FANFARE for the Common Flow

Elaine Shi **Bryan Parno** **Adrian Perrig**
Yih-Chun Hu **Bruce Maggs**

February 2005
Updated June 2005
CMU-CS-05-148

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This paper presents FANFARE[1], a suite of infrastructure-based primitives that empowers routers and receivers to secure and enforce various flow-control mechanisms, such as per-flow admission control, service differentiation, and congestion control, even in the face of sophisticated attackers. In FANFARE, a sender must receive capabilities from both a receiver and forwarding routers in order to acquire a certain bandwidth allocation, thus empowering both receivers and routers to control the rates of flows. FANFARE provides strong incremental deployment properties; in particular, FANFARE's automatic congestion response mechanism can protect a downstream legacy link from being flooded by FANFARE traffic. In FANFARE, routers use no per-flow state; they only need to rely on local information to make decisions, and hence do not have to trust other routers. FANFARE can be used to secure several known architectures for managing flows. In this paper, for example, we show how to use FANFARE to halt DDoS attacks and to secure a Diffserv infrastructure.

Keywords: Denial-of-Service, network infrastructure, capability, flow

1 Introduction

The Internet today relies on cooperative congestion control protocols to ensure that resources are allocated fairly between competing flows [33]. These protocols provide fair sharing, however, only when end hosts behave in a protocol-compliant manner. A single malicious host can send packets at an arbitrarily high rate to any receiver, potentially jeopardizing inter-flow fairness or overloading network links on the path to the receiver. A more sophisticated attacker may disguise the malicious host through the use of IP spoofing. Even more capable attacks might involve colluding sender/receiver pairs or leverage compromised routers. On a larger scale, attackers may employ zombies in Distributed Denial-of-Service (DDoS) attacks.

Several lines of work in the literature address these attacks. For example, networking researchers have promoted infrastructure-based flow-control mechanisms, where routers actively throttle "misbehaving" flows, i.e., those that use a disproportionate share of the bandwidth. Infrastructure-based flow control can also support useful features such as service differentiation [6] and service guarantees [7]. Unfortunately, much of the existing work in this category identifies flows based on their source and destination address, so that an attacker can spoof the IP address of a trusted high-priority host and thereby acquire the bandwidth necessary to facilitate its attack. Several of these mechanisms also rely on cooperation between routers [6, 40, 41], which introduces the possibility that a single compromised router can have a global impact.

Another approach focuses on protecting a receiver's network from being flooded. In practice, this is sometimes accomplished by requesting that an ISP upstream from the receiver manually install a filter to block attack traffic. Installing such filters, however, is cumbersome, and does not work well against large-scale DDoS attacks. Alternatively, various proposed protocols [1, 3, 26, 28, 45, 46] allow a receiver to prevent traffic from being sent towards it by handing out *capabilities* to legitimate senders. Other researchers propose IP pushback schemes [11, 18, 30, 31], where the receiver asks upstream routers to filter aggregates of attack packets. Most of these schemes rely on the receiver not only to protect itself, but also to act as a gatekeeper to prevent a malicious sender from flooding the links between the sender and the receiver. However, a malicious receiver may collude with the sender to allow it to use high-priority traffic to flood the intermediate network links. In particular, all of the previous schemes described above fail to prevent a pair of malicious endhosts from flooding the network path between them if the bottleneck link spans legacy routers.

Regardless of the specific approach, a fundamental difficulty in addressing these attacks is that current Internet standards require both routers and recipients to make decisions based on unsecured information, since almost all the fields in a packet may be forged by a sender or altered by a malicious router.

This paper presents FANFARE, a suite of primitives with which mechanisms for regulating flows can be secured and enforced. FANFARE is a capability-based scheme, like [1, 3, 26, 28, 45, 46]. Unlike previous schemes, however, FANFARE has all of the following properties:

- FANFARE participants make only local decisions. The receiver and the routers all operate independently and are not required to establish any trust relationships. Thus, none of the parties need to establish shared keys or exchange public keys. This approach greatly reduces vulnerability to router compromise, since a malicious router can only influence links directly under its control.
- FANFARE is incrementally deployable. It accommodates legacy routers, and does not prevent communication between legacy/upgraded clients and legacy/upgraded servers.
- FANFARE will not issue a capability to a flow if the requested capacity is not available on the entire path from the sender to the receiver, even if the bottleneck is a link between legacy routers. Furthermore, FANFARE routers can protect legacy links from malicious sender/receiver pairs and compromised routers, even if those pairs and routers are upgraded. Hence, FANFARE embodies the "do no harm" principle: upgrading senders and receivers does not enable new attacks on the legacy infrastructure.
- FANFARE maintains no per-flow state at the routers. FANFARE also requires only lightweight processing at the routers.
- FANFARE reacts very quickly to misbehaving flows and congestion, yet accomplishes this with relatively long-lived capabilities, which provide resilience to bursty loss.
- FANFARE can be used to secure existing protocols and architectures such as TCP [33], Diffserv [6], and CSFQ [40].

1.1 Secure Flow Architectures

FANFARE is a *secure flow architecture*. A secure flow architecture provides a framework for allowing routers and receivers to make nuanced decisions about the traffic they support. The architecture enforces these decisions on a per-flow basis even in the presence of strong adversaries. Ideally, a secure flow architecture should provide the following properties:

Secure flow identification. Routers and recipients should be able to securely identify a network flow and its associated parameters. The parameters may include the flow's current level of service, as well as the amount of time it has been active. Securing these identifiers allows routers and recipients to make sophisticated decisions about the flow. For example, routers and recipients can impose time limits on policy decisions or craft policies that gradually upgrade a flow's level of service based on its current level as well as its age.

FANFARE allows secure flow identification through a capability system. A *capability* is a marking embedded in a packet's header. The marking associates a packet with its flow identification in a way that is resilient to IP spoofing; the marking also ties a packet to its service parameters.

Secure admission control. Because routers and the network have finite resources, they cannot grant reservation requests for an arbitrary number of flows. Routers and receivers must therefore be able to decide which flows to admit, and what level of bandwidth or router resources to assign to each flow. This decision process is called admission control. Admission control consists of two components, an admission control algorithm and enforcement mechanism. An admission control algorithm takes flow reservation requests as input and outputs a set of flows to accept and a level of service to assign to each accepted flow.

Enforcement of admission control can be enabled through secure flow identification: routers must be able securely identify a flow's assigned service level whenever an admission control decision is made.

In the literature, researchers have proposed several algorithms for performing admission control [20, 17]. FANFARE does not dictate which specific admission control algorithm should be used. Instead, by building on the capability system as described above, FANFARE provides a granting and enforcement mechanism that accommodates different admission control algorithms. FANFARE allows routers and a receiver to accept a flow by jointly issuing capabilities to the sender. Routers and receivers can halt an accepted flow at any time by ending the provision of capabilities. Routers and a receiver can also upgrade or downgrade a flow's service level by changing the service level parameters in the capabilities issued.

Secure flow regulation. While admission control assigns service parameters to each flow, flow regulation involves scheduling and monitoring flows to ensure they obtain their level of guaranteed service. Secure flow regulation must also ensure that malicious flows do not use more than their allotted share of bandwidth. Furthermore, malicious hosts should not be allowed to continue sending packets if the recipient or a router along the path decides to halt the flow. Researchers have proposed various infrastructure-based mechanisms to rate limit flows [11, 29, 42]. It is beyond the scope of this paper to propose a new flow regulation algorithm. Instead FANFARE assumes the existence of such a flow regulation component on the routers and provides the primitives necessary to secure it.

Secure congestion control. Most current congestion control mechanisms rely on endhosts to faithfully back off when congestion occurs, but a malicious sender can violate the protocol to gain an unfair advantage over legitimate traffic. A secure flow architecture should enforce proper behavior in the face of congested links, even when malicious parties attempt to collude to flood an intervening link.

FANFARE provides a mechanism for enforcing congestion control. In FANFARE, a sender's probability of receiving a capability is inversely proportional to the congestion level between the communicating end-hosts, and the amount of bandwidth resource the sender is requesting. Therefore, when a link between two end-hosts is congested, the sender is unable to maintain a high bandwidth reservation and has to reduce its sending rate, even if the congestion occurs on a downstream link between two legacy routers.

1.2 Assumptions

In this paper, we assume a strong adversary model in which both endhosts and network routers may be compromised. After compromise, they may collude with each other. As two examples of potential misbehavior, we consider adversaries that may be selfish or malicious. A selfish adversary attempts to use an unfair share of the network bandwidth. For example, the attacker may send more traffic than permitted by its priority class or refuse to back off when congestion occurs. A malicious adversary attempts to perform a DoS attack by flooding one or more parts of the network. The attack may take place using a distributed set of compromised nodes (i.e., a DDoS attack). We also assume that compromised hosts can spoof the IP addresses of legitimate clients.

FANFARE information is transmitted in the headers of existing packets. Hence we assume that we have sufficient marking space in the IP packet. Our scheme requires space for a packet's priority class, two capabilities, a handful of flags and an optional field for additional information. We suggest a method for including these fields while maintaining IPv4 compatibility in Section 7.3. Since we piggyback our information on the existing traffic between a sender and receiver, FANFARE requires no additional packets.

We also assume short-term route stability between hosts for the time it takes to conduct a packet exchange. Short-term instability will not break our system, but will degrade performance. Finally, we assume that flows are bidirectional and send packets at regular rates. If communication is unidirectional, we assume that the receiver can send packets to the sender. We assume that endhosts deploy FANFARE; however, if either endpoint has not been upgraded, they can still communicate by falling back to legacy protocols.

1.3 Outline

The remainder of this paper is organized as follows. Section 2 describes the FANFARE protocol. Section 3 then provides guidelines for configuring FANFARE. An analysis of various attacks and FANFARE's ability to defend against them comes in Section 4. Next, Section 5 presents applications of FANFARE. An experimental evaluation of FANFARE appears in Section 6. We conclude with a discussion in Section 7, an explanation of related work in Section 8, and a summary in Section 9.

2 FANFARE: Flow-Aware Network Framework for Attack Resistance through Enforcement

The FANFARE framework provides two basic primitives for securely regulating network flows. The first primitive provides cryptographically secure path-dependent markings that are created by the routers between a sender and a receiver. These markings are combined to create capabilities that provide secure flow identification, prevent IP spoofing, and allow receivers and routers to enforce admission control decisions. The second primitive builds on the first and uses secret sharing to give routers a secure, automated congestion response system. With the second primitive, FANFARE can continue to prevent malicious behavior even with downstream bottleneck legacy links. In a full deployment of FANFARE routers, the first primitive would suffice to avoid congestion, but by providing the second primitive, we enable incremental deployment, since it allows FANFARE to guarantee strong properties even in partial-deployment environments.

We now introduce the two FANFARE primitives. A complete listing of the terminology used in this paper is provided in the appendix.

2.1 Full FANFARE Deployment

Overview. We delineate a hierarchy of priority classes for network traffic based on the number of packets allowed during a given time period. Traffic from any priority class takes precedence over legacy traffic. A sender can only send traffic using a given priority class if the receiver and all of the routers on the path have agreed to provide the corresponding level of service. We enforce these restrictions through the use of capabilities. To construct a capability, the sender sends an initialization request (INIT-REQ) packet to the receiver using non-prioritized (best-effort) traffic. Each router along the way inserts independently selected markings into the packet, and the receiver returns the aggregate of the markings (which together form a capability) to the sender. The sender then includes the capability in subsequent prioritized data packets.

When a router receives a prioritized data packet, it checks the capability by verifying that the marking it would normally insert is equal to the marking present in the packet. If the router notices a discrepancy or if the capability has expired, the router drops the packet. Thus, if any router along the path declines to include a marking, or if the receiver refuses to return the accumulated capability, then the sender's traffic will be quickly squelched, since it will not have the correct capability.

Figure 1 illustrates the fields in a FANFARE-enabled packet. A router validates a packet's current priority class by checking the Cap_{cur} fields; meanwhile, it inserts new markings into the Cap_{next} fields.

We now specify the procedure for initializing FANFARE traffic, increasing the priority of a sender's traffic, and maintaining a priority once it has been obtained.

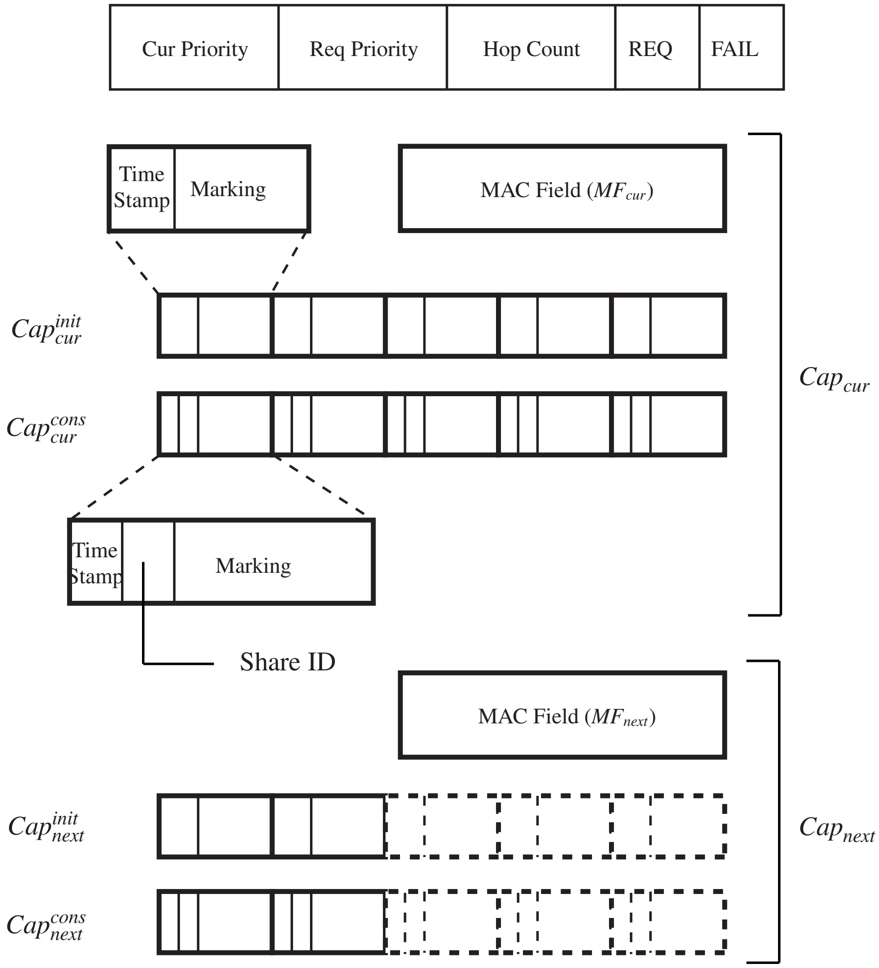


Figure 1: **FANFARE Packet Header** *The FANFARE header contains the sender's current priority class, as well as space for a requested class. The hop count records the number of routers along the path from the sender to the receiver. A one-bit flag (REQ) indicates whether the packet is part of an initialization round (an INIT-REQ packet) or part of a consolidation round (a CONS-REQ packet). Another flag (FAIL) indicates whether the current request has failed. We also use two MAC Fields to make router markings interdependent (see Section 2.3). The bulk of the header is composed of two capability sections, a* Cap_{cur} *and a* Cap_{next}. *Each capability section contains an Initialization Field,* Cap^{init}, *and a Consolidation Field,* Cap^{cons}. *The Initialization field consists of a pair of entries for each router along the path, one entry for a timestamp and the other for the router's marking. The consolidation field is constructed similarly, but each router's contribution also includes a share ID.*

Initialization. A sender initializes a FANFARE flow by using non-prioritized traffic to send an INIT-REQ packet to the receiver. In the INIT-REQ packet, the sender sets the current priority to 0 and fills in the requested priority field. Both capability fields (Cap_{cur}^{init} and Cap_{next}^{init}) are initially empty. When a router receives an INIT-REQ packet, it first makes an admission control decision. This decision must be based on the router's current capacity, but otherwise it may be dictated by the router's administrator. If the router decides not to admit the flow, it erases any markings in the Cap_{next}^{init} field that may have been inserted by previous routers, sets the failure bit in the header,[1] and forwards the packet along.[2] This prevents the sender from assembling a correct capability and effectively denies the sender the ability to send prioritized traffic. The router may optionally update the requested priority field to indicate the highest priority class it is willing to provide, so that the sender can make an informed request in its

[1]The failure bit alerts subsequent routers that this request has already failed, so they may also reject it without performing potentially expensive admission control decisions. While a malicious router might set the failure bit on legitimate packets, it could simply drop the packets, so we choose to include the bit in favor of the increased performance it enables.

[2]Even though the router has decided to reject the sender's request, it must still forward the packet, since the entire FANFARE system is designed to piggyback on the existing traffic between the sender and receiver. In other words, the router still has an obligation to deliver the data in the packet, even if it denies the FANFARE priority request.

next request.

If the router decides to allow the traffic flow, then it appends its marking to the list of markings in the packet's Cap^{init}_{next} field. We describe the details of the marking scheme below, but for now, it suffices that each router chooses a marking based on flow-specific information, the requested class, the router's secret key and the list of markings from previous routers. When the INIT-REQ packet arrives at the receiver, the receiver also makes an admission control decision as to whether it wishes to permit this flow. If it decides to allow the sender to transmit traffic, it collects the entries in the Cap^{init}_{next} field and includes them in a reply to the sender. In subsequent traffic, the sender includes the assembled markings in the Cap^{init}_{cur} field and sets the current priority class field to the value of the priority class it has requested.

Requesting a Higher Priority. If a sender already possesses a correct capability for some priority class, it may request an upgrade to a higher class. The sender constructs a new INIT-REQ packet that includes its current priority class, Cap^{init}_{cur}, and the new priority class requested. The round proceeds exactly like the initial round, and if the router decides to accept the upgrade request, it adds its markings to the Cap^{init}_{next} field. The receiver and the routers may base their admission control decisions on the sender's current priority class. For example, they might utilize a TCP-like mechanism and dictate that a sender may only request priority class PC_{i+1} if it currently has priority class PC_i. If the recipient and all of the routers on the path agree to the new priority class, then the sender updates the contents of subsequent packets to reflect the new priority class and includes the corresponding capability.

Maintaining a Priority. Once a sender achieves a certain priority class, it must perform regular maintenance or the associated capability will expire. To maintain its priority, the sender must regularly initiate a new initialization round with the request priority class set to its current priority class. The packet must also contain Cap^{init}_{cur}. If the round succeeds, the sender replaces the contents of the Cap^{init}_{cur} with those of the Cap^{init}_{next} and continues using its current priority class. If the request fails, it can retry the request or request a lower priority class.

2.2 Incremental Deployment

In a full FANFARE deployment, link congestion can be prevented if all routers do not over-commit local resources in their admission control, and enforce flow regulation to ensure that no flow gets more than its allotted bandwidth. However, in a legacy environment, several FANFARE-enabled routers may feed into a legacy router. Since the legacy router cannot distinguish between FANFARE flows and non-prioritized packets, prioritized traffic from the FANFARE-enabled routers may cause congestion at the legacy link. Even in the full deployment case, if a FANFARE router is malicious, it can cause congestion by admitting more prioritized traffic than allowed by local link capacity.

To achieve enforceable congestion control, we extend the scheme described above to include two rounds of capability negotiation: an initialization round and a consolidation round. The initialization round is identical to the scheme above, but the consolidation round uses secret sharing techniques to enforce automatic congestion control. To move from best effort to the lowest class of priority traffic, a sender must successfully complete an initialization round. To maintain its current priority class or ascend to a higher class, a sender must successfully complete both rounds.

The intuition behind the secret sharing scheme is to split a Cap^{cons} into multiple shares and attach only one share to each packet. Only when a sufficient number of shares have been collected can an end-host reconstruct the capability. An end-host has only a limited amount of time (dictated by τ_{init}) to collect the shares for the same capability. When a link is congested, a flow is unable to successfully transmit enough shares to construct a capability for its current priority class. As a result, it will be forced to slow its sending rate and acquire a capability for a lower priority class.

Initialization. The first initialization round allows a sender to elevate its traffic from best-effort service to the lowest level priority class of service. The sender can then use the resulting capability to request higher priorities. The sender activates the initialization round by sending an INIT-REQ packet to the receiver. In the INIT-REQ packet, the sender must set the current priority to 0 and set the requested priority to 1, the lowest priority class possible. This round is the same as the initialization round in the full deployment case, and if it completes successfully, the sender includes the assembled markings in the Cap^{init}_{cur} field and sets the current priority class field to 1.

Requesting a Higher Priority. Once a sender has obtained a priority class and corresponding capability, it may request an upgrade. This time, the sender must complete both an initialization and a consolidation round. First, the sender constructs an INIT-REQ packet that includes its current priority class, the associated capability, and the new priority class requested. The round proceeds exactly like the initial round. Assuming this round completes successfully, the sender can initiate the consolidation round.

To begin the consolidation round, the sender moves the accumulated markings from the initialization round into the Cap_{next}^{init} field of a CONS-REQ packet. The CONS-REQ packet also includes the sender's current capability and priority class, as well as the requested priority class. Both MAC fields are set to 0. The sender then sends n packets with the same CONS-REQ information to the receiver, where n is the maximum number of packets per time period permitted by the sender's current priority class. When a router receives a CONS-REQ packet, it verifies the current capability (see Section 2.3) and verifies that the Cap_{next}^{init} field contains the marking it would have assigned during the initialization round. It then calculates a second marking (again, we discuss the details in Section 2.3) and transforms the marking into m shares using a (t, m) secret sharing algorithm (e.g., Shamir's secret sharing [37]) such that any set of t or more shares suffice to reconstruct the marking, but any collection of $t-1$ shares reveals no information about the marking. The router appends a randomly chosen share (along with the share's ID) onto the list of markings in the packet's Cap_{next}^{cons} field and forwards the packet to the next router. The choice of t and m is dependent on the priority class, and in Section 3.4, we show how to derive appropriate values. For now, we assume that the number of shares provided for each class is known, so that the receiver knows how many pieces must be collected to assemble a complete capability.

Thus, if t distinct CONS-REQ packets arrive at the receiver, they can be combined to create a correct entry for the Cap_{next}^{cons} field, which the receiver can return to the sender. If t distinct CONS-REQ packets fail to arrive, then the sender can either retry the request or submit a request for a class with a lower threshold. If the sender does receive the completed entry for the Cap_{next}^{cons} field, then it moves both that field and the Cap_{next}^{init} field to the Cap_{cur} field and reinitializes the MAC field to 0. Now the sender can utilize the requested priority class.

Maintaining a Priority. As before, once a sender receives a capability for a priority class, it must still perform regular maintenance to prevent the capability from expiring. We require the sender to renew the Cap^{init} frequently, but allow a longer validity window for the Cap^{cons}. To renew a Cap^{init}, the sender initiates an INIT-REQ round identical to the initialization round, but this time it also includes its current class and associated capability. If the round succeeds, the sender replaces the contents of the Cap_{cur}^{init} with the markings from the Cap_{next}^{init} field that were returned by the receiver. To renew the Cap^{cons} field, the sender must also initiate another consolidation round and replace the contents of the Cap_{cur}^{cons} field with the new markings in the Cap_{next}^{cons} field.

2.3 Router Algorithm Details

In this section, we detail the router algorithm for issuing and validating capabilities.

2.3.1 Packet Update

When an INIT-REQ packet arrives at a router, the router must first decide whether to admit the flow at the requested priority level. This decision must be based on the router's available resources and may be based on the flow's previous priority class, an external mechanism utilized by the router's administrators, or both. If the router decides to deny the sender's request, it erases all previous markings from the packet's Cap_{next}^{init} field and forwards the packet. This prevents the sender from assembling a correct capability and thus prevents the sender from achieving the requested priority class. If the router does decide to admit the flow, it calculates an appropriate marking. For an INIT-REQ packet, the router uses the last b bits of

$$\text{(1)} \qquad \text{MAC}_{\mathcal{K}}(\text{source IP}||\text{dest IP}||\text{requested class}||\text{TTL}||\text{incoming interface}||\text{previous } Cap_{next}^{init} \text{ markings}||T)$$

where $\mathcal{K}$ is the router's secret key, T is the router's current timestamp, and previous Cap_{next}^{init} markings refer to all of the markings inserted into the Cap_{next}^{init} field by previous routers on the path. The router appends this marking, along with T_{init}, the last l_{init} bits of the timestamp T, to the entries in the Cap_{next}^{init} field, increments the hop count, and forwards the packet.

When a CONS-REQ packet arrives, the router first validates the current capability, as well as its own entry in the Cap_{next}^{init} field. If all of these fields validate, the router retrieves the timestamp T it used to calculate the marking in the Cap_{next}^{init} field. Note that a malicious sender cannot fake the timestamp, since we use the timestamp to validate the marking from the initialization round (see Section 2.3.2 for more details on the validation process). Using $\mathcal{K}$, the router calculates a new marking, M_{cons}, as the last b bits of

$$\text{MAC}_{\mathcal{K}}(\text{source IP}||\text{dest IP}||\text{requested class}||\text{TTL}||\text{incoming interface}||MF_{next}||T) \tag{2}$$

Using a (t, m) secret sharing scheme, the router transforms the new marking into m shares and appends a randomly selected share, the share's ID and the timestamp T_{cons}, the last l_{cons} bits of T, to the list of entries in the Cap_{next}^{cons} field. The router also updates the next MAC field (MF_{next}), such that

$$MF_{next} = \text{MAC}_{\mathcal{K}}(MF_{next}||T) \tag{3}$$

It then increments the hop count and forwards the packet along.

While similar to Equation 1, the calculation of Equation 2 includes the contents of the MAC field rather than the markings inserted by previous routers. We use the MAC field so that the marking is dependent on the markings of previous routers, making it even more difficult for an adversary to guess a correct capability.[3]

2.3.2 Capability Validation

When a packet arrives claiming to deserve a given priority class, a router must validate both of the Cap_{cur} fields (i.e., the initialization and the consolidation fields). If any field fails to validate properly, the router immediately drops the packet. To validate a Cap^{init} field, the router uses the timestamp T_{init} that accompanies the marking to calculate Equation 1. For the previous markings, it uses all of the markings in the Cap^{init} field that come before its own marking (it cannot use the markings that come after its own, since they were not used to compute the first marking). It also uses the current priority class in place of requested priority class. To validate the consolidation field, the router uses the timestamp T_{cons}. The router recalculates Equation 2 using T_{cons}, the current class instead of the requested class, and the current MAC field instead of the next MAC field. This allows it to exactly replicate the marking it would have provided to the sender when it first conducted the consolidation round. If the marking does not match the newly calculated value, the router immediately drops the packet.

Because T_{init} and T_{cons} only record the last l_{init} and l_{cons} bits of the router's internal timestamp T, the router must fill in the high order bits of the timestamp before using it to validate a capability. Equation 4 is used to compute the full timestamp T'_{init} by extending T_{init} with the upper bits of T. T in Equation 4 represents the router's current internal timestamp value.

$$T'_{init} = \begin{cases} \lfloor T/2^{l_{init}} \rfloor \cdot 2^{l_{init}} + T_{init} & T_{init} \leq T \bmod 2^{l_{init}} \\ \lfloor T/2^{l_{init}} - 1 \rfloor \cdot 2^{l_{init}} + T_{init} & T_{init} > T \bmod 2^{l_{init}} \end{cases} \tag{4}$$

The full timestamp T'_{cons} can be extended from T_{cons} in a similar way.

Therefore, a Cap^{init} is valid for at most $\tau_{init} = 2^{l_{init}}\Delta\tau$ time; and a Cap^{cons} is valid for at most $\tau_{cons} = 2^{l_{cons}}\Delta\tau$ time, where $\Delta\tau$ is the resolution of a timestamp.

3 Configuring FANFARE

In this section, we describe in detail how to configure FANFARE. We explain the interactions and implications of various FANFARE parameters. In particular, we present a typical configuration for FANFARE that achieves good performance, good security guarantees and TCP-friendliness over a downstream legacy link. The typical configuration proposed in this section also represents what is used in our simulation.

3.1 Timing Parameter Selection

FANFARE requires the selection of certain timing parameters; in particular, the lifetime of an initializer (τ_{init}) and consolidator (τ_{cons}), the length of identifiers for initializers (l_{init}) and consolidators (l_{init}), and the granularity of timestamps $\Delta\tau$. For the purposes of this discussion, we assume that the router's internal timestamp representation T has a sufficient number of bits such that it does not wrap around. A router increments its timestamp T every $\Delta\tau$

[3] We cannot base it on the contents of the Cap_{next}^{cons} field, since that contains a random share from each previous router.

seconds. We then pick τ_{init} and τ_{cons} to be multiples of $\Delta\tau$. Intuitively, a consolidator must have a longer lifetime than an initializer, because it takes more time to reconstruct a consolidator.

T_{init} and T_{cons} wraps around every $2^{l_{init}} \cdot \Delta\tau$ and $2^{l_{cons}} \cdot \Delta\tau$ seconds. A *Cap*^{init} is only valid if it is issued in the most recent $2^{l_{init}} \cdot \Delta\tau$ seconds; and a *Cap*^{cons} is valid only if it is issued in the most recent $2^{l_{cons}} \cdot \Delta\tau$ seconds. Therefore, $l_{init} = \lceil \lg(\tau_{init}/\Delta\tau) \rceil$; and $l_{cons} = \lceil \lg(\tau_{cons}/\Delta\tau) \rceil$.

First, we choose a timestamp interval $\Delta\tau$. Since τ_{init} and τ_{cons} are multiples of $\Delta\tau$, $\Delta\tau$ should be small enough to allow relatively rapid reaction to changes in link utilization, but large enough to keep the size of the timestamp field small. The timescale for changing priority classes (whether up or down) is a multiple of $\Delta\tau$, so the time FANFARE takes to react to changes in network conditions and the convergence of FANFARE after the introduction of new flows are all multiples of $\Delta\tau$. However, given fixed τ_{init} and τ_{cons}, the timestamp field is of length $\Theta(\lg(\frac{1}{\Delta\tau}))$, so we cannot make $\Delta\tau$ arbitrarily small. For this reason, we selected $\Delta\tau = 1s$ in our simulations.

τ_{init} is chosen as a multiple of $\Delta\tau$. Because we do not expect routers and senders to be time synchronized, an unlucky sender may only have $\tau_{init} - \Delta\tau$ time to acquire a capability. Thus, we must choose $\tau_{init} \geq 2\Delta\tau$. A smaller τ_{init} has two advantages: first, it allows routers and receivers to more quickly revoke a capability, and second, it allows a smaller choice of τ_{cons} (see below), which allows a more rapid reaction to changes in link utilization. As a result, we selected the smallest possible τ_{init} of 2 seconds for our simulations.

τ_{cons} is also chosen as a multiple of $\Delta\tau$, and must be greater than τ_{init}; otherwise, as soon as the capability is reconstructed, it will no longer be valid. Since a capability could be reconstructed as late as τ_{init}, it is guaranteed to be useful for at least $\tau_{cons} - \tau_{init}$. Ideally, a sender can use the capability until it can reconstruct the next capability; since the construction of an additional capability takes time $\tau_{init} - \Delta\tau$, a capability can be retained until the failure of $\left\lfloor \frac{\tau_{cons} - \tau_{init}}{\tau_{init} - \Delta\tau} \right\rfloor$ consecutive attempts to construct a capability. Our simulations used $\tau_{cons} = 4s$ to allow for 2 attempts to renew a capability.

Using this typical set of timing parameters, $l_{init} = 1$ and $l_{cons} = 2$, i.e., 1 bit is required to encode every *Cap*^{init} timestamp, and 2 bits are needed to encode every *Cap*^{cons} timestamp.

3.2 Capability-based Enforcement

Ideally, an attacker should be unable to correctly guess all of the markings that compose a legitimate capability. For a given router, the probability of guessing a marking of length b bits is $\frac{1}{2^b}$. For a path with r FANFARE routers, the probability that an attacker correctly guesses a complete capability is $\frac{1}{2^{b \cdot r}}$. Ultimately, the choice of b represents a tradeoff between security and space constraints in the packet header.

A colluding pair of endhosts can perform a capability guessing attack: the sender performs an exhaustive search by enumerating capabilities. If a packet succeeds in getting through to the colluding receiver, the receiver returns the corresponding capability to the sender. *To address the guessing attack, we propose to use more marking bits for higher bandwidth classes.* In the guessing attack, the expected number of packets the sender needs to send to succeed is $2^{b \cdot r - 1}$. However, due to the very limited lifetime of a *Cap*^{init}, the attacker has less than τ_{init} time to perform the attack. If the attacker currently possesses a bandwidth reservation of B, then he can succeed in blasting through $B \cdot \tau_{init}/pck\_size$ packets in τ_{init} time, where pck_size is the packet size. Therefore, the number of marking bits needed is $\Theta(\lg(B \cdot \tau_{init}))$.

3.3 Enforcing Flow Regulation

FANFARE does not dictate what specific flow regulation policy and algorithm should be used. However, in this section, we describe one possible way to instantiate a flow regulation policy and algorithm. We show in our simulation that using a similar flow regulation algorithm as described below, it is possible to achieve TCP-friendliness over a downstream bottleneck legacy link.

Decaying Capabilities. In FANFARE, a flow has two properties as stated by its capability: age and priority class. A router is allowed to make flow regulation decisions based on both properties.

In particular, given the age of a consolidation marking, a router can distinguish between capability's *stated* priority class (SPC) and its *effective* priority class (EPC). The capability's SPC is the current priority class as dictated by the packet header. The EPC reflects the actual class of service a flow with that priority class receives from the network at any given point in time. Initially, the flow's EPC is equivalent to its SPC. In general, we can downgrade the flow's EPC over time, rather than cutting off the flow immediately when the age of its Cap^{cons} exceeds the time limit. In other words, the router selects a downgrading function f that specifies a flow's EPC given its SPC and age. The function could provide the full SPC for the entire lifetime of the capability and then cut off service entirely, e.g.,

$$f(SPC, age) = \begin{cases} SPC & age \leq \tau_{cons} \\ 0 & age > \tau_{cons} \end{cases} \tag{5}$$

but a router could also choose a more gradual decay function, such as

$$f(SPC, age) = \begin{cases} SPC & age \leq \tau_{cons} \\ SPC\left(\frac{1}{\gamma}\right)^{age-\tau_{cons}} & age > \tau_{cons} \end{cases} \tag{6}$$

where γ is a parameter that specifies how fast a capability's EPC downgrades. A gradual downgrading function helps to compensate legitimate flows for the probabilistic nature of the consolidation round. For example, assume that routers implement an admission policy in which they refuse to give out priority class PC_{i+1} unless a flow currently possesses at least PC_i. Using Equation 6, if a high priority flow probabilistically fails to assemble a sufficient number of shares before the current Cap^{cons} times out, its effective rate will degrade, rather than abruptly falling back to PC_0, which would force it to complete several rounds of capability negotiation before it can regain a high priority class.

FANFARE does not dictate what flow regulation algorithm to be used, however, in Section 6, we show that by picking a decaying function similar to Equation 6, it is possible to achieve TCP-friendliness over a downstream legacy link.

Router Rate Limiting. To provide service differentiation for prioritized traffic, the routers must prevent flows from consuming more resources than allowed by their priority class. However, we would like to accomplish this goal without requiring per-flow state at the routers. Thus, we use a modified multistage filter [11] for each supported traffic class to track the largest flows within that priority group. The key insight is that as long as the largest flows within a priority class do not exceed that class's traffic threshold, then all of the flows within the class must also be below the limit. Since we can halt a misbehaving flow by erasing its capability and/or dropping its packets, we can eliminate the largest flows and then apply the multistage filter again to determine the new set of largest flows. We repeat this process until the largest flows respect the threshold for their priority class. We provide more details for the multi-stage filter algorithm in Appendix B.

3.4 Enforced Congestion Control

In this section, we explain how to pick parameters for the secret-sharing scheme. In essence, we use the secret-sharing scheme to provide automatic and enforceable congestion control. When a link is congested, a flow is unable to successfully transmit enough shares to construct a capability for its current priority class, and hence the sender is forced to slow down.

In Appendix C, we give the theoretical derivation for the probability that a sender successfully receives a capability. The result of the theoretical analysis is shown in Figure 2. As we can see from the figure, the larger the number of distinct shares, the steeper the slope of the curve. An intuitive explanation for this is that as the number of distinct fragments increases, each fragment is more likely to appear in a single packet only, so packet loss has a greater impact on the number of distinct shares received. Conversely, if the number of distinct fragments is small, multiple packets will contain the same share, attenuating the effect of packet loss.

We now discuss the selection of the parameters m and t, which determine the probability that a sender can obtain a capability, given a congested link. We want to ensure that a sender has a high probability of obtaining a capability if packet loss is low, and has a low probability of obtaining a capability if packet loss is high. When packet loss is low, $q \leq q_{low}$, we want the probability of obtaining the capability to be at least ρ_{low}. Similarly, if packet loss is high, $q \geq q_{high}$, we want the probability of obtaining the capability to be at most ρ_{high}.

To find the optimal values for t and m, we compute the values that minimize the squared error of the desired probabilities:

$$\operatorname*{argmin}_{t,m} \left((P_m(t, n \cdot (1 - q_{low})) - \rho_{low})^2 + (P_m(t, n \cdot (1 - q_{high})) - \rho_{high})^2 \right) \tag{7}$$

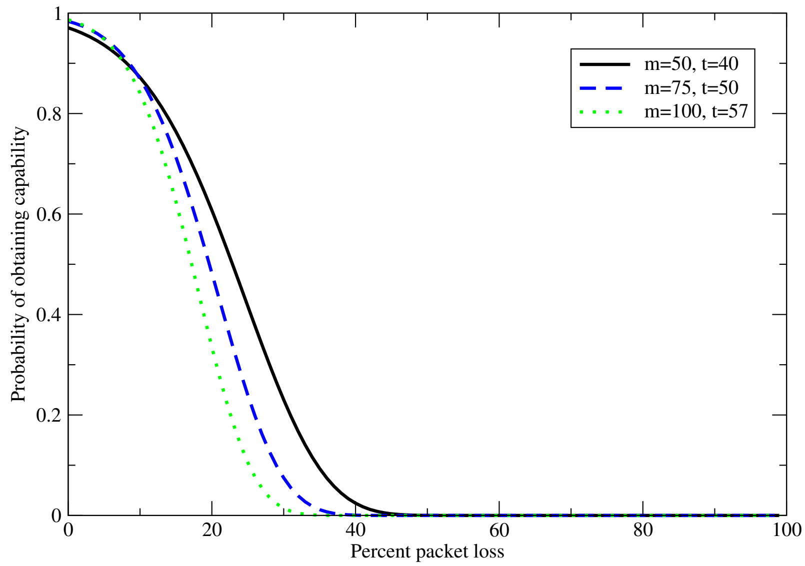


Figure 2: Probability of obtaining t distinct shares after a sender sends 100 packets for varying amounts of packet loss. For this figure, we vary m and compute t such that we achieve a reconstruction probability of at least $\rho_{low} = 95\%$ for packet loss probability of $q_{low} = 3\%$.

where $P_m(t, \alpha)$ is the probability that at least t distinct shares are received after receiving α packets. In our simulation we set $q_{low} = 3\%$, $\rho_{low} = 95\%$, $q_{high} = 30\%$, and $\rho_{high} = 5\%$. For these values, we obtain $m = 80$ and $t = 52$, given that 100 packets are sent.

4 Attack and Defense Analysis

In this section, we study possible attacks against FANFARE and explain how we defend against them. We focus on FANFARE specific attacks, since attacks that target vulnerabilities in the underlying architecture (such as those targeting the secure hash function or the multi-stage filtering technique) lie outside the scope of this paper. In this discussion, we assume that an established flow is unable to trick the router's flow regulation mechanism into giving it more than a minimal (ϵ) advantage over its maximum allowable bandwidth, since a flow that exceeds $1 + \epsilon$ of its allowable bitrate will be detected and throttled, with high probability. We consider attacks both by malicious endhosts and by malicious routers.

4.1 Defending against Malicious Endhosts

A malicious endhost can inject traffic with either a prioritized flow or through best effort traffic.

To attack using prioritized traffic, a sender needs to possess the correct capabilities. FANFARE's first tier defense against attacks through prioritized traffic is router- and recipient-based admission control. In FANFARE, a capability is jointly issued by the network and the recipient. When a FANFARE-enabled router detects congestion on a local link, it can halt or downgrade any flow by refusing to tag its packets with markings. When the recipient's network is under a DDoS attack, the recipient can also stop flows by not supplying capabilities to the attackers.

Sometimes an adversary may be well-concealed and it may attempt to illegally acquire a high priority class to facilitate its attack. To achieve this, the attacker may attempt to blast through more CONS-REQ packets than a legitimate sender would have done. However, multistage filtering will ensure that an attacker cannot send significantly more packets than a legitimate sender.

A colluding receiver may attempt to assist the malicious sender in an attempt to flood the route between them with high priority traffic. In FANFARE, a colluding receiver is subject to the routers' admission control decisions. Meanwhile, a malicious receiver is unable to accelerate the capability reconstruction process, for it can only receive as many consolidators as a legitimate receiver in the same position. So even when the receiver maliciously colludes with the sender, the sender cannot achieve a higher sending rate than if it were sending to a legitimate receiver.

In FANFARE, an attacker using IP spoofing is unable to flood network links through priority traffic, since she does not have the correct return path to obtain a capability. An attacker that fails to obtain a capability may attempt to guess the correct capability. We show in Section 3.2 that with our cryptographic mechanisms in place, the adversary is not likely to guess a correct capability. Thus, when an adversary attempts a DDoS attack using guessed capabilities, with high probability, the attack traffic will be dropped early on the path, before reaching the access link to the recipient's network. FANFARE prevents an attacker from spoofing its source IP address only when the destination is legitimate. A colluding receiver can return the capability to the real IP address of the IP spoofing attacker. Intermediate routers therefore cannot rely on the source IP address for admission control. None of the previous schemes [46, 45] prevent IP spoofing.

An attacker may attempt to perform IP spoofing to interfere with an established flow. Because the spoofer's packets traverse a different path than the legitimate traffic, the capabilities will differ. If the destination returns capabilities constructed using the spoofed route to the legitimate sender, the legitimate sender will lose priority. To prevent this attack, when a destination is returning a new initializer, it will ensure that the packet contains a valid, previously returned consolidator, and when a destination is reconstructing a new consolidator, it will ensure that the packet contains a valid, previously returned initializer. This links together all communications starting from the first returned initializer.

The adversary may also attempt to attack using best-effort traffic. In FANFARE, best-effort traffic will not disrupt prioritized traffic at a FANFARE-enabled router, since the router will always give precedence to priority traffic. With full deployment of FANFARE, we need not worry about best-effort traffic causing congestion for priority traffic downstream, since a FANFARE router will not admit more prioritized traffic than it can handle, and since FANFARE routers drop best-effort traffic in favor of prioritized traffic. During the early stages of deployment, however, best-effort traffic may cause congestion at legacy routers. As discussed in Section 7.1, this effect can be mitigated by limiting the percentage of bandwidth provided to best-effort traffic. Though FANFARE does not prevent legacy links from being flooded by malicious legacy traffic, we do show in Section 6 that FANFARE is friendly to legitimate TCP flows over a legacy link.

4.2 Defending against Malicious Routers

A malicious router might try to assist a malicious endhost by artificially elevating its privilege level. However, since each FANFARE router makes its admission decision independently of other routers, the malicious router's designation will not affect legitimate routers. This illustrates the benefits of our minimal trust assumption approach. A malicious router may also drop or delay the packets of a legitimate flow, but since a router has full control over packets it forwards, these attacks are outside the scope of this paper.

Malicious routers may also collude. Two colluding routers can use all of the priority bandwidth between them. First, the routers determine which privileged flows forwarded by the upstream router reach the downstream router. The upstream router then replaces the data in each packet from each such flow with data it wishes to send to the downstream router. When the packet reaches the downstream router, the downstream router processes the data added in this way, and forwards the corrupt packet normally (to allow the receiver to reconstruct the next capability).

An alternative (and generally equivalent) attack is for both routers to deny admission to any flows other than their own. If no policy control or admission control is applied at intermediate routers, then the collaborating malicious routers can obtain the same capabilities.

Unfortunately, none of the prior schemes defend against this attack, and we leave it as future work.

5 Applications of FANFARE

By designing FANFARE as a secure framework, we allow a wide variety of applications to integrate security into the network layer. In this section, we demonstrate the flexibility of FANFARE by securing several existing applications; in the future, new applications can utilize FANFARE as a secure foundation to build upon. Below, we discuss FANFARE's ability to quickly halt distributed denial-of-service attacks and show how to securely provide differentiated levels of service to network flows.

5.1 Halting Distributed Denial-of-Service Attacks

FANFARE provides strong properties for preventing and halting DDoS attacks. First, the sender relies on the receiver to return the capability assembled by the sender's INIT-REQ and CONS-REQ packets. If the receiver determines that it is facing a DDoS attack, or even decides that it has already committed too many network resources to its current set of clients, it can stop returning capabilities to senders. As we show in Section 3.2, without receiving the capability from the receiver, the sender has a low probability of forging a correct capability. Furthermore, as soon as an attack packet without the appropriate marking arrives at a FANFARE router, the router will drop the packet. As a result, attack traffic is dropped at routers near the attacker, before it can congest the link leading to the recipient.

The use of secret sharing during the consolidation round provides additional benefits. If FANFARE only used the initialization round without the consolidation round, a malicious endhost on the same subnet as the victim could provide an attacker with an appropriate capability. However, if the attack generates congestion in the network, then during the consolidation round, the attackers will be unable to receive enough shares to assemble a correct capability. Thus, the attacker will be forced to send at a lower rate. Furthermore, since FANFARE provides the primitives for securely differentiating traffic into multiple priority classes, an attacker cannot gain an advantage at the expense of a legitimate sender attempting to connect to the receiver. Since FANFARE-enabled routers give preference to prioritized traffic over legacy traffic, an attacker cannot use best-effort traffic to disrupt established priority connections.

5.2 Securing Diffserv

Diffserv Primer. The goal of Diffserv [6] is to provide scalable service differentiation without per-flow state at core routers. In Diffserv, a DS domain is a set of network nodes that run a common provisioning policy. Traffic entering a DS domain will be classified and conditioned by DS boundary routers (i.e., edge routers). To convey the classification decision, the edge router marks the packet with a DS code point value. The code point specifies which Per-Hop-Behavior (PHB) aggregate the packet belongs to. Thus DS interior routers (i.e., core routers) can select the forwarding behavior for each packet according to the code point marking. To provide scalability, sophisticated service provisioning and traffic conditioning functions that require per-flow state are only performed at the edge routers, where traffic volume is lower and flows are fewer. On the other hand, core routers do not keep state; they therefore apply a simpler forwarding behavior to each packet as specified by its code point marking. Simple resource management mechanisms such as RED [14] can be employed by core routers to provide service differentiation to different PHB aggregates.

IP Spoofing against Diffserv. Edge routers classify packets according to the content of some portion of the packet header such as the source and destination address. Without any authentication mechanism, a malicious endhost can easily forge the packet header and deceive a legitimate Diffserv router into providing prioritized service to its traffic. For instance, consider a DS domain configured with a policy where host A is promised a 50Mb channel to D, whereas M only has a 2Mb channel to D. Now if M spoofs the address of A, Diffserv routers will be enticed into believing that traffic from M to D is prioritized. Thus M can succeed in injecting significantly more traffic than it should.

In Section 6, we show through simulation how such an IP spoofing attacker can gain bandwidth at the expense of legitimate senders if a downstream legacy link is congested.

Securing Diffserv through FANFARE. We can leverage FANFARE to secure Diffserv against the IP spoofing attack. Basically, FANFARE's cryptographically secured path dependent markings can be directly added to Diffserv without requiring any changes to the Diffserv architecture or operational semantics. The only difference is that now, before classifying and conditioning a traffic flow, a FANFARE enabled Diffserv edge router needs to verify the source and destination addresses of a packet by verifying its capability. Since an IP spoofing attacker does not possess a return path to receive a capability, he is unable to acquire priority service through IP spoofing.

6 Evaluation

In this section, we first analyze FANFARE's header overhead and router storage and computational overhead. Then we present the simulation of FANFARE. The simulation demonstrates two critical properties of FANFARE: first, that FANFARE's congestion response mechanism provides fairness between competing flows in the face of congestion on a downstream legacy link, and second, that FANFARE can be parameterized to be friendly to TCP traffic across a congested legacy link. We show that fairness and TCP friendliness is preserved even in the presence of an adversary that controls two malicious end hosts communicating across the legacy link.

6.1 Overhead Analysis

Header Overhead. The communication overhead depends on many parameter choices. The maximum hop count has the largest impact on the FANFARE header size, since we allocate space for each FANFARE router in each capability field. However, after construction of the initial capability Cap_{cur}^{init} in the INIT-REQ round, the sender knows the number of FANFARE-enabled routers on the path. Moreover, the initial exchange only requires the Cap_{cur}^{init} field; the other three capability fields are not required. Subsequent FANFARE headers need only carry sufficient space to carry the exact number of markings of the routers on the path, resulting in much smaller overhead since most Internet paths are short, and FANFARE routers may be sparsely deployed.

Using 1 byte to encode the priority class, 1 byte for the hop count and flags, 3 bits for the timestamp, 2 bits for the marking, 8 bits for the share ID, and a maximum hop count of 32 hops, the header size for the INIT-REQ packet is 23 bytes. Assuming 4 FANFARE routers, the size of each subsequent header is also 23 bytes, and 43 bytes for 8 enabled routers.

Router Overhead. Even though the new generation of VLSI technology has enabled us to produce extremely powerful routers, requiring minimal state at each router is still a desirable property. FANFARE avoids using per-flow state to enable scalability to large networks.

FANFARE also requires minimal per-packet processing. In FANFARE, per-packet processing can be performed very quickly and does not slow down the fast path. FANFARE requires the computation of two functions for each packet: the message authentication code (MAC) and the secret sharing; both can be performed in parallel with the route lookup. MAC functions are computationally very efficient, and can be performed at line speed in dedicated hardware [38]. For example, we can use a high-speed block cipher to construct a CBC-MAC function. Secret sharing is another operation that requires attention. For large values of the number of possible secret shares m, secret sharing using a scheme like Shamir's [37] is computationally expensive, since Shamir's scheme evaluates a polynomial of order m to generate each share. To speed up the computation of these shares and to facilitate fast hardware implementation, we split up the secret into several chunks and perform secret sharing on each chunk. For example, we can split the marking into q chunks $C_0, \ldots, C_q$ such that all q chunks are necessary to reconstruct the marking. Fewer shares are computed for each chunk, allowing for an efficient hardware implementation. The tradeoff of this mechanism is that the receiver has to be able to reconstruct each chunk to be able to get the entire marking; however, by applying a second level of forward error correction we can resolve this shortcoming.

6.2 Simulation

Simulation Setup. To evaluate the security and efficiency of our design, we implement FANFARE in the *ns-2* simulator and test its effectiveness under several topologies and types of attacks. We evaluate the effectiveness of FANFARE's flow control as compared to legacy TCP and Diffserv with a legacy router. We parameterize FANFARE as follows:

- Timestamp interval T is 1 second
- Bandwidth for class $i \in [1, 127]$ is $50(i+1)$ kbps
- Bottleneck bandwidth is 1500 kbps

In our scenario, we model a single bottleneck link between two legacy routers. Figure 3 shows a simplified sketch of this scenario. Each flow enters the legacy router from a different FANFARE router. None of the FANFARE routers know the bandwidth of the bottleneck link. We also consider a different scenario in which competing flows merge at a single FANFARE router before entering the legacy router, but due to space constraints, we show the results from the first scenario only, as the results from the second scenario are nearly identical. We show the goodput of each flow as a moving average over ten seconds.

Throughout the simulation, we use a simple admission control policy where a FANFARE router gives out priority class no greater than PC_{i+1} when a requesting flow holds a current capability whose effective priority class is at least PC_i. We choose secret sharing parameters using the methodology described in Section 3.4, tuning the input to require lower packet loss rate for higher priority classes. The reason for requiring lower packet loss at higher priority classes is to ensure system stability: if two flows f_1 and f_2 share a common bottleneck legacy link downstream, they will statistically experience the same packet loss rate, and will therefore converge to the same priority class. Capabilities are downgraded over time as follows: for the first four timestamp intervals after the

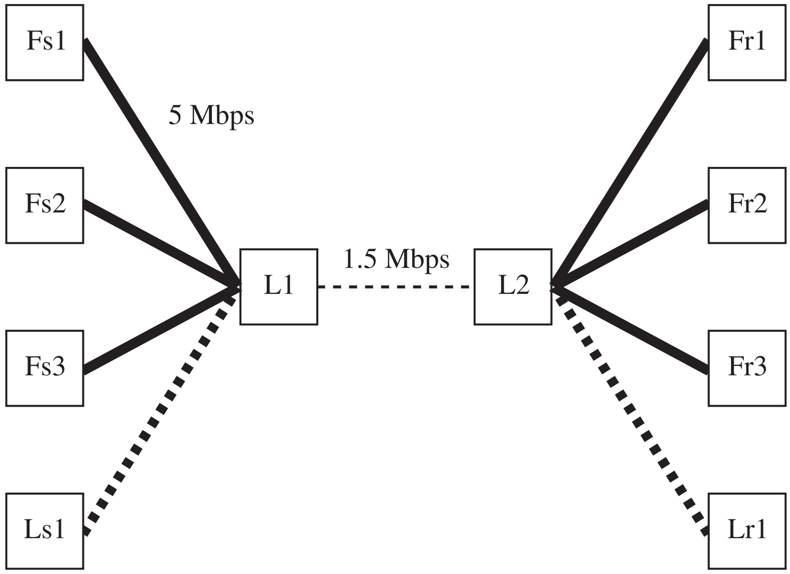
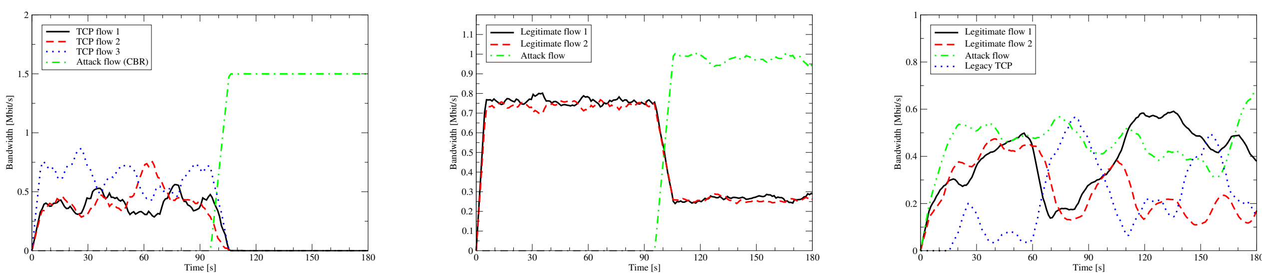


Figure 3: A simplified diagram of our simulation topology. FANFARE flows run from Fs1, Fs2, and Fs3 to Fr1, Fr2, and Fr3 respectively. Legacy TCP traffic runs between Ls1 and Lr1, and L1 and L2 are legacy routers on either end of the bottleneck link. The Fs1, Fs2, and Fs3 nodes each represent a series of FANFARE-enabled routers. The dashed lines represent the legacy links, and line thickness represents the relative bandwidths of the links.



(a) Three legacy TCP flows contend with an attacker that starts flooding at time 100s.

(b) Two legitimate Diffserv flows with an attacker that spoofs to increase its priority. The attacking flow starts sending at 100s.

(c) Two legitimate FANFARE flows using CBR and one legacy TCP flow contend with a malicious FANFARE flow using CBR. Legacy TCP flow starts after 20 seconds.

Figure 4: Simulation results for three different congestion control protocols under attack.

initializer is accepted, the effective priority class is equal to the stated priority class. After that time, each elapsed timestamp interval reduces the effective priority class by a factor of 2, rounding down each time. Though we use this specific set of policies in our simulation, FANFARE's design is general enough to allow the use of other admission and flow control policies.

Simulation Results. To demonstrate the effectiveness of FANFARE as compared to previous flow control mechanisms, we examine the flow control provided by legacy TCP as well as Diffserv. To evaluate legacy TCP, we allow a single TCP flow to compete with a malicious Constant Bit Rate (CBR) flow. To evaluate Diffserv, we allow two legitimate CBR flows to compete against a malicious CBR flow that forges its IP source address to take advantage of a PHB aggregate not intended for that malicious flow.

Figure 4(a) demonstrates the extreme unfairness to TCP when sharing a link with malicious flows. Legitimate TCP senders back off in the face of packet loss, whereas a malicious sender can aggressively inject traffic at a high rate without regard to the successful delivery of its packets. As a result, very shortly after the malicious CBR sender is introduced, legitimate TCP throughput drops to zero.

Like many other infrastructure-enforced flow control mechanisms, Diffserv partially addresses the problem of aggressive endhosts compromising inter-flow fairness. However, as demonstrated in Figure 4(b), Diffserv is prone to an IP spoofing attack. Here Diffserv routers are configured with a policy that promises a high-bandwidth channel between endhosts A and B. By spoofing the IP address of A, the attacker tricks routers into applying the high-bandwidth PHB to its packets, and as a result acquires a considerable fraction of the available bandwidth at the expense of legitimate flows.

Providing TCP friendliness over a legacy link is also a desirable feature of FANFARE. FANFARE can be parameterized to require that flows with capabilities back off faster than TCP flows in the face of congestion. Figure 4(c) shows the TCP-friendly performance of two legitimate CBR flows and one attack CBR flow when competing with one legacy TCP flow. This shows that FANFARE can enforce TCP-friendly behavior, even on FANFARE-enabled flows with a malicious sender and receiver. (Of course, an attacker able to send a legacy CBR flow will be able to compete unfairly with a legacy TCP flow.) This property allows an ISP to prevent its customers from flooding attacks by rate-limiting packets sent on non-FANFARE flows, since FANFARE flows are securely restricted to behave in a TCP-friendly manner, even when the bottleneck link is between two legacy routers.

7 Discussion

7.1 Restricting Best-effort Traffic for Incremental Deployment

As mentioned earlier, if all routers used FANFARE, we could entirely prevent link congestion, since routers can ensure that they do not over-allocate bandwidth resources. A malicious flow that attempts to inject more traffic than its allocated bandwidth will be throttled by the flow regulation algorithm on the routers. However, with legacy links in the system, there is still the potential for attacker flooding. If several FANFARE-enabled routers connect to a legacy router (as shown in Figure 3), and the enabled routers have very few prioritized flows, then the adversary could use best-effort traffic at multiple routers to flood the legacy link. This could negatively affect prioritized flows. As one potential remedy, we could have FANFARE-enabled routers restrict legacy traffic to a fixed percentage of link bandwidth, such that the combination of multiple routers feeding into a legacy link cannot exceed its capacity. This has the additional advantage in that it incentivizes endhosts to upgrade to the FANFARE system. On the other hand, until endhosts upgrade, the routers will underutilize the full bandwidth of the links, since they will restrict legacy traffic even when there is not enough prioritized traffic to use the remaining bandwidth. To mitigate this underutilization, routers can initially allocate a large percentage of bandwidth to legacy traffic, and then gradually decrease the percentage as more endhosts upgrade to the FANFARE system.

7.2 Route Stability

Since a capability is only valid on a specific sequence of routers, a capability is not valid for a path other than the one on which it was constructed. Thus, after a route change, a flow will soon get dropped after the point of change. An alternative approach would be to convert privileged flows to unprivileged as soon as the capability does not match any more, but an attacker could exploit this by guessing the early bits of a capability to get past the initial FANFARE router. Moreover, unprivileged traffic may cause congestion on legacy links, thus endangering the safety of privileged flows across those links.

Thus we recommend requiring an endhost to set up a new capability after a route change. Assuming route changes are not too frequent, this approach is the safest and most viable.

7.3 IPv4 Compatibility

FANFARE requires the use of many header bits, and such bits are not available in the IPv4 header. In this section, we sketch a solution that adds a new header to hold these bits. Our design is similar to the Hop-by-Hop Options header in IPv6 [9]. We allocate a new protocol number, which we place in the IP Protocol field. Our header then follows the IP header, and in addition to the fields described in Section 2 includes space for a *Next Header* field, which identifies the protocol immediately following the FANFARE header.

To use FANFARE with TCP, we add an option to TCP which is set by all FANFARE-enabled hosts. A FANFARE client attempting to connect to a server sends each TCP SYN packet twice: once embedded inside a FANFARE packet, and once as a legacy IPv4 packet. If the server speaks FANFARE, it will accept the FANFARE packet. It will also drop any TCP packet with the FANFARE-enabled option set, so the TCP packet sent over legacy IPv4 will be ignored. On the other hand, a legacy server will ignore the FANFARE packet and accept legacy IPv4 packet.

FANFARE packets must set the "Don't Fragment" bit in the IP header. A host deploying FANFARE must perform Path MTU Discovery [32] for each flow, and fragment all datagrams that would exceed the MTU after inclusion of the IP header and FANFARE header. FANFARE headers should optionally contain fragment offset fields to allow the destination to reconstruct the original packet before they are delivered to the application.

7.4 Compatibility with ICMP, NAT and Loose Source Routing

Internet routers use ICMP messages to inform a packet's source when the packet encounters certain problems; for example, when the packet has exceeded its maximum time-to-live, or when a packet with the "don't fragment" bit set traverses a link with a MTU smaller than the packet size. ICMP messages include the IP header and the first eight bytes of the protocol payload of the triggering packet. These values must allow the sender to determine which flow caused the error, and must not include any capabilities. To satisfy the first constraint, a node may cache the correlation between IP source, destination, and identifier fields with the corresponding flow. To satisfy the second constraint, we ensure that the first eight bytes of the FANFARE header does not contain any information from which the source can derive capabilities. In particular, it should include the next header field, the FANFARE header length, the current and requested priorities, the hop count and flags, and the MAC field.

Many Internet hosts are connected through Network Address Translators (NATs), which allow several hosts to share a single IP address. If the NAT does not understand FANFARE packets, then it will not be able to handle them. However, if the NAT deploys FANFARE, then it can cause all flows traversing it to be FANFARE-enabled, even if the end host does not deploy FANFARE.

With loose source routing, a sender can specify a set of routers its traffic should traverse. In this case, FANFARE admission control should look at the IP destination address, which represents the next hop along the path, to make admission control decisions. A FANFARE router may optionally inspect the loose source route option; however, there is no guarantee that the current IP destination will continue to forward the packet towards the destination indicated in the loose source route option. By forwarding a packet with a FANFARE header, an intermediate node may be providing some of its FANFARE resources to a flow that it is not actually participating in.

8 Related Work

Appendix E reviews research in Quality of Service (QoS), congestion avoidance and control, policing, and DoS/DDoS defense mechanisms. In this section, we focus on the most closely related work: capability-based mechanisms, where a capability grants the right to send priority traffic to a destination. Alexander et al. [1] present such an architecture in the context of active networks. Lakshminarayanan et al. [26] leverage the i^3 infrastructure [39] to enable a receiver to cut off unwanted senders. In their system, a private trigger is like a capability that can be revoked by the receiver. Anderson et al. [3] present an infrastructure where the sender uses a capability to set up a path to the receiver. While these systems achieve some of the properties of FANFARE, they are stateful and require trust relationships (i.e., secure keys) to be set up between infrastructure nodes and endhosts. Moreover, they do not address the issue of congestion across legacy links.

The most closely related works are by Machiraju et al. [29], Yaar et al. [45] and Yang et al. [46]. Machiraju et al. propose a secure QoS architecture [29]. They use lightweight certificates to enable routers to designate bandwidth reservations, and they propose a stateless recursive monitoring algorithm for routers to throttle flows that attempt to exceed their allotted bandwidth. However, they do not address incremental deployment and they do not enforce congestion control in the face of malicious routers that attempt to flood a link by overallocating bandwidth.

In SIFF [45], researchers propose a capability system to allow a receiver to enforce flow-based admission control. In both SIFF and FANFARE, routers mark packets to form a capability, which the receiver forwards to the sender. In contrast to FANFARE, SIFF only distinguishes between best effort and privileged flows, but does not provide multiple classes of service. Hence, in SIFF, a receiver cannot control a sender's rate, and must be very careful not to give capabilities to malicious senders. Also, SIFF routers cannot police traffic. They do not catch flows that send at a very high rate, and cannot slow down flows. Yang et al. address some of the above mentioned problems with SIFF. They allow a receiver to issue capabilities with fine-grained service levels, and propose a capability-based router queue management scheme to achieve per-source fair queuing. Furthermore, they have the routers perform rate-limiting to throttle malicious flows that attempt to exceed that allotted bandwidth. Their rate-limiting algorithm requires per-flow state on the order of 100 bytes, and hence is expensive. Meanwhile, Yang et al. only allow the receiver to perform admission control, and they do not address the issue of protecting a downstream legacy link from being flooded.

9 Conclusion

We present FANFARE, a framework for providing secure flow control. FANFARE enables routers and receivers to precisely control the upper bound on the bandwidth of each flow, without requiring any per-flow state on routers or any established cryptographic keys between routers or endhosts. Our system works well in legacy environments, even if only a small fraction of updated routers are deployed.

FANFARE enables several applications. Servers can use it to admit new flows liberally at the lowest rate, and only upgrade the allowed bandwidth once the user is authenticated; in such an environment, DDoS attacks cannot flood a server to squelch customers' flows.

FANFARE also enables construction of secure congestion control mechanisms, as FANFARE flows compete fairly with TCP, even if the bottleneck link is between legacy routers, and even if both endhosts are malicious. Thus, our system can provide secure versions of Diffserv, Intserv, TCP, XCP, as well as many other protocols.

References

[1] D. Scott Alexander, Kostas G. Anagnostakis, William A. Arbaugh, Angelos D. Keromytis, and Jonathan M. Smith. The price of safety in an active network. In *Proceedings of ACM SIGCOMM*, 1999.

[2] D. Andersen. Mayday: Distributed filtering for Internet services. In *Proceedings of USITS*, 2003.

[3] Tom Anderson, Timothy Roscoe, and David Wetherall. Preventing Internet denial-of-service with capabilities. In *Proceedings of Hotnets-II*, November 2003.

[4] T. Aura, P. Nikander, and J. Leiwo. Dos-resistant Authentication with Client Puzzles. In *Proceedings of Security Protocols Workshop*, 2001.

[5] D. Bansal, H. Balakrishnan, S. Floyd, and S. Shenker. Dynamic behavior of slowly-responsive congestion control algorithms. In *Proceedings of ACM SIGCOMM*, 2001.

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. Internet Request for Comment RFC 2475, Internet Engineering Task Force, December 1998.

[7] R. Braden, D. Clark, and S. Shenker. Integrated services in the Internet architecture: an overview. Internet Request for Comment RFC 1633, Internet Engineering Task Force, June 1994.

[8] J. Byers, J. Considine, G. Itkis, M. Cheng, and A. Yeung. Securing bulk content almost for free. *To appear in Journal of Computer Communications*, 2005.

[9] S. Deering, Cisco, and R. Hinden. Internet protocol, version 6 (IPv6) specification. Internet Request for Comment RFC 2460, Internet Engineering Task Force, December 1998.

[10] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology — CRYPTO '92*, 1993.

[11] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of ACM SIGCOMM*, November 2002.

[12] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume I of *Wiley Series in Probability and Mathematical Statistics*. John Wiley & Sons, 1968.

[13] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, 1999.

[14] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[15] V. Gligor. Guaranteeing access in spite of service-flooding attacks. In *Proceedings of the Security Protocols Workshop*, April 2003.

[16] S. Gorinsky, S. Jain, H. Vin, and Y. Zhang. Robustness to inflated subscription in multicast congestion control. In *Proceedings of ACM SIGCOMM*, 2003.

[17] Jay Hyman, Aurel A. Lazar, and Giovanni Pacifici. Joint scheduling and admission control for ats-based switching nodes. In *SIGCOMM '92: Conference proceedings on Communications architectures & protocols*, pages 223–234, New York, NY, USA, 1992. ACM Press.

[18] J. Ioannidis and S. Bellovin. Implementing pushback: Router-based defense against DDoS attacks. In *Proceedings of the Symposium on Network and Distributed Systems Security*, 2002.

[19] Van Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM*, August 1988.

[20] Sugih Jamin, Scott Shenker, Lixia Zhang, and David D. Clark. An admission control algorithm for predictive real-time service (extended abstract). In *Proceedings of the Third International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 349–356, London, UK, 1993.

[21] C. Jin, H. Wang, and K. Shin. Hop-count filtering: An effective defense against spoofed DDoS traffic. In *Proceedings of ACM Conference on Computer and Communications Security (CCS'2003)*, October 2003.

[22] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of Symposium on Network and Distributed Systems Security*, 1999.
[23] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In *The Eleventh International World Wide Web Conference (WWW 11)*, May 2002.
[24] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of ACM SIGCOMM*, August 2002.
[25] A. Keromytis, V. Misra, and D. Rubenstein. SOS: Secure Overlay Services. In *Proceedings of ACM SIGCOMM*, 2002.
[26] K. Lakshminarayanan, D. Adkins, A. Perrig, and I. Stoica. Taming IP Packet Flooding Attacks. In *Proceedings of ACM HotNets-II*, November 2003.
[27] J. Lemon. Resisting SYN flood DoS attacks with a SYN cache. In *Proceedings of Usenix BSDCON*, February 2002.
[28] S. Machiraju, M. Seshadri, and I. Stoica. A scalable and robust solution for bandwidth allocation. In *International Workshop on QoS*, May 2002.
[29] S. Machiraju, M. Seshadri, and I. Stoica. A scalable and robust solution for bandwidth allocation. Technical Report UCB//CSD02-1176, University of California at Berkeley, 2002.
[30] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *CCR*, 32(3):62–73, July 2002.
[31] J. Mirkovic, G. Prier, and P. Reiher. Attacking DDoS at the source. In *Proceedings of ICNP*, 2002.
[32] J. Mogul and S. Deering. Path mtu discovery. Internet Request for Comment RFC 1191, Internet Engineering Task Force, November 1990.
[33] J. Postel. Transmission control protocol. RFC 793, September 1981.
[34] K. Ramakrishnan and S. Floyd. Proposal to add explicit congestion notification (ECN) to IP. Internet Request for Comment RFC 2481, Internet Engineering Task Force, January 1999.
[35] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP congestion control with a misbehaving receiver. *ACM SIGCOMM Computer Communication Review*, 29(5), 1999.
[36] C. Schuba, I. Krsul, M. Kuhn, E. Spafford, A. Sundaram, and D. Zamboni. Analysis of a denial of service attack on TCP. In *Proceedings of IEEE Symposium on Security and Privacy*, 1997.
[37] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, November 1979.
[38] Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Stephen T. Kent, and W. Timothy Strayer. Hash-based IP traceback. In *Proceedings of ACM SIGCOMM 2001*, pages 3–14, August 2001.
[39] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *Proceedings of ACM SIGCOMM*, 2002.
[40] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high speed networks. In *Proceedings of ACM SIGCOMM 1998*, 1998.
[41] I. Stoica and H. Zhang. Providing guaranteed services without per flow management. In *Proceedings of ACM SIGCOMM*, 1999.
[42] I. Stoica, H. Zhang, and S. Shenker. Self-verifying csfq. In *Proceedings of IEEE INFOCOM*, 2002.
[43] X. Wang and M. Reiter. Defending against denial-of-service attacks with puzzle auctions. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2003.
[44] A. Yaar, A. Perrig, and D. Song. Pi: A path identification mechanism to defend against DDoS attacks. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2003.
[45] A. Yaar, A. Perrig, and D. Song. An endhost capability mechanism to mitigate DDoS flooding attacks. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2004.
[46] Xiaowei Yang, David Wetherall, and Tom Anderson. A DoS-limiting network architecture. In *Proceedings of ACM SIGCOMM*, 2005, to appear.

A Notation

Table 1 summarizes the notation used in the paper.

Table 1: Notation used in this paper

n	Number of packets a sender can send during one time period
$\Delta\tau$	Timestamp resolution at a router
b	Number of bits each router uses in its marking
t	Threshold number of shares required for reconstruction in our secret sharing scheme
m	Total number of shares created in our secret sharing scheme
r	Number of routers along a path
τ_{init}	Duration of an initialization marking
τ_{cons}	Duration of a consolidation marking
T	A router's internal timestamp value
T_{init}	Timestamp in an initialization marking
T_{cons}	Timestamp in a consolidation marking
l_{init}	Number of bits in T_{init}
l_{cons}	Number of bits in T_{cons}
PC_i	Priority class i
$\text{MAC}_{\mathcal{K}}(M)$	MAC of M using key $\mathcal{K}$

B Stateless Router Rate Limiting

Developed by Estan and Varghese [11], multistage filters can identify the largest flows within a packet stream using only a constant number of memory references per packet and a small amount of memory, while providing an error rate inversely proportional to the size of memory, whereas the error rate for traditional sampling is inversely proportional to the square root of the size of the memory. Each stage, s, of a multistage filter consists of a set of counters and an independent hash function, h_s. Each counter is initially set to 0. When a packet arrives, the multistage filter hashes the flow id with each of the hash functions. It computes the minimum value over all of the counters selected by the hash functions, and increments any counter equal to the minimum value. If the minimum value exceeds a certain threshold, we add the flow to a small pool of flows whose usage is tracked precisely (i.e., we allocate one counter per flow for the suspicious flows). Since all of the packets for a given flow must hash to the same counter in each stage, we are guaranteed never to miss a large sender (i.e., no false negatives). There is the potential for false positives, but by incrementing only the minimum-valued counters, we significantly reduce the probability of errors, and the precision of the individual counters will help eliminate most of the remaining errors. Periodically, we reset the hash counters to 0, but we continue to track any of the flows in the individual counter pool that exceed a second threshold. This prevents an attacker from sending all of his or her traffic immediately before the counters reset.

C Derivation of Secret Sharing Parameters

In the FANFARE system, a flow can send up to n packets per time interval, where n is dictated by the sender's priority class. Recall that when a router calculates a marking for a CONS-REQ packet, it creates m shares of the marking, and it randomly selects one of the shares to insert into the packet. The receiver must accumulate t distinct shares to reconstruct the marking (and thus provide the sender with a valid capability). If α packets reach the recipient, the probability of receiving exactly k distinct shares is [12]:

$$P[\text{exactly } k \text{ distinct shares after } \alpha \text{ packets}] = p_m(k,\alpha) = \binom{m}{m-k} \sum_{i=0}^{k} (-1)^i \binom{k}{i} \left(1 - \frac{m-k+i}{m}\right)^{\alpha} \tag{8}$$

Hence, the probability that the receiver obtains at least t distinct shares after receiving α packets is:

(9)
$$P[\text{At least } t \text{ distinct shares after } \alpha \text{ packets}] = P_m(t,\alpha) = \sum_{j=t}^{min(\alpha,m)} p_m(j,\alpha)$$
$$= \sum_{j=t}^{min(\alpha,m)} \binom{m}{m-j} \sum_{i=0}^{j} (-1)^i \binom{j}{i} \left(1 - \frac{m-j+i}{m}\right)^{\alpha}$$

If we assume independent packet loss with probability q, the probability of receiving a capability, P_C, is the sum over the product of the probability that at least l packets arrive at the recipient given the packet loss and the probability that those l packets suffice to reconstruct the marking for a given router, i.e.,

(10)
$$P_C = \sum_{l=t}^{n} \binom{n}{l} (1-q)^l q^{n-l} P_m(t,l)$$

D Compensating for Round Trip Time

When a sender using FANFARE requests an initializer, the sender does not receive a reply for one round trip time *rtt*. If each router includes the currently applicable initializer, the sender only has τ_{init} − *rtt* to send the packets necessary to accumulate the necessary consolidators. To prevent senders with long round trip times from being disadvantaged, we allow a sender to include in its initializer request an estimated round trip time *ertt*. After ensuring that *ertt* falls within a reasonable range (for example, less than 500 milliseconds), routers forwarding such an initializer request place in the packet an initializer that is valid after time *ertt*.

A sender choosing too large an *ertt* will receive an initializer which is not yet valid; however, this is self-correcting for two reasons. First, the sender can measure the actual round trip time of the initializer request and delay the use of the new initializer until the estimated round trip time has elapsed. Secondly, even if the sender begins early usage of the initializer, it will be rejected by an intermediate router if that router has not yet begun to use that initializer.

E Additional Related Work

QoS. The IETF has defined two architectures to provide QoS in the Internet: Intserv [7] and Diffserv [6]. Intserv provides fine-grained per-flow QoS guarantees, while Diffserv is more coarse-grained and operates on aggregates. FANFARE provides support for implementing Diffserv in a robust and secure fashion, even in malicious environments, while preventing malicious flows from flooding even if the bottleneck link is a legacy link. FANFARE can also be used to secure an approximate version Intserv, which provides stronger QoS guarantees, without requiring per-flow state on routers, provided that all routers on an Intserv path deploy FANFARE.

Congestion Avoidance and Control. Many researchers have studied the challenge of congestion avoidance and control, and seminal work was done on TCP [5, 19], RED [14], ECN [34], CSFQ [40, 41], and XCP [24]. TCP, ECN, and RED[4] assume that the sender cooperates with congestion control, so a malicious sender can still flood a victim even with these mechanisms. XCP and CSFQ cannot provide congestion control on a bottleneck link if that bottleneck is between two legacy routers. Under some of these schemes, even a malicious client can entice a server to send large quantities of data, potentially causing congestion [35].

[4] In RED, a malicious sender merely experiences a droptail queue of nondeterministic length.

Policing. To cope with malicious senders or sender-receiver pairs, Floyd et al. propose an infrastructure-based approach in which each router can detect and limit "unfriendly" flows [13] by calculating short-term statistics based on its scheduling queue. An alternative approach by Estan et al. [11] proposes an efficient mechanism for detecting high-rate flows. However, neither work provides protection for legacy links in times of congestion. One of the main goals of our work is to prevent two collaborating malicious endhosts from flooding the path between them using priority flows, even if the bottleneck link is across a legacy link. The idea of requiring receivers to prove reception of traffic to prevent oversubscription was proposed by Gorinsky et al. [16] in the context of multicast, and by Savage et al. [35] in the context of TCP Daytona. In a scheme similar to our secret sharing approach for detecting congested links, Gorinsky leverages forward error correction for detection. Recently, Byers et al. [8] proposed forward error correction for efficient multicast encryption as well.

DoS/DDoS Defense Mechanisms. To defend against DoS and DDoS attacks, researchers have proposed server-based, network-based, and server- and network-based countermeasures. To defend against TCP SYN flooding and IP spoofing attacks, several server-based filtering techniques have been proposed [21, 23, 27, 36]. Researchers have also proposed computational puzzles to slow down the attack rate [4, 10, 22, 43], but Gligor shows that these approaches are not sufficient to guarantee an upper bound on the maximum waiting time [15]. Other researchers propose a network-based filtering infrastructure, for example Mirkovic et al. [31], and Estan and Varghese [11]. Combinations of server-based and network-based DDoS defense mechanisms include the Pushback framework by Mahajan et al. [30, 18], and the path identification approach to detect spoofed source IP addresses by Yaar et al. [44]. Keromytis et al. present an overlay infrastructure to defend against DDoS attacks [25], and Andersen generalizes the architecture [2].