SCAMPI - A Scaleable Monitoring Platform for the Internet

Jan Coppens
IMEC
E-mail: Jan.Coppens@intec.ugent.be

Evangelos Markatos
FORTH
E-mail: markatos@ics.forth.gr

Jiří Novotný
Masaryk University
E-mail: novotny@ics.muni.cz

Michalis Polychronakis
FORTH
E-mail: mikepo@ics.forth.gr

Vladimír Smotlacha
CESNET
E-mail: vs@cesnet.cz

Sven Ubik
CESNET
Email: ubik@cesnet.cz

Abstract

In this paper we describe the architecture of SCAMPI (A Scaleable Monitoring Platform for the Internet). SCAMPI allows easy writing of monitoring applications, which can run on top of different network adapters without changing the code and which can provide detailed monitoring of high-speed Internet circuits. This is made possible by MAPI (Monitoring API) and the SCAMPI adapter, a programmable hardware monitoring adapter with built-in monitoring functionality.

1 Monitoring of high-speed networks

Most backbone Internet circuits currently operate at speeds ranging from 1 Gb/s to 10 Gb/s. In order to verify operational, performance and security characteristics of the network and to enable problem resolution we need a high-speed network monitoring system. We need to measure elementary network performance characteristics, such as throughput, delay, packet loss rate and jitter. And we also need to search for traffic patterns indicating possible security problems, such as intrusion or denial of service attacks. Finally, we need a platform for creation of monitoring applications that can provide a view on network state at higher level of abstraction based on network monitoring.

We can recognize three types of network monitoring according to how information about network is obtained:

- Processing data from network components (e.g., SNMP counters and Netflow records)
- Active monitoring, which injects testing packets into the network and processes them as they are received in another part of the network
- Passive monitoring, which analyzes existing traffic in the network

All types of monitoring have their advantages and difficulties. Processing data from network components provides continuous per-hop information, but tends to be unreliable due to problems with router software. Active monitoring is the easiest way to measure one-way delay, but it is generally unsuitable for other network characteristics, as it measures characteristics experienced by testing packets, rather than by existing traffic. Therefore, passive monitoring, which does not influence existing traffic, has become a popular method of precise and reliable network monitoring.

However, passive network monitoring is becoming increasingly demanding on computing resources. The reason is that the physical network speed tends to increase faster than the computer processor speed. We already cannot monitor current high-speed network links just by tapping traffic with a regular network adapter, catching all packets with tcpdump and processing them even on the most powerful PCs.

2 SCAMPI architecture

SCAMPI is a two-and-a-half year European project to develop a scaleable monitoring platform for the Internet. SCAMPI concentrates on passive monitoring. It has two main goals:

- To enable easy writing of portable monitoring applications

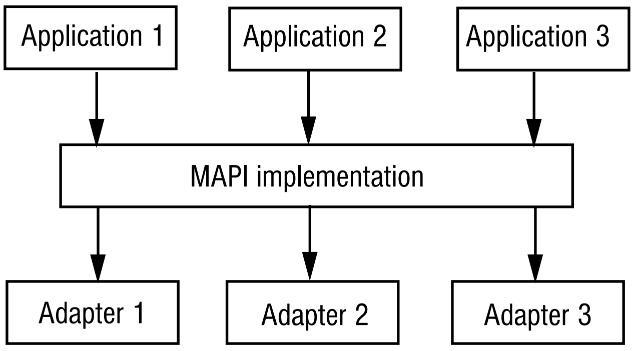

Figure 1. SCAMPI architecture

- To enable detailed monitoring of high-speed Internet

The first goal is realized by providing MAPI - Monitoring API, which enables application developers to start at higher level of abstraction of flows and monitoring functions.

The second goal is realized by performing certain time-critical functions needed for most monitoring tasks inside the SCAMPI adapter, a specialized programmable monitoring adapter. The data rate going further to the host computer is thus significantly reduced.

The project development effort therefore includes, going from bottom up, the SCAMPI adapter, MAPI and monitoring applications.

SCAMPI architecture is illustrated in Fig. 1. Several applications run concurrently on top of MAPI, which in turn runs on top of various network adapters. Currently, we support the SCAMPI adapter, DAG adapters and regular Ethernet NIC cards. Owing to the modular MAPI implementation, support for other network adapters can be added easily. Therefore, applications are portable between computers equipped with any of these adapters. When certain adapter provides some monitoring function in its hardware or firmware, MAPI will automatically use it. If it is not provided by the adapter, MAPI will use its own software implementation of the particular function.

We will describe individual components of the SCAMPI architecture in more detail, going again from bottom up.

3 SCAMPI adapter

An important part of the SCAMPI project is design, development and manufacturing of a specialized monitoring adapter. If we want to do detailed per-packet monitoring on high-rate traffic, we need to perform certain time-critical operations in hardware and its firmware. Pure software passive monitoring run on top of regular NIC card cannot even capture all packets on Gigabit Ethernet link even with advanced kernel-based accelerations [1]. And we need to monitor faster links, such as 10 Gigabit Ethernet and perform statistic calculations on captured packets.

Flexibility was one of the primary design goals. Therefore, the hardware is split into two cards - the universal motherboard and the interface card connected to the mainboard as a daughter board. In this way different interface cards can be used to connect to different network link types.

The SCAMPI adapters comes in two versions - Phase I and Phase II. The Phase I adapter consists of four components:

- COMBO6 mainboard
- 4-port SFP (optical) or TX (electrical) Gigabit Ethernet interface card
- Timestamp unit
- Firmware

The Phase I mainboard and two interface cards were actually developed as part of Liberouter project and 6NET European project. The original purpose was hardware accelerated IPv6 router, hence the name COMBO6. As the design was flexible, the adapter could be easily adapted for monitoring purposes. The timestamp unit, which provides each incoming packet with precise timestamp and firmware, which implements adapter functionality were developed in the SCAMPI project.

The Phase II adapter now being developed in the SCAMPI project will consist of three components:

- New mainboard
- 2-port XFP 10-Gigabit Ethernet interface card
- Timestamp unit
- New firmware

The new mainboard will support the faster interface card. One port on the interface card will be used for monitoring, whereas the other port can repeat packets from the first port. The new firmware will provide more functionality directly on the adapter including support of multiple simultaneous applications with different filtering requirements and it will enable faster packet processing.

Another well-known monitoring adapter is the DAG card from Endace [2]. When compared to DAG, the SCAMPI adapter will provide more functionality, it will be an open system allowing users to download their own firmware into the adapter and it is expected to be significantly less expensive. The estimated cost of the Phase II adapter (operating at 10 Gb/s) is 11000 Euro.

Interchangeable transceivers should allow monitoring of other physical and link layers, such as OC-192 and DWDM

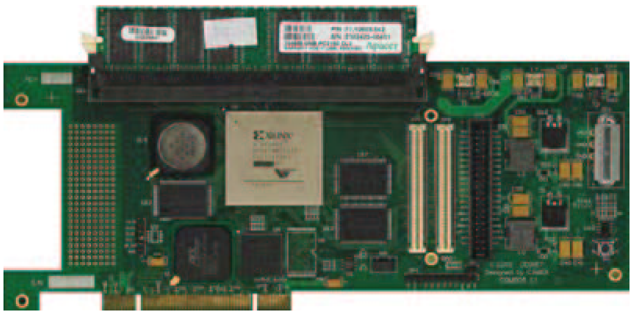

Figure 2. COMBO6 mainboard

links. Modifications in firmware will be required to support these links. We plan to work on these enhancements after the SCAMPI project.

The functionality of both the mainboard and the interface card has been programmed using Virtex II FPGA chips. The FPGA on the interface card has smaller number of gates than the FPGA on the mainboard. The adapter includes CAM (Content Addressable Memory) and other types of fast memory for packet processing. Sockets for regular DRAM chips for packet storage are provided. The COMBO6 mainboard is shown in Fig. 2.

4 Firmware

Firmware implements functionality of the adapter. During the SCAMPI project, we want to provide the following monitoring functions directly on the adapter:

- Header filtering (BPF syntax)
- Packet sampling (deterministic and probabilistic)
- Payload string searching (multiple strings)
- Various packet and byte statistics

Header filtering and possibly also sampling should significantly reduce volume of data transfered over the PCI bus to the host computer. Some applications will not need to capture any packets at all and will just read statistics computed on the adapter. Payload string searching allows to look for suspicious patterns, such as computer viruses.

The firmware is split into functional blocks which are separately designed. Some blocks can be used repeatedly in different projects. Recycling of blocks already designed (e.g., for IP header parsing) simplifies the VHDL design process.

Some of the blocks (such as HFE and LUP, which will be described later) are implemented as machines that we call "nanoprocessors", running dedicated programs. The nanoprocessor complexity lies between a Finite State Machine (FSM) and RISC processors. Nanoprocessors have

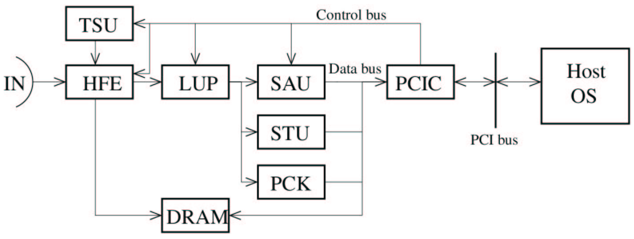

Figure 3. Structure of the adapter firmware

limited instruction sets. The nanoprogram is interpreted by a firmware block stored either in FPGA's BlockRAM or external SRAM. Instruction sets are designed especially for each nanoprocessor. The advantage of the nanoprocessor approach is the possibility to change block functionality at run time. There is no need to rewrite the source code (such as in VHDL), synthesize it and download the configuration data into the FPGA. It also makes the code design smaller and more efficient. The structure of the firmware is illustrated in Fig. 3.

HFE Header field extractor is a preprocessing unit which extracts valid data from IP and TCP headers and stores them in a unified header structure suitable for further processing.

LUP Lookup processor matches patterns in a packet header using 272-bit wide CAM with 8000 lines.

TSU Timestamp unit assigns high resolution timestamps to packets derived from the local clock. Timestamp is represented by a 64-bit value in a fixed point format, where 32 bits represent the fraction of a second.

DRAM Dynamic RAM stores received packets. It is directly accessible from the user space.

SAU Sampler unit is designed to reduce data exchange rate over the PCI bus when only samples are required. The unit provides both deterministic (i.e., each n-th packet is passed through) and probabilistic sampling (i.e., a packet is passed through with probability 1/n).

STU Statistic unit supports statistics computing. It counts packets and calculates $\sum x$ and $\sum x^2$ (where x is the length of the packet) for each of up to 256 categories of subflows defined by any 8 bits of the header.

PCK Payload checker performs content-based filtering. It can search up to 500 substrings of 16 bytes in the packet payload.

5 MAPI - Monitoring API

MAPI is SCAMPI middleware layer providing monitoring applications with an uniform access to monitoring capabilities across all low-layer network adapters and drivers. MAPI structure is illustrated in Fig. 4. MAPI is implemented as a `mapid` daemon running on the machine where network adapters are installed, reading data packets. Applications are linked with MAPI stub library and can run on the same or different machine. The stub library communicates with the `mapid` daemon using UDP sockets. It sends to `mapid` the application requests, such flow creation or application of a function to the created flow. It also reads from `mapid` the function results, such as computed statistics. MAPI provides a set of predefined functions for header filtering, sampling, payload string search and packet counters. Users can also write their own functions and apply them to flows. When whole packets or their parts should be passed all the way up to the application, they are transfered from `mapid` to the stub library using shared memory. We plan to implement a zero-copy packet mapping all the way from the kernel space.

The `mapid` daemon uses its `mapidcom` component to communicate with the stub library. The requests obtained from applications are forwarded to one of so called MAPI drivers, depending on which particular network adapter is used. MAPI drivers, such as `mapicombo6drv` for the SCAMPI adapter, `mapidagdrv` for DAG cards or `mapinicdrv` for regular NICs are user-space components providing a device-independent interface to network adapter functionality. MAPI uses a configuration file and its data structures to describe what functionality is provided by which network adapter and what functionality has to be provided by functions in MAPI itself. MAPI driver then communicates with a library and device driver provided by the manufacturer of a particular network adapter. For the SCAMPI adapter, we have developed our own `scampilib` library and `combo6drv` device driver.

MAPI also provides performance optimisation by eliminating duplicate functions on the same packet. For example, when more applications specify header filters including the same term, such as `dst port 2000`, the comparison of each packet against this term is performed only once for all applications. This optimisation can be performed either for filtering implemented in software or for filtering implemented on the adapter, in which case the specific architecture of CAM and lookup processor must be considered. Performance evaluation of the MAPI implementation was presented in [3].

Finally, MAPI implements admission control based on the KeyNote trust-management system [4]. A separate daemon `authd` communicating with `mapid` via shared memory is used for this purpose. The admission decision is

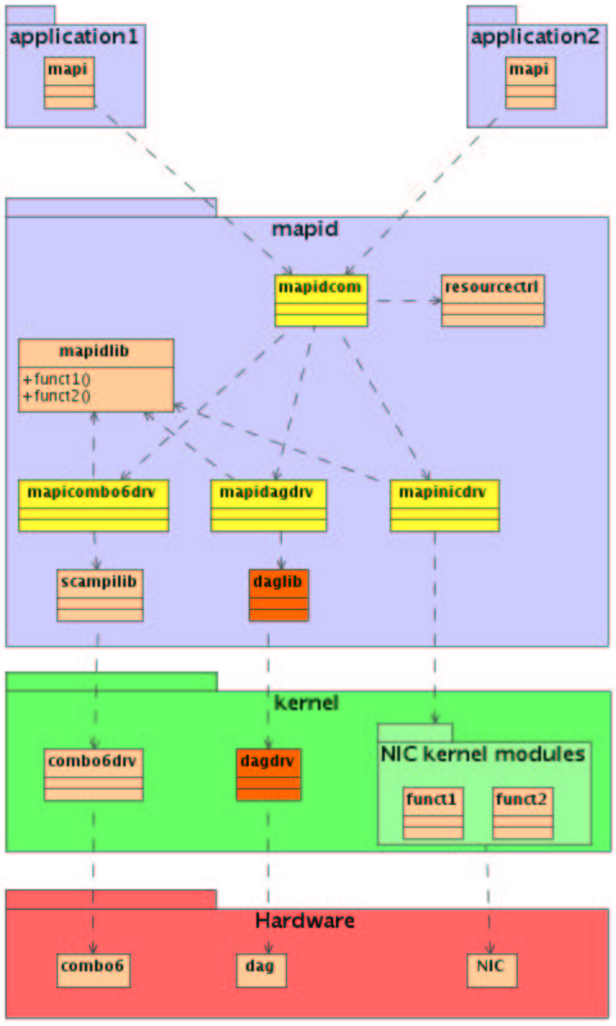

Figure 4. MAPI internal structure

made after the user opens a flow and specifies its options, so that they can be considered in decision process.

As an example, a simple application, which opens a flow, applies to it a header filter and a payload string search and which counts the number of passed packets can look as follows:

```
fd=mapi_create_flow("/dev/scampi/0");
mapi_apply_function(fd, BPF_FILTER,
   "src port 2000");
ctr_id1=mapi_apply_function(fd, PKT_COUNTER);
mapi_apply_function(fd, STR_SEARCH,
   "malicious string", 0, 1500);
ctr_id2=mapi_apply_function(fd, PKT_COUNTER);
mapi_connect(fd);

while(1) {
    sleep(1);
    mapi_read_results(fd, ctr_id1, &ctr_num1);
    mapi_read_results(fd, ctr_id2, &ctr_num1);
    /* . . . */
}
```

6 Applications

Next to the design and implementation of a scalable monitoring platform, the SCAMPI project also developed several monitoring applications. This section will roughly describe the functionality of these applications and the way they fit into the project. All developed applications focus on different capabilities of the monitoring system and use different techniques to deal with high-speed networks. Pure packet capture for instance focuses on the elimination of packets that are of no interest to the application. Flow record applications reduce the size of the captured data, without losing any valuable information, in order to deal for example with the limitations of memory capacity and PCI bus speed. Threshold alerting and QoS monitoring gather statistics of the flows and use sampling methods to reduce the captured network flow. Security applications, such as Intrusion Detection Systems, often need to analyze all captured data. In this case, efficient algorithms are needed in both the monitoring platform and the application.

6.1 Packet Capture and Libpcap-to-MAPI Interface

Packet Capture is a very basic monitoring application, which simply captures packets from the wire and does some additional processing. For instance, the incoming stream of packets can be filtered to obtain only those packets that the application is interested in. Furthermore, only certain parts of the packet (e.g., packet header) can be processed or saved to disk for a follow-up analysis. When we use a network adapter with built-in header filtering, such as the SCAMPI adapter, the volume of data is reduced before transferring to the host computer, enabling monitoring of high-speed traffic and reducing load of host computer CPU.

The libpcap-to-MAPI interface on the other hand translates libpcap functions to MAPI calls. This interface allows the execution of any libpcap-based application on a SCAMPI enabled platform. By using this interface, legacy applications can experience the performance of a SCAMPI system.

6.2 Flow Record Applications

Today, NetFlow is one of the most commonly used technologies for monitoring network usage and collecting information about network traffic. The work of collecting flow records is usually done by routers, which export the flow records to some collector. However, as the network speed increases, most routers are not able to do full flow analysis and have to use sampling to keep up.

The *flow record export application* uses the SCAMPI platform to export IPFIX flow records and will be used at high speeds where routers can not deliver flow records without using sampling. The *flow-based reporting application* on the other hand uses these flow records to produce a broad range of reports, showing various information about the network traffic. The user can access the reports through a web-based interface. Finally, the *host tracking application* focuses on the development of a flow probe and collector based on the CAPI (Collector API), a new API, defined by SCAMPI, that eases the creation of collector applications. CAPI is responsible for collecting flows, storing them persistently and providing facilities for performing queries on the flows for the purpose of traffic accounting and activity tracking.

6.3 Billing and Accounting

Internet and Application Service Providers use an accounting application to bill their customers based on their actual traffic or network usage. Porting such an application on SCAMPI will provide an ISP the ability to accurately measure various characteristics of network traffic, improve their services and provide advanced billing mechanisms and policies. The accounting application will gather input data from many SCAMPI probes, in order to obtain aggregated statistics or a more comprehensive “picture” of the whole network usage.

SCAMPI can provide the means to efficiently measure network performance and behaviour at high speeds, in order to feed billing components with the meaningful events and billable information. Henceforth, the perspective role of SCAMPI is to empower and encourage the adoption of modern economic and product models in the telecommunications and networking market.

6.4 Threshold Alerting for Traffic Engineering

Threshold alerting is a common part of network management platforms and is supported in both SNMP and COPS. There are several uses for these type of alerts, which all share a common requirement to the SCAMPI platform: applications want to receive information asynchronously (events).

The above mechanism is incorporated as part of a short-term traffic engineering research. In this context we need to split traffic entering a router on two outgoing paths according to a given weight distribution. The main goal of this application is to show the functionality of the event-based reaction to monitoring. This is accomplished by building a feedback loop between the MAPI, a local tunnel management point, and a possible resulting reconfiguration (re-mapping).

6.5 Quality of Service Monitoring

QoS monitoring analyzes the behaviour of a specified (e.g., SLS monitoring) or random stream (e.g., CoS monitoring) throughout a system under observation (ranging from a single link to a concatenation of ISPs). In this application, a two-layered architecture for QoS monitoring is implemented, i.e. a QoS monitoring layer and an application layer. The monitoring layer, belonging to a single Internet Service Provider (ISP), provides end-to-end QoS statistics of the observed network to the application layer. Statistics such as packet loss, delay and jitter are provided in a non-intrusive way throughout hashing-based sampling (trajectory sampling) in the ISP access/peering points. The measurements of the individual observation points are correlated and processed in a centralized database. Any application in the upper layer can request these end-to-end QoS statistics from the monitoring layer.

6.6 Security Application

NDISs (Network Intrusion Detection Systems) are an important part of any modern network security architecture. A NIDS constantly monitors network traffic, trying to detect attacks or suspicious activity by matching packet data against well-defined patterns. Such patterns, or *rules*, identify attacks by matching fields in the header and payload of the packet. For example, a packet directed to port `80` and containing the string `/bin/perl.exe` in its payload is probably an indication of a malicious user attacking a web server. This attack can be detected by a rule which checks the destination port number, and defines a string search for `/bin/perl.exe` in the packet payload.

Implementing a NIDS is rather a complicated task. Several basic operations like packet decoding, filtering, and classification, TCP/IP stream reconstruction, and string searching, must be crafted together to form a fully functional system. Each one of these operations alone requires deliberate decisions for its design, and considerable programming effort for its implementation. Furthermore, the resulting system is usually targeted to a specific hardware platform. For instance, the majority of current NIDSes are built on top of `libpcap` [5] packet capture library using commodity network interfaces set in promiscuous mode. As a result, given that `libpcap` provides only basic packet delivery and filtering capabilities, the programmer has to provide considerable amount of code to implement the large and diverse space of operations and algorithms required by a NIDS.

In contrast, MAPI inherently supports the majority of the above operations in the form of functions which can be applied to network flows, and thus, can be effectively used for the development of a complete NIDS. Consequently, a great burden is released from the programmer who has now a considerably easier task. As a matter of fact, we have developed a Network Intrusion Detection System based on MAPI. Based on the observation that a rule which describes a known intrusion threat can be represented by a corresponding network flow, overall implementation is straightforward. As an example, consider the following rule taken from the popular Snort [6] NIDS, which describes an IMAP buffer overflow attack:

```
alert tcp any any -> 139.91/16 143 (flags: PA;
      content:"|E8C0FFFFFF|/bin";
      msg: "IMAP buffer overflow";)
```

All packets that match this particular rule can also be returned by the following network flow, after the application of the appropriate MAPI functions:

```
fd = mapi_create_flow(dev);
mapi_apply_function(fd, BPF_FILTER,
   "tcp and dst host 139.91 and dst port 143");
mapi_apply_function(fd, TCP_FLAGS, "PA");
mapi_apply_function(fd, STR_SEARCH,
   "|E8C0FFFFFF|/bin");
```

Our MAPI-based NIDS operates as follows: During program start-up, the files that contain the set of rules are parsed, and for each rule, a corresponding network flow is created. Rules are written in the same description language used by Snort. Snort rules are converted by a separate module to the appropriate MAPI function elements, which are then applied to the related network flow. The rest of the functionality is left to MAPI, which will optimize the functional components of all the defined rules and deliver the packets that match any of them.

Implementing the above intrusion detection application using `libpcap` would have resulted in longer code and higher overheads. As shown in Figure 5, our implementation takes no more than 2000 lines of code, while the

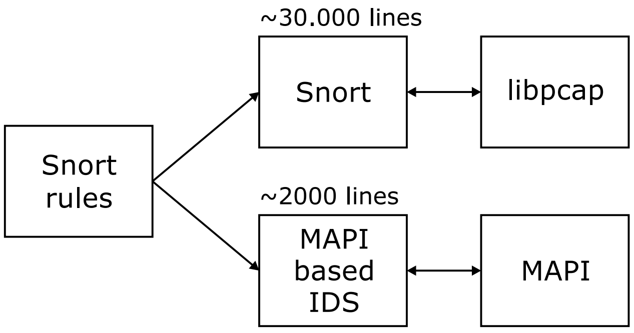

Figure 5. Comparison between Snort NIDS and MAPI-based NIDS.

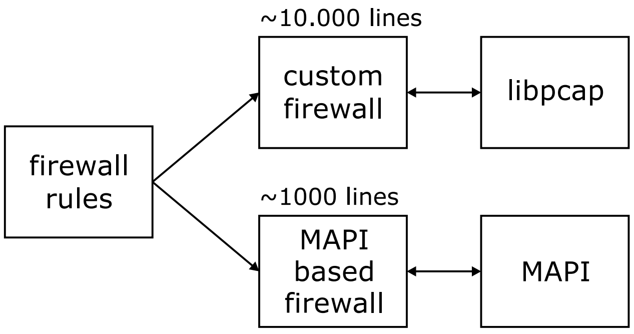

Figure 6. Comparison between custom and MAPI-based firewall applications.

core functionality of other popular NIDSes, such as Snort, consists of roughly 30.000 lines of code[1]. For example, `libpcap` does not provide any string searching facility, and thus the programmer would have to provide a significant chunk of code for the implementation of the chosen string searching algorithm. Instead of forcing the programmer to provide all this mundane code, MAPI already provides this frequently used functionality.

Note that a NIDS based on MAPI is not restricted to a specific hardware platform. MAPI operates on top of a diverse range of monitoring hardware, including more sophisticated lower level components like network processors, and thus, can further optimize overall system performance, considering that certain MAPI functions can be pushed to the hardware. Additionally, the functionality of the MAPI daemon can be shared by multiple concurrently running applications. For example, along with the intrusion detection application, one can develop a firewall application in the same fashion (i.e., in a few lines of code), as shown in Figure 6, adding this way extra capabilities to the overall system. Again, instead of providing code for the whole firewall operations, the programmer can use MAPI to reduce the development effort, and to effectively share resources by pushing the core firewall functionality into the MAPI daemon.

[1] Although the functionality of the two systems is not identical, it is clearly depicted a difference in code length of at least one order of magnitude.

7 Status of work and future plans

The Gigabit Ethernet version of the SCAMPI adapter is ready and available for monitoring. The 10 Gigabit Ethernet version and firmware blocks for packet processing are being developed and planned to be released in Fall 2004.

We plan to continue work on SCAMPI-based passive monitoring platform in the proposed Lobster (Large Scale Monitoring for Broadband Internet Infrastructure) project, which concentrates on deployment of passive monitoring for security applications. We also want to add more features for performance and security monitoring to the programmable monitoring adapter within the proposed GN2 project.

References

[1] L. Deri. Improving passive packet capture: beyond device polling. To be published.

[2] DAG cards. Endace Measurement Systems, `http://www.endace.com`.

[3] M. Polychronakis, K. G. Anagnostakis, E. P. Markatos, Arne Øslebø. Design of an application programming interface for IP network monitoring. To appear in the *Proceedings of the 9th IEEE/IFIP Network Operations and Management Symposium (NOMS2004)*, Seoul, Korea, 19.-23. April 2004.

[4] M. Blaze, J. Feigenbaum, J. Ioannidis, A. Keromytis. The KeyNote trust-management system version 2. *Request For Comments 2704*, Internet Engineering Task Force, September 1999.

[5] S. McCanne, C. Leres and V. Jacobson. libpcap. Lawrence Berkeley Laboratory, Berkeley, CA, available via anonymous ftp to `ftp.ee.lbl.gov`.

[6] M. Roesch. Snort: Lightweight Intrusion Detection for Networks. November 1999, available from `http://www.snort.org`.