Practical Network Applications on a Lightweight Active Management Environment

K. G. Anagnostakis, S. Ioannidis, S. Miltchev, and J. M. Smith

CIS Department, University of Pennsylvania
200 S. 33rd Street, Philadelphia, PA 19104, USA
{anagnost,sotiris,miltchev,jms}@dsl.cis.upenn.edu

Abstract. The main focus of active networking research so far has been at the infrastructure level, facing the challenges of designing suitable node operating system structures and the study of different programming models. This has left exploration of the actual utility of active networks to rather simple applications that have yet to exploit the full potential of the programmable network.
In this paper we present an application-driven study of active networks, identifying unique and practical applications that make full use of the active infrastructure. We explore a class of applications in network monitoring that indicate a clear need for programmability as offered by active networking technology.
We have built several monitoring applications on an active substrate that is synthesized from off-the-shelf components. We demonstrate the flexibility provided while showing that for certain application workloads such a system can efficiently operate at modern backbone network speeds. Our performance study also leads to design considerations for scaling up the infrastructure to future network speeds.

1 Introduction

While there is generally no doubt about the increased potential and flexibility of active networks compared to the state-of-the-art [25], adoption of this new technology is, to date, still limited. Performance and security considerations may have been important reasons for this, however, the lack of unique and appealing applications is equally or possibly more important. Motived by the so far rather dry focus on infrastructure issues, we argue for an application-driven approach. We identify and experiment with a class of applications in the field of network monitoring that we believe have the key properties for making active networking arguments.

Network monitoring is an increasingly important, yet difficult and demanding task on modern network infrastructures. As argued in [9], *the scalability of the stateless IP networks has been bought at the expense of observability*, which has in turn led to the recent study of several ways of monitoring networks in order to support control and management functions [10, 12]. Most routers offer built-in monitoring functionality, accessible using mechanisms such as SNMP [27] or

NetFlow [7]. However, the domain of network monitoring has exposed fundamental weaknesses of these static-functionality, protocol-based, parametric designs. User needs vary and are uncertain at design time, causing the available management interfaces to fall short of user expectations. Additionally, tasks attractive only to minorities of users are usually not cost effective 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.

Operational experience indicates a clear demand for a dynamically extensible system to support network monitoring and measurement-based applications. Our hypothesis is that allowing these applications to *share* a common, open and programmable monitoring platform can be more cost-effective and efficient than employing closed purpose-specific monitoring tools. The main idea is to allow *soft real-time* processing of traffic measurement data by application modules, as close to the information source as possible. This is in line with management by delegation (MbD) models [11] with the key difference lying in the level of abstraction: we argue for a traffic measurement approach, rather than using the already diluted SNMP-based information. With respect to deployment, it is important that for a significant number of applications the proposed system can be introduced as a local enhancement. Of course, more wide-spread deployment is desirable to support distributed applications such as the traffic regulation mechanism described in this paper.

Some unique characteristics of the application domain and the system we propose make this study both interesting and challenging. While formally belonging to "management plane" functionality, our system needs to operate at the tempo of packet forwarding. This means that on one hand, it can be developed as a separate system, without requiring modification of existing routers. On the other, it carries some of the performance concerns associated with the forwarding function. However, unlike packet forwarding, the need for real-time processing is less strict. For example, extensive buffering can be used to address transient peak demands. The task is also a good target for parallelization, which, while being interesting as a feasibility argument, is not the focus of this paper.

In this paper we describe a proof-of-concept implementation of simple monitoring applications on a programmable infrastructure. These applications are: traffic analysis, usage accounting and traffic regulation for distributed "denial-of-service" attack detection and prevention. We also present a lightweight active substrate designed to support experimentation with these applications, that is based on off-the-shelf components. Our initial experiments demonstrate that such an approach to active networking is indeed feasible. The flexibility provided by the system compared to its simplicity is admirable. We believe that the general methodology as well as the specific application domain is a promising avenue for further research.

The remainder of this paper is organized as follows. In Section 2 we present the applications and in Section 3 the active substrate used for building these applications. In Section 4 we discuss experiments demonstrating the feasibility and merits of our approach and studying performance issues. In Section 5 we present related work and in Section 6 we conclude and discuss further work.

2 Applications

2.1 Traffic Regulation

The current Internet architecture offers very few protection mechanisms against ill-behaved traffic. Especially in recent years there has been an increase in Distributed Denial of Service (DDoS) attacks [19, 20] as well as *flash crowd* effects. A denial of service attack occurs when a single or multiple hosts transmit large amounts of traffic targeting a specific network node. Flash crowds occur when a large number of users access the same server simultaneously[1], overwhelming the available resources.

The two main problems are to detect the attack source(s), despite the fact that IP source addresses are spoofed by the attacker, and to respond, by confining or blocking traffic from the attacking sites. Both problems are a natural fit for an active networking approach. In fact, recent studies [26, 24] essentially *assume* some router programmability for implementing these schemes. Network nodes need to obtain information from the network to reconstruct the path towards the attacking sites. This can be done by monitoring traffic and recording the upstream router for each packet. Sampling techniques can be used to reduce the cost of monitoring. We show that this can be accomplished without modifying the router functionality. In response to an attack and once the attacker has been traced, router control commands can be initiated from our active substrate to block or otherwise confine the attacking traffic.

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

2.2 Traffic Analysis

While primarily a tool for Internet research, traffic analysis is also becoming important for service providers to support functions such as traffic engineering [10]. SNMP [27] and NetFlow [7] impose an *a-priori* specification of the granularity of the traffic data in the form of standardized objects. This was also the main trigger for the development of *passive measurement* systems such as OC3MON [3]. These systems passively "listen" on a network link and dump packet headers to disk. Trace files can then be downloaded for off-line analysis. The flexibility lies in that these systems capture the full stream of data for later processing. There are, however, certain constraints with this approach:

[1] This is also known as the *Slashdot* effect.

– Analysis of the data has to be performed post-mortem. There is no way to perform the measurements 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.
– As the mismatch in growth trends between bandwidth, processing, disk speed and disk capacity increases, it is going to be increasingly impractical to maintain huge data-sets for post-processing. On-the-fly reduction of data by processing may be more efficient, especially if the processing task is limited, as is the case in the example module presented below.

The contribution of an active networking approach is that users, depending on established trust relationships, can install modules on the monitoring system for efficiently performing *real-time* traffic analysis.

We have built a simple analysis module to demonstrate the power of an active networking solution. The goal of our module is to observe 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. This kind of information is not available through the standard network management mechanisms. Also, a typical passive measurement system requires communication of all traffic to the analysis host. Our implementation is clearly more efficient, easy to implement, and adds minimal overhead to the measurement system. The module consists of about 40 lines of code (with an overhead of about 101 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 easy to build. Similar functionality is not present in existing systems.

2.3 Accounting

The task of obtaining network usage statistics per accountable entity is important for managing an operational network. Often, this forms the basis for *billing* users for network usage. Volume-based billing schemes in use today rely on NetFlow or SNMP-type accounting for calculating the billing 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. A representative example in this category is based on the ECN [22] mechanism as described in [16]. 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 provides incentive for users to make reasonable use of network resources during congestion.

To support ECN-based charging schemes, one must collect accounting information per network prefix as well as marked vs. non-marked packets. This is not

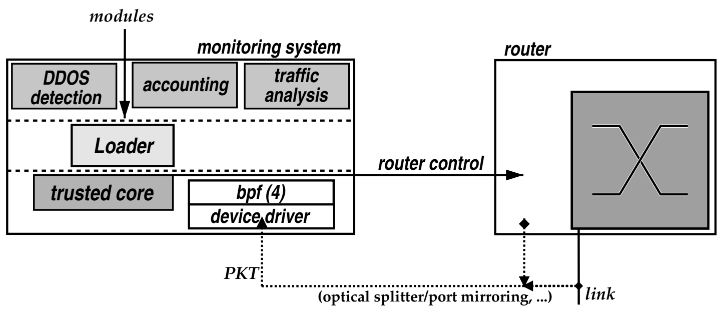

Fig. 1. Structure of the active substrate

supported by any of the existing router accounting mechanisms: NetFlow allows either per-ToS *or* per-AS *or* per-network accounting tables. We will demonstrate that the implementation of this scheme based on our lightweight active substrate is fairly simple.

3 The Lightweight Active Management Environment

The active substrate for the experiments in this paper is built entirely from off-the-shelf components. Two reasons influenced this choice: firstly, the goal of the paper is not to build a new active networking infrastructure and secondly, our claim is that an active network *can* and *should* be built in an incremental fashion. The overall architecture of our system is shown in Figure 1, and is roughly equivalent to the structure of Switchware [2] and other active networking prototypes. The system is built around the OpenBSD 2.8 [1] operating system. OpenBSD provides an attractive platform for developing secure applications because of the well-integrated security features and libraries (e.g. IPsec stack, SSL, KeyNote, *etc.*). Similar implementations, however, are possible with other operating systems or active network platforms. In the following paragraphs we will describe the various components of our system.

Loader The module loader is implemented as a system daemon which accepts TCP connections for loading and controlling modules. Users need to authenticate themselves to the active node by establishing a secure IPsec tunnel. The modules are implemented as native code shared objects. Upon receipt of a module, the loader can either execute it in its own virtual address space or load it inside the operating system kernel. The decision depends on the kind of credentials the objects use to authenticate to the loader. In our current prototype we assume that code executing in the operating system kernel will not accidentally harm the system. There are however a number of techniques to shield against this type of errors, like software-based fault isolation [28] and the use of large address spaces [29], which can be easily adopted in our active substrate.

```
KeyNote-Version: 2
Authorizer: NET_MANAGER
Licensees: TrafficAnalysis
Conditions: (an_domain == "an_exec" && module == "capture" &&
  (srcip  == 158.130.6.0/24 ||
   dstip  == 158.130.6.0/24)       /* own network  */
   && snaplen  == 40)              /* headers only */
-> "ANONYMIZE";
Signature: "rsa-md5-hex:f00f5673"
```

Fig. 2. Example credential that grants "TrafficAnalysis" capture access to the network interface for traffic to/from 158.130.6.0, packet headers only. The packet source and destination addresses must be anonymized.

Trusted Core The applications that use our system require access to the packets going through the node. On UNIX this can be accomplished using the Packet Capture library (`pcap(3)`) which provides wrapper functions for the Berkeley Packet Filter (`bpf(4)`). We extended the `bpf_tap` function inside the `bpf(4)` device driver to process the packet and packet headers according to the privileges of the application. For example, an application might be permitted access to specific flows only, as shown in Figure 2, or be allowed to access packets headers (instead of full packets) and only after the IP addresses are anonymized for privacy (*e.g.* Figure 3). Traffic information can subsequently be passed on to the applications.

Security Policy We take a *Trust Management* [5] 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.

We have chosen to use KeyNote [4] as our trust management system. 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

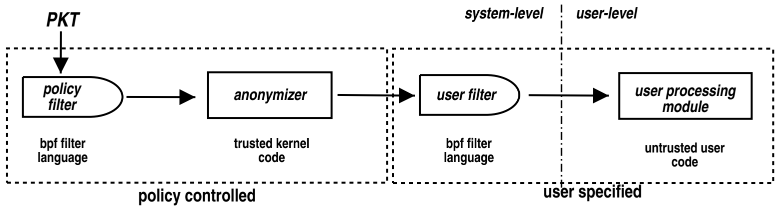

Fig. 3. The cycle of a packet being processed by the system and user code.

as time of day and node name, that are important to the policy management function.

We use KeyNote for performing policy compliance checks and settings when loading up the incoming object code. The KeyNote credential specifies what resources will be allocated to the newly created process. Object modules that execute inside the kernel have no resource bounds. However user level processes are assigned specific resource permissions depending on the credentials they carry. We rely on the `rlimit` structure of the UNIX operating system, where limits for CPU time, memory size, number of allowed processes, *etc.*, are specified. We also forbid user processes from modifying those values. We did not attempt to make the environment totally tamperproof as that would have been beyond the application-oriented scope of this work. However, there has been extensive research in this field which can be easily incorporated in our system [15, 14].

4 Experimental Study

A number of experiments were performed with the implementation of our system on the test-bed shown in Figure 4. The experiments aim primarily to validate our design and study system performance.

Our test-bed consists of 7 x86-based routers and an "edge" machine, all running OpenBSD 2.8. Five of these machines are 1GHz Pentium III with 256MB SDRAM, two are 400MHz Pentium III with 256MB of SDRAM (*Kerkyra* and *Ithaki*) and one is a 166MHz Pentium with 256MB of SDRAM (*Naxos*). All links in this topology are point-to-point. The core of the network (resembling a network backbone) is comprised of 1 Gbit/s Ethernet links (3Com 3C985-SX 33MHz 64 bit PCI) and the surrounding Ethernet links are 100 Mbit/s. The edge machine is connected with a 10 Mbit/s Ethernet to one of the backbone routers. We used the ALTQ [6] implementation for providing the router functionality assumed by the traffic regulation and ECN-based accounting applications.

4.1 System demonstration

Traffic Regulation To demonstrate the traffic regulation module we designed the following experiment. The attack progress and system response is illustrated

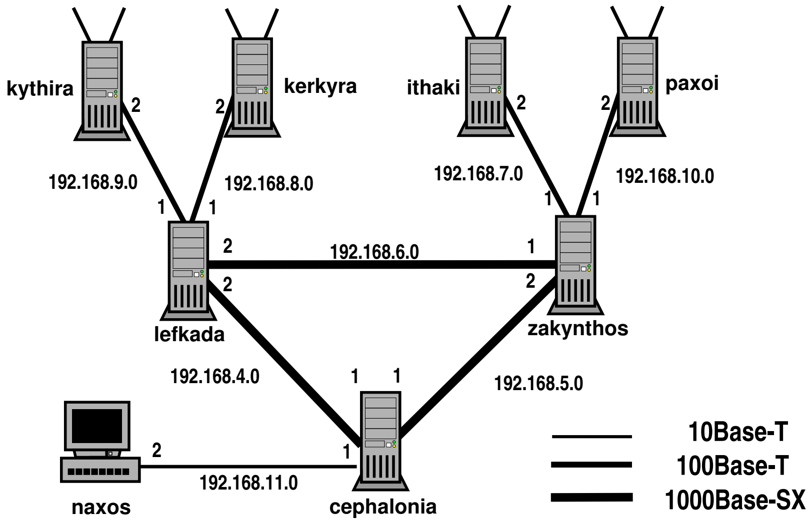

Fig. 4. DSL Test-bed layout.

in Figure 5. *Kythira* (source) starts a TCP connection with our edge host, *Naxos* (sink). We limit the flow to 10 Mbit/s using ALTQ on the source. At times 5 sec, 15 sec and 25 sec, *Kerkyra*, *Ithaki* and *Paxoi*, start a UDP flood attack to *Naxos*. Since the IP source addresses are forged the host under attack (*Naxos*) must use a form of IP traceback to single out the offending hosts. *Naxos* registers abnormal link utilization by employing a simple monitoring module that measures overall load on the 10 Mbit/s link. Upstream router *Cephalonia* is requested to start sampling packets. A second monitoring module on *Cephalonia* samples the packets destined to *Naxos* and determines that offending packets arrive through *Zakynthos* and *Lefkada*. *Cephalonia* subsequently sends the result to *Naxos*. *Naxos* requests *Zakynthos* and *Lefkada* to start sampling and they return *Kerkyra*, *Ithaki* and *Paxoi* as sources of the offending packets. Once the offending sources are identified, *Naxos* asks *Zakynthos* and *Lefkada* to block traffic from *Kerkyra*, *Ithaki* and *Paxoi*. As the offending traffic is blocked, TCP traffic slowly recovers to its pre-attack levels.

Traffic Analysis A simple traffic analysis experiment was performed with the packet train analysis module (code is presented in Appendix A). As our testbed does not carry real network traffic we generated traffic from host *Lefkada* to host *Cephalonia*, using a 3 hour long IP header trace captured at the University of Auckland [2]. The module code, running on *Cephalonia* is shown in Appendix A and the results of the measurements in Figure 6. Note that even a non-privileged

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

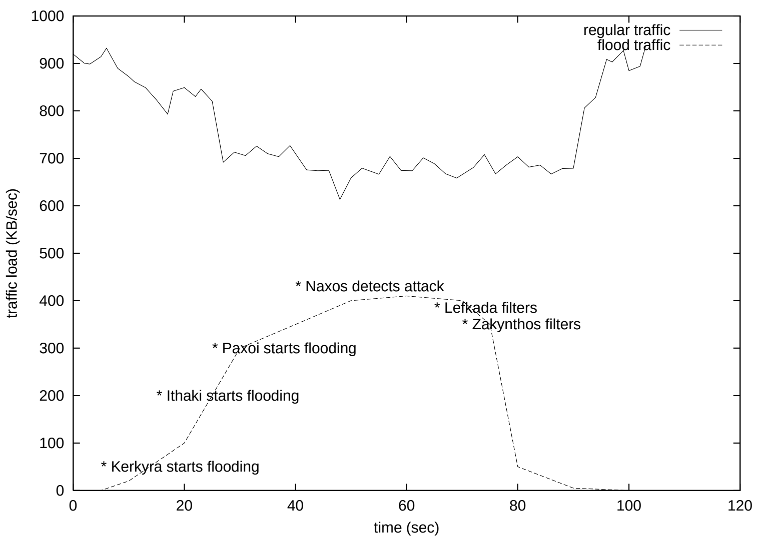

Fig. 5. Traffic flood escalation, detection and filtering events for traffic regulation experiment

user could be allowed to install and run such a measurement module: no access to the packet payload is needed, and the algorithm works with anonymized packets as well. This demonstrates how such an approach can provide the technical basis for allowing researchers to use similar modules on real network links.

Accounting The implementation of the ECN accounting module is remarkably simple (the module code is included in Appendix A). For each packet, the module checks whether the CE (Congestion Experienced) bit is set in the IP header, and calls the accounting function `add_to` with the appropriate table of marked or unmarked traffic as an argument. To test this module in our experimental set-up, we enabled ALTQ's ECN on the 100 Mbit/s link between *Ithaki* and *Zakynthos* (see Figure 4). We generate traffic from the hosts behind *Ithaki* in the following way:

- A simulated *Network A*, where a mixture of TCP connections is generated based on `ttcp`, with a Poisson distribution for connection arrivals and an exponential distribution for the connection duration (in bytes).
- A simulated *Network B*, with a mixture of TCP connections with short-lived bandwidth-savvy UDP streams , all using `ttcp`. UDP traffic is non-congestion controlled, and this is clearly reflected in the resulting charges in Table 1.

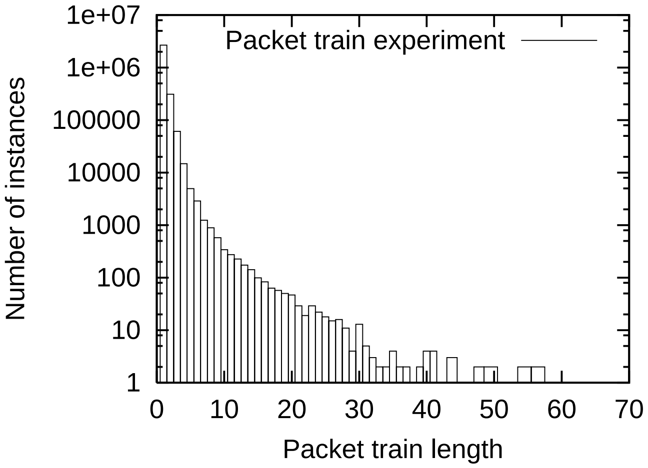

Fig. 6. Graphical representation of packet train distribution.

user	marked (charged)	unmarked (free)	total traffic	charge
Network A	104.29 MB	2.88 GB	2.99 GB	USD 52.14
Network B	283.84 MB	1.39 GB	1.67 GB	USD 141.92

Table 1. Charges matrix for ECN experiment.

4.2 Performance study

The questions that we attempt to answer with respect to performance is what cost the applications have in terms of processing and how much and what kind of overheads the system structure imposes. The results are summarized in Table 2. We send traffic from *Zakynthos* to *Cephalonia* and analyze performance at the receiving monitoring system.

To measure the system overhead, we have implemented a simple module that consumes a certain number of cycles for every packet received. We vary this number of cycles burned and measure how many packets are dropped by `bpf`. We calculate the maximum number of cycles available per packet on our system as the maximum number of cycles before the drop rate starts to grow above 4 % .Figure 7 shows an experiment using traffic with a packet size of 1000 bytes and a rate of 90 Mbit/s. In this experiment, performance starts to deteriorate at around 55000 cycles.

Using 473-byte packets at 155 Mbit/s rate (about 40000 pkts/sec) the number of cycles available is about 15000. Since *Cephalonia* is a 1.3 GHz P4 processor system, the theoretical maximum number of cycles available per packet is around 30000. The large overhead is due to context switches, kernel-user space crossings,

Task	Cycles/packet	
max. no. cycles (on 1.3Ghz P4)	31737	
max. no. cycles (after system overheads)	15277	± 1052
null module	1831	± 89
anonymization	131	± 8
dump to disk	3293	± 416
ecn accounting	850	± 43
pkttrain	101	± 0
ddos detection	390	± 1303

Table 2. Cost breakdown. Numbers in rows 1 and 2 refer to a 40 kpps traffic stream, equivalent to a 155 Mbit/s link fully utilized by 473-byte packets

memory copies and `bpf`-internal overhead that is unavoidable as we rely on a composition of off-the-shelf components. The results indicate that a specialized system structure could improve system performance up to 100 % , but also shows that a single, purely software-based system may not be sufficient to support heavy workloads at higher network speeds. However, considering the application workload (shown in Table 2) the system is still sufficiently fast to support a fairly extensive application workload at 155 Mbit/s line rates.

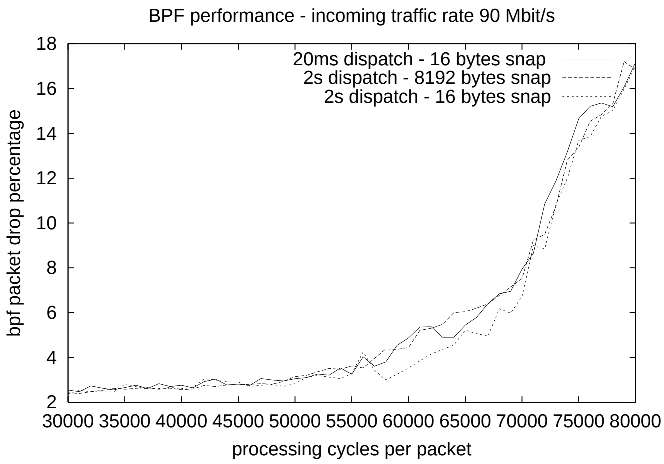

Fig. 7. BPF drop rate vs number of cycles consumed by each packet for 1000-byte packets arriving at a rate of 90Mbit/s

5 Related Work

Numerous system tools have been developed for network monitoring and performing management functions. The first generation of a measurement tools such as `tcpdump(8)` were based on the Berkeley Packet Filter [18]. `bpf(4)` provides operating system functionality for traffic monitoring. Users employ filters specified in a *Filter Language* that execute on a *filter machine.* inside the kernel. OC3MON [3] is a dedicated host-based monitor for 155Mbit/s OC3 ATM links. The host records the captured headers into files for post-processing. No real-time functionality for packet processing is considered in the original design. `ntop(8)` [8] is an end-system network monitoring tool that shares a subset of the motives of our approach to extensibility. `ntop(8)` allows plugins, in forms of shared libraries, to be developed independently from the system core. The difference lies in that full trust in plugins 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 [17] is an extensible network probe environment which allows loading of "experiments" on the probe, with the purpose of analyzing protocol performance. The NIMI project [21] provides a platform for large-scale Internet measurement, with Java and Python based modules. NIMI only allows *active measurements* (i.e. handling of probe packets, in contrast to getting real traffic from the wire). The security policy and resource control issues are addressed using standard ACL-like schemes. In the active networking arena, ABLE [23] provides an environment for network management applications. It allows applications to monitor the network using SNMP. Executing the SNMP applications closer to the managed elements reduces the communication burden. However, ABLE is limited to what is exposed through the SNMP MIBs.

6 Conclusions and Future Work

We have shown that network monitoring offers excellent soil for active networking solutions. This is mostly due to the continuously evolving and highly diverse nature of the monitoring applications. These characteristics impede designers and users from predicting and agreeing on a set of functions that can satisfy the needs of users in the long-term. If the quality of a design is judged by its ability to absorb the next wave of demand from its users, an active networking approach, as demonstrated in this paper, is a promising alternative to the existing designs. We also believe that such an application-centric, incremental methodology, can lead to experimental deployment of active networking technology while also advancing our understanding of the most crucial issues in the field.

To validate our claims, we have built an active substrate, based on off-the-shelf system components, and developed a set of practical applications. Our study establishes evidence on the feasibility of providing and the utility from exploiting a programmable monitoring infrastructure. We show that, from a performance perspective, our implementation is able to support a reasonable application workload at speeds of modern networks.

Developing practical network applications has given us insight on several aspects to explore further. For instance, our current implementation can control host-based OpenBSD routers for demonstration purposes. Extending our system to perform *active router control* functions involving routers is possible on the same architectural principles, although it may require different mechanisms for obtaining access to the actual network traffic. Furthermore, our performance study has indicated that a purely software-based approach can be much more efficient than the specific system structure used for our experiments. The design of a purpose-specific system can improve performance by 100 % and becomes an interesting topic for further work.

Departing from a purely software approach, we also see tremendous potential in the use of network processors such as [13]. Our system can be improved by moving the filtering function as well as some of the processing of the modules 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 diverging requirements on the design of network processors. We expect that monitoring workloads can be both processing and memory intensive which can be different from forwarding-type functions considered in the current network processor designs.

Finally, except for network processors, we would like to study other distributed designs. Due to the fact that the application workload will consist of several independent modules, it can naturally be distributed among different processors. The applicability of our approach can hereby 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 and Jon Moore for fruitful discussions, and the anonymous reviewers for their valuable comments that helped improve the quality of this paper.

References

1. The OpenBSD Operating System. http://www.openbsd.org/.
2. D. Scott Alexander, Michael W. Hicks, Pankaj Kakkar, Angelos D. Keromytis, Marianne Shaw, Jonathan T. Moore, Carl A. Gunter, Scott M. Nettles, and Jonathan M. Smith. The SwitchWare Active Network Implementation. In *Proceedings of the 1998 ACM SIGPLAN Workshop on ML*, 1998.

3. 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.
4. M. Blaze, J. Feigenbaum, J. Ioannidis, and A. D. Keromytis. The KeyNote Trust Management System Version 2. Internet RFC 2704, September 1999.
5. M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proc. of the 17th Symposium on Security and Privacy*, pages 164–173, 1996.
6. 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.
7. Cisco Corporation. NetFlow services and applications. (white paper), http://www.cisco.com/warp/public/cc/pd/iosw/ioft/netflct/tech/napps - wp.htm, 2000.
8. Luca Deri and Stefano Suin. Effective Traffic Measurement using ntop. *IEEE Communications Magazine*, pages 2–8, May 2000.
9. Nick Duffield and Matthias Grosslauer. Trajectory sampling for direct traffic observation. In *Proc. ACM SIGCOMM'00 Conference*. August 2000.
10. 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.
11. G. Goldszmidt and Y. Yemini. Distributed management by delegation. In *Proc. of the 15th International Conference on Distributed Computing Systems*, pages 333–340, 1995.
12. Matthias Grossglauser and David Tse. Measurement-based admission control. In *Proc. ACM SIGCOMM'97 Conference*. August 1997.
13. Intel Corporation. Intel IXP1200 Network Processor. (white paper), http://developer.intel.com/design/network/products/npfamily/ixp1200.html, 2000.
14. Sotiris Ioannidis and Steven M. Bellovin. Sub-Operating Systems: A New Approach to Application Security. Technical Report MS-CIS-01-06, University of Pennsylvania, February 2000.
15. J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy. The flash multiprocessor. In *Proc. 21-th International Symposium on Comp. Arch.*, pages 302–313, Chicago, IL, April 1994.
16. 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.
17. G. Robert Malan and Farnam Jahanian. An Extensible Probe Architecture for Network Protocol Performance Measurement. In *ACM SIGCOMM'98*, 1998.
18. 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.
19. 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.
20. 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.
21. V. Paxson, J. Mahdavi, A. Adams, and M. Mathis. An Architecture for Large-Scale Internet Measurement. *IEEE Communications Magazine*, 1998.

22. K.K. Ramakrishnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. Technical report, RFC 2481, January 1999.
23. Danny Raz and Yuval Shavitt. An active network approach for efficient network management. In *IWAN'99, LNCS 1653*, pages 220 –231, Berlin, Germany, 1999.
24. S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *Proceedings of ACM SIGCOMM'2000*, August 2000.
25. J. M. Smith, K. Kalvert, S. Murphy, H. Orman, and L. Peterson. Activating networks: a progress report. *IEEE Computer*, 32(4), April 1999.
26. Dawn Song and Adrian Perrig. Advanced and authenticated techniques for ip traceback. In *Proceedings of IEEE INFOCOM'2001*, April 2001.
27. W. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2.* Addison Wesley, 3rd edition, 1999.
28. Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham. Efficient Software–Based Fault Isolation. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 203–216, December 1993.
29. 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.

Appendix A - Module code

Packet train module code:

```
void foolet_pkt(u_char *myparams, void *pcaphdr, u_char *pkt)
{
        struct ip *iphdr = (struct ip *) (pkt + 14);
        static struct in_addr tr_src, tr_dst;

        if ((iphdr->ip_src.s_addr == tr_src.s_addr) &&
            (iphdr->ip_dst.s_addr == tr_dst.s_addr))
                train++;
        else {
                 pktstats[train] += train; train = 1;
                 tr_src = iphdr->ip_src;
                 tr_dst = iphdr->ip_dst;
        }
}
```

ECN accounting module code:

```
void foolet_pkt(u_char *myparams, void *pcaphdr, u_char *pkt)
{
        struct ip *iphdr = (struct ip *) (pkt + 14);

        if (iphdr->ip_tos & IPTOS_CE)
                add_to(marked_tbl, iphdr);
        else
                add_to(unmarked_tbl, iphdr);
}
```