Intrusion Prevention with Active Networks: A Performance Comparison between User and Kernel-Space Implementation

A. Hess, T. Gingold, S. R. Garzon und G. Schäfer

Telecommunication Networks Group, Technische Universität Berlin, Germany
Email: [hess,gingold,rodriguez,schaefer]@tkn.tu-berlin.de

Abstract: Recent experiences with attacks in the Internet and especially the tremendous increase in the propagation speed of self-distributing attacks clearly show that the problem of exploiting vulnerabilities of hosts connected to the Internet can not be countered appropriately with an approach that is only aiming to defend against attacks by fixing security holes when patches become available. In order to overcome this situation, various researchers are working on network based intrusion prevention. One specific approach in this respect aims at deploying programmable networking technology (sometimes also called active networking technology) that allows to dynamically deploy task-specific services on so-called active routers for intrusion prevention purposes.
However, as an intrusion prevention system (IPS) necessarily has a certain impact on the network performance because each packet is analyzed in terms of malicious content before being forwarded, the important criterion of processing efficiency has to be taken into account in the design and implementation of such a system while at the same time also considering further requirements like robustness and security. One particular design question arising in this context is if specific intrusion prevention modules should be executed in user- or in kernel-space. In order to thoroughly discuss this question we realized two prototypes: a user- and a kernel-space implementation. In this paper we discuss both architectures and we present performance results in terms of throughput, delay and loss rate.

1 Introduction

Self-spreading worms like Blaster [CER03] in 2003 or MyDoom [CER04] in 2004 are getting a serious threat for todays communication networks. On the one hand the time intervals between the occurrence of worms like these are getting smaller and on the other hand the attacking speed accelerates. In order to cope with security issues like this one, we propose to realize and deploy intrusion prevention functionalities in active network routers (prevention is the task of keeping the attacker away from private network resources). An underlying active network allows to dynamically distribute new security modules as new attack signatures or algorithms to active routers.

In contrast to a normal network intrusion detection system (NIDS) — as for example Bro [Pax99] — an intrusion prevention system (IPS) has capabilities that go beyond simply monitoring attacks as an IPS can actually block them. An NIDS sniffs the network and evaluates copies of the packets transmitted, whereas in the case of an IPS, all traffic is routed exclusively through the IPS and thus, it has the possibility to drop malicious packets. Hence, an ISP influences the network performance, as each packet is analyzed in terms of malicious content before being routed and this must be considered when designing and realizing an IPS, as applications, users and administrators have varying demands with respect to throughput, latency, loss-rate, etc.

Consequently, the question arises how to implement an IPS in order to meet these requirements. In [HJS03], we introduced and motivated the FIDRAN architecture which is a generic IPS framework - in contrast to a monolithic system - for active networking with modules that can dynamically be added, exchanged or removed . We showed that FIDRAN allows to realize a demand-driven IPS, which means that FIDRAN can be fine-grained adjusted to the security requirements posed by the private network.

Realizing a demand-driven IPS (only to monitor the network traffic for attacks which really could do harm to the private network) is the first step in order to keep the impact on the network performance small. Another specific design decision in this respect is to realize the IPS functionality either in user- or kernel-space. Generally, the following differences between user- and kernel-space implementations exist.

Regarding the fundamental requirement that an IPS must work as efficient as possible in order to keep the impact on the network performance small it has to be noted that during execution each user-space process on most Unix based systems is regularly interrupted in order to allow preemptive multitasking; this is not the case for kernel "processes". Also transferring data from kernel into user-space most often requires a set of copy operations which further reduce execution performance. However, efficiency is not the only aspect to be considered in the design, but appropriate attention needs also to be paid to security and robustness. With regards to both aspects the fact that code which is executed in kernel space cannot be restricted or supervised strongly demands for execution in user space. There are, however, research projects like the open kernel environment (OKE) which try to restrict kernel space modules but they are still evolving and have not yet reached sufficient maturity. Processes that run in user-space, on the other hand, can be supervised and restricted (e.g. sand-boxing, access control, etc.), but of course this supervision results in performance degradation. In the context of FIDRAN, the security issue arising of modules running in kernel space is addressed by requiring that only modules that originate from trusted sources and have been checked for integrity are accepted for execution. Finally, regarding robustness, the effects of a system-fault in kernel space are much more serious than in user-space, a fact that also strongly votes for user space implementation of intrusion prevention modules.

In order to thoroughly discuss the question of execution performance, we made a comparison between FIDRAN [HJS03] and MIPSA. FIDRAN is a flexible intrusion detection and response framework for active networks which analyzes packets in kernel-space. Although it is possible to install user-space modules, a part of the packet processing always takes place in the kernel. Thus, we realized a second prototype of an IPS, a modular intrusion prevention system based on AMnet (MIPSA), which runs completely in user-space and which is designed to cooperate with the active networking environment AMnet [FHSZ02].

The paper is structured as follows. The next section shortly introduces the two intrusion prevention systems that were used for the performance comparison. Subsequently, we present the testbed configuration, the conducted experiments and finally, we discuss the achieved results.

2 The Architectures

In this section, we give a short introduction to both architectures we used for the performance comparison. Both systems were designed and realized for a Linux-based operating system and both systems use the netfilter architecture [net], which is part of Linux ($\geq 2.4.xx$). The netfilter framework provides a set of hooks within the Linux kernel, that allows to do packet filtering, network address translation (NA[P]T) and other packet mangling operations. It also allows kernel modules to register callback functions within the network stack. Accordingly, the registered callback function is invoked for every packet that traverses the respective hook within the network stack.

Another design requirement that is fulfilled by both systems is a modular architecture in order to allow the distribution of security operations among several active routers, which is the main particularity of our approach. Snort-Inline [BFP$^+$03] for example, which is a modified version of the intrusion detection system Snort, is a monolithic system.

2.1 MIPSA

MIPSA is a modular intrusion prevention system for the active networking environment AMnet. A programmable node that is running the AMnet software can execute services which are loadable on demand from a service module repository and enhance the functionality of intermediate systems in a flexible manner. A service is composed of one or more service modules which are linked to a process chain. This chain is connected to an Execution Environment (EE) which hands packets to the first module in the chain and picks up the packet from the last.

The basic functionality of MIPSA is realized in the IP-checker, the interpreter and the send module (see figure 1). The other integrated modules are loadable on demand (realized as active networking services) and they form a tree with the IP-checker as root node. The underlying active networking environment passes all packets to the IP-checker, which does some preprocessing on each packet (setting of variables, pointers, etc.) and forwards the packet to the next registered module (TCP, UDP, etc.). Moreover, each running module has the capability to log, forward or drop a packet (also combinations). The last module of the processing flow is the send module, which is the outgoing interface between MIPSA and the underlying active networking infrastructure. Finally, the interpreter introduces a simple description language into MIPSA which is used to specify attack signatures. Each module contains a packet switching part, specifying the successor module for specific packets (e.g. TCP and Port 80 $\rightarrow$ HTTP-module).

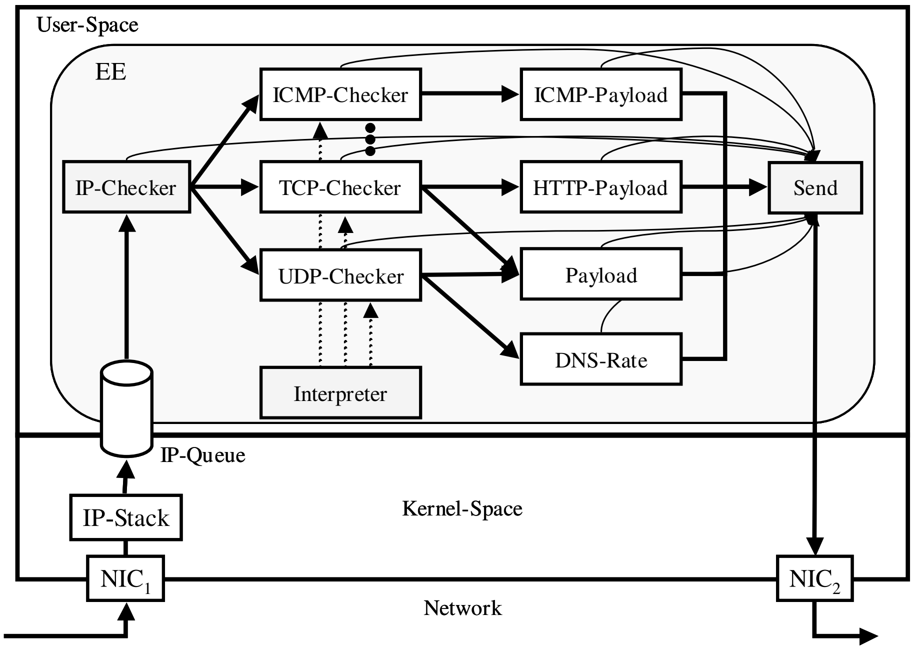


Figure 1: The MIPSA Architecture

2.2 FIDRAN

FIDRAN utilizes the mechanism of loadable kernel modules that is provided by the Linux OS. The architecture is depicted in figure 2. The system is composed of the following units:

- configuration and managements components:
  - management module,
  - control module,
  - traffic selector,
  - security policy and
  - FIDRAN queue.
- intrusion prevention components:
  - varying set of loadable on demand op-modules (realized as active networking services).

Configuration and management components are in charge of administrative tasks, whereas the op-modules contain the intrusion prevention intelligence which in fact realizes the protection against attacks. As most attacks base on a vulnerability that is specific for an operating system (OS), protocol or application, a modularization of the protection services is possible. Now, in order to realize a demand-driven intrusion prevention system, the required op-modules for a specific set of targets are concatenated and stored in an op-module process chain. An op-module can be part of one or more process chains. It is the task of the control module to pass a packet to the correct op-module process chain. The last module of a process chain passes the packet back to the control module, which in turn forwards the packet to the kernel packet sending routine. A detailed description of FIDRAN is given in [HJS03].

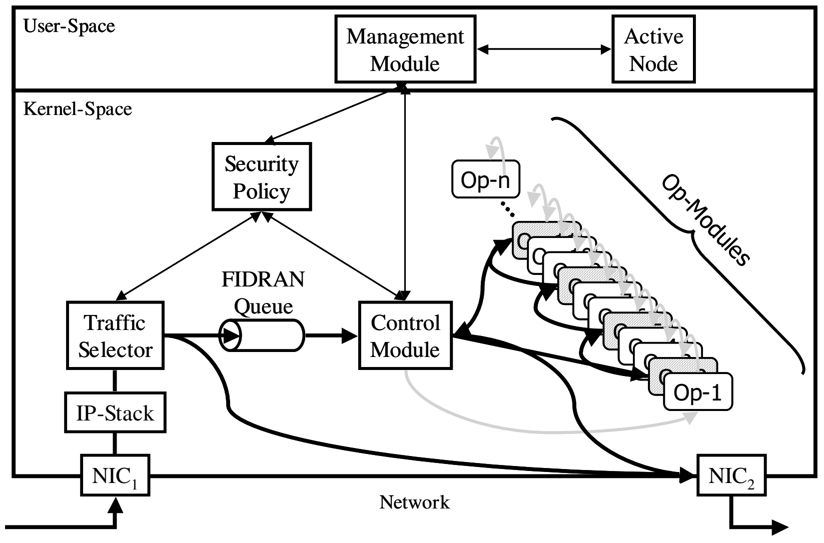


Figure 2: The FIDRAN Architecture

3 Implementation and Measurements

In this section, we start with a short description of a packet's processing path in both systems. Afterwards, we explain the testbed configuration and finally, we discuss the measured results.

A packet that is received by a network interface card (NIC), triggers the execution of a hardware interrupt (disregarding NAPI and interrupt mitigation for the moment). According to the event of a hardware interrupt a corresponding interrupt handler routine is executed which transfers the packet from NIC to kernel-space. There, the kernel verifies the IP-header checksum and passes the packet — assuming that the IP-header checksum was correct — to the netfilter architecture Up to this point both systems process the packets in the same manner.

The MIPSA system uses the *ip_queue* module of the netfilter framework in order to forward the packets to user-space. The netfilter framework copies the packets and inserts them into the user-space queue for the IP-checker module (see figure 1). The IP-checker analyzes the packets and forwards them to the appropriate successor module. Finally, each packet reaches (if not dropped) the send module, which passes back the packets to the netfilter architecture in kernel-space. As MIPSA is realized as a regular user-space process, it is intercepted by the scheduler (regularly) and by events that are of higher priority (irregularly).

In case of the FIDRAN system, it is the traffic selector which is the interface between FIDRAN system and IP stack. It is a kernel module that is registered to the *PREROUTING* netfilter hook. Each packet is analyzed by the traffic selector whether the packet must be analyzed or not (specification via IP address in the security policy). If so, it is inserted into the FIDRAN queue (no copy operation required). The control module takes the packets from the FIDRAN queue and passes them to the correct op-module chain. Each op-module performs its own analyzing functions and the last module of a chain passes the packets back to the control module, which in turn forwards them to the kernel sending routine. The intrusion prevention part of the FIDRAN system is running as a kernel thread, which means that it has all I/O privileges and cannot be pre-empted by the scheduler.

Regarding both systems the behavior on a per packet basis of both systems is defined through the following factors:

1. traffic load,
2. depth of module chain and
3. complexity of integrated algorithms.

In order to compare both systems we used three Pentium III 800 machines running Linux 2.4.24 and being connected via 100 MBit/s ethernet links. The middle host was running the intrusion prevention system (either MIPSA or FIDRAN) and thus, it was equipped with two network interface cards. In addition, we used the *Real-time UDP Data Emitter (RUDE)* [LSP] and the *Collector for RUDE (CRUDE)* for traffic generation and evaluation.

3.1 Basic Performance Degradation

In a first experiment we measured the influence of both systems on the network performance without doing intrusion prevention — meaning both systems were tested without integrated modules. Figure 3 shows that for loads higher than $50MBit/s$ a noticeable difference between MIPSA and FIDRAN can be remarked. The decrease of the delay curve for higher loads is due to the fact that MIPSA looses packets when being operated in this range.

3.2 Performance Comparison on the Basis of Signatures

In a further experiment both systems were tested with the same set of signatures and algorithms. In a first run a varying number (0-200) of TCP-Null scan signatures were used. Each signature checks for every packet if it is a TCP segment and, if so, whether at least one of the TCP-flags SYN, ACK, FIN, RST, URG, or PSH is set. Beyond this, in a second experiment a varying number (0-200) of payload signatures were used. Each packet is analyzed whether it contains a specific sub-string or not. Both systems were using the Boyer-Moore algorithm.

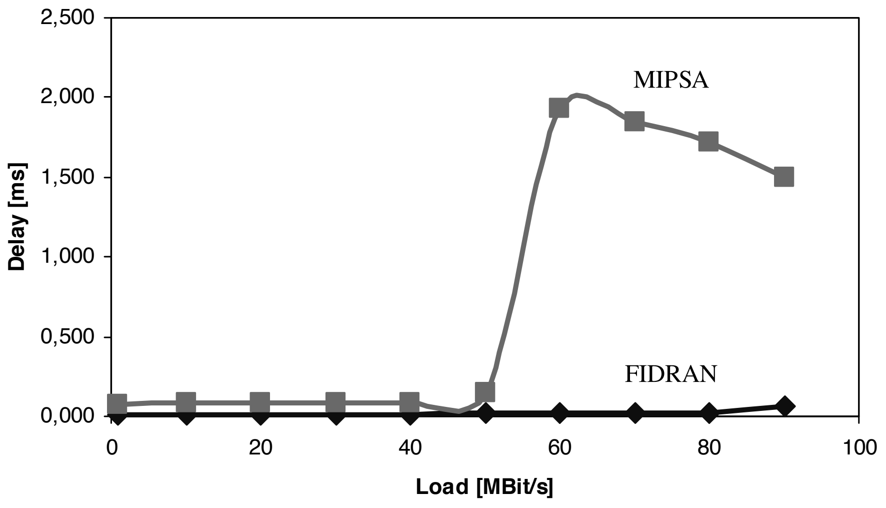


Figure 3: Basic Packet Delay

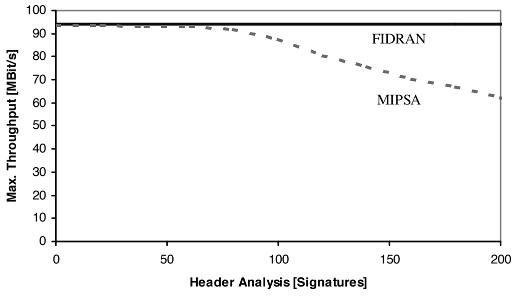
The results are depicted in figure 4. The left picture shows the results for the header analysis and the right one those for the payload analysis. Apparently, FIDRAN, which processes the packets in kernel-space, attains in both experiments the higher throughput. Regarding the results for the inspection of the packet headers, no obvious decline of the throughput was remarked, whereas this is the case for the payload analysis.

In contrast to FIDRAN, MIPSA decrements the network throughput while analyzing packet headers. However, the performance difference between both systems gets smallers when payload analysis has to be performed. Figure 4(b) depicts, that with an increasing number of substring searches that have to be performed, the relative difference between MIPSA and FIDRAN decreases.

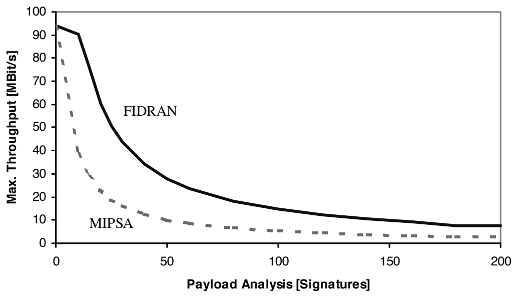
3.3 Performance Comparison on the Basis of Modules of Constant Delay

In order to compare both system independent of signatures or algorithms used, we realized a so-called *sleep* module for both systems. A sleep module introduces an exact delay into the system. In our configuration, it delays each packet for $0.1ms$ before passing it to the next module. In a first run both systems were compared with only one sleep module integrated. The results are depicted in figures 5(a), 5(b) and 5(c).

Figure 5(a) depicts the maximum throughput over the offered load for one integrated sleep module. Up to an offered load of about $40MBit/s$ both systems show the same comportment. But for higher loads FIDRAN reaches a maximum throughput of about $70MBit/s$ whereas the maximum value for MIPSA is about $35MBit/s$.



(a) Header Analysis



(b) Payload Analysis

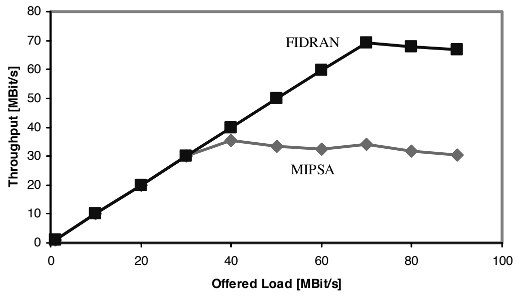
Figure 4: Performance Comparison based on Signatures

In a next step we analyzed the delay that is caused by both systems. Figure 5(b) depicts the delay that is introduced by either MIPSA or FIDRAN over the load. The basic delay for a load of $1MBit/s$ and one integrated sleep module is:

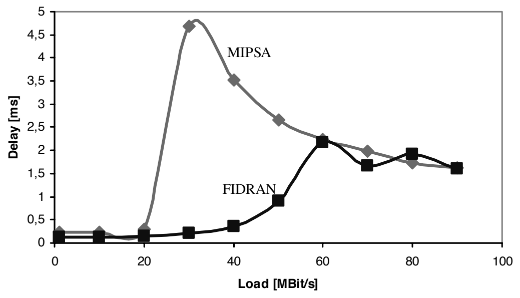
- $0.230ms$ for MIPSA and
- $0.167ms$ for FIDRAN.

Furthermore, it can be seen that for a load up to $60MBit/s$ the FIDRAN system obviously causes a smaller latency than MIPSA. For load rates higher than $20MBit/s$ the latency that is caused by MIPSA grows in a non-linear manner. The MIPSA graph has a peak at a load of about $20MBit/s$ and for higher loads it approaches the FIDRAN curve. This comportment is caused by packet loss as depicted in figure 5(c).

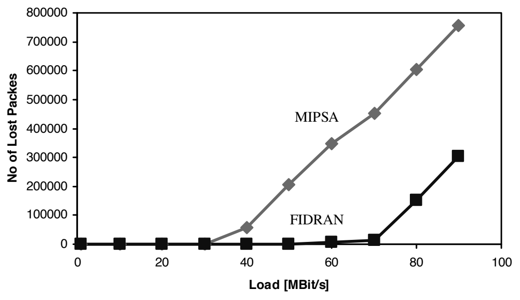
Figure 5(c) depicts the packet loss over the load for both systems. Up to a load of about $30MBit/s$ the packet loss stays within a negligible range, but then in case of MIPSA the packet loss increases strongly in a linear manner. It must be said, that it is the netfilter



(a) Throughput



(b) Delay



(c) Packet Loss

Figure 5: Comparison 1 Sleep Module: MIPSA ↔ FIDRAN

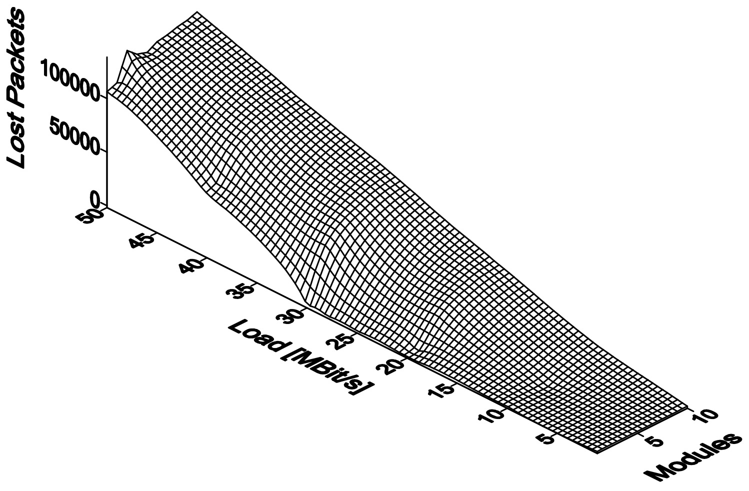


Figure 6: MIPSA Packet Loss

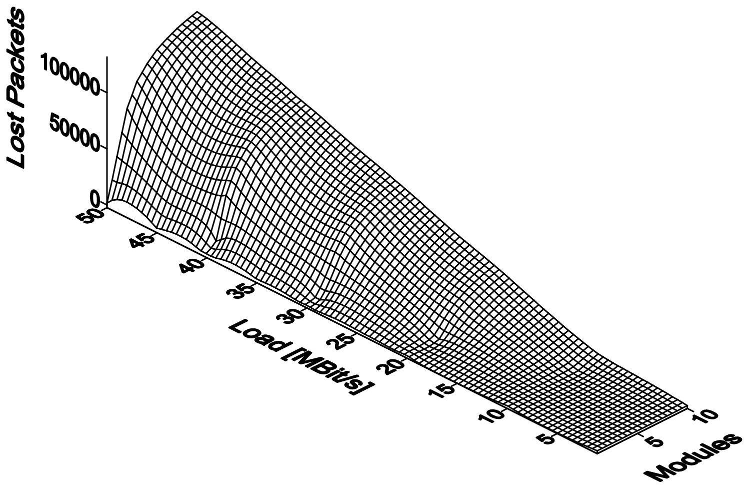


Figure 7: FIDRAN Packet Loss

architecture respectively the integrated user-space queuing mechanism that is dropping the packets. MIPSA still processes any packet that is forwarded by the *ip_queue*. Regarding FIDRAN, the point where the packet loss starts to increase significantly in a linear manner is at a load of about $70 MBit/s$.

Finally, figures 6 and 7 depict the packet loss for MIPSA and FIDRAN over load and a varying number of sleep modules (1-10). It can be seen that the packet loss for the MIPSA system is throughout higher than for the FIDRAN system. However, in case of FIDRAN the integration of further sleep modules is noticeable whereas this is not necessarily the case for MIPSA.

4 Conclusions

MIPSA and FIDRAN are modular intrusion prevention systems which can dynamically be reconfigured, i. e. modules can be inserted or removed. MIPSA is designed as a user-space process and works in combination with the active networking environment AMnet, whereas FIDRAN runs in kernel space. In this paper we compared the performance of the two in terms of throughput, delay and packet loss.

Recapitulating the results of section 3, for one integrated sleep module the two systems show a similar comportment for load rates up to $20MBit/s$. For higher loads MIPSA gets lossy (see figure 5(c)), whereas FIDRAN remains loss-free. Furthermore, FIDRAN reaches twice the maximal throughput of MIPSA (figure 5(a)).

FIDRAN performs conspicuously better than MIPSA in terms of loss rates over sleep modules and load (figures 6 and 7). However, for extreme configurations (high load and many integrated sleep modules) the packet loss of FIDRAN approximates that of MIPSA. Summarizing, the main difference between MIPSA and FIDRAN is that the former requires packet copying for user-space queuing and is steadily interrupted by the Linux scheduler. The results show that these costs are noticeable.

References

[BFP$^+$03] Jay Beale, James C. Foster, Jeffrey Posluns, Ryan Russell, and Brian Caswell. *Snort 2.0 Intrusion Detecion.* Syngress, 2003.

[CER03] CERT. W32/Blaster worm. http://www.cert.org/advisories/CA-2003-20.html, August 2003.

[CER04] CERT. W32/Novarg.A / W32/Shimg / W32/Mydoom. http://www.cert.org/incident_notes/IN-2004-01.html, January 2004.

[FHSZ02] T. Fuhrmann, T. Harbaum, M. Schöller, and M. Zitterbart. AMnet 2.0: An Improved Architecture for Programmable Networks. In *Proceedings of the 4th Annual International Working Conference on Active Networks (IWAN)*, 2002.

[HJS03] A. Hess, M. Jung, and G. Schäfer. FIDRAN: A flexible Intrusion Detection and Response Framework for Active Networks. In *8th IEEE Symposium on Computers and Communications (ISCC'2003)*, Kemer,Antalya,Turkey, July 2003.

[LSP] J. Laine, S. Saaristo, and R. Prior. RUDE & CRUDE. http://rude.sourceforge.net/.

[net] The netfilter/iptables project. http://www.netfilter.org.

[Pax99] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23–24):2435–2463, 1999.