

## Chapter 6

# Furies: Malicious Flow Detection via Aggregate Policing

As described in [22], a control architecture for provisioning network resources is essential for ensuring end-to-end Quality-of-Service (QoS) for IP-based latency sensitive applications. In Chapter 4, we proposed a Clearing House architecture to coordinate bandwidth allocation within and across domains to achieve statistical delay and packet loss bounds. Chapter 5 discussed how the CH-nodes adapts intra- and inter-domain aggregate reservations based on Gaussian predictors. In that chapter, we also presented the Traffic-Matrix based Admission Control (TMAC) scheme that takes into account network-wide traffic distributions within an ISP.

In addition to bandwidth reservations and admission control, an equally important resource control task is *traffic policing*, i.e., verifying that each admitted flow uses only its allocated share of network resources, and does not lie about its bandwidth requirement. This chapter focuses on malicious flow detection and traffic policing mechanisms, as shown in our thesis roadmap (Figure 6.1). We have designed a new control framework, called Furies<sup>1</sup>, for

---

<sup>1</sup>Furies is the Roman name of the three Greek goddesses responsible for tormenting evildoers.

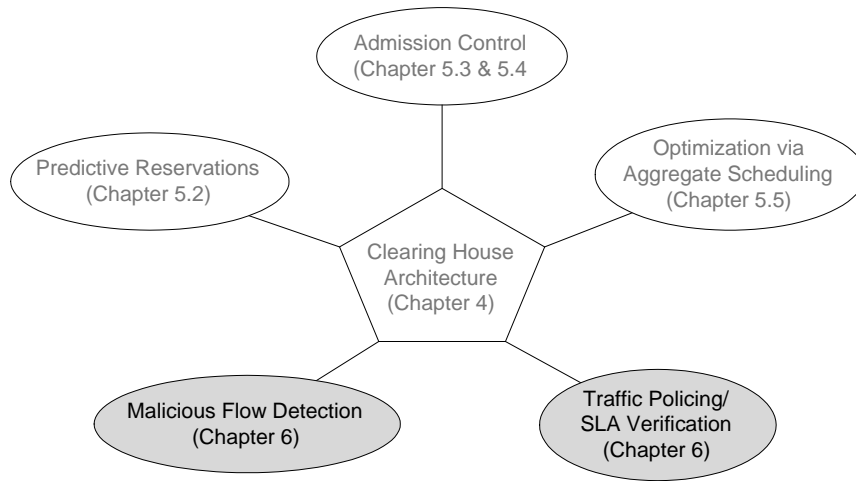


Figure 6.1: Thesis roadmap: The CH architecture and its various resource control mechanisms.

detecting and policing malicious flows without having to keep per-flow state information at any edge routers. The words “malicious” and “misbehaving” are used interchangeably in our discussions to describe an admitted flow that violate its allocated share of bandwidth.

We show the scalability and the practicality of Furies through simulations, a prototype, and an implementation. Section 6.1 provides a brief overview of the design challenges, our contributions and evaluation methodology. Section 6.2 highlights the key features of Furies, and discusses how we arrive at several design choices based on our understanding of current Internet infrastructure. We also discuss how Furies can be extended to police inter-domain traffic subject to SLAs, but our main focus is on detecting individual flows.

Malicious flow detection (MFD) is an example of on-line change detection problems [116], in which one needs to detect the occurrence of abnormal traffic behavior as soon as possible, but with a low rate of false alarms. Section 6.3 describes the detection algorithm, called *Malicious flow Detection via Aggregate Policing* (MDAP), that we have developed. The design of MDAP is driven by two goals: (a) to protect the well-behaved flows against resource depletion due to malicious activity, and (b) to identify and eventually penalize the malicious flows. Another desirable property is robustness with respect to noise

conditions and errors in workload modeling. The simulations discussed in Section 6.4 are designed to evaluate the effectiveness and robustness of MDAP. We also study the trade-offs between different performance criteria by tuning the parameters of MDAP.

Section 6.5 provides a brief description of our implementation of Furies based on Click Router [24], and demonstrates that the processing overhead that Furies introduced at an edge router is insignificant. Deployment issues are addressed in Section 6.6. Section 6.7 summarizes key results presented in this chapter.

## 6.1 Introduction

### 6.1.1 Motivation

Designing a control framework that performs scalable traffic policing in the edge networks faces numerous challenges. The most important is scalability. The overhead imposed by implementing the policies, such as amount of states maintained and processing time, should be bounded as the number of flows grows, i.e., should scale at most linearly with the number of flows. Another important requirement is compatibility with the existing Internet architecture. By compatibility, we mean that the proposed policies should be incrementally deployable and be able to reuse rather than replace the primitives supported by the existing network. The scalability, performance and deployment issues that might affect the design of a MFD scheme are not well understood. We attempt to bridge this gap in our work.

### 6.1.2 Traffic Policing and Malicious Flow Detection

After a high-priority flow is admitted, it is important to verify that this flow uses only its allocated share of network resources. A natural approach is to monitor every admitted flow at its ingress router, i.e., the entry point to a network domain. *Traffic policing*

in the Diff-Serv literature usually refers to parameter-based packet filter mechanisms, which are useful in tracking and shaping per-flow usage. However, this requires every edge router to maintain per-flow information and incurs significant processing overhead, leading to poor scalability. In this chapter, *policing* refers to monitoring aggregate groups of admitted flows and detecting specific groups that violate their total allocated bandwidth. The ultimate goal is to uniquely identify the malicious flows within these detected groups. It is crucial to detect and penalize these malicious flows because they can potentially cause other flows that share the same link to suffer from congestion, resulting in delays and packet losses.

### 6.1.3 Performance Indexes

In general, the four intuitive performance indexes for designing and evaluating an on-line change detection system [116] are:

1. probability of *false alarms* (i.e., a flow is detected as malicious when it is not),  $P_{fa}$
2. probability of *non-detection* (i.e., a malicious flow is not detected),  $P_{non}$ ,
3. delay for detection,  $t_{det}$ , and
4. magnitude of *detected change* (e.g., how much do malicious flows exceed their allocated bandwidth).

The choice of the parameters for a detection algorithm usually involve a set of trade-offs among these performance indexes. For example, if we choose a large value as the threshold for detection, the probability of false alarms decreases, but the probability of non-detection increases. Therefore, these indexes may be in different order of importance depending on the applications.

Since the main goal of this dissertation is to deliver end-to-end QoS assurance to latency sensitive applications, the first criteria is to protect the well-behaved flows that use

legitimate amount of resources against malicious behavior. Identifying and penalizing the malicious flow itself is secondary, as long as the impact of the malicious activity of undetected flows on other well-behaved flows is negligible. This implies that malicious flows that cause the most damage should be detected as soon as possible and penalized, e.g., through packet drops, while smaller malicious flows are tolerable even if they are not identified. The exact magnitude of violation (detected change) is not important. To quantify how well the performance of well-behaved flows is preserved against malicious activity, we define  $P_{\text{mis}}$  as the probability of incorrectly dropped packets from the good flows. In summary, an ideal MFD algorithm in this case should achieve the following (in the order of importance):

- close to zero  $P_{\text{fa}}$ ,
- close to zero  $P_{\text{mis}}$ ,
- maximum possible successful detection probability,  $P_{\text{fa}}$ , (or  $1 - P_{\text{non}}$ ), and
- small  $t_{\text{det}}$ ,

#### 6.1.4 Our Contributions

We propose a new control framework, Furies, that can be deployed by an Autonomous System (AS)<sup>2</sup> to detect malicious flows in an efficient and scalable manner. Our approach exploits the Internet’s hierarchical structure and the economic relationships between Internet Service Providers (ISPs) to form special grouping of flows for aggregate policing. Within the CH-architecture, Furies resides in the core of a Local Clearing House (Section 4.4: Figure 4.7) as the main resource controller. An LCH can also host other logical entities that provide services such as security and billing, but a detailed analysis of those entities is out of the scope of this dissertation.

---

<sup>2</sup>An AS is a network domain administered by a single organization, e.g., an ISP, a transit provider, a campus or corporate network.

The main contribution of this chapter is designing, developing and evaluating MDAP (*Malicious flow Detection via Aggregate Policing*), a mechanism within the Furies framework that detects malicious behavior through intelligent coordination of edge routers and aggregate policing. MDAP requires edge routers to maintain only a small amount of aggregate state information, and chooses its parameters based on the performance criteria listed at the end of Section 6.1.3 (as for an ideal MFD algorithm). Two other desirable properties that we consider are robustness and minimal processing overhead. Since MDAP does not require a priori knowledge of the individual flow characteristics, we evaluate its sensitivity to modeling errors in Section 6.4 by considering a variety of source models. We also simulate different extreme scenarios to investigate whether MDAP is robust with respect to noise conditions, e.g., when large and small flows are aggregated together for policing, or when the percentage of malicious flows are varied. For MDAP to scale well with a large user population, its processing overhead should be minimal. We study this issue through implementation in Section 6.5.

In our model, the source provider's network is responsible for monitoring the admitted flows and detecting malicious behavior. The subsequent ASs (ISPs or transit providers) only attempt to verify whether the inter-domain traffic from their upstream peers or customers adhere to their respective Service-Level Agreements (SLAs) [10]. An SLA specifies the expected level of service offered by a provider to the customer traffic it carries, e.g., maximum delay or packet loss guarantees, provided that the customer does not exceed the total subscribed bandwidth.

The key insight behind Furies is a coordinated way of assigning a flow-identifier, *Fid*, to every admitted *flow*, which allows aggregation of flows for traffic policing without compromising the ability to uniquely identify a flow if it is malicious. Each *Fid* has two subfields: *FidIn* and *FidEg*. At ingress routers, admitted flows are aggregated based on their *FidIn* for group policing. Similarly, egress routers police flows with the same *FidEg*

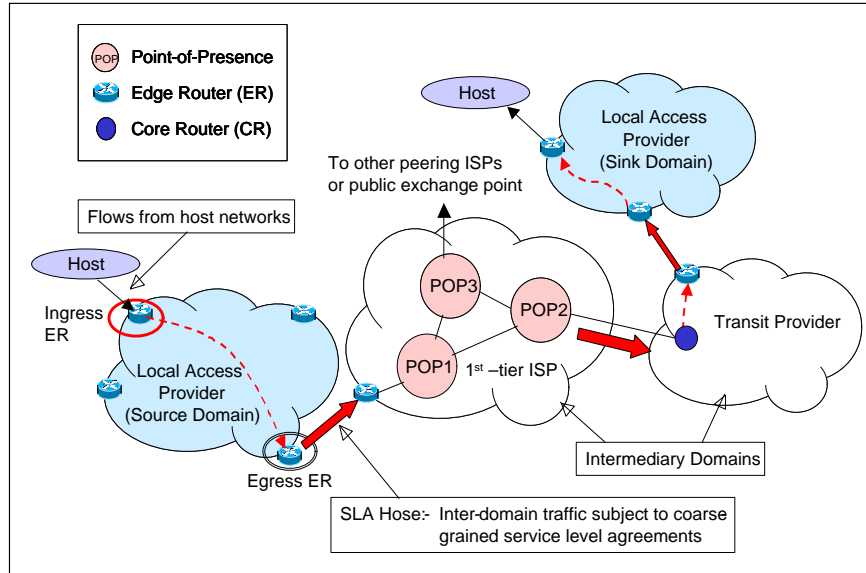


Figure 6.2: An example logical map of the Internet infrastructure.

as an aggregate. The fact that each edge router maintains only the aggregate state for each *group*, identified by *Fid* subfields, is crucial for the reduction of state from  $O(n)$ , which would be required if each flow were policed individually, to  $O(\sqrt{n})$ , where  $n$  is the number of admitted flows.

Furies does not strive to provide hard end-to-end guarantees but rather statistical QoS, as provided by soft real-time services. We assume that only the high-priority traffic needs resource reservation and are admission controlled. Furies can co-exist with any Measurement-Based Admission Control (MBAC) [62] algorithms previously proposed. For evaluation purposes, we use the Measured Sum (MS) algorithm [63]. A detailed comparison of the different MBAC algorithms is out of the scope of this chapter.

## 6.2 Design Rationale

Our design decisions are influenced by the inherent Internet hierarchy as described in Chapter 4.2.1. Figure 6.2 illustrates the logical relationship between host networks, LAPs,

transit providers and ISPs. For ease of discussion, we introduce the following definitions:

- A **flow** refers to a high-priority stream identified by its source and destination IP addresses.
- A **source domain** refers to the first AS (usually LAP or ISP) that a flow from a host network is routed to.
- A **hose** is a collection of flows that crosses from one domain to another, e.g. between two peering ISPs, from LAPs to transit providers or to first-tier ISPs.
- An **Ingress ER** is the edge router where a flow enters a domain and an **Egress ER** is where it exits.

Furies attempts to preserve end-to-end performance assurance by auditing the high-priority traffic that is admitted into a domain. The following characteristics are responsible for the scalability and robustness of our architecture.

### 6.2.1 Furies Service Model

We treat the traffic coming from another provider's domain differently from the individual flows from host networks, because resource allocation decisions for these cases happen in vastly different time-scales and granularity. In the former, the transit traffic (hose) is usually subjected to peering agreements or SLAs [10] that reflect aggregate traffic performance (e.g., average packet loss) and stay effective over long time-scale, e.g., weeks or months. In the latter, the reservation requests from individual flows usually need fast resource control decisions, e.g., within milliseconds, and finer-grained performance guarantees.

We assume that the source domain (e.g., the LAP shown in Figure 6.2) is responsible for policing individual flows from host networks and ensuring that the aggregate traffic

(hoses) entering the subsequent ISP domains do not violate the associated SLAs. In other words, it is important for the source domain to identify and stop malicious flows before they leave its network. The subsequent ISPs will treat these flows as part of a “hose” coming from the source domain and police them as an aggregate.

Furies uses the “core-stateless” principles [54], in that our architecture differentiates between edge and core nodes. While edge nodes do perform per flow management, core nodes do not and therefore can be efficiently implemented at high speeds. This significantly reduces the implementation complexity since no state information is maintained in the core. Our architecture builds on many of the existing Diff-Serv primitives, including:

**Packet marking and classification** The Diff-Serv Code Point (DSCP) [15], which is the first six bits in the TOS byte in the IP-header, is marked to differentiate packets from different classes of applications, e.g., high-priority voice flows vs. best-effort data traffic. Bits in the TOS octet are set at the network edges or administrative boundaries. Core routers simply classify the packets into different queues based on their DSCP values.

**Expedited forwarding PHB** The granularity of service provisioning is a “class” in Diff-Serv, and multiple flows with the same DSCP value are mapped on to a single per-hop behavior (PHB) at a router. We consider expedited forwarding PHB [18] because it is appropriate for applications that require a hard guarantee on the delay and jitter, such as VoIP. It can be implemented using a priority scheduler that always schedule packets from high-priority queue first whenever it is not empty.

**Traffic policer** In Diff-Serv, the Traffic Conditioning Agreement (TCA) [15] provider and the customer may specify that packets submitted for a certain service level (as specified by the DSCP) and are deemed to be non-conforming may be re-marked to a lower service level. This remarking is performed by a traffic policer. The simplest form of

policing is dropping the packets, which is sufficient most of the time.

## 6.2.2 Flow-Identifiers and Group Policing

Since a typical ER in a local POP can observe up to 5000 [97] simultaneous flows, per-flow policing is not ideal because it incurs a huge processing overhead. Instead, we propose to aggregate flows for group policing. To be able to uniquely identify a malicious flow, Furies assigns each admitted flow a unique 32-bit flow-identifier, *Fid*, which is inserted in the packet header. This *Fid* is explicitly assigned by Furies and is not assumed as a random number by the flow. Every *Fid* consists of two 16-bit sub-fields: *FidIn* and *FidEg*. All flows that enter or exit at a particular ER are aggregated into different groups based on their *Fids*. Each of these groups is identified with a unique group-identifier. The *FidIn* and *FidEg* subfields of a flow refer to the group identifiers used by its ingress and egress ER, respectively.

Furies aggregates all flows that share the same subfield *FidIn* (or *FidEg*) and polices them as a group at the associated ER. Every flow is policed at both its ingress and egress ER in two distinct groups, thereby increasing the chances of detecting malicious flows. Every ER maintains only the aggregate state for each group and hence does not store any per-flow state.

## 6.2.3 Assumptions

We make the following assumptions in our design:

- All routers can support two QoS classes: Best-effort and High-priority. From Chapter 3.2.2, we found that the perceived quality of VoIP application is satisfactory if the packet loss rate is less than 1% and per hop queuing delay is minimal (less than 5 ms).

We use this as a guideline for the performance requirement of high-priority class.

- For any particular flow that enters a domain, we can infer the associated egress ER from the underlying routing protocol. We do not modify any routing decisions.
- Packets that violate traffic profiles can either be dropped or re-marked to lower priority levels at the traffic policers. Our architecture can support both cases equally well, but for simplicity, we choose packet dropping as the default.
- Furies requires explicit REQUEST and TEARDOWN messages for admitting and releasing every flow. The signaling messages are generated by either the customer router or a proxy and sent as UDP packets at the same level of priority (high).
- We assume the reservation agent or proxy that issues the REQUEST message (on behalf of the host) is also responsible for inserting the assigned 32-bit *Fid* in the packet header if the flow is admitted.

## 6.3 Furies Architecture and MDAP Mechanisms

In this section, we provide a detailed description of our algorithms for malicious flow detection.

### 6.3.1 Components of Furies

Furies adds two components to edge routers (ERs) to implement its policies: Traffic Monitors (TMs) and Traffic Policers (TPs). Each ISP domain has a Resource Manager (RM) that interacts with the ERs continuously to admit new flows and coordinate edge policing.

The RM is a logical unit that can be physically placed at the fault monitoring point or policy server in an ISP. The RM maintains the repository of assigned *Fids* and their allocated bandwidths in the Fid-Repository (FR). It also contains the admission control policy. Figure 6.3a illustrates how the control messages flow between the various Furies components. When a new REQUEST message arrives at ER-s, it is forwarded to the RM,

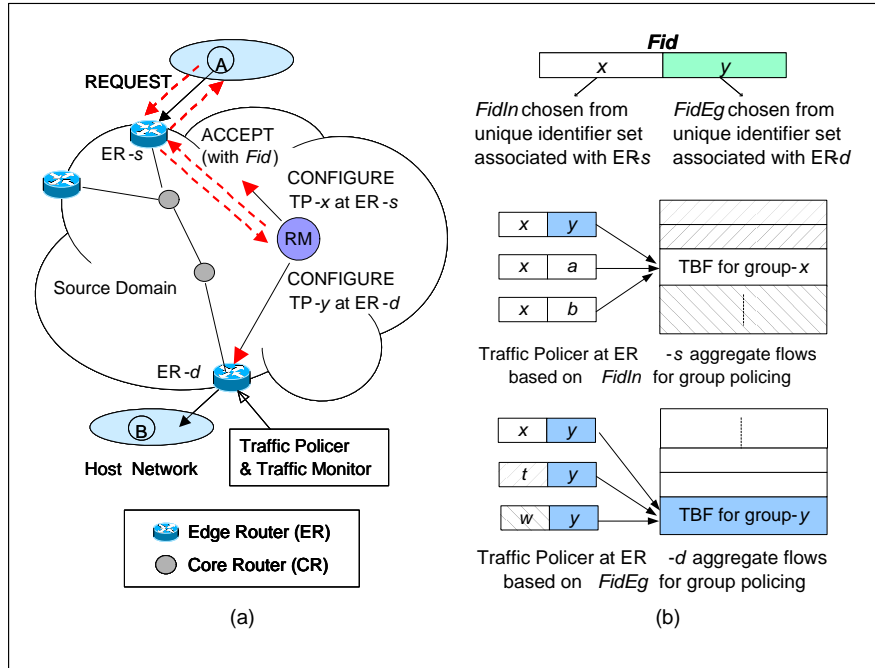


Figure 6.3: (a) Components of Furies. (b) MDAP mechanisms.

which performs admission control and assigns an *Fid* if admitted. Upon successful admission, the RM then sends an ACCEPT message along with the *Fid* back to the host. It also updates the traffic policers at the associated ingress and egress ER using the CONFIGURE message. Otherwise, the RM sends a REJECT message.

A Traffic Monitor (TM) at each ingress ER passively measures the rate of admitted traffic and updates the RM periodically. These updates are used by the RM for measurement-based admission control (Chapter 5.3). A Traffic Policer (TP) is introduced at each ER to police groups of flows identified by subfields of their *Fids*. In the example shown in Figure 6.3b, the new flow is assigned an *Fid* with the first subfield, *FidIn* equals to  $x$ , and *FidEg* equals to  $y$ . At ER-s, the new flow is aggregated with other flows with  $FidIn = x$  for group policing. At ER-d, this flow is grouped with other flows with  $FidEg = y$  for policing. Figure 6.4 highlights the major control blocks within the RM, which will be discussed in detail in the following section.

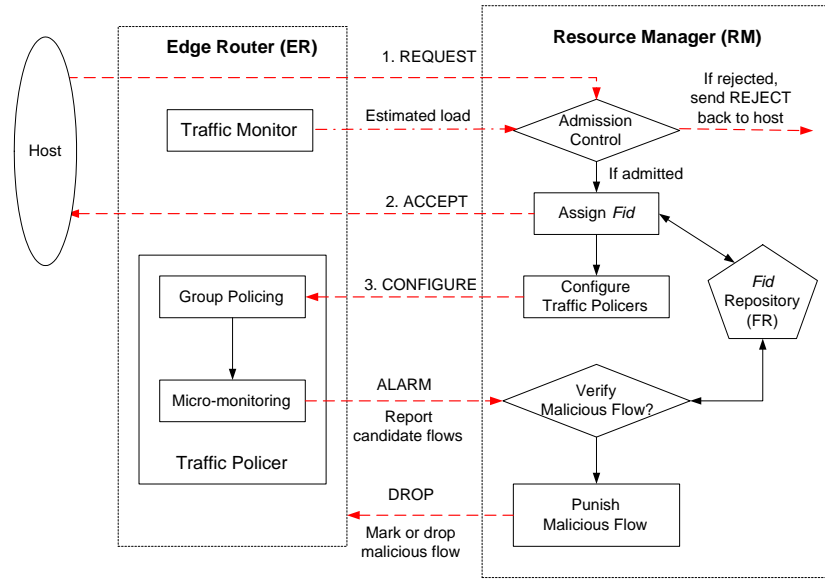


Figure 6.4: The logical flow of control messages between an edge router (ER) and a Resource Manager (RM).

### 6.3.2 Fid Assignment and Releasing

Furies assigns each edge router ER- $i$  a set of  $M$  unique group identifiers, denoted as  $\mathcal{A}_i = \{x_{i1}, x_{i2}, \dots, x_{iM}\}$ , where each member is a 16-bit binary number and is unique across the set  $\mathcal{A}_i$ . The sets  $\mathcal{A}_i$  and  $\mathcal{A}_j$  associated with any two ER- $i$  and  $j$  are mutually disjoint.

Any *Fid* of an admitted flow should satisfy the following properties:

1. If the flow is routed from ER- $s$  to ER- $d$ , then  $FidIn \in \mathcal{A}_s$ , and  $FidEg \in \mathcal{A}_d$ .
2. No other flow should have the same *Fid*.
3. Flows with the same *FidIn* (or *FidEg*) have similar bandwidth requirements.

When a new flow from ER- $s$  to ER- $d$  is admitted, the RM picks a random  $x_s$  from  $\mathcal{A}_s$  and a random  $y_d$  from  $\mathcal{A}_d$  such that the *Fid* with  $FidIn = x_s$  and  $FidEg = y_d$  satisfies the above properties. This *Fid* is assigned to the flow, and a new entry with this *Fid* and its allocated bandwidth is added to the Fid-Repository (FR). The total number

of flows that can be uniquely identified in this scheme is  $K \cdot M^2$  for a particular ingress ER, where  $K$  is the total number of potential egress ERs, each having its own unique set of identifiers. We assume the admitted flow will send a TEARDOWN message to the ingress ER when it terminates. The TEARDOWN message contains the *Fid*, and its allocated peak rate. Upon receiving the TEARDOWN message, the RM updates the FR accordingly and releases the *Fid*.

### Demand for Fids

In [97], the authors built a passive monitoring infrastructure over their backbone to collect and analyze real Internet traffic. They observed that the typical number of simultaneously active flows on an ingress link (between an edge router and a core router) is between 300 and 5000. From the trends in application usage seen at the NASA Ames Internet Exchange [117], we estimate that about 10% of these will be latency sensitive applications such as streaming media and online gaming <sup>3</sup> Even if the demand for *Fids* increases 10 times in future, we need at most  $M = \sqrt{5000}$ , which is roughly 71 unique identifiers per  $\mathcal{A}$  set. Based on the discussion in Section 6.2<sup>4</sup>, the total number of ERs within an ISP,  $K$ , is in the order of 150-500. Therefore, the total number of unique identifiers required for the whole ISP ( $M \times K$ ) is roughly 35,500 in this case. Allocating 16-bits ( $2^{16} = 65536$ ) for each subfield should be more than sufficient for producing mutually disjoint  $\mathcal{A}_i$  for all routers.

If the demand ever exceeds the available *Fids*, we can either (a) increase the size of *Fids*, or (b) allow a small probability of collisions by recycling used *Fids*. A comprehensive analysis of malicious flow detection with *Fid* collisions is part of our future work.

---

<sup>3</sup>Results in [117] are based on analysis of Internet traffic trace data collected at Ames Internet Exchange (AIX) over 10 months, from May 1999 through March 2000. On average, the fraction of Real Audio Traffic packets seen at AIX is between 0.5-6% of the total traffic, while the fraction of online gaming traffic is between 0.5-4%.

<sup>4</sup> $K = \text{number of POPs} \times \text{number of ERs/pop}$ .

### Explicitly Assigned vs User-Selected Fids

In our approach, we attempt to maximize the level of flow aggregation without compromising the uniqueness of *Fids*, thereby minimizing the number of groups to be policed at every ER. An explicit assignment of *Fids* can achieve this since the RM can keep track of the availability of individual *Fids* and allocate unused ones to new flow requests. For example, if there are  $n$  flows from an ingress ER to a specific egress ER, an explicit assignment can preserve uniqueness by maintaining only  $\sqrt{n}$  groups at each of the two routers. It also allows the aggregation of flows with similar bandwidth requirements into a common group for policing to increase effectiveness of group policing.

On the other hand, if flows were allowed to assume their own *Fids*, then it would be necessary to maintain a membership function to map the random *Fids* to a particular group. The cost of performing an online grouping of flows based on these functions would be very high because the *Fids* are continuously changing. Techniques like Stochastic Fair Blue (SFB) [75] that use random hash functions cannot be applied because they do not provide an inverse mapping from group identifiers to actual *Fids*. Thus, using SFB alone does not provide a direct mechanism to verify whether a suspected flow is truly misbehaving.

### 6.3.3 Group Policing

Furies deploys a set of Token Bucket Filters (TBF) [118]) at both ingress and egress ERs for policing the traffic generated by admitted flows with the matching group identifiers (*FidIn* or *FidEg*). Every group identifier,  $x \in \mathcal{A}_i$ , is associated with a TBF with two parameters,  $r_{\text{tot}}$  and  $b_{\text{tot}}$ , where  $r_{\text{tot}}$  is the total average rate of admitted flows and  $b_{\text{tot}}$  is the total burst size. When a new flow between ER- $s$  and ER- $d$  is admitted, the RM sends a CONFIGURE message that specifies the *FidIn* and *FidEg* of the admitted flow to the ingress ER- $s$  and egress ER- $d$ , respectively, along with its average rate  $r$  and burst-size  $b$ .  $\text{TP}_s$  and  $\text{TP}_d$  will then update the the TBF with the matching *FidIn* and

*FidEg* accordingly. Packets that violate the associated traffic profile are discarded. Each TP keeps track of the dropped packets and reports the statistics to the RM.

Since the policing at the TP is performed on a group of flows sharing the same 16-bit subfield of *Fid*, the amount of state information maintained at the ingress ER is proportional to  $M$ , the number of unique identifiers in the set,  $\mathcal{A}$ . Consider an example ISP domain with  $K$  edge routers and  $M=100$ . Each ER maintains only 100 pieces of state information, but an arbitrary router can admit up to  $K \times 10,000$  flows with unique *Fids*. A per-flow policing scheme would have require each ER to maintain all  $K \times 10,000$  states.

### 6.3.4 MDAP Detection Process

There are two stages in the MDAP detection process:

- Guess candidate flows within the group that violate aggregate bandwidth allocation, and
- Verify which of these flows are truly malicious.

#### Providing Best Guesses

As an example, let a flow with  $Fid = [f, g]$  be malicious. All flows with the same  $FidIn = f$  will be policed as an aggregate in the same Token Bucket at the ingress ER,  $TP_s$ , regardless of what their *FidEg* is. If the total allocated rate of  $FidIn = f$  is violated, the affected TP reports  $f$  to the RM using an ALARM message (Figure 6.4). However, this information alone is insufficient for pinpointing the exact misbehaving flow, because there can be as many as  $K \cdot M$  flows with the same *FidIn* that could potentially be malicious. If the TP at an egress ER  $FidEg = g$  also sends an ALARM message, the RM guesses that a flow with *Fid* that contains both  $f$  and  $g$  as its subfield is malicious, and submits this *Fid* for verification.

However, a malicious flow may not always be detected at both its ingress and egress ERs. To improve the effectiveness of MDAP, we introduce a “micro-policing” mode. Whenever a group TBF that violates the aggregate rate is identified, Furies requires the edge routers to randomly sample individual flows within this group for a duration of  $t_{mp}$ . At the end of this period, the associated ER identifies  $n_{mp}$  largest flows, and report their *Fids*, along with the sampled peak rate, to the RM. Normally the potentially malicious flows are the ones that transmit at a much higher rate relative to other members. The value of  $t_{mp}$  should be as small as possible to ensure short detection time ( $t_{det}$ ), but it has to be large enough to observe malicious behavior, especially if the flow is bursty. From our trace analysis (Chapter 3.2.3, [119]), we observed that the audio trace with the minimum activity cycle has mean silence period of 4.9 seconds. Using this as a rough guideline, we choose  $t_{mp} = 5$  seconds. On the other hand, the choice of  $n_{mp}$  depends on the relative number of malicious flows in the network. Large value of  $n_{mp}$  provides better chances of catching all the malicious flows (if they are many), but incurs higher processing overhead. We perform sensitivity analysis by varying the value of  $n_{mp}$  from 1 to 10 and found that the performance difference is negligible for  $n_{mp} \geq 5$ , given that 10% of the flows are malicious. We use  $n_{mp}=5$  in all our experiments.

### Verifying Malicious Behavior

For each reported flow with  $Fid = b$ , the RM compares the allocated rate,  $r_b$  with the measured peak rate  $m_b$  reported in the ALARM message:

$$m_b < (1 + \epsilon) \cdot r_b \quad (6.1)$$

where  $\epsilon$  is a hysteresis parameter to absorb transient behavior of bursty traffic. If the condition in (6.1) is violated, the flow is considered misbehaving.  $\epsilon$  is typically between 0 and 0.05. A counter associated with this flow is incremented for every such violation of condition (6.1). To reduce the probability of false alarm, we introduce a second hysteresis

parameter,  $\eta$ , which determines the minimum number of violations before a flow is reported as misbehaving. A reasonable range for  $\eta$  is between one and five.

Several actions can be taken against a detected flow, e.g., dropping all the packets of this flow, demoting all its packets to best-effort, or charging more for the connection. Such a penalty would require keeping some state information at the edge router, but only for a very small subset of flows that misbehave. The choice of the penalty function is dependent on the business goals of the providers, which will not be addressed in this dissertation. Instead, we focus on providing insights into the technical design of MFD scheme itself.

### 6.3.5 Policing of SLA Traffic

Service Level Agreements (SLAs) are contracts between service providers and customers, in which the providers stipulate their commitment to deliver numerous service levels to their customers with an agreed-upon pricing scheme. In return, the customers are bound to only use the service they pay for and will be penalized otherwise. For example, consider a customer that signs up for fractional T-1<sup>5</sup> line, say for 25%, which is equivalent to 386 Kbps. The customer is required by the SLA to keep the peak link utilization below 386 Kbps. Delivery of excess traffic beyond that is not guaranteed by the service provider.

#### SLA Performance Guarantees

Previously, SLAs have primarily focus on backbone performance such as reliability and availability, e.g., a typical SLA requirement for T-1 and frame relay service should be 99.93% availability or 60 seconds of disrupted service in 24 hours. In recent years, ISP competition has fueled stronger SLAs that specify packet losses and delay guarantees. For instance, Cable & Wireless is offering its dedicated Internet access customers a protective guarantee that they will experience an average latency of no more than 70 ms per month

---

<sup>5</sup>A T1 line can carry 24 digitized voice channels or it can carry data at a rate of 1.544 Mbps.

across the ISP's Internet backbone. It has also added a packet-loss protection guarantee of no more than 1% over one month. However, this implies that the customer packets can experience 1 full second delay for one day, and 35 ms for all the other 29 days in a month, and still average less than 70 ms per month. It would be impossible for the customer to run any real-time audio or video applications for that particular day where the delay is greater than the acceptable range for satisfactory perceived quality (Chapter 3.2.2). With the increasing diversity of Internet applications, there is a strong need for the ISPs to provide finer-grained performance guarantees that reflect application-level requirements.

### SLA Traffic Policing

SLA validation tools are currently available (e.g., from VisualNetworks<sup>6</sup> and TeleChoice<sup>7</sup>) for business users to measure the latency, packet loss and network availability, and to validate whether the service providers meet the SLAs. One area where all ISPs are lacking is in real-time monitoring tools to verify whether the customers abide by the SLAs in terms of generating traffic. This is crucial, so that the ISPs can offer more specific SLAs with finer-grained performance assurance (as discussed earlier) through admission control, load balancing, and capacity planning.

Furies does not address how the new SLAs are specified or negotiated. Instead, we propose a way to police the SLA traffic that enter a domain. As a basic rule, Furies favors individual flows (as discussed in previous section) over SLA-traffic, i.e., SLA traffic has lower priority than flows with assigned *Fids*, and SLA packets will be remarked or dropped first when congestion happens.

We assume a domain  $k$  will issue an SLA request to reserve the required bandwidth,  $R_{kq}$ , to carry a “hose” from its domain to the neighboring domain  $q$ . For a domain  $q$ , each admitted SLA-hose from its neighboring domain  $k$  is assigned an identifier  $S_k^q$ . A traffic

---

<sup>6</sup><http://www.visualnetworks.com/>

<sup>7</sup><http://www.telechoice.com/>

profile consisting of two parameters:  $S_k^q$  and the allocated rate  $R_{kq}$  is maintained at the ingress ER- $i$  where the SLA traffic enters. All the packets from SLA-flows with the matching  $S_k^q$  are policed as an aggregate. If the sampled aggregate rate exceeds the allocated rate  $R_{kq}$ , the packets are marked non-conformant and dropped. A control message that indicates the violation is sent to the originating domain  $p$ .

### 6.3.6 Other Issues

In this section, we discuss how MDAP can be modified to cope with diverse traffic loads, untrusted networks, and changing routing policies.

#### Hiding in the Aggregate

Although group policing allows the Furies architecture to scale, it limits the effectiveness of MDAP because a malicious flow can “hide” in the aggregate. This can happen when:

- the aggregate usage of the group is less than the total allocated rate because certain flows are under-utilizing their resources or the percentage of over-utilization is less than the threshold  $\epsilon$ , and
- the malicious flow is relatively small compared to other larger, yet legitimate, flows in a misbehaving group.

To address this problem, Furies introduces redundancy by deploying TP at every egress point as well. By assigning a unique *Fid* to every flow, we ensure that no two flows are in the same group in both the associated ingress and egress ERs. Essentially every flow is policed in the aggregate at two distinct points to maximize the number of malicious flows that are detected. Secondly, we assign flows with similar bandwidth requirement into the same group (Property 3 of *Fids* in Section 6.3.2).

## Bogus Identifiers

It is possible for an external malicious user/application to create its own *Fid* which is valid but has not been explicitly assigned by the RM. We refer to such identifiers as *bogus Fids*. This problem can be easily solved by using a secure hash function with a secret key within an ISP. Assume that the RM and the ERs share a secret key,  $K$ , for a hash function  $h()$ . Given an  $Fid = [f, g]$ , the secure-*Fid* allocated to the flow is set to  $[h_K(f), h_K(g)]$ , from which  $f, g$  cannot be inferred without knowing  $K$ . The associated ingress (egress) ER can authenticate the flow by comparing the *FidIn* (*FidEg*) of the secure-*Fid* with the hashed values of valid group identifiers (e.g.,  $f$  and  $g$ ) of the ER.

## Routing Changes

When a routing change occurs and causes an active flow to change its ingress or egress points, the previously assigned *Fid* may not match the group identifiers in one of the new ERs or both. Whenever this happens, we can either (1) remark the packets of this flow as best-effort, or (2) contact the RM for re-admission of this flow with the new endpoints. Further investigation is needed to understand the the performance and security issues of both approaches. However, we believe such events are rare based on our discussions with two first-tier ISPs.

## 6.4 Simulation Study

The aim of the simulation study is to evaluate the performance and robustness of Furies. We use the ns-network simulator [89] to implement the basic mechanisms of Furies. The TP<sup>8</sup> is implemented as a connector in front of a node, and a time-window estimator is introduced at each input link to estimate the rate of existing flows. The admission control

---

<sup>8</sup>We modify the Diff-Serv module contributed by Sean Murphy, <http://www.teltec.dcu.ie/~murphys/ns-work/diffserv/index.html>.

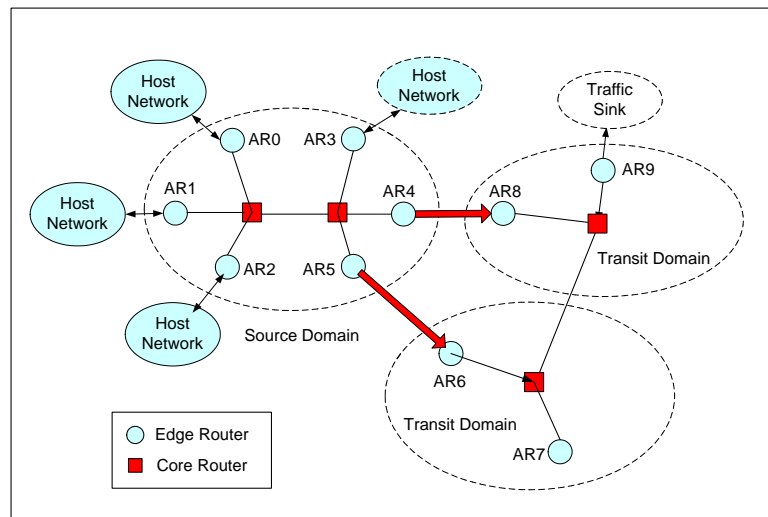


Figure 6.5: Simulation topology.

module is created as an `NsObject` and inserted before the ingress ERs. The various tasks of the RM in Furies are implemented at the Tcl-level. Our Furies-patch works for ns-2.1b6.

#### 6.4.1 Network Topology

We use ns to simulate a simple subgraph of the Internet topology as shown in Figure 6.5. It consists of 4 ASs (a source domain, two transit domains and one traffic sink) and captures the general properties of the Internet, including the POP structure of large ISPs and peering relationship with transit domains. Flows from host networks enter and exit the source domain through edge routers, AR0-AR5, where they are aggregated for policing using the MDAP scheme. The transit traffic (or hoses) that traverse from the source domain (at AR4 and AR5) to another two AS domains are subject to SLA policing at AR6 and AR8, respectively. All routers support priority scheduling and all control signaling between Furies-RM (not shown on the figure) and the ERs is carried in UDP messages.

### 6.4.2 Traffic Generation

Similar to Chapter 5.4, the arrival process of the admission-controlled traffic is modeled as Poisson with arrival denoted as  $\lambda_i(t)$ . We use the indices  $t$  to indicate the time-of-day dependence of the traffic demand as reported in [102] and [97]. To reflect the realistic traffic demand, we introduce  $\pm 10\text{-}15\%$  changes to  $\lambda_i(t)$  at a regular interval of 30 minutes. The traffic distributions from an ingress ER to a set of egress ERs are based on a random probabilistic model. We also analyze two extreme cases where: (a) the traffic is equally distributed to all egress ERs and (b) some egress ERs attract heavy traffic while the other egress ERs are relatively idle. Neither cases affect the results. This is not surprising since the design of MDAP is independent of the traffic distribution model.

Since MDAP does not assume a priori knowledge of source traffic we need to evaluate its robustness with respect to variation in workload characteristics. We use four kinds of traffic source models in our experiments: EXP1, EXP2, CBR and PARETO. In Chapter 3 we discussed the characteristics of each model and how it can be used to describe certain types applications. This information is summarized in Table 5.1, which can be found in Chapter 5.4.2.

### 6.4.3 Performance Evaluation

To quantify the effectiveness of the malicious flow detection (MDAP) scheme in Furies, we are interested in the following two events: successful detection and false alarm (Section 6.1.3). The probability of successful detection  $P_{sd}$  is approximated as the fraction of malicious flows that are actually detected. Similarly, the probability of false alarm  $P_{fa}$  is the fraction of normal flows that are incorrectly reported as misbehaving. Since the flows are policed as an aggregate, malicious flows can cause packets from complying flows to be dropped. The probability of a packet being incorrectly dropped, denoted as  $P_{mis}$ , quantifies the impact of malicious flows on end-to-end performance seen by other member flows. The

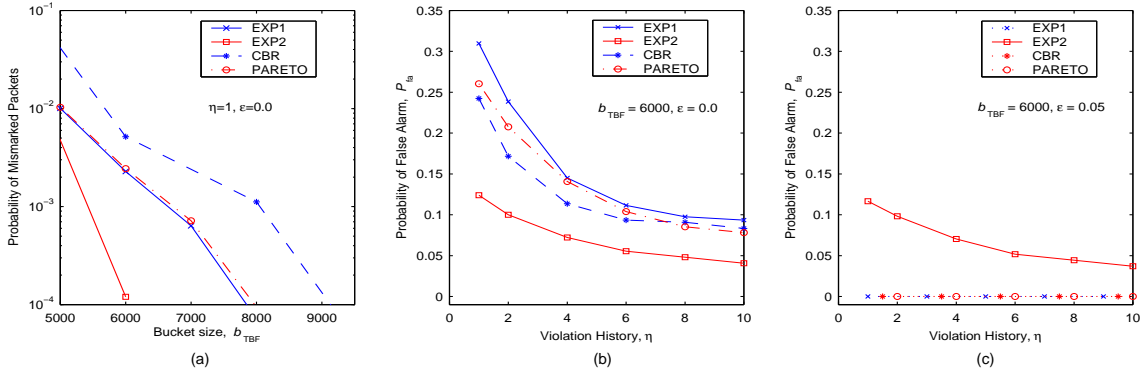
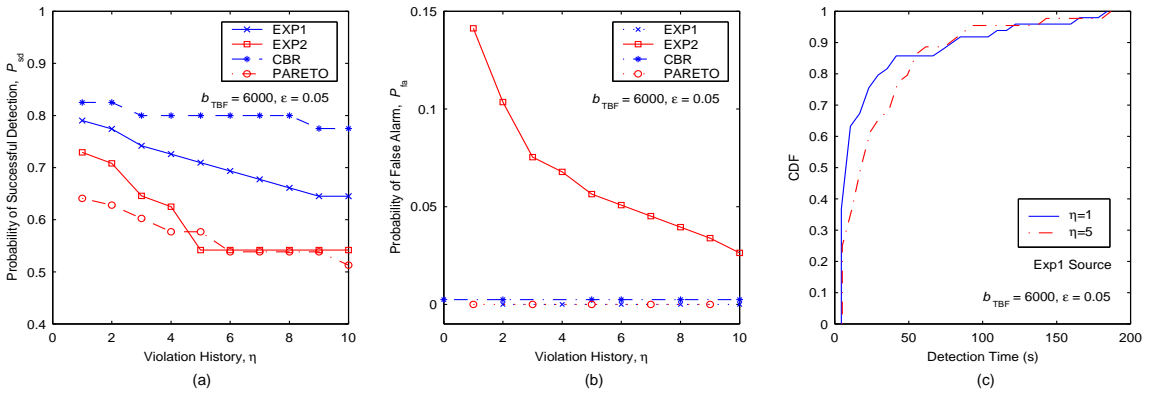


Figure 6.6: Case 1: Zero Malicious Flows.

Figure 6.7: Case 2: Many small homogeneous flows; a small fraction,  $\gamma = 0.1$ , misbehave.

goal of MDAP is to maximize  $P_{sd}$ , while minimizing  $P_{fa}$  and  $P_{mis}$ .

## Test Scenarios

To examine the robustness of MDAP, we simulate four extreme cases:

**Case 1:** This is the reference case with zero malicious flows.

**Case 2:** This arrangement is similar to Case 1, but now a small fraction,  $\gamma$ , of the flows are misbehaving.

**Case 3:** We assign *Fids* such that one large flow and many simultaneous small flows are grouped together for policing. The peak rate of the large flow is 10 times larger than

the peak rate of a small flow. All of the small flows are compliant, and only the large flow misbehaves.

**Case 4** Again, we consider a mixture of one large flow and many simultaneous flows like Case 3. However, the large flow is compliant this time, and a fraction  $\gamma$  of the small flows are malicious.

We repeat each experiment using four different source models: EXP1, EXP2, CBR and PARETO. For each scenario, the simulation was repeated 10 times with different random seeds, and the average  $P_{sd}$ ,  $P_{fa}$ , and  $P_{mis}$  was computed. Each simulation ran for 1000s. All experiments were performed under high load with 20% blocking probability. A malicious source requests allocation for  $r$  Kbps but sends traffic at a higher rate, randomly chosen between  $1.1 \cdot r$  and  $1.2 \cdot r$  Kbps (10-20% violation). The average and peak rate for each source model is the same as described in Section 6.4.

## Results and Discussions

**Case 1:** The experiments in Case 1 are intended for understanding the limitations of the MDAP and its performance sensitivity with respect to different choices of design parameters. Ideally, none of the flows should be reported as “misbehaving”, but the transient behavior of bursty traffic could momentarily overflow the Token Bucket Filters (TBFs) and be interpreted as malicious, leading to a “false alarm”. Figure 6.6a shows how the choice of bucket size,  $b_{TBF}$  at the Traffic Policers (TP) affects the probability of mis-marked packets,  $P_{mis}$ . A smaller value of  $b_{TBF}$  is more effective in detecting misbehaving flows, but there should be enough tokens to allow the legitimate packets to pass, and keep the  $P_{mis}$  low. Except for the CBR traffic,  $P_{mis}$  is below 0.01 for other source models. For the CBR source, increasing  $b_{TBF}$  to 6000 is sufficient to reduce  $P_{mis}$  to 0.005. The two hysteresis parameters  $\eta$  and  $\epsilon$  determine under what conditions a flow is reported as “malicious” (Section 6.3.4), but have no effect on  $P_{mis}$ .

We can relax the condition for MDAP by increasing  $\epsilon$  and  $\eta$ , and this helps to reduce the number of false alarms. Figure 6.6b and 6.6c study how  $P_{fa}$  varies as a function of  $\eta$  for  $\epsilon = 0.0$  and  $0.05$ . For a 0% tolerance level in MDAP (i.e.,  $\epsilon=0$ ),  $P_{fa}$  decreases gradually as  $\eta$  is increased. However, we notice that  $P_{fa}$  drastically decreases for all the source models when the tolerance level is increased to 5%. This indicates that  $P_{fa}$  is more sensitive to  $\epsilon$  than  $\eta$ . For the rest of the experiments, we choose  $\epsilon=0.05$  and  $b_{TBF} = 6000$ .

**Case 2:** Increasing  $\eta$  causes a delay in reporting misbehaving flows and may adversely impact the effectiveness of MDAP. We examine this issue in Case 2. We set the value of  $\gamma$  (fraction of misbehaving flows) to be 0.1. Figure 6.7a and 6.7b show the variation of  $P_{sd}$  and  $P_{fa}$  as  $\eta$  is increased from 1 to 10. The effect of  $\eta$  on  $P_{sd}$  for the CBR source is minimal. For the other source models,  $P_{sd}$  decreases sharply when  $\eta$  is increased and the rate of decrease varies across the source models. From Figure 6.7b, we can infer that only in the case of the EXP2 source model is  $P_{fa}$  sensitive to the value of  $\eta$ . With  $\eta = 1$ , we can detect most of the malicious flows with EXP1 (79%), CBR (83%) and PARETO (64%) sources with virtually zero  $P_{fa}$ . In the case of EXP2, there is a trade-off between maximizing  $P_{sd}$  and minimizing  $P_{fa}$  as we choose the value for  $\eta$ . This indicates that burstier sources are more difficult to detect.

We also measured the detection time for each correctly identified malicious flow and plotted the distributions in Figure 6.7c. With  $\eta = 1$ , the average detection time is 26.9 seconds, which is less than 1/10 of the average duration of a flow. 90% of the flows are detected within 78.9 seconds. When we increase  $\eta$  to 5, the average detection time increases to 33.8 seconds, which is still reasonably fast. The 90<sup>th</sup>-percentile detection time is 80.4 seconds in this case.

The simulations in Case 2 show the basic results of MDAP hold across different source

Table 6.1: Case 3: One large malicious flow and many small complying flows.  $\eta = 5$ ,  $b_{\text{TBF}} = 6000$ ,  $\epsilon = 0.05$ .

Source	EXP1	EXP2	CBR	PARETO
$P_{\text{sd}}$	1.0	1.0	1.0	1.0
$P_{\text{fa}}$	0.0077	0.027	0.012	0.0013
$P_{\text{mis}}$	0.003	0.00021	0.0072	0.0037

models. Although long range dependent traffic like PARETO is harder to detect, we can achieve a reasonable success rate (0.64) with zero false alarms. The presence of burstier sources, EXP2, pose challenges to the MDAP scheme, and we need to choose the value for  $\eta$  carefully to maximize  $P_{\text{sd}}$  while keeping  $P_{\text{fa}}$  reasonably small.

**Case 3:** We have so far considered only homogeneous flows with the same rates. In the next two cases, we consider an extreme case where there is one large flow with peak rate  $r_1$  and many small flows with peak rate  $r_s$ , where  $r_1 = 10 \times r_s$ . We model the small flows the same way as in Case 1 and repeat our experiments for four different source models. In Case 3, only the large flow is misbehaving. The results are summarized in Table 6.1. For all source models, we always successfully detect the misbehaving large flow and  $P_{\text{fa}}$  is less than 0.03.

**Case 4:** Case 4 addresses the scenario where misbehaving small flows “hide” in the aggregate with another large flow. The probability of a small flow being malicious is  $\gamma$ . Intuitively, we suspect that detection is harder in this case, because the misbehaving flows can “steal” the idle bandwidth allocated to the large flow. Since the traffic policer can only enforce the total allocated rate, the malicious flows may not be detected. Figure 6.8 shows the  $P_{\text{sd}}$  achieved for different values of  $\gamma$  and Table 6.2 summarize  $P_{\text{sd}}$ ,  $P_{\text{fa}}$  and  $P_{\text{mis}}$  for  $\gamma = 0.1$  and 0.5. Surprisingly, we notice that the  $P_{\text{sd}}$  achieved with  $\gamma = 0.1$  for EXP1, CBR, and PARETO sources are fairly close to the results in Case 2, where there is no large flow. But for EXP2, the success rate is significantly

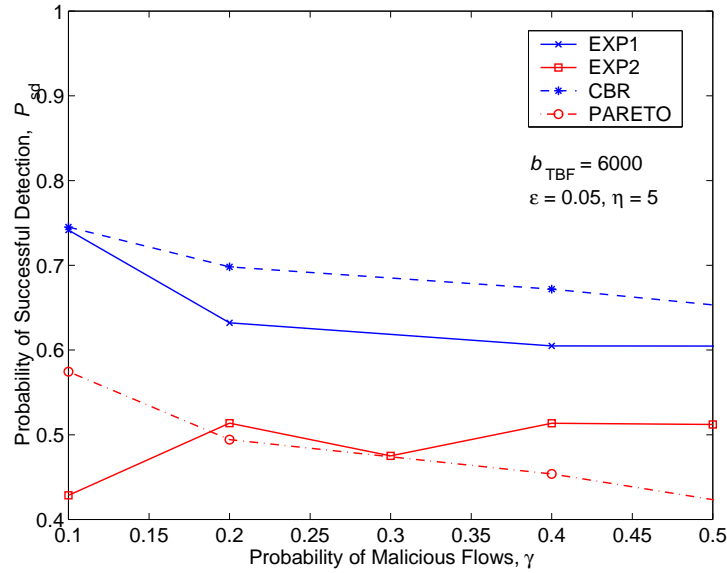


Figure 6.8: Case 4: One large flow ( $r_l$ ) and many small flows ( $r_s$ );  $\gamma$  of small flows misbehave.

smaller ( $P_{sd}=0.43$  in Case 4 as supposed to 0.54 in Case 1). When  $\gamma$  increases, the success rate  $P_{sd}$  decreases for EXP1, CBR and PARETO source models. With EXP2,  $P_{sd}$  fluctuates as we vary  $\gamma$ , and is actually higher at  $\gamma=0.5$  than  $\gamma=0.1$ . This is because the active cycle of EXP2 is very short (10%), and can easily go undetected if it coincides with the idle period of the large flow. However, when the fraction of malicious flows increases, there is an increased likelihood that some of the malicious flows will synchronize or overlap in their active cycles, leading to overflow of the TBF at the traffic policer. When the aggregate rate is violated, all the flows sharing the same subfield (*FidIn* or *FidEg*) will be monitored individually (micro-monitoring) and the malicious flow can be correctly identified. The probability of false alarms  $P_{fa}$  and mis-marked packets  $P_{mis}$  are negligible in this case across different values of  $\gamma$  and source models.

Table 6.2: Case 4: One large flow and many small flows.  $\gamma$  of small flows misbehave.  $\eta = 5$ ,  $b_{\text{TBF}} = 6000$ ,  $\epsilon = 0.05$ .

Source Model	EXP1	EXP2	CBR	PARETO
$\gamma = 0.1$				
$P_{\text{sd}}$	0.74	0.43	0.75	0.57
$P_{\text{fa}}$	0.00066	0.011	0.0	0.0
$P_{\text{mis}}$	0.0032	0.00016	0.0047	0.0028
$\gamma = 0.5$				
$P_{\text{sd}}$	0.61	0.51	0.67	0.39
$P_{\text{fa}}$	0.00067	0.025	0.0	0.0
$P_{\text{mis}}$	0.0030	0.00047	0.0088	0.0022

Table 6.3: Comparisons between heterogeneous and homogeneous source models:  $\gamma = 0.1$ ,  $b_{\text{TBF}} = 6000$ ,  $\epsilon = 0.05$ .

Source Model	HET	EXP1	EXP2	CBR	PARETO
$\eta = 1$					
$P_{\text{sd}}$	0.55	0.79	0.73	0.83	0.64
$P_{\text{fa}}$	0.0	0.0	0.14	0.0025	0.0
$P_{\text{mis}}$	0.0030	0.0028	0.00016	0.0079	0.0031
$\eta = 5$					
$P_{\text{sd}}$	0.55	0.71	0.54	0.80	0.58
$P_{\text{fa}}$	0.0	0.0	0.060	0.0025	0/0
$P_{\text{mis}}$	0.0030	0.0028	0.00016	0.0079	0.0031

### Further Sensitivity Analysis

So far, we have been considering flows with homogeneous source characteristics in our simulations. The next experiment uses a random mixture of the four different source models (EXP1, EXP2, CBR and PARETO), each with different peak rates, idle times and burst times. Each arriving flow chooses among these source models at random. We repeat the experiment in Case 2, with  $\eta=1$  and 5 using heterogeneous flows (HET), and compare the results with Case 2 where homogeneous flows are used. Results are summarized in Table 6.3. With HET sources, the success rate  $P_{\text{sd}}$  is lower than all the other homogeneous source models, but the differences in  $P_{\text{fa}}$  and  $P_{\text{mis}}$  are negligible. It is difficult to tune

the hysteresis or TBF parameters to optimize the overall performance since the source characteristics are not known a priori.

We also repeat the Case 2 experiment using the EXP1 source with the following modifications:

- a. MDAP without micro-policing mode, and
- b. MDAP deployed at ingress ERs only.

Results show that only 23% of malicious flows are detected in (a), and 53% in (b), which is significantly lower than 79%, when MDAP is deployed at both ingress and egress ERs (Figure 6.7).

## 6.5 Implementation and Prototyping

In this section, we provide a brief description of how we implement the Furies architecture on top of the Click modular router [24]. We use this implementation to evaluate certain performance metrics which could not be accurately quantified through simulations. One such important metric is the overhead incurred at an edge router by adding Furies control functionalities. The current implementation works on Linux 2.2.16 and 2.2.17 kernels.

We also built a proof-of-concept Furies prototype on Millennium<sup>9</sup>, a powerful computational testbed in our laboratory. We verified all the features of Furies by setting up a virtual network of seven ingress-egress POPs over this testbed.

### 6.5.1 Overview of Implementation

Click is a Linux-based software architecture developed by Kohler, et al. at MIT [24] for building flexible and configurable routers. A Click router is assembled from packet processing modules called elements. Individual elements implement simple router functions

<sup>9</sup>Millennium Project, <http://www.millennium.berkeley.edu/>.

like packet classification, queuing and scheduling. We extend the Click router to support two additional elements: the reservation agent (RA) and the monitoring agent (MA). The RA is responsible for directing flow requests to the Furies Resource Manager (RM) and forwarding responses from the RM to the client. The MA handles the traffic monitoring and policing of admitted flows, i.e., the functions of the TMs and TPs in the Furies architecture (Section 6.3.1).

The communication between the Click router and the Resource Manager is performed through SNMP. In order to enhance the throughput of the Click router, we reduce the number of context switches required for processing the control packets from the RM by batching the messages from the RM to the Click router.

### **6.5.2 Performance Evaluation**

Using our implementation, we measure the performance overhead of adding the RA and MA in an edge router. To obtain a realistic picture of this overhead, we compare the maximum throughput obtained from our implementation to a basic implementation of Click which did not contain any of the monitoring tools (hereafter referred to as default Click). A quantification of this overhead is necessary to determine whether it is practical to deploy Furies.

#### **Experimental Setup and Methodology**

For evaluation purposes, we set up our own cluster of machines over a 10.2.2.0/24 network. The experimental setup consisted of a total of six Intel PCs running Linux. A Pentium-III 650Mhz machine was used as the Click router. This machine used a 3com 3c905 100 Mbps Ethernet controller and had 256 KB of L2 cache. Another machine, a Celeron-400 Mhz with a 128 KB of L2 cache was used as one of the traffic generators. The other traffic generators were Dell 6350, 4-processor 650 Mhz machines. We use two more

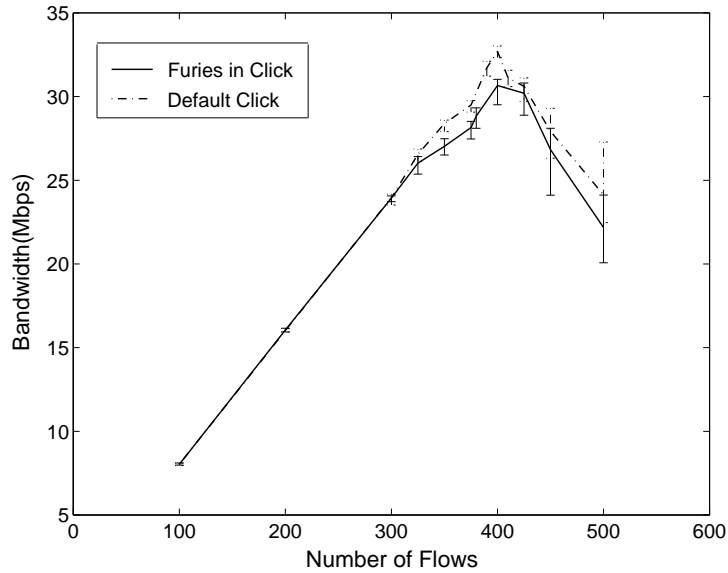


Figure 6.9: Throughput comparisons between Furies+Click and default Click.

machines as Sink and Resource Manager. All these machines are connected to a backbone router using 100 Mbps connections. The router is a Bay Networks Accelar-1100B router with the capacity to support 16 100 Mbps Ethernet ports.

To make our measurements more realistic, we used an IP routing table from a BBN planet edge router [120]. The routing table contained 43,872 entries. We disabled the IP-lookup cache, and compared the lookup time with the time taken for traffic monitoring. The traffic statistics was periodically sent to the RM every 100 ms. We modified Mgen [121], a publicly available traffic modeling software, to generate traffic for our experiments.

## Experimental Results

In our first experiment, we measured the maximum throughput of our implementation at different loads and compared it to a default Click router. A basic flow in our setup has a bandwidth of 80 Kbps and a packet size of 1024 bytes. As the number of flows increases, the amount of policing and state needed at the edge router also increase.

Figure 6.9 compares the throughput of our Click implementation of Furies and

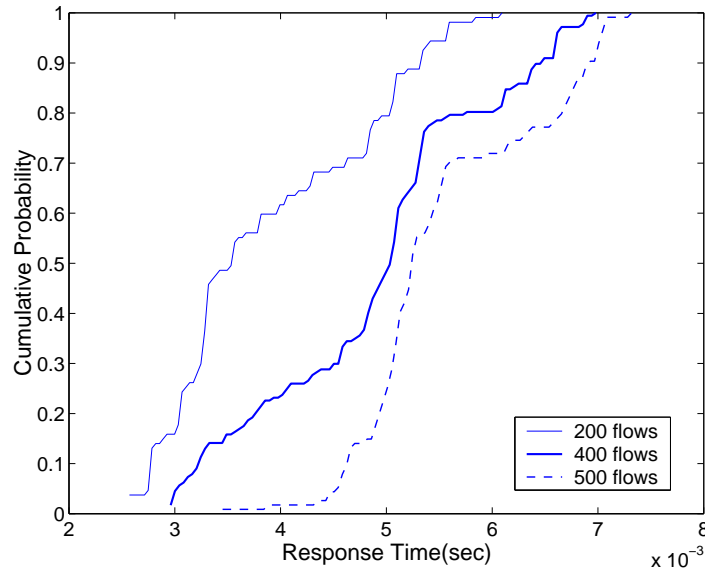


Figure 6.10: Response time for processing flow requests for varying loads.

the default Click. We vary the number of flows to generate varying loads. Since an ER can observe up to 500 latency-sensitive flows (Section 6.3.2), we consider a load of 100-500 flows<sup>10</sup>. From the figure, we can infer that the percentage overhead (percentage difference in throughput) is insignificant. The maximum percentage difference observed in our experiments is 5%. This occurs at a load of 400 flows. The default Click achieves a peak throughput of 32 Mbps while our implementation achieves 30.5 Mbps. At smaller loads, the overhead is negligible.

In the second experiment, we measured the response time for flow allocation at different loads. We achieved a particular load in the system by maintaining a constant number of active flows and sending dummy flow requests to the Click router from the Traffic generator. There are three stages in the process of obtaining a response for a flow request: the RA in Click forwards the request to the RM, the RM performs admission control on the flow, and the RM sends the response to the requesting entity through the RA.

<sup>10</sup>Our setup did not allow us to generate beyond 500 flows.

In Figure 6.10, we plot the cumulative distribution of the response time at three different loads: 200 flows (small load), 400 flows (saturation point) and 500 flows (very high load). From the graph, we can observe that the mean response time increases as the load increases and the CDF shifts to the right. In all our experiments, we observe a minimum response time of 2.5 ms and a maximum of 7.2 ms. Our results indicate that the standard deviation of our response time is high. This can be attributed to the batching of responses at the RM and timer-based processing of flow requests at the Click router. However, this variance is tolerable since the mean response time is small.

### 6.5.3 Discussions

We found it easy to build extra router mechanisms on top of Click. Our entire implementation consists of 4463 lines of C++ code and the effort taken to integrate our code with Click was minimal. From our experiments, we observed a negligible performance difference between the interrupt-driven kernel mode and the interrupt-driven user mode of Click. In both cases, the processing overhead of deploying the Furies mechanisms on a Click router is minimal ( $\leq 5\%$ ). Our implementation did not use the *Poll-device* element optimization of Click. We also used a much larger routing table than the one used in [24]. Studies in [122, 123] have shown that increasing the size of the routing table can adversely affect the throughput of the system. However, Click does not support many optimizations for fast table lookups. The performance of Click has also been optimized for DEC 21140 Tulip 64-bit PCI controllers and our setup used a 32-bit 3com Ethernet controller. Though these issues may affect the net throughput of our system, we believe that they will not change the percentage difference of the throughput.

## 6.6 Deployment Issues

This section describes some of the issues involved in deploying Furies in the existing Internet.

### 6.6.1 Distributed RM Implementation

Although the Resource Manager (RM) is described as a single logical entity within a domain (Section 6.3.1), it can be implemented as a distributed architecture across POPs. Every POP of an ISP usually has a fault monitoring facility to continuously manage the link and router status in its network. The additional functions of the RM can be implemented as additional pieces of software that reside in these monitoring facilities. For example, a *local-RM* of a POP can maintain part of the domain's database, FR (Section 6.3.1), by tracking *Fids* where at least one of the *FidIn* or *FidEg* is a valid group identifier of an edge router within the same POP. For every flow that is admitted, a new entry with its *Fid* and allocated bandwidth is created in the partial FR databases at both its ingress and egress POPs. Similarly, when a flow stops, we remove the flow's entry from the local RMs in its ingress and egress POPs, respectively.

### 6.6.2 Changes to Routers

To deploy Furies, no changes are required in the core routers, while the policing and monitoring need to be added to the edge routers. From our Click implementation, we infer that the modifications needed to add the extra Furies mechanisms in an edge router is minimal.

### 6.6.3 Virtual Private Networks

Furies can be deployed within an ISP to support Virtual Private Network (VPN) customers. The mechanisms of Furies can be applied to provide an abstraction of a VPN

overlay network by treating the VPN endpoints as ingress and egress points. We do not need to change any of the underlying admission control or traffic policing mechanisms.

#### 6.6.4 Multiple ISPs

If indeed Furies is deployed, it will be done incrementally and not all ISPs will be Furies-enabled. A flow that passes through multiple ISPs may not receive the best performance guarantees if some intermediary ISPs do not support Furies. Furies can be extended to support a *Confederation* of ISPs, which is a group of ISPs that coordinate among each other to set up a larger virtual ISP through peering agreements. Every peering point is associated with a set of SLAs, and a flow that is admitted in one ISP can be guaranteed its level of service within the confederation. Furies also provides a way for sharing resources across ISPs and does not assume any trust relationships between ISPs.

### 6.7 Summary

Although detection of malicious flows has long been recognized as an important aspect of resource control, a practical and scalable way of implementing it has not been studied in great detail. This chapter proposes the Furies architecture for policing incoming flows and detecting malicious behavior without requiring per-flow state maintenance at any edge routers. By aggregating flows for group policing, Furies only requires  $O(\sqrt{n})$  state maintenance at edge routers (where  $n$  is the number of flows), which is substantially better than previous approaches. Extensive simulations show that Furies-MDAP is effective and robust across a variety of source models and extreme cases. For VoIP type traffic (EXP1), MDAP can successfully detect 79% of malicious flows with zero false alarms and less than 0.1% incorrectly dropped packets. However, further study is needed to improve the detection of bursty malicious sources. Our approach has significant practical value since Furies incurs very minimal processing overhead at edge routers and can be incrementally deployed. 90%

of the malicious flows are detected within  $1/4$  of their average life time, and the average detection time is 26.9 seconds or  $1/10$  of flow life time.

MDAP provides a way to select a small subset of the flows, which are most likely to be malicious, for individual sampling and further verification. Since flows are aggregated for policing in general, the processing overhead and state maintenance required are minimal. Therefore, the optimal operational range for MDAP is when only a small fraction of the total admitted flows is malicious. The advantage of MDAP over per-flow policing schemes is the most significant in this case. Otherwise, if the majority of the flows are malicious, a large subset of the flows would be subjected to individual sampling and the overhead of MDAP approaches that of per-flow policing schemes.

Tuning the parameters of MDAP involves tradeoffs among different performance indexes, e.g., frequency of false alarms vs non-detections. The optimal choice of parameters are often dependent on the operational goals and business models of the network providers.