Efficient Packet Monitoring for Network Management

K. G. Anagnostakis* S. Ioannidis* S. Miltchev* J. Ioannidis† J. M. Smith*

*CIS Department, University of Pennsylvania
{anagnost,sotiris,miltchev,jms}@dsl.cis.upenn.edu

†AT&T Labs – Research
ji@research.att.com

Abstract

Network monitoring is a vital part of modern network infrastructure management. Existing techniques either present a restricted view of network behavior and state, or do not efficiently scale to higher network speeds and heavier monitoring workloads. Considering these shortcomings we present a novel architecture for programmable packet-level network monitoring. Our approach allows users to customize the monitoring function at the lowest possible level of abstraction to suit a wide range of monitoring needs: we use operating system mechanisms that result in a programming environment providing a high degree of flexibility, retaining fine-grained control over security, and minimizing the associated performance overheads. We present the implementation of this architecture as well as a set of experimental applications.

Keywords: network monitoring, active networking

1 Introduction

Network monitoring is an increasingly important, yet difficult and demanding task on modern network infrastructures. As argued in [16], *"the scalability of the stateless IP networks has been bought at the expense of observability"*, which has in turn led to the study of several ways of monitoring networks in order to support control and management functions. Most routers offer built-in monitoring functionality, accessible using mechanisms such as SNMP [10], RMON [42] or NetFlow [13]. However, this predefined functionality is often either too specific or not specific enough for modern network monitoring requirements. User needs vary and are often not anticipated at design time, causing the management interfaces to fall short of these needs. Additionally, tasks attractive only to minorities of users are frequently not cost-effective enough to be integrated in routers. Finally, in cases such as detection and prevention of denial-of-service attacks, the need for timely deployment cannot be met at the current pace of standardization or software deployment.

The desire for more flexibility has led to the rise of non-router-based techniques, broadly classified as *active* and *passive* measurements.[1] The degree of flexibility offered by these non-router-based techniques is still limited. Active measurement systems such as *pings* have restricted capabilities by their very nature as they both interfere with and also do not always give an accurate picture of network conditions [45]. Passive measurement systems do not suffer from these limitations, but measurements can only be analyzed off-line [19]. The drawback in this case is that there exist applications that *require* real-time analysis, while

[1] See, *e.g.*, [34] and [5], respectively.

for a number of other applications, the ability to trade off storage and communication for computation results in increased efficiency. Customizing the function of a passive monitor for a particular application is a possibility; however, it requires significant effort. To our knowledge, the use of passive measurement systems is still limited.

In this paper, we explore a solution to these problems, namely, the use of a dynamically extensible system for passive traffic monitoring. Users are given the ability to install modules that will perform monitoring in real time. Naturally, the installation of these modules is subject to policy constraints. Putting these modules in the monitoring system, hence close to the information source, is similar in principle to Management-by-Delegation (MbD) models [20]. The key difference lies in the level of abstraction: we argue for a packet-level traffic measurement approach instead of relying on the already abstracted SNMP-based metrics. With respect to deployment, the proposed system can be introduced as a local enhancement, *e.g.*, as a passive measurement system or as an enhanced interface card. Of course, wide-spread deployment is needed to stimulate more distributed applications such as the IP Traceback [37, 6, 14] and the Pushback [26, 18] mechanisms described in Section 4. The FLAME system described here attempts to provide an efficient system structure and a fine-grained protection model. We rely on a safe C-like language called Cyclone [17] and an efficient kernel-level packet handling technique derived from ASH [43]; these maximize safety without sacrificing functionality. The implementation and the experimental applications instantiated on this system show that this approach adds substantial value to a measurement infrastructure and is a promising venue for further investigation.

The rest of this paper is structured as follows. In Section 2 we elaborate on the rationale and motivation behind our work. We present the system architecture in Section 3 and a set of experimental applications in Section 4. In Section 5 we study the performance of the resulting system, showing the improvement over current systems. We present related work in Section 6 and we conclude in Section 7.

2 Motivation

There are three main thrusts for this work. First, as briefly described in Section 1, there are several fundamental limitations and trade-offs of existing techniques for network monitoring that require investigation of alternatives. Second, developing a monitoring system that users can program at the packet granularity poses several interesting design challenges from a systems perspective, as the standard OS mechanisms are too heavyweight and intrusive. Last, we view network monitoring as a case for active networking, testing existing arguments and knowledge. We further elaborate on each of these motives in the following paragraphs.

2.1 Limitations of existing techniques

We discuss in turn why SNMP-like abstractions, existing MbD-based models, and active and passive measurement techniques are inflexible or inefficient for modern measurement-based applications.

The most widely used management mechanisms today are based on SNMP, RMON and Netflow. Management functionality is *hard-coded* into the managed elements and follows a standardized information model. The need for standardization often results in significant delays; it takes months to years from the development and standardization of a protocol to the time the corresponding MIBs can be derived. This is understandable, as the required management functionality can often not be accurately predicted without significant operational experience. The lack of management hooks hinders the use of management tools as new features are added to the network. A good example for this is the interaction of ECN [35] with Netflow: NetFlow only allows either per-ToS (which includes the CE bit used by ECN) or per-network accounting tables, and ECN affects both. Experimentation with ECN-based accounting and charging [25] using Netflow is thus made impossible. Another example is if the need arises to measure the rate of TCP SYN packets (for

SYN attack [31] detection); unless such a feature was deemed necessary when the MIB was designed, there is no way to add this kind of functionality.

The problem with the existing forms of MbD designs is that they only target the overhead and latency of communication between the management system and the managed element. The underlying information model remains the same: the information that is available through SNMP/RMON/Netflow. Therefore, the problems described in the previous paragraph also hold for MbD designs.

While being the easiest to implement, active measurements have certain limitations. First, as these techniques create probe traffic, this traffic may interfere with regular traffic in uncertain ways. Also, probe traffic is often treated differently than regular traffic. The quality and validity of the information provided by these probes also raises some issues. For example, rare or exceptional phenomena (*e.g.*, drops, delay variations, other anomalies) may not be visible through active measurements.

There are three main problems with passive measurement systems. First, analysis of the data has to be performed off-line, after the measurements have been taken. There is no way to perform monitoring in real-time, unless there is "login" access to the measurement system. This restricts the use of the system to the actual infrastructure owner and trusted parties if real-time functionality has to be built into the system. Second, maintaining the huge data sets for post-processing is often inefficient. On-the-fly data reduction by processing may be more efficient, especially if the processing task is limited; we present an example module that illustrates the power of this approach. Finally, in most passive monitoring systems policy as well as functionality is hard-coded into the system. For instance, OC3MON [5] only captures packet headers (IP and TCP) but some applications may require access to the payload.

2.2 Implementation challenges of programmable packet monitoring

Previous work in this context has demonstrated the benefits but also revealed limitations of programmable packet monitoring. In particular, the LAME system [4] supports a fair mix of applications at typical network speeds but has significant base and per-application overheads. These overheads are due to commodity operating system abstractions and system mechanisms that provide security and fault-isolation. The Windmill system [27] is similarly structured and pays a similar cost without providing protection.[2] Since computation is expensive, especially as link speeds continue to grow, it is crucial to minimize waste of resources. Although it would be preferable to rely as much as possible on off-the-shelf components, having to examine alternative system mechanisms for the needs of programmable packet monitoring appears inevitable.

2.3 Relationship with active networking

We study network monitoring as a specific class of in-network functionality that could benefit from dynamic extensibility within the broader perspective of active networking. In this direction, techniques that are well understood in the active networking context apply directly to the design of an extensible monitoring system. For example, fine-grained security and resource control models [22, 3] can safely extend the application range of the system, thus "opening" the infrastructure for third parties to perform monitoring functions. The trade-offs between flexibility, security and performance have been studied extensively in active networks [2], and experimentation has shown that these functions can be appropriately restricted without excessive design complexity and performance cost.

Note that ad hoc forms of "active" networking, in the sense of dynamic extensibility, have been explored to some extent in the case of active measurements [34] as well as passive measurements [27]. In the spirit of minimizing complexity, it is desirable to find a unified general solution to accelerating network evolution, as advocated by active networking arguments. Network monitoring, along with other functions such as routing [33], represent network-side functions that have already stressed the need for extensibility. The

[2] The reason for this appears to be that protection was not an objective in Windmill's design.

target system structure should be easy to integrate in extensible router designs such as [1, 24]. This opens further opportunities as the function of the router itself can be adapted to the needs of monitoring, while also removing the burden of operating separate infrastructures for forwarding and monitoring. The amount of resources that could be made available for monitoring in this scenario is an interesting question, however, we do not touch upon these issues in this paper.

3 Design

The following section presents the FLAME system architecture. Our design tries to optimize the following, often conflicting, dimensions:

Flexibility. For a monitoring system to offer the needed flexibility it is important to provide a programming abstraction at the lowest possible level. This is why our system structure focuses at the packet-level. Deviation from this abstraction can easily lead to problems, therefore we avoid investigating alternatives such as aggregates.

Performance. Performing per-packet computations in real time (or near-real time) presumes adequate processing capabilities as well as efficient handling of communication, computation and memory resources. Furthermore, executing multiple applications on the same infrastructure introduces further complications. It increases the system complexity and the overall execution overhead. For the architecture to be scalable both such characteristics must be kept at a minimum.

Security. Security is needed in order to make the system robust, maximize usability and address concerns for privacy. Robustness is needed against misbehaving (e.g. poorly implemented or malicious) modules that interfere with the system operation by overconsuming its resources. Fine-grained security models increase usability because more precise policies can be specified and enforced. For example, for some users the system should allow access to the payload (under specific conditions) and for others, the IP addresses of the packet must be anonymized. The challenges in this dimension lie both in the specification, as well as the enforcement of policy for the loaded applications. This is a critical issue as security must be carefully balanced with performance and flexibility.

3.1 Architecture overview

The architecture of FLAME is shown in Figure 1. Modules consist of kernel-level code K_x, user-level code U_x, and a set of credentials C_x. Code is written in Cyclone [17] and is processed by a trusted compiler upon installation. The kernel-level code takes care of time-critical packet processing, while the user-level code provides additional functionality at a lower time scale and priority. This is needed so that applications be able to communicate with the user or management system (e.g. using the standard library, sockets etc.). We describe how safe execution of in-kernel code is accomplished in Section 3.2. The set of credentials C_x is used at compile time to verify that the module is allowed by system policy to perform the functions it requests. The dark units in Figure 1 before each K_x represent code that is inserted before each module code segment for policy checks. These units appropriately restrict access of modules to packets or packet fields, provide selective anonymization of fields, and so on. We describe how these credentials are used and how policy is specified in Section 3.3.

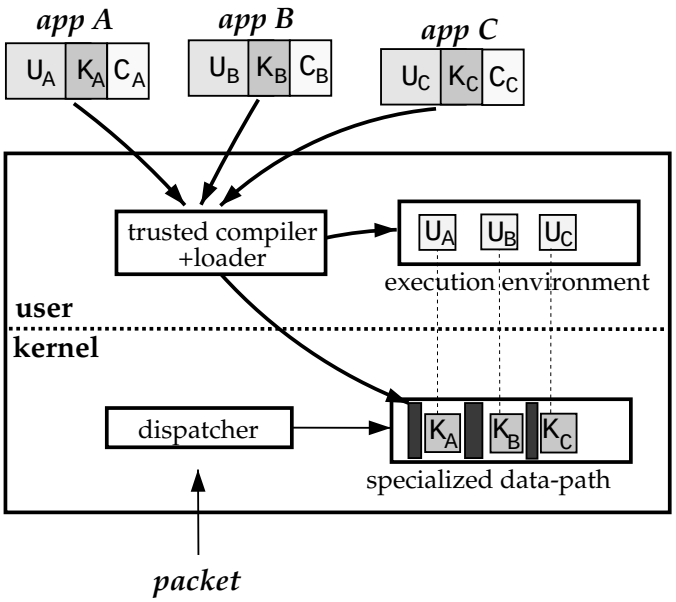

Figure 1: **FLAME Architecture**

3.2 Safe Execution

There are a number of major design challenges for allowing user code to execute inside the operating system kernel. The system needs to guard against excessive execution time, privileged instructions, exceptions and random memory references. There has been extensive work in the operating system and language communities that addresses the above problems [17, 38, 30, 11, 44]. FLAME leverages these techniques to satisfy our security needs.

Bounding Execution Time. A simple method for bounding execution time is eliminating backward jumps [21]. This has the advantage of providing us with an upper bound for the execution time: linear to the length of the program. However such a programming model is hard to program in and rather limiting. Another possible approach could be executing each installed module as a kernel thread and context switch between threads when they exceed their allocated time slot. Unfortunately this can prove too heavy weight for our purpose, which is round-robin execution of monitoring functions on incoming packets. We take a different approach, namely, we augment the backward jumps with checks to a cycle counter; if the module exceeds its allocated execution time we jump to the next module (Figure 2). This adds an overhead of 5 assembly instructions for the check and another 6 if the check succeeds,r to initiate the jump to the next module.

Exceptions. To handle exceptions caused by the module code executing in the kernel we modified the trap handler of the operating system to catch possible exceptions originating from the loaded code. Instead of causing a system panic we terminate the module and continue with the next one.

Privileged Instructions and Random Memory References. To guard against instructions that arbitrarily access memory locations or try to execute privileged assembly instructions we use Cyclone [17]. Cyclone is a language for C programmers who want to write secure, robust programs. It is a dialect of C designed to be *safe*: free of crashes, buffer overflows, format string attacks, and so on. All Cyclone programs must pass

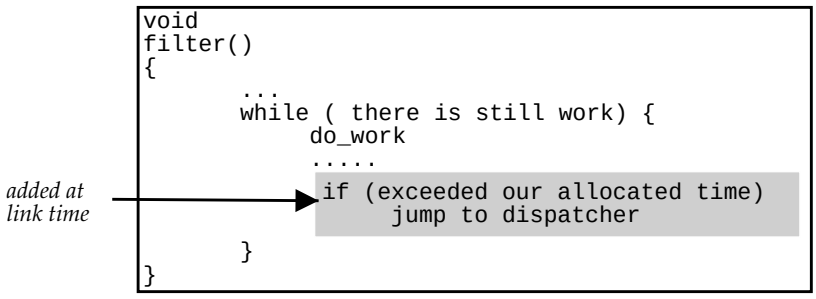

Figure 2: **Our system patches backward jumps in order to guarantee termination of the module execution. If a module has exceeded its allocated execution time out system jumps back to the dispatcher in order to schedule the next task.**

a combination of compile-time, link-time and run-time checks to ensure safety.

3.3 Policy control

The network operator controls what packets a module can access, what part of the packet a module is allowed to view and in what way, what amount of resources (*e.g.*, processing, memory) the module is allowed to consume on the monitoring system, and what other functions (*e.g.*, socket access) the module is allowed to perform.

We have chosen a *Trust Management* [8] approach to mobile code security. Trust Management is a novel approach to solving the generalized authorization and security policy problem. Entities in a trust-management system (called *"principals"*) are identified by public keys, and may generate signed policy statements (which are similar in form to public-key certificates) that further delegate and refine the authorization they hold. This results in an inherently decentralized policy system: the system enforcing the policy need only consider the relevant policies and delegation credentials, which the user needs to provide. The KeyNote [7] implementation is used as our trust management system. An example KeyNote credential is shown in Figure 3. In the current FLAME design, we perform policy compliance checks at the time of loading up the incoming object code.

KeyNote provides a simple notation for specifying both local policies and credentials. Applications communicate with a "KeyNote evaluator" that interprets KeyNote assertions and returns results to applications. A KeyNote evaluator accepts as input a set of local policy and credential assertions, and a set of attributes, called an "action environment," that describes a proposed trusted action associated with a set of public keys (the requesting principals). The KeyNote evaluator determines whether proposed actions are consistent with local policy by applying the assertion predicates to the action environment. In our system, we use the action environment as the place-holder of component-specific information (such as language constructs, resource bounds, *etc.*) and environment variables (such as time of day, node name, *etc*.) that are important to the policy management function.

4 Applications

4.1 IP Traceback

IP traceback [40, 37, 26, 41] is one possible application of FLAME that provides a useful function without requiring modification of router functionality. The current Internet architecture offers very few

```
KeyNote-Version: 2
Authorizer: NET_MANAGER
Licensees: TrafficAnalysis
Conditions:
  (app_domain == "flame" && module == "capture" &&
   (IPsrc  == 158.130.6.0/24 || IPdst  == 158.130.6.0/24))
-> "HEADERS-ONLY";
Signature: "rsa-md5-hex:f00f5673"
```

Figure 3: **Example credential that grants "TrafficAnalysis" the right to capture traffic to/from 158.130.6.0. The user is authorized to receive packet headers only.**

protection mechanisms against ill-behaved traffic. Especially in recent years there has been an increase in Distributed Denial of Service (DDoS) attacks [31, 32] which have led to the study of appropriate *traceback* mechanisms. A traceback mechanism has to detect the attack source(s), despite the fact that IP source addresses may be spoofed by the attacker, and to respond, by confining or blocking traffic from the attacking sites.

Neither of these functions is supported by existing routers. Using our system, they could be implemented by simply monitoring traffic and recording the upstream router for each packet. In this particular case, sampling techniques are applicable for reducing the cost of monitoring. Out-of-band communication is needed to recursively trigger traceback functionality on neighboring routers, to find the source of the attacks. Once the attackers have been traced, router control commands need to be initiated from the monitor to an appropriate management interface to block or otherwise confine the attacking traffic. Although in our own implementation we use the ALTQ [12] QoS API for blocking the attacking sites, in the general case this interface is not necessarily coupled with the router itself.

The key benefit of using the FLAME approach in implementing IP packet traceback and traffic rate limiting is the ability to adapt depending on the type of ill behaved traffic. For example, FLAME give users the flexibility of dynamically deploying the appropriate module to counter a possible attack. As new attacks are being invented it is easy to develop and deploy mechanisms to counter them.

4.2 Worm detection

Recently, the Internet has observed a wave of "worm" attacks [29] although the concept and techniques have generally existed since the early description in [9]. A worm compromises a system such as a Web server by exploiting system security vulnerabilities and once a system has been compromised the worm attempts to replicate by "infecting" other hosts. The "Code Red" worm and its variants infected over 300,000 servers in July-August 2001.

The worm could be locally detected and prevented if the packet monitor could obtain access to the TCP packet content. Unfortunately, most known packet monitors only record the IP and TCP header and not the packet payload. In FLAME, we implemented a module to scan packets for the Worm signature:

```
... GET /default.ida?NNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNN .....
```

If this signature is matched, the source and destination IP addresses are recorded. The module also has the option of blocking traffic from the attacking as well as the attacked site. The ALTQ QoS API is used for this purpose.

4.3 Traffic Analysis

While originally a tool for research, traffic analysis is now needed for management functions such as traffic engineering [19]. As discussed in Section 2, efficient extraction of the needed information is not always possible through SNMP or NetFlow interfaces, and the use of passive measurement systems can be unnecessarily costly.

We have built a simple analysis module to demonstrate the simplicity of providing this function on FLAME. The purpose of the module is to measure the existence of *packet trains*. A packet train can be loosely defined as a set of consecutive packets belonging to the same *flow* as observed on a network link. This is typical, for instance, for TCP traffic, which in its slow-start phase injects two packets for each acknowledgement received. Note that this kind of information is not available through the standard network management mechanisms. A typical passive monitoring system would require communication of all traffic to the analysis host. Our implementation is trivial to implement and adds minimal overhead to the monitoring system. The module consists of about 40 lines of code, costs about 100 cycles per packet and results in less than 400 bytes of measurement data. Other functions, such as extracting statistics on TCP window sizes or analyzing traffic burstiness, are similarly easy to build. To our knowledge, such flexibility is not provided by existing systems.

4.4 ECN-based Charging

The task of obtaining network usage statistics per accountable entity is important for managing an operational network. Often, this forms the basis for *charging* users for network usage. Volume-based charging schemes in use today rely on NetFlow or SNMP-type accounting for calculating the charging scheme parameters, usually in terms of volume per network prefix. Recently, more dynamic pricing algorithms have been studied, especially with regard to providing differentiated services or controlling congestion. Recent proposals are based on the ECN [35] mechanism as described in [25]. When the network becomes congested, routers mark packets probabilistically, with the probability depending on the level of congestion. Marked packets are charged a fixed amount of currency. Users who consume more resources during times of congestion are charged more than others. This scheme aims at providing incentive for users to make reasonable use of network resources during congestion.

To support this charging scheme, we collect accounting information per network prefix as well as marked vs. non-marked packets. As mentioned in Section 2 this is not supported by any of the existing router accounting mechanisms: NetFlow allows either per-ToS *or* per-AS *or* per-network accounting tables. The implementation on FLAME is fairly simple ; the kernel-level part takes care of looking up the appropriate entry in the accounting hashtable, while user-level code attempts to flush the tables to the user application on time- and/or memory thresholds.

5 Experimental evaluation

In this section, we describe experiments with the FLAME implementation. The purpose of the experiments is two-fold. First, we measure the *monitoring capacity* of the system compared to LAME, and attempt to identify parameters that affect system performance. Second, we measure the cost of the different modules that we run on the system. The test platform consists of two 1GHz Pentium III PCs with 256MB SDRAM, and 1 Gbit/s Ethernet interfaces (NetGear GA620 32-bit PCI based with Alteon chipset). One PC is used as

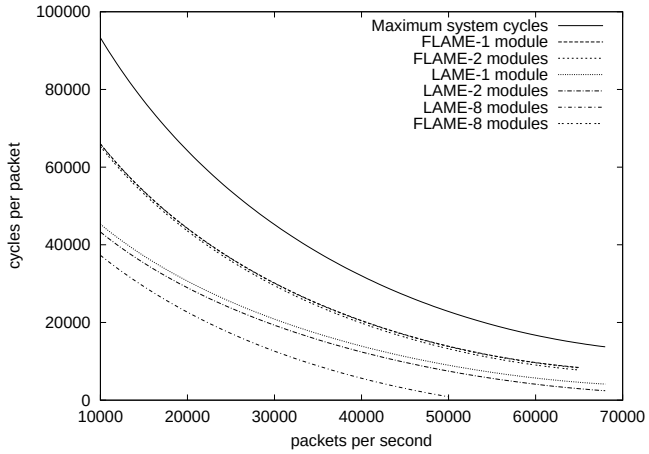

Figure 4: **Monitoring capacity of FLAME vs. LAME for different traffic rates and configurations**

the traffic generator and the other one runs the FLAME prototype. For experiments whose results depend on the actual traffic content (*i.e.* measurements on the packet train module and ECN accounting) we used the `auckland-2` traffic trace from NLANR. [3] In all other cases, we used a modified version of `ttcp` whose sending rate is controlled through the command line.

Figure 4 presents the monitoring capacity of FLAME compared with LAME for different traffic rates. We define monitoring capacity as the maximum number of processing cycles that the system provides to modules without experiencing packet loss (*e.g.*, overflowing the receive queue). This is measured using a "null" module whose function is to simply consume processing cycles in a *for* loop. To infer the monitoring capacity, we increase the number of cycles that this module consumes until the packet loss rate becomes non-zero. The results show that FLAME consistently outperforms LAME, imposing a significantly smaller system overhead. The top line represents the maximum number of cycles available for the specific processor for each packet at a given traffic rate. These results also confirm that FLAME scales well with increasing number of applications. This is of particular importance as in practice the system may have to accomodate many small monitoring modules and/or few large ones (in terms of processing needs).

Table 1 summarizes the cost per individual module for FLAME and LAME. The cost of the "null" module illustrates the per-module overhead ; this is a major overhead factor in LAME which is minimized in FLAME. Additionally, it is shown that, for some applications, the cost of each module can be higher in FLAME than in LAME. This is natural, as FLAME moves protection inside or around the module; hereby the fixed base- and per-module costs are traded for more efficient case-specific overheads. In other cases, such as the "dump-to-disk" example, the kernel-level structure of FLAME provides further performance benefits.

In Section 3.2 we described our approach for guaranteeing bounded execution time. Figure 5 presents the evaluation of this approach. We execute sequentially ten modules similar to that presented in Figure 2, with and without our backward jumps patch. Furthermore, we simulate a workload inside the modules which we gradually increase to monitor the system behavior. We notice only a slight increase in execution time, which stays constant across a wide range of load.

To investigate the overheads imposed by using the Cyclone language, we compiled and executed the packet train module using C (Unsafe) and Cyclone (FLAME). Figure 6 presents the normalized overheads. Cyclone has an additional 19% overhead over the unsafe case, due to the checks that are necessary for type

[3] Traces and tools are available from http://moat.nlanr.net/Traces/Kiwitraces/auck2.html

Task	LAME **cpp**	**error**	FLAME **cpp**	**error**
null module	1831	± 89	80	± 15
anonymization	131	± 8	138	± 10
dump to disk	3293	± 416	2418	± 208
ECN accounting	850	± 43	952	± 55
pkttrain	101	± 0	120	± 0

Table 1: **Cost breakdown per-application in cycles per packet (cpp)**

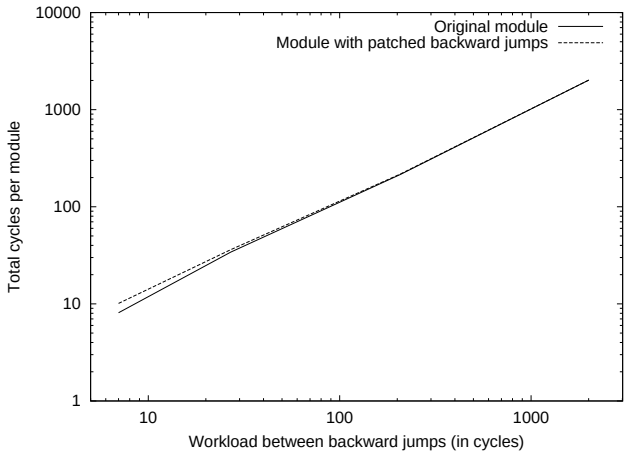

Figure 5: **The cost of patching backward jumps**

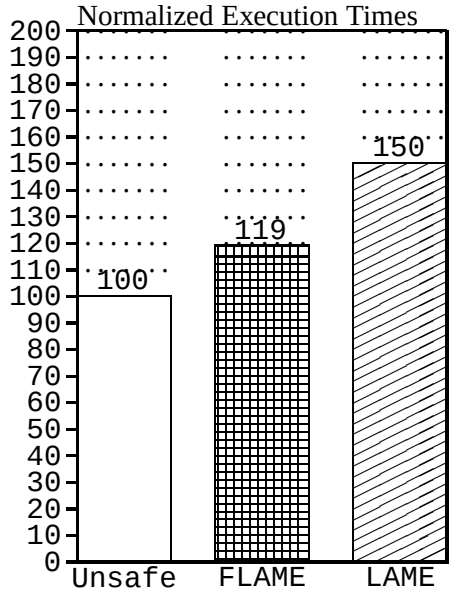

Figure 6: **Overheads for the packet train module**

safety. It is, at this point, unclear whether a typical workload would require this amount of overhead. A detailed study of applications on this platform is therefore needed to correctly estimate the average system overhead. Note however that efficient implementation and efficient compilation techniques are key to reducing this overhead. For instance, using types rather than arrays (whose bounds have to be checked) on accessing the packet data and techniques such as loop unrolling can significantly reduce run-time costs. In the LAME approach, the system imposes a *de-facto* fixed cost for the system as a whole as well as per-module which cannot be similarly reduced.

6 Related Work

Numerous techniques have been developed for flexible network monitoring. The first generation of a tools such as `tcpdump` were based on the Berkeley Packet Filter [28] (BPF). The Packet Filter provides operating system functionality for user-level traffic monitoring. Users define filters in a *filter language* and pass them to the system, to execute on a *filter machine* inside the kernel. The filters specify which packets the user is interested in. These packets are then passed to the user-level application for processing. BPF-based tools were suitable for shared media networks such as Ethernet and FDDI, however, with switched networks now dominant this method is no longer useful for traffic monitoring.

OC3MON [5] is a dedicated host-based monitor for snooping on 155Mbit/s OC3 ATM links. The host is attached to the link using an optical splitter so that the monitoring function does not interfere with regular service. The host monitors packets and records the captured headers in files for post-processing. No real-time functionality for packet processing is considered in the original design.

`ntop` [15] is an end-system network monitoring tool that shares a subset of the motives of our approach to extensibility. `ntop` allows plug-ins, in forms of shared libraries, to be developed independently from the system core. The difference lies in that full trust in plug-ins is assumed and no further security-enhancing function is build into the system. Also, there is no notion of using the measurement data for router control and no provisions for distributed applications across the node boundaries.

Windmill [27] is an extensible network probe environment which allows loading of "experiments" on the probe, with the purpose of analyzing protocol performance. As discussed throughout this paper, Windmill does not provide safety and does not efficiently scale to higher network speeds or application workloads.

The NIMI project [34] provides a platform for large-scale Internet measurement using Java- and Python-based modules. NIMI only allows active measurements, while the security policy and resource control issues are addressed using standard ACL-like schemes.

In the active networking arena, Smart Packets [39] and ABLE [36] provide programmable platforms for network management applications, following the MbD principles. ABLE allows applications to poll SNMP interfaces from inside the network, while Smart Packets operate on a management interface at the language level. In both cases, the management information and control settings are pre-defined subsets of the managed element state.

7 Summary and concluding remarks

We have described an architecture for a programmable packet-monitoring system: we provide a mechanism for loading code in the system kernel; we guarantee safety by using a type-safe language and run-time checks; finally, we use a Trust Management system for fine-grained policy control.

The field of network monitoring provides an excellent proving ground for programmable-infrastructure systems because of the continuously evolving and highly diverse nature of the monitoring functions that users need to embed in the network infrastructure. These characteristics make it hard for designers and users

System	Flexibility	Safety	Performance
pre-programmed	Low	High	Low to Medium
Windmill	Medium	Medium	Low
LAME	High	High	Low
FLAME	High	High	High

Figure 7: **Comparison of network monitoring systems**

to predict and agree on a set of functions that can satisfy the long-term needs of monitoring applications. In the design of the FLAME system, we examined the trade-offs between safety, performance and flexibility and we believe that FLAME achieves a good balance, as shown in Figure 7. The structure of FLAME follows typical design patterns for active-network prototypes while incorporating crucial system-level optimizations for high performance.

Although there are numerous open questions in the software structure of our system, further work in programmable packet monitoring could also depart from a purely software approach. We see tremendous potential in the use of network processors such as the Intel IXP 1200 [23]. Our system can be improved by moving the filtering functionality, as well as some of the module processing, to the network processor micro-engines. Adding or reserving processing capabilities on router interfaces for monitoring purposes and exposing them for programming can also be highly useful for certain limited workloads. A detailed study and characterization of potential application workloads may, in turn, reveal different requirements on the design of network processors. We expect that monitoring workloads can be both processing- and memory-intensive, unlike the forwarding-oriented functionality considered in current network processor designs.

Finally, in addition for network processors, we would like to study other distributed designs. Since the application workload will consist of several independent modules, it can naturally be distributed among different processors. Our approach can thereby be extended to higher network speeds and more intensive workloads.

Acknowledgements

This work was partially supported by DARPA under Contracts F39502-99-1-0512-MOD P0001 and N66001-96-C-852 and OSD/ONR CIP/SW URI through ONR Grant N00014-01-1-0795. We would like to thank Michael Hicks, Jessica Kornblum and Jon Moore for valuable comments and suggestions throughout the course of this work.

References

[1] ACIRI. XORP: An Extensible Open Router Platform. http://www.xorp.org/, January 2001.

[2] D. S. Alexander, Paul B. Menage, W. A. Arbaugh, A. D. Keromytis, K.G. Anagnostakis, and J. M. Smith. The Price of Safety in an Active Network. *Journal of Communication Networks (JCN)*, March 2001.

[3] K. G. Anagnostakis, M. W. Hicks, S. Ioannidis, A. D. Keromytis, and J. M. Smith. Scalable resource control in active networks. In *Proceedings of the 2nd International Working Conference on Active Networks*, October 2000.

[4] K. G. Anagnostakis, S. Ioannidis, S. Miltchev, and J. M. Smith. Practical network applications on a leightweight active management environment. In *Proceedings of the 3rd International Working Conference on Active Networks*, October 2001.

[5] Joel Apisdorf, k claffy, Kevin Thompson, and Rick Wilder. OC3MON: Flexible, Affordable, High Performance Statistics Collection. In *Proceedings of the 1996 LISA X Conference*, 1996.

[6] Steve M. Bellovin. ICMP Traceback Messages. Work in Progress, Internet Draft draft-bellovin-itrace-00.txt, March 2000.

[7] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust-Management System Version 2. RFC 2704, http://www.rfc-editor.org/, September 1999.

[8] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proc. of the 17th Symposium on Security and Privacy*, pages 164–173, 1996.

[9] John Brunner. The shockwave rider. This is the original reference for the term ”tape-worm”, 1975.

[10] J. D. Case, M. Fedor, M. L. Schoffstall, and C. Davin. Simple Network Mangement Protocol (SNMP). RFC1157/STD0015, http://www.rfc-editor.org/, May 1990.

[11] Jeff Chase, Hank Levy, Miche Baker-Harvey, and Ed Lazowska. Opal: A single address space system for 64-bit architectures. In *Proceedings of the Fourth Workshop on Workstation Operating Systems*, pages 80–85, 1993.

[12] Kenjiro Cho. A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers. In *Proceedings of USENIX 1998 Annual Technical Conference*, June 1998.

[13] Cisco Corporation. NetFlow services and applications. http://www.cisco.com/, 2000.

[14] Drew Dean, Matt Franklin, and A. Stubblefield. An algebraic approach to ip traceback,. In *Proceedings of NDSS ’01*, February 2001.

[15] Luca Deri and Stefano Suin. Effective Traffic Measurement using ntop. *IEEE Communications Magazine*, pages 2–8, May 2000.

[16] Nick Duffield and Matthias Grossglauser. Trajectory sampling for direct traffic observation. In *Proc. ACM SIGCOMM’00 Conference*. August 2000.

[17] Greg Morrisett et al. Cyclone: A next-generation systems language. In *http://www.cs.cornell.edu/projects/cyclone*, 2001.

[18] Sally Floyd, Steve Bellovin, John Ioannidis, Kireeti Kompella, Ratul Mahajan, and Vern Paxson. Pushback messages for controlling aggregates in the network. Internet Draft, *work in progress*.

[19] Chuck Fraleigh, Christophe Diot, Bryan Lyles, Sue Moon, Philippe Owezarski, Dina Papagiannaki, and Fouad Tobagi. Design and Deployment of a Passive Monitoring Infrastructure. In *PAM 2001 Workshop*, April 2001.

[20] G. Goldszmidt and Y. Yemini. Distributed management by delegation. In *Proc. of the 15th International Conference on Distributed Computing Systems*, pages 333–340, 1995.

[21] Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. PLAN: A packet language for active networks. In *Proceedings of the Internation Conference on Function Programming (ICFP)*, September 1998.

[22] Mike Hicks and Angelos D. Keromytis. A Secure PLAN. In *International Working Conference on Active Networks (IWAN)*, 1999.

[23] Intel Corporation. Intel IXP1200 Network Processor. http://developer.intel.com/ixa/, 2000.

[24] Scott Karlin and Larry Peterson. VERA: An Extensible Router Architecture. In *Proceedings of the 4th International Conference on Open Architectures and Network Programming (OPENARCH)*, pages 3–14, April 2001.

[25] Koenraad Laevens, Peter Key, and Derek McAuley. An ECN-based end-to-end congestion-control framework: experiments and evaluation. Technical report, Microsoft Research, TR MSR-TR-2000-104, October 2000.

[26] Ratul Mahajan, Steven M. Bellovin, Sally Floyd, John Ioannidis, Vern Paxson, and Scott Shenker. Controlling high bandwidth aggregates in the network – extended version. `http://www.aciri.org/pushback/`.

[27] G. Robert Malan and Farnam Jahanian. An Extensible Probe Architecture for Network Protocol Performance Measurement. In *ACM SIGCOMM'98*, 1998.

[28] Steven McCanne and Van Jacobson. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *Proceedings of the Winter 1993 USENIX Conference*, pages 259–270, January 1993.

[29] David Moore. The spread of the code-red worm (crv2). In *http://www.caida.org/analysis/security/code-red/*. August 2001.

[30] Greg Morrisett, Karl Crary, Neal Glew, Dan Grossman, Richard Samuels, Frederick Smith, David Walker, Stephanie Weirich, and Steve Zdancewic. A realistic typed assembly language. In *1999 ACM SIGPLAN Workshop on Compiler Support for System Software*, pages 25–35, May 1999.

[31] CERT Web Pages. CERT Advisory CA-1996-21: TCP SYN Flooding and IP Spoofing Attacks. http://www.cert.org/advisories/CA-1996-21.html, September 1996.

[32] CERT Web Pages. CERT Advisory CA-1998.01 smurf IP Denial-of-Service Attacks. http://www.cert.org/advisories/CA-1998.01.smurf.html, January 1998.

[33] Craig Partridge, Alex Snoeren, Tim Strayer, Beverly Schwartz, Matthew Condell, and Isidro Castineyra. FIRE: Flexible Intra-AS Routing Environment. In *Proc. ACM SIGCOMM'00 Conference*, pages 191–203. August 2000.

[34] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis. An Architecture for Large-Scale Internet Measurement. *IEEE Communications Magazine*, 1998.

[35] K.K. Ramakrishnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. RFC 2481, http://www.rfc-editor.org/, January 1999.

[36] Danny Raz and Yuval Shavitt. An active network approach for efficient network management. In *IWAN'99, LNCS 1653*, pages 220 –231, Berlin, Germany, 1999.

[37] Stefan Savage, David Wetherall, Anna Karlin, and Tom Anderson. Practical Network Support for IP Traceback. In *ACM SIGCOMM*, August 2000.

[38] Fred B. Schneider, Greg Morrisett, and Robert Harper. A language-based approach to security. *Informatics: 10 Years Back, 10 Years Ahead*, pages 86–101, 2000.

[39] B. Schwartz, A. Jackson, T. Strayer, W. Zhou, R. Rockwell, and C. Partridge. Smart packets: Applying active networks to network management. *ACM Transactions on Computer Systems*, 18(1):67–88, February 2000.

[40] Dawn Song and Adrian Perrig. Advanced and authenticated techniques for ip traceback. In *Proceedings of IEEE INFOCOM'2001*, April 2001.

[41] Van C. Van. A defense against address spoofing using active networks. Bachelor's Thesis, MIT, 1997.

[42] S. Waldbusser. Remote Network Monitoring Management Information base. RFC2819/STD0059, http://www.rfc-editor.org/, May 2000.

[43] Deborah A. Wallach, Dawson R. Engler, and M. Frans Kaashoek. Ashs: Application-specific handlers for high-performance messaging. In *Proc. 1996 ACM SIGCOMM Conference*, August 1996.

[44] Curtis Yarvin, Richard Bukowski, and Thomas Anderson. Anonymous rpc: Low-latency protection in a 64-bit address space. In *Proceedings of 1993 Summer USENIX Conference*, June 1993.

[45] Yin Zhang, Vern Paxson, and Scott Shenker. The stationarity of internet path properties: Routing, loss, and throughput. Technical report, ACIRI, May 2000.