SCAMPI: A Scalable and Programmable Architecture for Monitoring Gigabit Networks

Jan Coppens[1*], Steven Van den Berghe[1], Herbert Bos[2], Evangelos P. Markatos[3], Filip De Turck[1], Arne Oslebo[4], Sven Ubik[5]

[1]Department of Information Technology (INTEC), Ghent University, Gent, Belgium.

[2]Leiden Institute of Advanced Computer Science, Leiden University, Leiden, The Netherlands.

[3]Institute of Computer Science (ICS) Foundation for Research & Technology Hellas (FORTH), Heraklion, Greece.

[4]UNINETT, Trondheim, Norway [5]CESNET, Prague, Czech Republic

Abstract - *Effective network monitoring is vital for the growing number of control and management applications typically found in present-day networks. Increasing link speeds and the diversity of monitoring applications' needs have exposed severe limitations of existing monitoring techniques. As a response, the EU IST SCAMPI project designs and implements a scalable and programmable architecture for monitoring multi-gigabit networks. The SCAMPI architecture has an expressive programming interface, uses intelligent hardware, provides user policy management and resource control, and achieves scalability through parallelism. This paper addresses the problems with current high-speed network monitoring and presents the system architecture and components of the SCAMPI platform.*

Keywords: High-speed Network Monitoring, Programmable Monitoring, Open Monitoring Platform.

1 Introduction

With the widespread use and deployment of the Internet, traffic monitoring has been increasingly used as the mechanism to improve the performance and security of computer networks. In modern networks, different kinds of users would like to monitor different aspects of the network traffic for different purposes, e.g. traffic engineering, intrusion detection, denial of service detection, monitoring of service level agreements (SLAs), charging, etc. Some of these applications require

*Corresponding Author: Jan.Coppens@intec.UGent.be; Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium.

aggregate traffic statistics only, while others may be interested in monitoring each and every byte that travels through the network.

Existing tools tend to focus on a single subclass of network monitoring problems, while ignoring the others. Each of these monitoring tools is generally based on a specific set of primitives and functions, different from the ones used in other tools. To make matters worse, commercial vendors frequently use their own libraries and standards.

Besides the plethora of incompatible monitoring tools, current solutions are finding it increasingly difficult to keep up with modern link rates. As networks get faster and traffic analysis more complex, network monitoring gets slower. Monitoring techniques that used to be effective at low speeds (10 to 100Mbit) are becoming less and less useful when applied to current high-speed backbones [10].

The European Union IST project known as *SCAMPI* [15] addresses both of these problems by developing an affordable, high-performance and open network monitoring platform. SCAMPI started in April 2001 with the aim to implement all the necessary infrastructure to monitor networks at speeds exceeding 10 Gbps, including: (1) hardware (NIC), (2) middleware, (3) monitoring API or *MAPI*, and (4) applications that validate the SCAMPI approach. The key ideas behind SCAMPI are that programmable hardware is used to thin the datastream feeding into the host processor, while the middleware offers support for running multiple high-speed monitoring applications simultaneously (including resource control and policy management), and the MAPI offers a standardised API to applications that is much more expressive than what is offered by existing solutions. The MAPI, definition and implementation, is one of the main contributions of the project. While simple enough to support most existing network monitoring tools without having to change the code (for example, `libpcap` runs directly on top of the MAPI), it is expressive enough to allow applications to specify advanced queries on the data streams (e.g. 'scan all packets in a flow for a signature and return the packet if there is a match'). To our knowledge, such a query is impossible to express in existing APIs.

Of existing APIs, CoralReef [8] is probably the one that has the most in common with MAPI. CoralReef is a software suite for developing applications for capturing and analyzing network traffic. Analysis can be done in real time or from trace files and multiple hardware adapters are supported. The expressiveness of CoralReef is however limited to specifying BPF filters for removing unwanted traffic and there is currently no support for sampling, flow records or utilizing intelligent network adapters.

The applications that will be implemented in SCAMPI are derived from real-world problems

and chosen so as to cover a wide range of measuring requirements (e.g. traffic sampling, header capture, full payload capture of every packet, etc.).

The SCAMPI platform can be used by Internet Service Providers (ISPs) and Application Service Providers (ASPs), to provide for instance more advanced billing mechanisms. At the same time, it may improve the end-user experience through better network performance achieved by informed network management and traffic engineering. It also allows administrators to enhance the security of computer systems connected to the Internet by providing a better defence against cyberattacks, such as Denial-of-Service attacks and Intrusion attempts [2].

The remainder of this paper describes the problems of current monitoring architectures and the solution provided by SCAMPI. Section 2 addresses high-speed network monitoring and the hard- and software limitations when dealing with these kind of speeds. In section 3 the overall SCAMPI architecture is presented, while section 4 describes individual components such as the MAPI, the Module Organizer and the Job Control subsystem. A summary and conclusion is presented in section 5.

2 High-speed Network Monitoring

Monitoring is complicated by the fact that network speed increases at a rate that exceeds Moore's Law. For example, some research suggests that bandwidth doubles roughly every 9-18 months [3], an observation which is usually referred to as "Gilder's Law". At the same time, network monitoring applications tend to become more complex and demanding. Where early monitoring applications commonly required little information from the network (e.g., aggregated traffic or statistics), more recent tools may need a much more significant amount of information, possibly including both header and payload for each and every packet. Worse still, the amount of processing needed on this data tends to increase. For example, for detecting Internet worms and various other forms of cyberattacks, we may need such computationally intensive processing as string matching on the entire payload.

2.1 Hard- and Software Optimizations

In order to monitor gigabit networks, we have to overcome the hardware limitations of current monitoring devices. In PC-based monitoring platforms, captured packets are sent over the PCI bus to main memory.

Present-day PCI buses are 64 bits wide and run at 66Mhz, yielding a maximum throughput of 0.5Gbps at best, which is way too little to monitor gigabit backbones. Even though new PCI standards are being developed (e.g PCI-133 and PCI-533, designed for a maximum of roughly 1 GBbps and 4.3 GBps, respectively), the latter one is currently not commonly available [1]. In fact, it is doubtful whether PCI-X 533 will be widely available before the year 2005, by which time high-speed link rates may well have increased to 40 Gbps and beyond. Even ignoring the bus speed, the memory is another bottleneck, which is unable to keep up with the growth of link speed.

The obvious solution to this problem is to limit the number of packets and bytes that are transferred to the main CPU. By using configurable hardware devices or network processors (e.g. Intel IXPs [14]), the captured data that is not of interest to any of the applications can already be discarded in the hardware itself. To reduce the network flow even further, techniques such as packet sampling, counting and calculation of statistics can also be applied in the hardware. SCAMPI therefore aims to provide optimizations by configuring the hardware on behalf of the applications' requests.

In environments without such programmable hardware (e.g. lower-speed networks with traditional network cards), SCAMPI is still useful, albeit less efficiently. Note this is not a problem for lower-speed networks, as these are not capable of saturating the bus and memory bandwidth anyway. It is important to realise, however, that the SCAMPI programmable hardware may take many different forms and may not look like network cards at all. For example, Juniper routers allow for traffic filtering based on header fields [17]. The filtered traffic, in turn, can be mirrored to a router's port, on which we may connect a PC that has its interface in promiscuous mode. This way, Juniper routers can be used to make an effective first-pass monitor, and delegate the rest of functions to a general-purpose computer (e.g. a PC). In addition, Juniper routers provide possibilities for obtaining counters and statistics which can be exploited by SCAMPI. Similarly, although the cards by themselves offer fixed functionality, there are efforts underway to make ENDACE's DAG cards programmable, by way of daughter cards [21].

Besides programmable hardware, software optimizations, and in particular packet buffering techniques, can provide other means in order to keep up with high-speed networks. Commonly used applications for network monitoring purposes, such as the intrusion detection mechanisms and the network protocol analyzers exploit the libpcap API to communicate with the network devices. Libpcap and other user-level monitoring APIs gain access to the captured packets by copying them from kernel- to user-space. By doing so, a lot of valuable processing time is wasted due to this

redundant operation. As processing time is vital in high-speed networks, copying packets in memory will degrade the performance of the network monitor. In the proposed architecture, zero-copy packet handling, i.e. *mmap* [20], and advanced packet buffering techniques are used to transfer packet from hardware or kernel-space to user-space. In the SCAMPI project we implemented a driver that mmaps the IXP buffer to the application, as well as a driver, for commodity network interfaces, that provides packet filtering in the Linux kernel, with a buffer that is mmapped to the applications.

2.2 SCAMPI goals

Addressing the challenges posed by the state-of-the-art tools in network monitoring, SCAMPI represents a step towards building an affordable network monitoring system for high-speed networks that will enable the development of portable applications. SCAMPI employs a three-pronged approach in order to achieve its goals:

- **Standard Monitoring API (MAPI).** SCAMPI defines and implements a set of monitoring calls/primitives that are collectively called the MAPI. Monitoring applications are written using this MAPI. The MAPI is implemented on top of several platforms, decoupling the development of the monitoring applications from the monitoring environment on top of which the applications will be executed. Monitoring applications are written once, and are able to run on top of any monitoring environment without the need to re-write or re-compile the application.

- **Expressive power.** Current monitoring application programming interfaces provide little (if any) expressive power to application programmers. Application programmers are not able to communicate their monitoring requirements to the underlying network monitoring system. As a result, frustrated application programmers end up receiving all network packets in the address space of their application where they perform the operations they need. As a simple example of the poor expressive power of current network monitoring systems, consider a user that wants to sample one out of every 100 packets in order to find the most popular applications that use his/her network. Current network monitoring systems (like libpcap/Berkeley Packet Filters [16, 19], and Linux Socket Filters [13]) do not enable users to express such simple sampling requirements. Therefore, users that are interested in receiving just one out of every 100 packets are required to read all packets, and just discard 99 out of every 100 of

them. To overcome these limitations, SCAMPI's MAPI will enable monitoring application programmers to express their requirements to the underlying monitoring system, which in turn will decide how these requirements are more efficiently implemented. The MAPI will be discussed in more detail in Section 4.1.

- **Scalability through special-purpose hardware and parallelism.** Although network monitoring can be performed on top of traditional network adapters, SCAMPI, wherever possible, will exploit specialized network adapters that provide some monitoring functionalities in hardware. These devices contain on-board processors and FPGAs that can be programmed to perform monitoring functions and off-load the host processor, memory system, and I/O bus from much of their load. An important part of the SCAMPI project is the design, development and manufacturing of our own specialized network adapter. The adapter will be able to do packet filtering, sampling, payload inspection and produce traffic statistics in hardware at a rate of 10Gbit/s.

3 The SCAMPI Architecture

To monitor a high-speed backbone, a SCAMPI system will be connected to a splitter, which mirrors all the traffic of that link. The SCAMPI system will be composed of a regular PC coupled with a Hardware Monitor connected to the system's I/O (e.g. PCI) bus. Different instantiations of the SCAMPI system will require different hardware monitors. Low-speed monitoring systems will probably use a regular network interface, while high-speed systems may employ a special-purpose adaptor. This adaptor has the computing capacity to not only capture packets, but also to process them and perform some simple (but fast) monitoring functionalities. In the SCAMPI monitoring system, software will run on the host processor (a PC), both at kernel-level and at user-level, as well as on the SCAMPI monitoring hardware if possible.

User-level software will be composed of the following major modules: the job control system, the protection subsystem and the system that translates MAPI calls into the underlying system calls.

Figure 1 depicts the overall SCAMPI system architecture. The MAPI separates the monitoring applications from the monitoring infrastructure. Independent of the underlying infrastructure, monitoring applications will be written using the function calls provided by the MAPI. The MAPI enables programmers to write their application once and run them on top of different monitoring infrastructures without any changes. Generally speaking, the MAPI is a library which is linked in

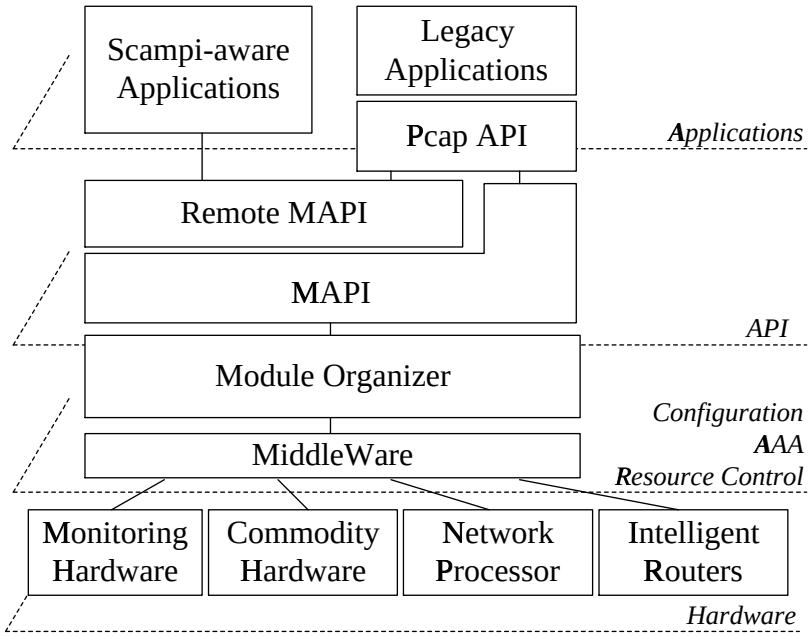

Figure 1: Generic Architectural Decomposition

with the application, providing (optionally distributed) access to the SCAMPI system. A special subset of the functions offered through this subset provide the same signature as `libpcap` in order to support legacy applications.

The MAPI communicates with the module organizer in order to aggregate the requests of the different clients into a single configuration. Other tasks handled by the module organizer are:

- optimize the configuration to handle current and future requests, taking into account the capabilities of the available hardware and exploit possible commonalties in the different simultaneous monitoring jobs;
- perform admission control based on user policies and current state of the system.

The actual monitoring jobs are executed in either software (the middleware), or in hardware, which can be either internal or external to the monitoring agent. From the implementations point of view, the functions in the MAPI give a view of the different capabilities below, without saying that a given function is available in soft- or in hardware. This means that the MAPI should offer three categories of functions:

1. Monitoring Job Control: including definition of monitoring jobs, installation and removal,

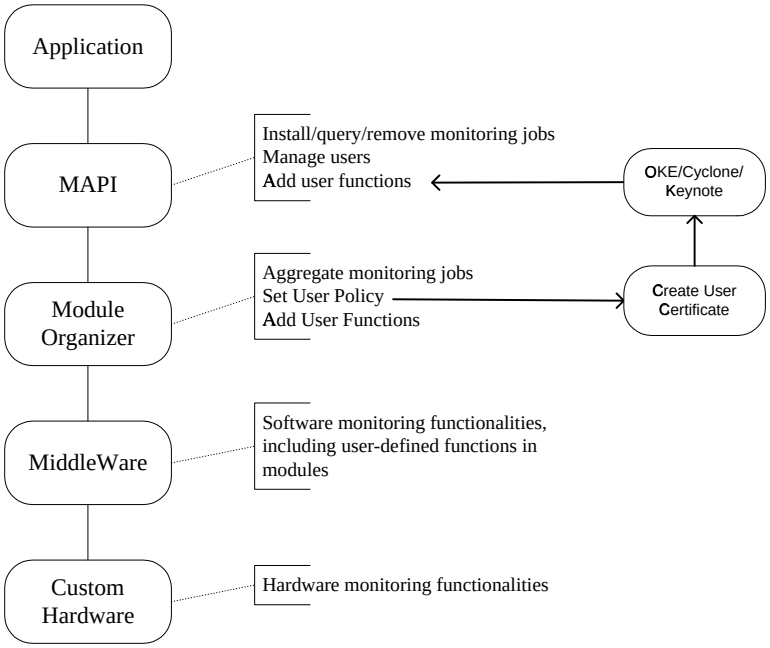

Figure 2: Implementation view on the architecture

result handling.

2. Programmability: adding and removal of user-defined analysis functions.

3. User Handling: including access control, authentication and profiles (or policies as discussed later)

While the first two categories will have repercussions in either hardware or software, the management of users is done in software only by the module organizer. The implementation view on the architecture is illustrated in figure 2. An application can use the MAPI to install, remove and query monitoring jobs. By calling the right MAPI function, user-defined functions can be loaded in the monitoring agent. To allow secure access to the monitoring agent and manage the operational SCAMPI system, administration applications can add and remove authorized users and configure their privileges. If one of the previous functions is called, the MAPI propagates it to the module organizer. If for example an administrative application adds a new user, the module organizer will create a credential containing the privileges and credentials of this new user. This credential is sent back to the requesting application using the same mechanism as currently found in the OKE trust management scheme, which in turn builds on KeyNote [5, 4].

4 A Programmable Monitoring Architecture

4.1 Definition of the Monitor API

Most applications will interact with the SCAMPI system through its Monitoring API or MAPI. The Monitoring API is a library that can be compiled into any application to encapsulate the communication with the centralized SCAMPI components. The MAPI can either run co-located with the SCAMPI monitoring device or can be accessed remotely. The following functions form the basic infrastructure of the MAPI.

- Network flow creation: creates a new flow containing only the packets of interest.
- Apply function: append one of the functions defined below or a user-defined function added to the system, to the chain of operations to be executed on the network flow.
- Network flow removal: releases the resources related to the network flow.
- Information retrieval: gets information from a given network flow, including statistics, packet traces or flow records.
- Events: get asynchronous information from the given network flow.

Once an application has created the subflow, containing only those packets the application is interested in, one or more functions can be executed on this flow of packets. These functions can be subdivided in the following categories:

- Cooking: re-order and select packets in such a way that a transport or application-level flow can be analyzed, e.g. to recreate a TCP flow all duplicate packets have to be discarded, packets that are out of order have to be re-ordered and fragmented packets need to be de-fragmented.
- Flow selection: this category classifies incoming packets into a network flow. This can be based on IP/TCP/UDP header fields, a fixed bit pattern or a pattern to be matched in a cooked stream. Also a set of sampling selectors are available, i.e. periodic, probabilistic or based on the hashing of packets header fields [12]. Flow selections can be concatenated to refine the selection. The resulting selected packets are sent to the following steps in the chain.
- Flow analysis: get analysis results from a network flow. This includes byte and packet counters and bandwidth and packets per second analysis.

- Flow export: export resulting packets or statistics into a flow record conform to the IPFIX standard [11], a tcpdump file or proprietary format.

The first two functions can be used to create the subflow of packets of interest, the last two functions will be applied on this subflow.

4.2 The Module Organizer

The module organizer is a component that translates a monitoring job (created by calling MAPI functions) to a equivalent module representation. A first step in creating a monitoring job is obtaining the required network flow, i.e. all packets are selected that are needed in the monitoring job. Modules such as classifiers, samplers and pattern matchers will be used to create this target network flow. Once the monitor obtains the target network flow, several functions will operate on that subflow. These functions will process the flow in order to obtain the required monitoring information. Functions such as 'return the entire network flow', 'return only packet headers', 'count packets or bytes' and 'measure bandwidth of the flow' will be supported.

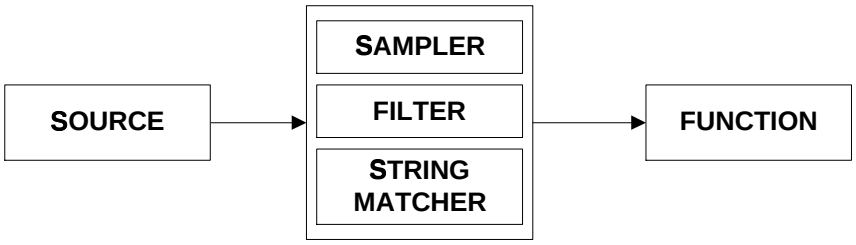

Figure 3: Monitoring Job Representation

Figure 3 depicts the module representation of a generic monitoring job. It consists of different packet processing components that are linked together. If a packet conforms the selection criterion in a certain component, it is sent to the next component in the chain. In order to create such a graph representation of a monitoring job, a graph library is developed. This library is implemented in C++ and supports all functions needed in order to add, delete, replace and find nodes and merge and clone existing graphs.

Because a SCAMPI monitoring agent will process multiple monitoring jobs simultaneously, a global monitoring graph will be created. The global monitoring graph will be a tree with the Raw Network Flow as root and consists of all merged monitoring jobs. This tree has to be optimized in order to support as many monitoring jobs as the monitor (hardware and software) can handle.

Every node in the graph has an associated cost or processing time, which conforms the number of processing cycles needed to process a packet. A patternmatcher for example has a higher cost than a simple counter, a filter will have a higher cost than a periodic sampler. Each packet will traverse the tree top-down. It is possible that a packet will branch in a certain node and travel on multiple paths simultaneously. The ultimate optimization goal is to minimize the total processing time for each packet.

4.2.1 Module Optimizer

A straightforward solution to create the global monitoring graph is to replicate the raw network flow for each monitoring job. In this case each monitoring job will operate on this raw network flow. This approach is very easy to implement, but will definitely not be the most efficient solution. Other techniques and algorithms are studied to optimize the monitoring graph.

4.2.2 Module Mapper

The module mapper will map a monitoring graph to the corresponding hard- and software representation. In the ideal case, the whole monitoring graph will be mapped to the hardware, because this will result in the fastest packet processing. Because hardware resources are limited or not available in the case of off-the-shelf network interfaces, part of the graph will be mapped to middleware, i.e. software. In SCAMPI, the Click modular router is used as middleware [18]. The Click kernel module can be configured by writing the configuration to the "/proc" filesystem. A Click configuration is described in "Click language". Due to the fact that both the monitoring graph and Click configurations are directed graphs of elements, the conversion is very straightforward.

4.3 Job Control

If SCAMPI is to support multiple applications that may be active simultaneously, it needs to provide some form of resource control, both for safety (in terms of not being able to bring the system in a corrupt or inconsistent state), sharing (i.e. ensuring applications only use those resources that are allocated to them) and security (in terms of not being able to touch other applications' sensitive data/packets). We consider monitoring applications to be the clients of the SCAMPI platform, which acts as a server. In this section, we focus on admission control (AC), which deals with questions like:

1. does the client have the required privileges to perform a specific operation at the server?

2. given the total resource capacity and the resources currently in use, can we accommodate the client's request?

The former aspect is handled by explicit trust management and in essence depends only on the credentials presented by a client. As we shall see, the latter is dealt with in a similar fashion, but with the subtle difference that per-request state may be required at the server side. Besides AC, we also need to enforce the resource control.

4.3.1 Trust Management

All authorization in SCAMPI will be based on delegated trust management via explicit credentials as implemented by Keynote [4]. In such a scheme, a server initially contains a policy, stating explicitly which clients are allowed to do what. Whenever a client wants to perform an operation (e.g. `createFlow()`), it has to provide the corresponding credential, proving that it was authorised to perform this operation. In SCAMPI, trust may also be delegated, i.e. an authorising party may delegate part of its privileges to another party (the licensee), with or without extra constraints.

4.3.2 Operation authorisation

For security reasons, each operation in the MAPI should be authorised. In this way, it is possible to restrict the access to resources (including packets) to specific clients only. However, we do not want to send a specific set of credentials for each and every request. We therefore use an explicit authorisation operation which provides the appropriate authorisation settings:

At admission-control time, we may also check whether or not sufficient resources are available in the system to accommodate the new request. For example, it may be specified in a policy that the maximum number of flows that can be created by a client should not exceed N, or the number of functions applied to a flow should not exceed M. This also means that when a request is accepted by admission control, the available resource capacity changes and should be updated. In other words, whichever resources are taken into consideration (simple examples could be number of flows, or data rates in the flows), some accounting of resources is needed, e.g. to keep track of the reserved resources (on a system-wide or per-request basis). This is also needed for the admission control component, which needs to keep state about the currently available resource capacities.

For this purpose, a resource database component is added to the SCAMPI server. The resource database is responsible for keeping track of the resources in the server's domain. Each request that leads to an explicit resource reservation (e.g. loading code that instantiates a queue of specific size) may be entered in the database, together with the resources reserved/allocated on its behalf and dependencies between requests. All of this is transparent to the clients. It is only relevant to the admission control procedure.

4.3.3 Custom Code Loading

One of the goals of SCAMPI is to allow new functionality to be loaded below the kernel-userspace boundaries by clients with the appropriate privileges. There may be two ways in which this is done. If security is not an issue, clients should be able to insert new modules that automatically link with the predefined functions. For example, in a Click-like software model, it may be possibly to link a new component in between two existing elements. In case security and strict resource control is needed, the components that will be linked will be components in the Open Kernel Environment (OKE) [5]. The OKE allows third parties to load and execute fully optimised native code in environments without explicit support for code isolation (such as the kernel of an operating system, or microengines on a network processor) without jeopardising safety. The resources that can be accessed by such modules is restricted by the OKE. A version of the OKE for the Linux kernel is available under GPL. In SCAMPI we have implemented an interface between the OKE and a Click configuration, that allows one to extend a Click configuration with OKE modules at runtime. In fact, using an implementation that combines OKE with a Click-like software model, known as Corral, we are able to maintain a consistent software model throughout the middleware [6].

We distinguish between code loading and code application. When loading a new function, we only add (and link) some object code that was previously unknown to the system, without actually executing it (beyond what is needed for registration). Both operations need to be appropriately authorised.

The load operation takes two parameters that define what sort of code this is and where it is supposed to be running: (1) the type of code to be loaded, which can be OKE code or plain code, and (2) the place where this code will be run (e.g. in the kernel on an x86 host processor, or in the kernel of the StrongArm control processor on an IXP). Whether or not one is permitted to load such code at this location is a separate issue and handled by the credential scheme. Whether or not the appropriate execution environment exists will be kept in the resource database.

5 Conclusion

In this paper we have addressed the problems that arise when using today's monitoring tools in future networks. The EU IST SCAMPI project boldly proposes a solution for these emerging problems by designing and implementing a flexible, scalable and programmable monitoring architecture. We defined and partially implemented an expressive Monitoring API (MAPI), which scales to multi-gigabit networks by using configurable hardware and parallelism. Moreover, the MAPI is backward compatible with current libpcap-based monitoring applications.

In the remainder of the project life-time, multiple monitoring applications, such as pure packet capture, traffic summary reporting, threshold alerting, QoS monitoring and security related applications will be developed on top of the MAPI. These portable applications will illustrate the ease-of-use of the SCAMPI architecture, while keeping up with high-speed networks. We will evaluate the architecture by conducting isolated and multi-domain experiments. These experiments will validate and provide feedback to the architecture design and demonstrate the scalability of the SCAMPI platform.

Acknowledgement

This work was supported by the IST project SCAMPI (IST-2001-32404) funded by the European Union. Work of the fifth author is also supported in part by the Fund Of Scientific Research - Flanders (F.W.O.-V., Belgium).

References

[1] S. Arramreddy and D. Riley, "PCI-X 2.0 White Paper", April 5 2002.

[2] R. Bace and P. Mell, "Intrusion Detection Systems", National Institute of Standards and Technology (NIST), Special Publication 800-31, 2001.

[3] J. S. Bayne, "Unleashing the power of networks", http://www.johnsoncontrols.com/ Metasys/articles/article7.htm.

[4] M. Blaze, J. Feigenbaum, J. Ioannidis and A. D. Keromytis, "The KeyNote trust-management system version 2", Network Working Group, RFC2704, September 1999.

[5] H. Bos and B. Samwel, "Safe kernel programming in the OKE", In Proceedings of OPENARCH'02, New York, USA, June 2002.

[6] H. Bos and B. Samwel, “The OKE Corral: Code Organisation and Reconfiguration at Runtime using Active Linking”, Proceedings of IWAN’2002, Zuerich, Switzerland, December 2002.

[7] K. claffy, J. Apisdorf, K. Thompson and R. Wilder, “Oc3mon: fexible, affordable, high performance statistics collection”, 1996. http://www.nlanr.net/NA/Oc3mon/.

[8] CoralReef, http://www.caida.org/tools/measurement/coralreef/.

[9] Endace measurement systems, “DAG 4.2GE dual gigabit ethernet network interface card”, 2002.

[10] G. Iannaccone, C. Diot, I. Graham, and N. McKeown, “Monitoring Very High Speed Links”, ACM SIGCOM Internet Measurement Workshop, 2001.

[11] IETF IPFIX Working Group, http://www.ietf.org/html.charters/ipfix-charter.html.

[12] IETF PSAMP Working Group, http://www.ietf.org/html.charters/psamp-charter.html.

[13] G. Insolvibile, “Kernel korner: The linux socket filter: Sniffing bytes over the network”, The Linux Journal, 86, June 2001.

[14] “Intel IXP1200 Network Processor Family - The Foundation of a Total Development Environment for Next-Generation Networks”, Intel Product Overview, 2001.

[15] IST-SCAMPI: A Scaleable Monitoring Platform for the Internet, http://www.ist-scampi.org.

[16] Jacobson V., Leres C. and McCanne S., “pcap manual page”, Lawrence Berkeley National Laboratory, University of California, Berkeley, CA, 2001

[17] Juniper, M5 and M10 Internet Routers Hardware Guide, 2002, http://www.juniper.net.

[18] E. Kohler, R. Morris, B. Chen, J. Jannotti and M. F. Kaashoek, “The Click Modular Router”, IEEE/ACM Transactions on Computer Systems 18(3), pages 263-297, August 2000.

[19] S. McCanne and V. Jacobson, “The BSD packet filter: A new architecture for user-level packet capture”, In USENIX Winter, pages 259-270, 1993.

[20] A. Rubini and J. Corbet, ”Linux Device Drivers, 2nd Edition”, ”Chapter 13: mmap and DMA”, O’Reilly, 2nd Edition , June 2001.

[21] The DAG Project, http://dag.cs.waikato.ac.nz/.