering the Internet [?]. (See also Section IV-E.1.)

B. Security Requirements

The authentication mechanism must have certain security properties in order to be considered a solution to our problem. These follow from the assumptions given above, and include:

- **Cryptographic strength.** It must be infeasible for an attacker to consistently forge packets without breaking some believed-to-be-strong cryptographic primitive(s). Any such primitive(s) should be used in the strongest possible way. Brute-force methods must yield a sufficiently low probability of success that the service delivered to legitimate packets is unaffected.
- **Stateless verification.** Authenticity should be determined solely from the packet contents and static per-class information (e.g. public key or shared secret), and not via any state that changes per-packet.
- **Replay prevention.** The mechanism must include a means to detect re-use of codes. (This implies that state must be updated on a per-packet basis, but does not violate the previous requirement, provided the verification and anti-replay checks are separate, and update happens only *after* verification of authenticity.) Given a fixed field size in the packet, this requirement implies that the number of packets that can be verified is bounded; however, it should be large, so that resynchronization is required infrequently, even for high-data-rate classes.

- **Limited damage.** Given the stateless verification requirement, a successful forgery can only cause legitimate packets to fail the anti-replay check. The number of legitimate packets that can be excluded in this way must be small; if the attacker gets lucky, the results should not be catastrophic.

- **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.

C. Performance Requirements

Our model of router processing implies some performance constraints on the authentication check. We assume that authentication can be pipelined with other operations, and consider only requirements that follow from the basic architecture of the router. For simplicity, all incoming and outgoing links are assumed to operate at the same rate, ρ bytes/s. Let α 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/ρ is the time between packet arrivals on each link.

If classification/authentication is performed on the output side, and packets arrive continuously on each interface, it must be possible to authenticate at least α packets every S/ρ seconds. If authentication is performed on the input side, however, only one packet needs to be authenticated in that time. In the sequel, we consider the more stringent of these constraints.

We can model the time required to authenticate a packet as a simple linear function of packet length, $\delta + \nu S$, with fixed overhead δ and per-byte time ν . Thus, we have the constraints:

$$\begin{aligned}\delta &< S/\rho\alpha - S\nu \\ \nu &< 1/\rho\alpha - \delta/S\end{aligned}$$

It is clear that these bounds are tightest when the packet length S is minimized. Thus the performance requirement for authentication—like other packet processing components in routers—is constrained most strongly by the maximum *packet* arrival rate and not the maximum *byte* arrival rate.

One way to minimize the per-packet cost is to set ν to zero, thereby making the authentication check independent of the length (and therefore the content) of the packet. For conventional end-to-end MACs, this is not possible, because part of their function is to protect the integrity of the whole packet. We therefore propose an authentication code designed specifically for this purpose. Before doing so, however, we consider some alternative solutions.

D. Alternative Solutions

- **Public-key signatures.** 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.

- **Ingress filtering.** Packets that arrive at a 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 cannot be used under certain conditions that are not at all unusual, including asymmetric routes. Second, they must be applied throughout the network to be effective. If it is possible to reach any router on the protected path via a path that does not contain reverse-path checks, then spoofed packets can be injected 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.

- **Hop-by-hop authentication.** Hop-by-hop authentication has the advantage that it does not require distribution of any shared secret to routers along a path. However, as with reverse path checks, 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.

- **Conventional end-to-end MAC.** Conventional MACs, such as HMAC [?] or UMAC [?] provide high security and protection against tampering, but have relatively high costs for minimum-sized packets, and therefore open the possibility of an attack based on saturating the verification mechanism itself. Moreover, conventional MACs do not protect against replay, and are therefore not sufficient in themselves as a solution to our problem.

- **Traceback.** Recently various methods have been proposed for *tracing* denial-of-service attacks to their source by reconstructing the paths followed by packets. Some of these require storing per-packet information at routers [?], while others place the information in the packets themselves [?]. These methods may be useful for reacting to an attack, although they represent only the first step in a long and difficult process required to identify and punish an attacker. In any case, they are fundamentally *reactive*—they cannot prevent an attack from affecting the enhanced services for which customers have paid. Before such services can be offered for profit, a provider needs to be reasonably confident that it can provide them consistently. Traceback mechanisms, important though they may be, do little to provide this assurance.

IV. FIXED-COST PACKET AUTHENTICATION

In this section we present FPAC, an authentication code intended for use in authorizing access to reserved resources. FPAC is unlike traditional MACs in that it does not protect against tampering. It also has a higher probability of forgery than most other MACs—but still low enough to serve its purpose. It must be emphasized here that FPAC achieves high performance and fixed cost by eliminating some security guarantees that—as we have argued above—are superfluous in the attack scenarios we consider to be most likely against enhanced services. It is thus *not* intended to be a replacement for conventional MAC algorithms; rather, it is intended to be used hop-by-hop to reduce the likelihood of forged packets obtaining access to reserved resources. To emphasize this distinction, we refer to conventional MAC checks like HMAC [?] 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 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. Some secure means of distributing the required secrets (the block cipher key and a key for a random function) to the routers along the path is required. This is a nontrivial requirement that we discuss briefly at the end of this section, but a general solution is beyond the scope of this paper.

A. Overview

FPAC is based on the standard technique of encrypting a bit-string having a particular property known to the sender and the verifier, using a symmetric cipher. The sender chooses a random nonce X , and uses it to construct an *authenticator* V having the property. The sender then computes

$$E_{k_0}[V|R]$$

and places X together with the resulting ciphertext block in the packet. Here k_0 is a secret key known only to the sender and the verifier, $E_k[\cdot]$ is encryption with a block cipher under the key k , R is the *anti-replay* token (whose construction is described below) and ‘|’ is concatenation of bit strings. The verifier decrypts the ciphertext and checks that the required property of X and V holds. If so, the verifier then checks that the anti-replay token R has not been seen before; if not, the packet is accepted and R is recorded in the anti-replay state. Thus the cost of verification of FPAC is one block decryption, plus the cost of checking that the desired relationship holds between X and V .

Security of FPAC rests on the block cipher—it must not leak information about the secret k_0 , and it should prevent an attacker from forging a valid authenticator without knowing k_0 . To make it more difficult for an attacker to obtain information about the block cipher key k , the particular property of X and V is also secret; this is discussed further below.

Given a strong cipher and no information about k_0 , 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, anti-replay tokens cannot be re-used, so the size of R determines the number of packets that can be sent before resynchronization/rekeying is necessary. Thus, 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 R implies that fewer packets can be sent before resynchronization. In our implementations with 64-bit block ciphers, we have used two different splits: a 24-bit V and 40-bit R , 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.

We want to use the block cipher $E_{k_0}[\cdot]$ in the strongest possible way—in particular, both V and R should be constructed so that their values are unpredictable to an attacker. This minimizes the amount of known plaintext available to the attacker. To achieve this the authenticator V should be constructed using a secret k_1 , chosen from a large space, in addition to X . For example, V might be a keyed hash of X , or the encryption of X under some simple substitution cipher. The point is that

given X , the attacker should not be able to determine whether any particular V (there might be more than one) has the required property, without knowing k_1 . Note that the overall security still rests on the security of the block cipher. (The techniques used to make R unpredictable are described below in Section IV-D.)

We now describe two versions of FPAC, which differ in the construction of V . In the first, which we call “FPAC unattached” (FPAC_u), V depends solely on X and the secret k_1 . In the second, which we call “FPAC attached” (FPAC_a), the contents of the packet itself are used (in addition to X and k_1) in constructing V . This binds the authenticator to the packet contents, but in a way that does not depend on the length of the packet and provides essentially no protection against tampering. Both algorithms are described in terms of a 64-bit block cipher, such as DES, Blowfish or Khufu [?]. Other block sizes could be used as well.

B. FPAC Unattached

In FPAC unattached the authenticator V is computed as $V = f_{k_1}(X)$, where f is a random function keyed by k_1 that need not be invertible. The purpose of f is to make V unpredictable. One possibility is to let f be chosen (via k_1) from a family of *universal hash functions* [?]. These functions are ideal for construction of V because they are very inexpensive to compute and they produce outputs of about the right size.

In our implementations we have used $V = e_{k_1}[X]$, where $e[\cdot]$ is a simple 8-bit substitution cipher. Strings longer than eight bits are encrypted 8 bits at a time. (Note that this transformation is significantly weaker than a universal hash; for example, if two 8-bit blocks of X happen to be equal, that equality is evident in V .) We describe the algorithm for a 32-bit V , using this substitution cipher as the inner function f .

The sender performs the following steps:

1. Choose a random 32-bit nonce X .
2. Compute $V = e_{k_1}[X]$.
3. Construct R in the manner described below.
4. Place $(X|E_{k_0}[V|R])$ in the packet as the FPAC code.

The verifier performs the following steps.

1. Parse the packet to get X and Z .
 2. Verify that $e_{k_1}[X]$ is equal to the first 32 bits of $D_{k_0}[Z]$.
- Note that the verifier’s computations of $e_{k_1}[X]$ and $D_{k_0}[Z]$ can be carried out in parallel.

C. FPAC Attached

The “attached” version of FPAC uses the packet contents in computing V , and thus provides some very modest protection against using the code with a completely different packet. It does *not* provide any significant protection against tampering. The basic idea is that X and k_1 are used to compute random offsets into the packet, and the bytes at those offsets are included in constructing V .

Note: Although we believe that FPAC_u is adequate for most purposes, FPAC_a is included here for completeness, and as an example of an authentication code that—in some sense—binds to the contents of a packet, but does not require reading every byte of the packet.

We need two random functions for FPAC_a, keyed by secrets k_1 and k_2 . (Again, a universal hash function such as NH [?] is a

good choice for such a function.) Let $f_{k_2}(\cdot)$ be such a function with a 64-bit output. For the second function, again let $e_{k_1}[\cdot]$ be an 8-bit substitution cipher as described for FPAC_u.³ Let l be the actual length of the packet in bytes, excluding the length of the FPAC information and any fields that change in an unpredictable way as the packet travels through the network. Let p_i denote the i th byte of the packet, numbered from 0, and excluding the bytes of any fields whose length is excluded from l . The maximum packet size is assumed to be 65,535 bytes.

The sender performs the following steps:

1. Choose a random nonce X .
2. Compute $Y = f_{k_2}(X|l)$, and split it into four 16-bit blocks a_0, \dots, a_3 , where a_i is bits $16i$ through $16i + 15$ of Y .
3. Compute $V = e_{k_1}[p_{a_0 \bmod l}] \dots | e_{k_1}[p_{a_3 \bmod l}]$.
4. Compute R as described in the next section.
5. Place $X|E_{k_0}[V|R]$ in the packet as the FPAC.

The verifier performs the following steps:

1. Parse the packet to get X and Z .
2. Perform steps 2 and 3 of the sender computation, and verify that the result is equal to the first 32 bits of $D_{k_0}[Z]$.

By including the packet length in the computation of Y in step 2, changes in the length of the packet should result in a completely different set of offsets. However, this does not necessarily imply a different result for V . For example, if all bytes of the packet are equal, the value of V is independent of the value Y computed in step 2. Thus the strength of the binding between the packet and the FPAC_a code depends on the contents of the packet itself. Higher-entropy packet contents provides a higher likelihood of detection of packet modifications.

D. Protection Against Replay

Protection against replayed packets is handled through the use of sequence numbers, as in the Encapsulating Security Payload (ESP) of IPsec [?]: 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 number 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 R . In this description all arithmetic is understood to be modulo 2^w , where w is the size of R .

³A universal hash function would work fine here as well.

The sender maintains a *send window* of sequence numbers in the range L_s to $L_s + T - 1$ (inclusive); initially L_s is set to the ISN. Each verifier maintains a window of acceptable sequence numbers, defined by the range L_r to $L_r + W - 1$ (inclusive), where $T < W$. In addition, each verifier knows the value of T . Initially $L_s = L_r$; this initial sequence number (ISN) is chosen randomly and kept secret. For each packet, 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 R . 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 window must not be advanced if doing so would cause the ISN 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.)

The above policy for advancing the window at the verifier is not sufficient if losses can occur—the window would never advance past the sequence number carried by any lost packet. Therefore the verifier must also be prepared to advance the window when it sees evidence that the send 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 $j - T + 1$ (because $j < L_s + T$) without passing L_s . 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 should 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 that can be left to the implementor. W needs to be enough bigger than T to accommodate a case where the sender advances the window in a large jump; $W \geq 2T$ should suffice. A large value of T is desirable to maximize the entropy in the sequence number; 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. (See also the discussion below on detection of attacks.)

Finally, consider the construction of R . If we simply let R be the sequence number, the high-order bits of R will contain very little entropy. On the other hand, if the sender chooses sequence

numbers carefully the low-order bits of R 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 R .

Let the sequence number be N , and divide N into bytes: assuming a 32-bit sequence number, 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 $i = 1, 2, 3$ let $n'_i = n_i \oplus n'_{i-1}$. Then set $R = e_k(n'_3) | \dots | e_k(n'_0)$, where $e_k(\cdot)$ is an 8-bit substitution cipher known only to the sender and verifier.

The verifier similarly divides R into r_0, \dots, r_3 , and applies the inverse substitution to get $n'_i = d_k(r_i)$, $i = 0, \dots, 3$. Then the receiver sets $n_0 = n'_0$, and $n_i = n'_i \oplus n_{i-1}$ for $i = 1, 2, 3$. The verifier uses $N = r'_3 | r'_2 | r'_1 | r_0$ as the sequence number.

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.

E. Discussion

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 preclude linear and differential cryptanalysis, 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 being accepted is about $2^{-B}W$, 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 forgery is about 2^{-53} . This is sufficiently low to ensure that forged packets do not steal a significant amount of reserved resources from a class.

A number of practical issues remain, and we consider each briefly in the remainder of this section.

E.1 Nonlinear Paths and Multiple Sources

When the packets belonging to a class follow different paths through the network—such as when multiple sources send packets to multiple destinations using the same differentiated services 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. Even with coordination, different sets of sequence numbers will be presented at different routers; this may not prevent legitimate packets from getting through, but it does open the possibility of degrading service by replaying packets snooped from one path into a different path. It is therefore desirable to explicitly divide the sequence number space into subspaces, so that each path has its own subspace and a verifier can easily determine which subspace to check against.

One possible solution is to partition the anti-replay sequence number space among source-destination (or ingress-egress) pairs, using some bits of R as 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 maintains a separate, independent window of acceptable sequence numbers for each tag that it sees. The same ISN 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, perhaps at the cost of a more expensive verification step. 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.

If the number of sources is large, another possibility is to add the FPAC code at the ingress router of the differentiated services domain, via a so-called *bump in the wire*, as suggested for IPsec [?]. Note, however, that this opens the possibility of attacks from the originating domain.

E.2 Secret Distribution

Use of FPAC requires that each router that authenticates classification share secret information (keys) with the originator(s) and other verifiers of packets belonging to that class. Some means is required to distribute this secret information to routers along the path while maintaining confidentiality. This is a non-trivial problem whose difficulty should not be underestimated: key distribution protocols are an active research topic, especially for group services [?], [?]. Although RSVP supports secure establishment of resource reservations, it is not designed to provide confidentiality. Either RSVP would need to be extended to also provide for key distribution, or a new protocol would need to be developed.

E.3 Detecting Attacks

When a denial-of-service attack is under way, routers using FPAC to authenticate packets should experience a high rate of rejection of packets. Depending on the nature of the failure (authentication or anti-replay), the type of attack can be determined, and appropriate remedial action (e.g. traceback) can be started. Thus per-packet authentication can be used in conjunction with other tools [?], [?].

V. IMPLEMENTATION AND EVALUATION

We added authentication to the *traffic control* facility of the Linux kernel (version 2.2.14). Traffic control provides queuing disciplines, classes within queuing disciplines, classification filters, and policing. Each network device has an associated queuing 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 specified manually in our implementation; anti-replay is a configurable option. The MAC algorithm and secrets 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.

TABLE I
AVERAGE VERIFICATION COSTS FOR 1024-BYTE PACKETS

Algorithm	Cycles/Packet
HMAC-MD5	20951
FPAC _a (Khufu-8)	1145
FPAC _a (Blowfish-16)	2162
FPAC _u (Khufu-8)	999
FPAC _u (Blowfish-16)	2056
FPAC _u (Khufu-8) with anti-replay	1521
FPAC _u (Blowfish-16) with anti-replay	2520

At the source node, the FPAC code is added by the kernel; 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_a, in two versions based on 16-round Blowfish and 8-round Khufu [?] as block ciphers, respectively
- FPAC_u, with 8-round Khufu as the block cipher

Publicly-available versions of Blowfish, Khufu, and MD5, with minor hand-tuning, were used. It must be noted that as freely available code, the performance of these implementations should not be interpreted as any kind of limit on what can be achieved with these algorithms. Rather, our purpose here is to illustrate the general differences between approaches and the costs of 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 various 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 II. For each algorithm, we measured the cycles required for each of 1000 packet of 1024 bytes each; the numbers presented in Table I 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_u 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 is about an order of magnitude higher than the slowest version of FPAC (HMAC does not include anti-replay).

We also measured the effect of packet size on performance. Table II shows the average per-packet verification cost, with-

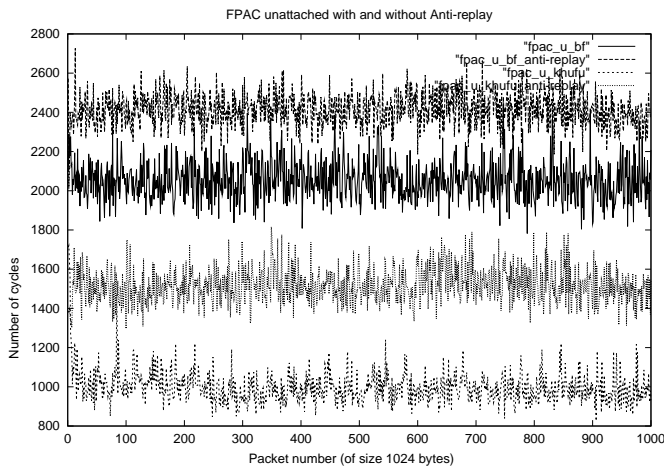


Fig. 4. Cycle count per packet

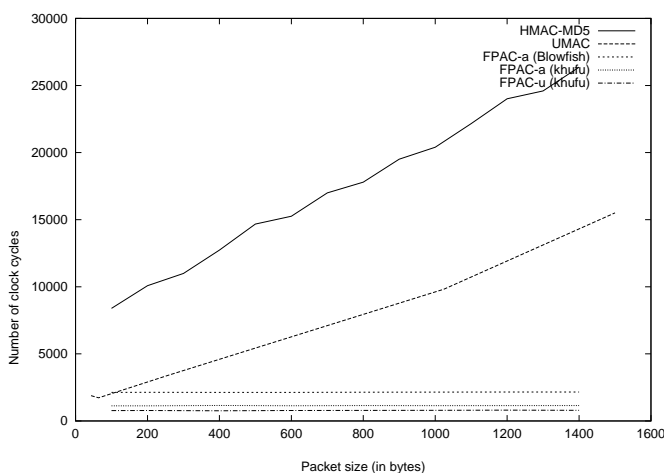


Fig. 5. Verification cost vs. packet size

out anti-replay, for HMAC-MD5, $FPAC_a$ (Blowfish), $FPAC_a$ (Khufu), and $FPAC_u$ (Khufu) for 100-byte packets.

TABLE II

AVERAGE VERIFICATION COSTS FOR 100-BYTE PACKETS

Algorithm	Cycles/Packet
HMAC-MD5	8391
$FPAC_a$ (Khufu-8)	1124
$FPAC_a$ (Blowfish-16)	2130

Figure 5 shows that the per-packet cost of FPAC is independent of the packet size. We measured the cycle count for packets of sizes from 100 bytes to 1400 bytes in intervals of 200 packets. Each point is the average of 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 PFAC 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-to-end MAC designed specifically for extreme speed in software implementations [?]. 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 for the smallest packet sizes.

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 con-

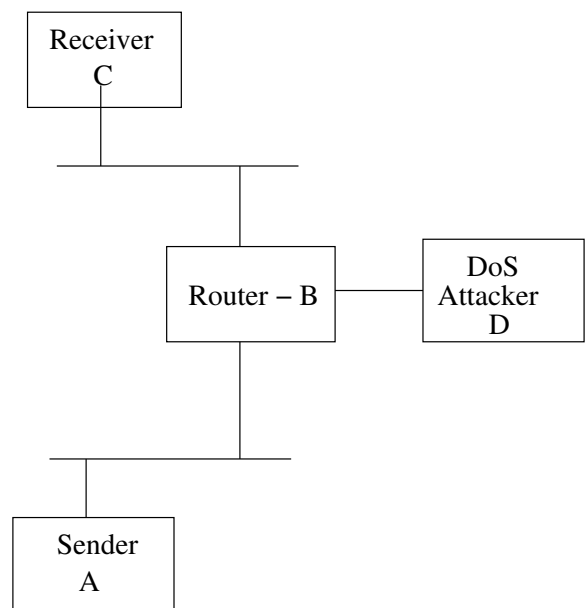


Fig. 6. Test setup for measurements

nected 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. 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 show data arriving at C at the reserved rate. The fourth curve shows that when the DoS stream is present and FPAC is not used, the rate of arrival of legitimate packets at C is significantly reduced. This is shown by the increase in time required to deliver a number of packets to C.

VI. RELATED WORK

We are unaware of any existing detailed analysis of the problem of preventing denial of access attacks against reserved resources. The proposed standards for Integrated Services in the

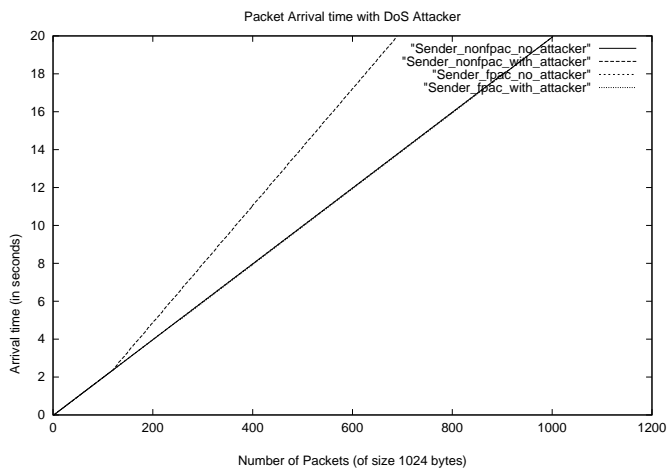


Fig. 7. Packet arrivals vs. time

Internet [?], [?], [?], [?] 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.

The high computational cost of conventional end-to-end MAC methods based on cryptographic hash functions is a well-known problem [?]. Additional evidence of this is provided by the competition to design the fastest hash function. UMAC [?] 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 PFAC rest on the same properties of an underlying block cipher. Schneier [?] 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.

VII. 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 per-packet 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 per-packet 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 potential speed advantage is even greater when the authentication check is implemented in hardware. Because it requires just one block cipher operation per packet, it should be possible to implement FPAC easily in hardware.