WSIM: A software platform to simulate all-optical security operations

Antonis Krithinakis*, Lubomir Stroetmann*†,
Elias Athanasopoulos*, Georgios Kopidakis*, and Evangelos P. Markatos*,
*Institute of Computer Science, FORTH
Greece
Email: {krithin, lubomir, elathan, kopidaki, markatos}@ics.forth.gr
†University of Aachen
Germany

***Abstract*—Network throughput rates increase every day in contrast to electronic chip processing speed and electronic I/O. Today's firewalls operate by using traditional electronic circuits just like any common PC. However, performing these operations in a fast fiber optics network on the scale of 40Gbps is impossible. In this paper, we propose a novel system that is currently being researched and tries to perform the security operations of a firewall using optical components. We describe the basic limitations of the optical domain that make this project difficult to implement. We outline the basic software platform called WSIM which is a simulator that offers theoretical support of the project's feasibility. The marriage of an all-optical firewall with the traditional digital systems' architecture can offer significant benefits to the network from both a security and a performance perspective.**

I. Introduction

This novel system is called **WISDOM**[6] (***Wi****respeed* ***S****ecurity* ***D****omains using* ***O****ptical* ***M****onitoring*) and is designed to develop optical processing modules which will be placed at the front end of a node firewall to provide high-speed information filtering purely inside the optical domain. These photonic firewalls will operate using novel algorithms and protocols, to extract and process security information at wire speed.

The main challenge of this project is to demonstrate that it is possible to bridge the gap between the very fast (at least 40 Gbps) optical data transmission and the much slower electronic data security checks by pushing critical security functions all the way down to the optical domain.

The paper is structured in the following order. We present the background in Section II. In Section III we present in detail a software modelling platform, its architecture and some application scenarios that can be modelled. We discuss the software API, which we refer to as SAPI (Security Application Programming Interface), which is the mini operating system for the initial hardware prototype. SAPI is described in Section V. Finally we conclude in Section VI.

II. Background

In this section we discuss the challenges which appear in the optical domain and present information needed as a background to understand the algorithm used by the simulator platform.

A. Optical Algorithms

Operations such as pattern-matching are considered fundamental for security-oriented applications like firewalls and Intrusion Detection Systems. Thus, it is vital for the community to investigate new ways of performing these tasks for high speed links like the ones provided by optical systems.

The simulator software is made to simulate an all-optical pattern recognition system. The components of this system are able to detect and indicate the position of a specified pattern of bits in a high-speed optical data signal. The target pattern only needs to be generated at a much lower speed, readily achievable with standard electronics. This pattern matching component is what we call a pattern matching box.

Every box in the system runs the pattern matching algorithm whose complexity is proportional to the length of the pattern. The algorithm is implemented using optical components only which we are not going to discuss in this paper but are explained in detail in [10].

B. Optical Limitations

One of the fundamental constraints of the optical domain is the absence of state[8]. This is due to the

fact that building a memory storage device in the optical domain is not as trivial as it is in the electronic domain. More precisely, the only notion of *memory* in the optical domain is to use serial, time-of-flight storage which delays the light pulses inside a loop for a relatively short period of time.

Optical memory capacity is thus severely limited compared to electronic memory, with integrated optical devices being able to store a few hundreds of bits of information (albeit at extremely high data rates). The difficulty in having a flexible storage medium in the optical domain results in a system that almost completely lacks the notion of state.

Almost all systems and algorithms developed in the electronic domain reside to a great extent in having a controllable state. Therefore, developing security algorithms in the optical domain requires a new method of thinking and learning time in order to understand how these algorithms can be implemented.

The lack of memory in the optical domain also results in very limited flexibility in the configuration of a device and the dynamic re-adjustment of its setup. Since there is no memory - in the sense of the term used in the electronic domain - to hold a system's state, it is hard to modify a system's behavior by altering its properties at run-time. As far as the implementation of a security device in the optical domain is concerned, the limited configurability reduces the flexibility in implementing even trivial features, enabled by default in all similar devices of the electronic domain, such as adding, deleting or modifying the *rule-set* of the device.

Aside from the lack of memory in the optical domain, it is also quite difficult to perform logic operations, which again is trivial in the electronic domain. The WISDOM hardware architecture uses Semiconductor Optical Amplifiers (SOAs)[9] in order to implement simple operations like AND, XOR and other gates expressing Boolean logic.

However, they are not as functional as digital gates, and they can be used only in certain combinations. Using SOAs we have managed to implement the pattern-matching algorithm we described in Section II-A.

III. Software Modeling

In this section we present the software platform we have developed for modelling the optical firewall. It is a custom framework developed specifically for the needs of our architecture. It runs on any PC equipped with a network interface and a Microsoft Windows operating system. The framework does not aim to model the optical core of the system; there are several of options currently on the market for simulating optical devices[7]. Instead, our initial goal was to produce a software environment that illustrates the operation of our architecture as a whole system.

By using the framework, we are able to reproduce application scenarios in a controlled software environment. We can estimate time costs and verify the pattern matching optical algorithm.

This section is organized as follows: First, we present the framework's features through a quick guide using limited screen shots of the software environment, due to space constraints. We proceed to explain the system architecture and then depict some possible usage configurations and application scenarios.

A. Overview

One of our first major concerns during the development of the framework was that it had to target a mixed scientific audience and not just computer scientists. This is, mainly, because our collaborative project is composed of physicists, optical engineers, electrical engineers and computer scientists. Thus, we took the decision to develop a graphical and user friendly application for the Microsoft Windows operating system. The framework can be used by anyone without requiring deep technical knowledge of computer software.

With the assistance of the graphical environment the user is able to construct instances of all-optical pattern matching boxes. We will further refer to such an instance as a filter or rule.

Each filter is configured using a dialog box. First you give the box a unique name. Then you select the target properties (protocol and service) that the box will search for (e.g. ICMP[4], TCP[5], UDP[2], PORT number of required service) and then the source of the packets.

In the beginning, the only source available is the trace file captured from the network, but after the creation of each new box you can additionally select as source the matched or unmatched packets of any of the other boxes, thus creating processing chains. Optionally, you can put an extra time delay for a box. In addition, if you choose the PORT target you will be asked to select between source port and destination port and put in the port number as shown in Figure 1.

After a new box has been set up, an iconic representation of an all-optical pattern matching box is displayed on the program's workspace. The user may construct an unlimited amount of filters in this fashion. However, the actual system's capacity is limited. The real optical

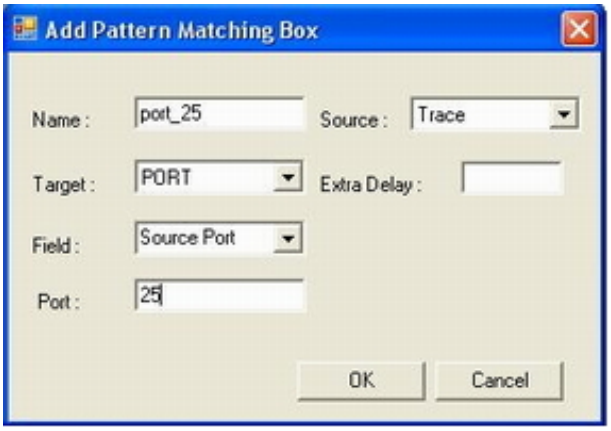

Fig. 1. Search for a specific port number

firewall is expected to be able to serve simultaneously only a few filters.

Upon having set the preferred filters, the framework can inject traffic into the simulated system. This can be achieved either by passively capturing the traffic experienced by the host, which runs the framework, or by processing already captured traces. Each filter, depicted in the user's desktop as an optical firewall instance, updates its statistics in real-time. Statistics include packets captured, packets which matched the filter and packets that did not.

In addition, a global tick counter accounts for the cost of the whole session in terms of spent processing time. We define a tick as the amount of time spent for processing one bit of information in the optical domain. For every tick, each pattern matching box performs the pattern matching algorithm on one bit of the data. From an algorithmic view we can imagine this operation as an iteration by the main processing loop of the system where each box operates separately from the others in a pseudo-parallel way.

However, the tick described above corresponds to real time, which in the optical domain is on the scale of tens of picoseconds. For example, if we operate in 40Gbps then the one bit time is: $\frac{1}{40*10^9} = 25$ picoseconds.

This correlation of the one bit operation and the physical time is a configurable feature of our application so the user can change the virtual network throughput.

IV. System Architecture

The Simulator process has two threads. The first one is the graphical interface of the application and the second one is the main processing unit that consists of the sniffer and the pattern matching box component.

After the user configuration is made and the simulation is started, the Sniffer thread is created. The Sniffer is initialized with a trace file from the user. Then the file is processed and internal structures that represent the packet header are created. Each packet from the trace is processed and translated to a sequence of bits representing its header. At present, the system architecture deals with the IP protocol[3] for the packets' network layer and the TCP[5] and UDP[2] protocols for the packets' transport layer. Packets in the trace file using other protocols are excluded.

When all internal structures have been created the global timer described above begins and starts counting the time spent on all operations until there are no unprocessed packets remaining in the stack.

Each packet is processed using the complete optical algorithm for pattern matching and not using simple equality relations of computer programming.

Every box keeps basic information like its name, the target and several counters that build the state of the box and tree lists. The first list contains the source of the packets to capture. The other two lists contain the IDs of boxes whose source list has to be refreshed with the packet being currently processed after the box has finished with it. There is one list for matched packets and one list for unmatched packets for every pattern matching box.

After a box has captured a packet and finished processing it and there is a match, then the packet feeds the source list of the boxes that the matched list refers to. The same happens with packets that do not match the target. The source of the boxes can be fed with packets from a trace depending on the user's configuration.

If the box is still in operation and its source list is fed with a new packet, the packet gets dropped. The source list must be empty when the pattern matching box operates on a packet otherwise we have a packet drop. If a box has finished with a packet and its source list is empty, the box will be in an idle state until its source list is refreshed.

The simulation ends when all packets from the trace have been processed through all the pattern matching boxes. Then the user can examine the log files generated which contain all the information on the packets of the trace including timestamps. We depict the system's architecture in Figure 2.

A. Application Scenarios

We use a small trace containing 131 packets for the examples. The scenarios represent an optical firewall that

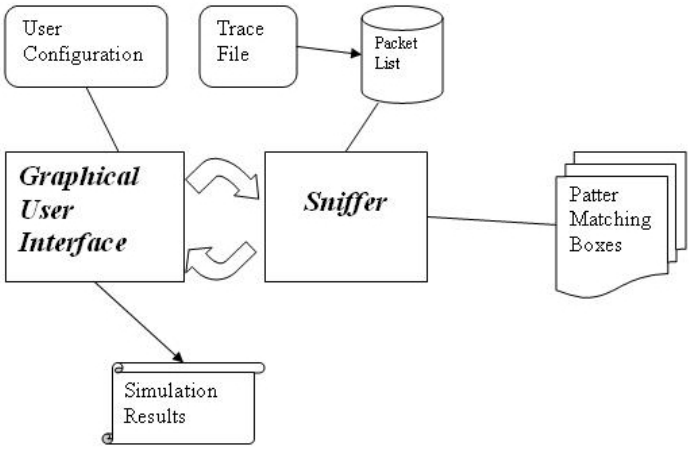

Fig. 2. WSIM Architecture

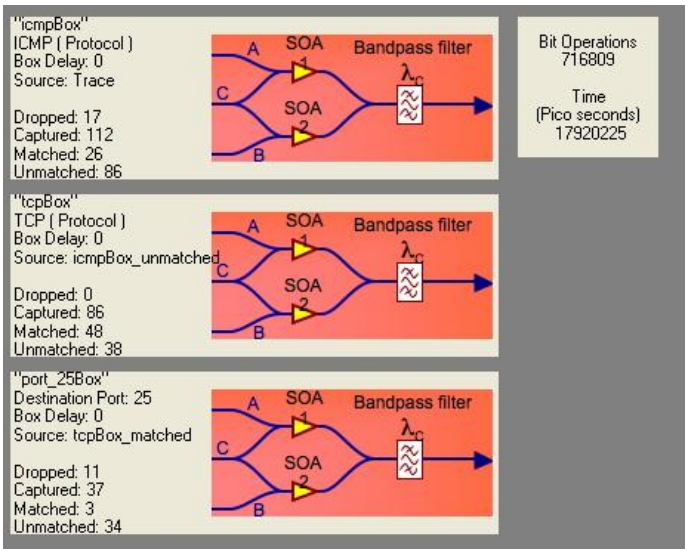

Fig. 3. First scenario interface layout

drops all incoming ICMP packets before they manage to reach any router operating in the digital domain. After that we need to separate the traffic and search for specific source or destination ports that may corresponds to well-known attacks.

1) First Scenario: We are going to set up 3 pattern matching boxes. The first one is called icmpBox and searches for ICMP protocol packets. The source of the packets is the trace file.

Secondly, we want to filter out the unmatched packets produced from the first box and keep only the TCP protocol. So the second box is called tcpBox and the source of the packets is the icmpBox_unmatched field. All matched packets from the tcpBox will go through the last pattern matching box that is called port_25Box and is searching for destination port 25.

Then we start the simulation and see the results in Figure 3 and Table I below.

Our trace contains 131 packets, but as we can see from the table of results in Table I below, the first box has dropped 17 packets because of time issues and has captured 112 packets, a total of 129. That means that 2 packets are not of IP protocol and were not captured by the filter. 26 packets matched the icmpBox target and got dropped because we assumed that the optical firewall drops all incoming ICMP packets.

The rest of the packets went through the second rule without any dropping and then the matched packets of this rule through the last pattern matching box which dropped 11 packets. That can be easily explained because the first 2 rules need the same time to complete processing a packet according to the complexity of the pattern matching algorithm. This is because the target of the first two rules is 8-bit long (protocol field in packet's IP header) and the target of the third rule is 16-bit (destination port field in packet's transport layer header), so the algorithm in the third box needs twice the time to complete processing each packet.

	Target	**Packet Source**	**Packets Dropped**
icmpBox	"ICMP"	"Trace"	17
tcpBox	"TCP"	"icmpBox_unmatched"	0
port_25Box	"PORT"	"tcpBox_matched"	11

	Packets Captured	**Packets Matched**	**Packets Unmatched**
icmpBox	112	26	86
tcpBox	86	48	38
port_25Box	37	3	34

Network throughput	40 Gbits per second
Bit Operations	716,809
Time (picoseconds)	17,920,225

TABLE I
FIRST SCENARIO RESULTS OPERATING IN 40GBPS

Now we are going to simulate the same scenario but change the network throughput to 80Gbps so that each bit operation lasts 12.5 picoseconds. The number of dropped packets increases and the time for completing the total operation is approximately 25% of the initial one. Results are shown in Table II.

2) Second scenario: Now we are going to set up 5 pattern matching boxes. The network throughput is set to

	Target	Packet Source	Packets Dropped
icmpBox	”ICMP”	”Trace”	22
tcpBox	”TCP”	”icmpBox_unmatched”	0
port_25Box	”PORT”	”tcpBox_matched”	14

	Packets Captured	Packets Matched	Packets Unmatched
icmpBox	107	21	86
tcpBox	86	48	38
port_25Box	34	3	31

Network throughput	80 Gbits per second
Bit Operations	358,405
Time (picoseconds)	4,480,062.5

TABLE II
FIRST SCENARIO RESULTS OPERATING IN 80GBPS

40Gbps. The first pattern matching box is called tcpBox and searches for TCP protocol packets. The second one is called udpBox and searches for UDP protocol packets. The source of the packets of both boxes is the trace.

	Target	Packet Source	Packets Dropped
tcpBox	”TCP”	”Trace”	17
udpBox	”UDP”	”Trace”	17
port_25Box	”PORT”	”tcpBox_matched”	11
port_80Box	”PORT”	”tcpBox_matched”	11
port_53Box	”PORT”	”udpBox_matched”	0

	Packets Captured	Packets Matched	Packets Unmatched
tcpBox	112	48	64
udpBox	112	38	74
port_25Box	37	3	34
port_80Box	37	12	25
port_53Box	38	19	19

Network throughput	40 Gbits per second
Bit Operations	717,001
Time (picoseconds)	17,925,025

TABLE III
SECOND SCENARIO RESULTS OPERATING IN 40GBPS

All matched packets from the tcpBox will go through the next two pattern matching boxes that are called port_25Box and port_80Box and are searching for destination port 25 (default for mail processing) and destination port 80 (default for HTTP traffic) respectively. Then all matched packets from the udpBox will go through the last pattern matching box which is called port_53Box and is searching for destination port 53.

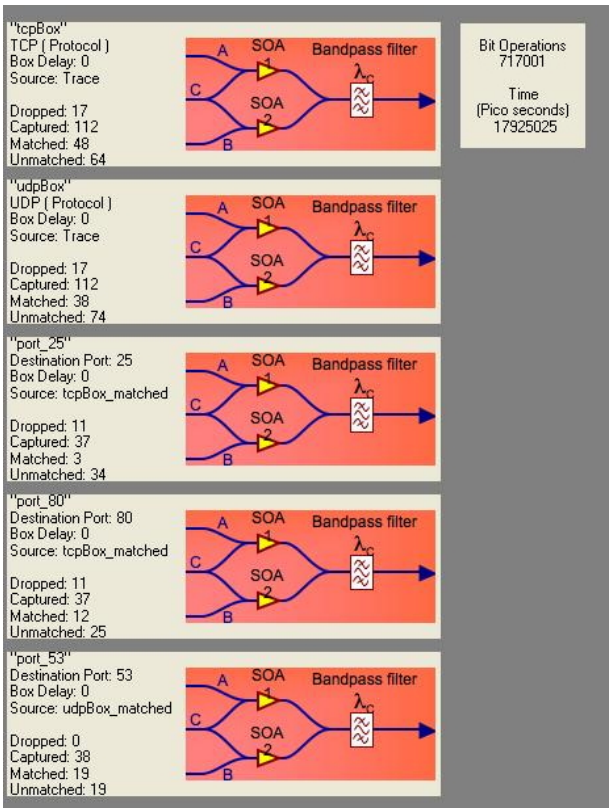

Fig. 4. Second scenario interface layout

Then we start the simulation and see the results in Figure 4 and Table III above.

V. SOFTWARE API

The SAPI (**S**ecurity **A**pplication **P**rogramming **I**nterface) is the mini operating system of the WISDOM firewall and will be used together with the first WISDOM hardware prototype developed by our project partners. It creates an interface between the low-level hardware controls and the user by providing an easy to use API for creating new sets of filter rules. The SAPI uses an object-oriented design and provides several classes: *SAPI_Init*, *SAPI_Filter* and its three subclasses *SAPI_IPv4_Filter*, *SAPI_TCP_Filter* and *SAPI_UDP_Filter*. For a simplified UML diagram of the SAPI, see Figure 5.

SAPI_Init is used to initialize the hardware and performs various low-level tasks such as turning on heater and peltier controls on the circuit board and resetting any previously set filter rule. *SAPI_Filter* provides a structure for a general packet filter. The WISDOM hardware only deals with simple bit patterns on which it performs its

bit-by-bit optical pattern matching. In order to create a filter rule for the firewall, one can use *SAPI_Filter* which then translates a human-readable rule to a bit pattern to be passed on to the hardware. Additionally, one can use the more specific *SAPI_IPv4_Filter*, *SAPI_TCP_Filter* or *SAPI_UDP_Filter* which are aware of the packet header structure of their respective protocol types in order to create more specific filter rules.

For example, in order to create a rule that will filter out any outgoing e-mail traffic, one creates a new instance of *SAPI_TCP_Filter* with `destination_port="25"`. (Port 25 is the one used by the SMTP protocol used to send email). Another very common firewall rule example would be to create an ICMP filter. ICMP is very often abused and by blocking ICMP ping packets one can also hide the status of internal hosts from the outside. To do this using the SAPI, you create a *SAPI_IPv4_Filter* object with `protocol="ICMP"`. The SAPI knows that ICMP has the protocol number `0x01`[1] and knows the offset of the "protocol" field inside an IP packet[3]. It uses this knowledge to create an appropriate binary pattern that it can pass down to the hardware. Once an object with the desired filter rule has been created, all you have to do is call its `apply()` function and the filter rule is converted and sent to the hardware.

Furthermore, the SAPI provides a set of predefined functions for easily creating some of the most commonly used filter rules. Those include the aforementioned Mail filter and ICMP filter and also feature a HTTP web traffic filter, FTP filter, IRC filter, DNS filter, Microsoft File & Printer Sharing filter, a generic TCP filter to be used to pre-filter TCP traffic and two IP address filters for filtering by source or destination IP address, respectively. Not only do these functions simplify the filter creation process, they also act as "wizards" for normal users so that they do not need to know which ports are used by Microsoft's File & Printer Sharing services or what makes up an ICMP packet.

The SAPI will eventually be accompanied by a GUI interface which is currently being developed. It will allow filter rule creation at the click of a button, offer the possibility to change the order of the set of rules before it's written to the consecutive set of hardware devices and will make configuring and using the WISDOM firewall possible even for regular users not versed in the field of network administration.

VI. Conclusion

Currently, important security operations such as pattern matching for filtering packets are difficult to perform in network links experiencing throughput greater than 40 Gbps.

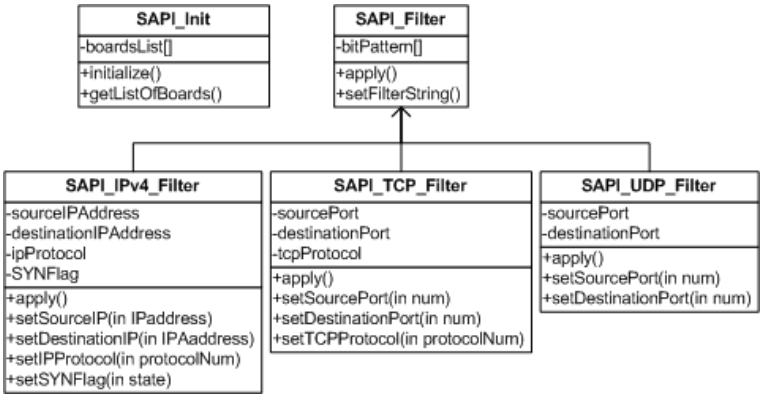

Fig. 5. Simplified UML diagram of the SAPI architecture

In this paper we have presented a novel idea which suggests the creation of a system architecture that deals with security operations in the optical domain. We have also presented the mini operating system of the firewall and the software platform which can be used to build and model scenarios in a controlled environment. A user-friendly simulator of the optical firewall's operation, which uses actual network traffic traces, is useful for security algorithm design purposes as well. It is expected that it will also provide a hardware validation tool later in the project.

The bad news is that, as we have described, the development of such a system has to deal with a lot of optical limitations and thus is a very challenging project. The good news is that every day clever ideas come along and solve fundamental problems. In addition, the current simulation platform results make us very hopeful. WISDOM is the beginning of all later optical based hardware, which can perform sophisticated all-optical processing for security purposes.

VII. Acknowledgements

We would like to thank the anonymous reviewers for their valuable feedback. Antonis Krithinakis, Elias Athanasopoulos, Georgios Kopidakis and Evangelos P. Markatos are also with the University of Crete. Lubomir Stroetmann assisted to this work, while interning with FORTH. This work is funded by the FP6 EU project WISDOM. Elias Athanasopoulos is also funded by the Microsoft Research PhD Scholarship project, provided by Microsoft Research Cambridge.

REFERENCES

[1] Iana assigned internet protocol numbers. http://www.iana.org/assignments/protocol-numbers/.

[2] RFC768 - User Datagram Protocol. http://www.faqs.org/rfcs/rfc768.html.

[3] RFC791 - Internet Protocol. http://www.faqs.org/rfcs/rfc791.html.

[4] RFC792 - Internet Control Message Protocol. http://www.faqs.org/rfcs/rfc792.html.

[5] RFC793 - Transmission Control Protocol. http://www.faqs.org/rfcs/rfc793.html.

[6] The WISDOM Project. http://www.ist-wisdom.org.

[7] VPIphotonics. http://www.vpiphotonics.com/.

[8] C. C. Carroll. R68-40 sequential machines and automata theory. *IEEE Trans. Comput.*, 17(9):922–923, 1968.

[9] D. Cotter et al. Non-linear optics for high-speed digital information processing. In *Science 286, pages 1433-1636)*, 1999.

[10] R. P. Webb et al. 42gbit/s all-optical pattern recognition system. In *Proceedings of Optical Fibre Communications (OFC)*, 2008.