Efficient Packet Monitoring for Network Management

K. G. Anagnostakis, S. Ioannidis, S. Miltchev, M. Greenwald, J. M. Smith
CIS Department, Univ. of Pennsylvania
Philadelphia, PA, USA
{*anagnost,sotiris,miltchev,mbgreen,jms*}*@dsl.cis.upenn.edu*

J. Ioannidis
AT&T Labs – Research
Florham Park, NJ, USA
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. We present a novel architecture for programmable packet-level network monitoring that addresses these shortcomings. 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 an 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 [14], "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 [8], RMON [35] or NetFlow [11]. However, as we will describe in Section 2.1, this predefined functionality has not been flexible enough for modern network monitoring requirements.

The desire for more flexibility has led to the rise of non-router-based techniques, broadly classified as "active" and "passive" measurements (see, e.g., [28] and [4], respectively). The degree of flexibility offered by these non-router-based techniques is still limited. Active measurements are restricted by their very nature as they both interfere with, and also cannot always capture network phenomena. Passive measurement systems do not suffer from these limitations, but measurements can only be analyzed off-line. The drawback in this case is that there exist applications that require real-time analysis, while 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 possible but requires significant effort. To our knowledge, the use of passive measurement systems is still limited.

In this paper, we explore the use of a dynamically extensible system for passive traffic monitoring as a solution to these problems. The FLAME system described here attempts to provide an efficient system structure and a fine-grained protection model. We rely on a C-like language called Cyclone [12] for allowing safe kernel-level packet handling , hereby maximizing efficiency without sacrificing safety or flexibility. Users can install modules that perform monitoring in real time. The installation of these modules is subject to policy constraints. The proposed system can be deployed initially as a local enhancement, e.g., as a passive measurement system or, ideally, as an enhanced interface card. Wide-spread deployment also enables applications such as the IP Traceback [31, 13] and Pushback [23] mechanisms described in Section 4. 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 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

This work is motivated by three observations. First, as mentioned in Section 1, the fundamental limitations and unsatisfactory trade-offs of existing techniques for network monitoring require investigation of alternatives. Second, standard operating system mechanisms that could support a programmable monitoring system with packet-level granularity are too heavyweight and intrusive. Developing a custom system poses several interesting design challenges from a systems perspective. Last, we view network monitoring as a case for active networking, testing existing arguments and knowledge.

2.1 Limitations of existing techniques

We discuss in turn why SNMP-like abstractions, existing Management-by-Delegation (MbD) 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 is often hard to predict without significant operational experience. The lack of management hooks hinders the development of appropriate management tools as new features are added to the network. The interaction of ECN [29] with Netflow provides a good example of this: 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 using Netflow is thus difficult. Another example is measuring the rate of TCP SYN packets (for SYN attack 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 [15] 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 strictly the information exported by SNMP/RMON/Netflow. Therefore, the problems described in the previous paragraph also hold for MbD designs.

Active measurements are easy to implement, but have certain limitations. First, these techniques create probe traffic which may interfere with regular traffic in unpredictable ways. Also, probe traffic is often treated differently than regular traffic. The quality and validity of the information provided by these probes raises further 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 new real-time functionality is needed. Second, maintaining the huge data sets for post-processing is often inefficient. On-the-fly data reduction may be more efficient, especially if the reduction has little processing overhead; in Section 4.3 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 [4] only captures packet headers (IP and TCP) but some applications may require access to the payload.

2.2 Design challenges in programmable packet monitoring

Previous work context has demonstrated some of the benefits of programmable packet monitoring, but has also exposed certain design challenges. In particular, the LAME system [3] supports a fair mix of applications at typical network speeds. However, the emphasis in LAME on off-the-shelf components dictated the use of commodity operating system abstractions and system mechanisms to provide security and fault-isolation. These mechanisms introduced significant base and per-application overheads that render the architecture unsuitable for higher network speeds. The Windmill system [24] is similarly structured and pays a similar cost without providing protection, as protection was not an objective in Windmill's design. 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 [16, 2] 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 [1], 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 [28] as well as passive measurements [24]. 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 [27], represent network-side functions that have already stressed the need for extensibility. The target system structure should be easy to integrate in extensible router designs such as [18, 21]. 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 engineering question, however, we do not touch upon these issues in this paper.

3 Design

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 [12] 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 slower time scales and lower priority. This is needed so applications can 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 3.1. 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 3.2.

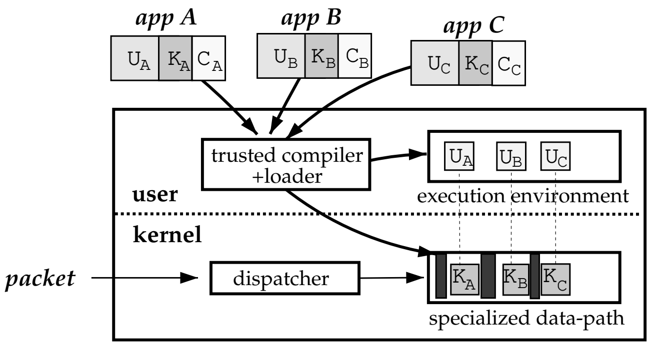

Figure 1: **FLAME Architecture**

3.1 Safe Execution

Allowing user code to execute inside the operating system kernel raises a number of major design challenges. The system must 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 [20, 32, 9]. FLAME leverages these techniques to satisfy our security needs.

Bounding Execution Time. A simple method for bounding execution time is eliminating backward jumps. This has the advantage of providing us with an upper bound for the execution time: linear in the length of the program. However such a limitation makes programming cumbersome. Alternatively, we can execute 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 — we only need sequential execution of monitoring functions on incoming packets. We take a different approach similar to [17]: we augment the backward jumps with checks of a cycle counter; if the module exceeds its allocated execution time we jump to the next module . This adds an overhead of 5 assembly instructions for the check and another 6 if the check succeeds, 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 [12]. 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

```
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 2: **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.**

crashes, buffer overflows, format string attacks, and so on. All Cyclone programs must pass a combination of compile-time, link-time and run-time checks to ensure safety.

3.2 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 [6] 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 must provide. The KeyNote [5] implementation is used as our trust management system. An example KeyNote credential is shown in Figure 2. In the current FLAME design, we perform policy compliance checks while loading 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

The current Internet architecture offers little protection against ill-behaved traffic. In recent years Distributed Denial of Service (DDoS) attacks have increased, which has led to the study of appropriate "traceback" mechanisms. A traceback mechanism detects the attack source(s), despite the fact that IP source addresses may be spoofed by the attacker, and responds by confining or blocking traffic from the attacking sites. FLAME allows an implementation of IP Traceback [31, 23, 34] without requiring modification of router functionality to support confinement or traffic blocking.

We implement Traceback on FLAME 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. We use out-of-band communication to recursively trigger traceback on neighboring routers, to find the source of the attacks. Once the attackers have been traced, router control commands are 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 [10] 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 gives 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 [26] although the concept and techniques have generally existed since the early description in [7]. 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. 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. As discussed in Section 2, both active and passive measurement systems have certain drawbacks.

We have built a simple traffic analysis module to demonstrate the ease of implementation on FLAME. It also demonstrates how our per-packet monitoring is capable of safely maintaining state across multiple packets. 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”. 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

It is important to obtain network usage statistics per accountable entity when managing an operational network. These statistics often form the basis for charging users for network usage. Today, volume-based charging schemes rely on NetFlow or SNMP-type accounting for calculating the cost of service, 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 for charging (such as the approach described in [22]) are based on the ECN [29] mechanism. When the network becomes congested, routers mark packets with a probability depending on the level of congestion. Marked packets are charged at an elevated rate, so resources consumed during periods of congestion are more expensive. This scheme provides incentive for users to shift their use of network resources to periods of reduced congestion.

To support ECN-based charges, we index accounting information by network prefix as well as by 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 FLAME implementation is fairly simple. The kernel-level module looks up the appropriate entry in the accounting hashtable, while user-level code flushes 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 1 GHz Pentium III PCs with 256 MB SDRAM, and 1 Gbit/s Ethernet interfaces (NetGear GA620 32-bit PCI cards with the

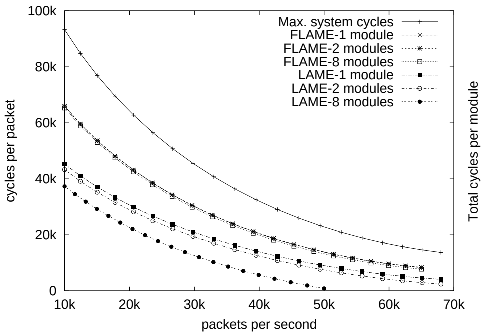

Figure 3: **Monitoring capacity of FLAME vs. LAME for different traffic rates**

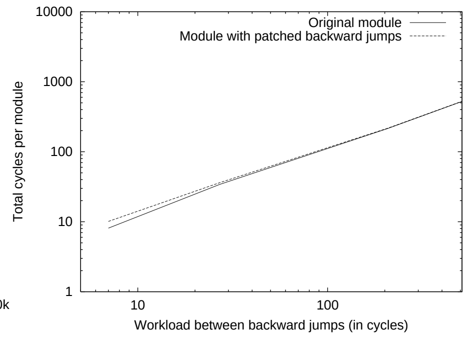

Figure 4: **The cost of patching backward jumps**

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

Figure 5 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 accommodate 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" (DDUMP) example, the kernel-level structure of FLAME provides further performance benefits.

In Section 3.1 we described our approach for guaranteeing bounded execution time. Figure 5 presents the evaluation of this approach. We sequentially execute ten modules with and without our backward jumps patch. Furthermore, we simulate a workload

Task	LAME cpp	LAME error	FLAME cpp	FLAME error
NULL	1831	± 89	80	± 15
IPANON	131	± 8	138	± 10
DDUMP	3293	± 416	2418	± 208
ECNACC	850	± 43	952	± 55
PTRAIN	101	± 0	120	± 0

Table 1: **Cost breakdown per-application in cycles per packet (cpp). Cycle counts reported are in *addition* to the base overhead reported in NULL.**

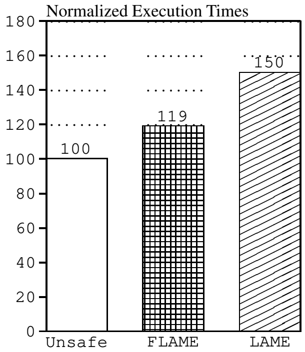

Figure 5: **LAME, Cyclone and FLAME overheads for the packet train module**

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 workloads.

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 5 presents the normalized overheads. Cyclone has an additional 19% overhead over the unsafe case, due to the checks that are necessary for type 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 [25] (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 [4] is a dedicated host-based monitor for snooping on 155 Mbit/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.

LAME [3] shares many architectural features with FLAME – dynamically loading user-provided modules, a Trust Management [6] approach to security — but the emphasis in LAME on off-the-shelf components introduced significant overhead that rendered the architecture unsuitable for high performance applications. Windmill [24] is an extensible network probe environment which allows loading of "experiments" on the probe, with the purpose of analyzing protocol performance. As discussed earlier in this paper, Windmill does not provide safety and does not efficiently scale to higher network speeds or application workloads.

The NIMI project [28] 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 [33] and ABLE [30] 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.

FLAME installs modules in the monitoring system, close to the information source. This is similar in principle to Management-by-Delegation (MbD) models [15]. 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.

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.

We observe that 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 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 6. The structure of FLAME follows typical design patterns for active-network prototypes while incorporating crucial system-level optimizations for high performance.

Although open questions in the software structure of our system remain, we also expect to investigate programmable packet monitoring systems that use hardware support. We see tremendous potential in the use of network processors such as the Intel IXP 1200 [19]. Finally, we would like to study other distributed designs. Since the

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

Figure 6: **Comparison of network monitoring systems**

application workload will consist of several independent modules, there is a natural decomposition into independent, concurrent components. These can be run, in parallel, on different processors. We anticipate that our approach can thereby be extended to higher network speeds and more intensive workloads.

Acknowledgements

This work was supported by the DoD University Research Initiative (URI) program administered by the Office of Naval Research under Grant N00014-01-1-0795, and DARPA under Contracts F39502-99-1-0512-MOD P0001 and N66001-96-C-852. The `ipanon` code was kindly provided by Joerg Micheel. We would like to thank M. Hicks, J. Kornblum, J. Moore, E. Markatos and H. Bos for valuable comments and suggestions throughout the course of this work.

References

[1] D. S. Alexander, P. B. Menage, W. A. Arbaugh, A. D. Keromytis, K. Anagnostakis, and J. M. Smith. The price of safety in an active network. *IEEE/KICS Journal of Communications and Networks (JCN)*, March 2001.

[2] 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 (IWAN)*, October 2000.

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

[4] J. Apisdorf, k claffy, K. Thompson, and R. Wilder. OC3MON: Flexible, affordable, high performance statistics collection. In *Proceedings of the 1996 LISA X Conference*, 1996.

[5] 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.

[6] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings of the 17th Symposium on Security and Privacy*, pages 164–173, 1996.

[7] J. Brunner. *The Shockwave Rider*. Del Rey Books, Canada, 1975.

[8] 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.

[9] J. Chase, H. Levy, M. Baker-Harvey, and E. 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.

[10] K. Cho. A framework for alternate queueing: Towards traffic management by PC-UNIX based routers. In *Proceedings of USENIX 1998 Annual Technical Conference*, June 1998.

[11] Cisco Corporation. Netflow services and applications. *http://www.cisco.com/*, 2000.

[12] Cyclone user's manual. Technical Report 2001-1855, Department of Computer Science, Cornell University, Nov. 2001. Current version at *http://www.cs.cornell.edu/projects/cyclone/*.

[13] D. Dean, M. Franklin, and A. Stubblefield. An algebraic approach to ip traceback,. In *Proceedings of NDSS '01*, February 2001.

[14] N. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. In *Proceedings of the ACM SIGCOMM'00 Conference*. August 2000.

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

[16] M. Hicks and A. D. Keromytis. A secure PLAN. In *International Working Conference on Active Networks (IWAN)*, 1999.

[17] M. Hicks, J. T. Moore, and S. Nettles. Compiling PLAN to SNAP. In *Proceedings of the 3rd International Working Conference on Active Networks (IWAN)*, October 2001.

[18] ICIR. XORP: An extensible open router platform. *http://www.xorp.org/*, January 2001.

[19] Intel Corporation. Intel IXP1200 network processor. http://developer.intel.com/ixa/.

[20] T. Jim, G. Morrisett, D. Grossman, M. Hicks, J. Cheney, and Y. Wang. Cyclone: A safe dialect of C. In *Proceedings of USENIX 2002 Annual Technical Conference*, June 2002.

[21] S. Karlin and L. Peterson. VERA: An extensible router architecture. In *Proceedings of IEEE OPENARCH 2001*, pages 3–14, April 2001.

[22] K. Laevens, P. Key, and D. McAuley. An ECN-based end-to-end congestion-control framework: experiments and evaluation. Technical report, Microsoft Research, TR MSR-TR-2000-104, October 2000.

[23] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network (extended version). *http://www.aciri.org/pushback/*, July 2001.

[24] G. R. Malan and F. Jahanian. An extensible probe architecture for network protocol performance measurement. In *ACM SIGCOMM'98*, 1998.

[25] S. McCanne and V. 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.

[26] D. Moore. The spread of the code-red worm (CRv2). In *http://www.caida.org/analysis/security/code-red/*. August 2001.

[27] C. Partridge, A. Snoeren, T. Strayer, B. Schwartz, M. Condell, and I. Castineyra. FIRE: Flexible intra-AS routing environment. In *Proceedings of the ACM SIGCOMM'00 Conference*, pages 191–203. August 2000.

[28] V. Paxson, J. Mahdavi, A. Adams, and M. Mathis. An architecture for large-scale internet measurement. *IEEE Communications Magazine*, 38(8):48–54, August 1998.

[29] K. Ramakrishnan and S. Floyd. A proposal to add explicit congestion notification (ECN) to IP. RFC 2481, *http://www.rfc-editor.org/*, January 1999.

[30] D. Raz and Y. Shavitt. An active network approach for efficient network management. In *Proceedings of the 1st International Working Conference on Active Networks (IWAN)*, pages 220 –231, 1999.

[31] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. In *ACM SIGCOMM*, August 2000.

[32] F. B. Schneider, G. Morrisett, and R. Harper. A language-based approach to security. *Informatics: 10 Years Back, 10 Years Ahead*, pages 86–101, 2000.

[33] 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.

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

[35] S. Waldbusser. Remote network monitoring management information base. RFC2819/STD0059, *http://www.rfc-editor.org/*, May 2000.