A Parallel Architecture for Stateful Intrusion Detection in High Traffic Networks

Michele Colajanni Mirco Marchetti
Dipartimento di Ingegneria dell'Informazione
University of Modena
{*colajanni, marchetti.mirco*}*@unimore.it*

***Abstract*— In a scenario where network bandwidth and traffic are continuously growing, network appliances that have to monitor and analyze all flowing packets are reaching their limits. These issues are critical especially for *Network Intrusion Detection Systems* (NIDS) that need to trace and reassemble every connection, and to examine every packet flowing on the monitored link(s), to guarantee high security levels. Any NIDS based on a single component cannot scale over certain thresholds, even if it has some parts built in hardware. Hence, parallel architectures appear as the most valuable alternative for the future. In this paper, we propose a parallel NIDS architecture that is able to provide us with fully reliable analysis, high performance and scalability. These properties come together with the low costs and high flexibility that are guaranteed by a total software implementation. The load balancing mechanism of the proposed NIDS distributes the traffic among a configurable number of parallel sensors, so that each of them is reached by a manageable amount of traffic. The parallelism and traffic distribution do not alter the results of the traffic analysis that remains reliable and stateful.**

I. Introduction

Network Intrusion Detection Systems (NIDS) are becoming a valuable element in a modern network infrastructure for guaranteeing the security of complex information systems. A NIDS is used to inspect network traffic with the goal of looking for evidences of illicit activities and malicious network packets. To control all the traffic flowing through a network, a NIDS has to perform a stateful analysis on each packet. This requires a NIDS to track and reassemble each distinct connection. (For example, in a LAN, a NIDS has to get and analyze every Ethernet frame.) The throughput of the monitored traffic and the number of concurrent connections affect the amount of memory and computational power that are required by each NIDS.

Various trends are affecting the capacity of present NIDS appliances and their possibility of being applied to the most modern network infrastructures. The increasing number of connected devices, the augment of link capacities, and the diffusion of network-related applications and services are causing a continuous growth of traffic flowing through wide and even local area networks. As a consequence, traffic generated in large network installations may easily overwhelm the memory and power capacity of a typical NIDS. For example, a NIDS implemented through standard hardware can barely deal with 100 Mbps traffic [1], even through special configurations (e.g., fast logging in binary format) [2].

The most common solutions to achieve high performance NIDS are being directed towards custom hardware components, that are specifically designed for intrusion detection on high speed links. Some valuable results are described in Section II. On the other hand, we think that custom hardware does not represent a long-term solution for scalability issue. These hardware oriented solutions only push further the maximum manageable throughput, but they will be overwhelmed by the traffic volumes of the near future (10 Gbit Ethernet technologies are behind the corner). Moreover, they are characterized by high costs and low flexibility that are typical of hardware-based solutions.

In this paper, we propose an innovative parallel NIDS architecture that achieves high performance by combining conventional sensors in parallel, with no need of ad hoc hardware components. We think that parallel architectures are the most valuable alternative for guaranteeing scalability and large diffusion of NIDS even to face future high capacity networks. We demonstrate that the proposed NIDS can effectively scale and deal with increasing traffic volumes thanks to a fine-grain traffic distribution algorithm and an innovative load balancing technique, that dynamically dispatches incoming traffic to the available sensors. This architecture allows the NIDS to inspect high speed links with no packet loss and no negative impact on the accuracy of the traffic analysis, that remain reliable and stateful.

The rest of this paper is organized as following. Section II compares our proposal against other works in the field of high speed NIDS. Section III analyzes the details of the parallel NIDS architecture. Section IV describes the configuration rules that may be used to guarantee the efficacy of the proposed architecture. Section V presents experimental results achieved by a prototype implementation of the proposed NIDS. Finally, Section VI outlines main conclusions and results of this paper.

II. Related work

The most common solutions to achieve high performance NIDS rely on hardware-based components. For example, *Application Specific Integrated Circuits* (ASIC) appliances can inspect high traffic throughput [3], [4], but they do not represent an exhaustive solution to scalability. Moreover, ASIC appliances are characterized by high costs and low flexibility. Similar problems affect others hardware-based architectures, such as FPGA [5]–[7] and *Network Processors* (NP) [8], [9].

It is important to observe that the *parallel* NIDS architecture proposed in this paper should not be confused with *distributed* NIDS architectures that have been extensively studied in literature [10]–[13], and can be implemented through commercial or open source software [14]–[16]. A distributed NIDS architecture is composed by sensors deployed at different places of a network. Their typical scenario is represented by a large network consisting of small interconnected subnets where a distributed NIDS architecture deploys a sensor in each subnet. These sensors may be connected to a central manager that concentrates and correlates sensors alerts. On the other hand, a parallel NIDS architecture may be thought as a single, logical NIDS sensor, composed by a traffic distribution device that is connected to conventional sensors that operate in parallel, each analyzing only a subset of the traffic on the same link.

To the best of our knowledge, there are not many other parallel NIDS architectures. The first parallel implementation of a NIDS described in [17] is characterized by some drawbacks in the traffic dispatching algorithm, that is able to classify packets just on the IP address basis. This is a main limit in many network configurations, especially if we consider that Network Address Translation (NAT) mechanisms hide entire computer networks behind a single IP address [18]. Hence, even the traffic flowing between two IP addresses may be too much for the capacity of a single NIDS sensor, with the possibility of system bottlenecks and limits to the architecture scalability. On the other hand, the proposed system achieves an effective, fine-grain traffic sharing between its NIDS sensors. Moreover, the traffic dispatching algorithm of the architecture in [17] does not adapt to the current traffic pattern because the traffic distribution depends on a set of static rules that assign different traffic flows to different sensors. The problem is that traffic composition may change rapidly, thus overloading some sensors with significant packet loss and leaving other sensors almost unused. The proposed system addresses even this issue.

The other interesting parallel architecture presented in [19] is characterized by a custom hardware load balancer that feeds conventional NIDS sensors. To ensure that every packet belonging to a certain flow is analyzed by the same sensor, the hardware load balancer calculates a set of hashing functions on different fields of the packet. The destination NIDS sensor is selected on the basis of the resulting hashes. However, although the balancer may dynamically adapt to the traffic dispatching rules, the redirection of already established connections to a different sensor makes it impossible to perform a stateful analysis.

A serialization of the internal NIDS state was proposed in [20]. This is an important contribution because a serialized representation can be propagated to other NIDS sensors to achieve better coordination in distributed NIDS architectures. However, this paper does not consider state serialization as a mean to provide stateful and dynamic load balancing of established connections among different NIDS sensors.

The proposed parallel NIDS architecture differentiates from all the previous proposals. It is a total software based solution where every element is built with standard hardware, which is flexible and inexpensive. The dispatching algorithm is able to classify network packets on the basis of many features, such as protocols, IP addresses, and port numbers. This mechanism allows the NIDS to achieve an effective load sharing among multiple NIDS sensors. Another innovative feature is represented by the introduction of a load balancing mechanism that can dynamically reassign an already established connection to an arbitrary sensor. Moreover, this algorithm allows sensors to perform a stateful, in-depth traffic analysis, that guarantee maximum detection accuracy together with load balancing properties.

III. Parallel NIDS architecture

The proposed architecture for traffic distribution and analysis consists of various components that are described in Figure 1. The *traffic source* is installed on the high-speed monitored link. Its purpose is to provide the scatterer with a copy of the traffic that must be inspected. A suitable traffic source may be implemented by mean of a network tap, a switch with SPAN port or a hub. The preference for a network tap instead of a hub or a switch is motivated by its higher throughput and availability.

The traffic source feeds a *scatterer* that, in our implementation, captures every Ethernet frame and sends the frame to one of the directly connected *slicers*. The scatterer is the only centralized element, and has to manage traffic at the same throughput of the network link. Architecture scalability is limited by the number of slicers that the scatterer is able to handle. In order to keep the computational cost reasonably low, scattering operations must be as simple as possible. Here, the presented results are based on a simple round-robin dispatching policy, because the comparison of different algorithms is out the scope of this paper.

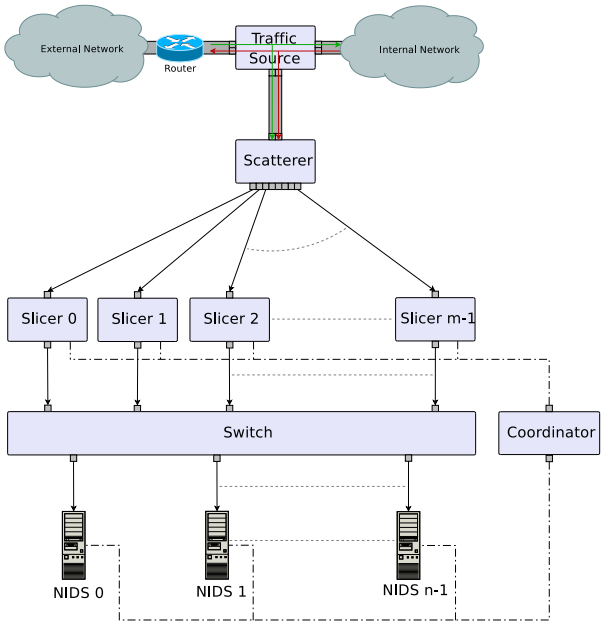

Fig. 1. Parallel NIDS architecture

The slicers capture every frame from the input interface and have to determine the destination NIDS sensor of each of them. That operation cannot be executed by the scatterer, because in our implementation the destination sensor is selected on the basis of a frame analysis, that is a quite complex operation. Indeed, each slicer implements a set of slicing rules, that are used to classify every received Ethernet frame. In particular, our implementation of the slicer makes it possible to select the destination sensor on the basis of various features, such as protocols, IP addresses and port numbers. Slicing rules need to be carefully designed, so to route every frame towards the NIDS that may need it to detect an attack (see Section IV). This is one of the most critical task because we want to perform a stateful traffic analysis. Hence, we must guarantee that every frame belonging to the same transport level connection is routed to the same NIDS sensor.

Once applied the slicing rules to determine the destination sensor, a slicer writes a logical indicator of the selected sensor in the MAC destination address field of the Ethernet header. It is also possible that a single Ethernet frame has to be sent towards two or more sensors. In those instances, the slicer creates a copy of the frame for each destination sensor, and applies a different indicator to each copy.

After the slicer layer, the frames are sent to the *switch* that is used to enforce routing decisions that have been previously taken by slicers. Every frame coming from one of the switch input interfaces is routed to the sensor indicated by the value written in the MAC destination address of the Ethernet frame header. If the switch is programmable (that is, if the switch is provided with a static routing table), then sensor indicators used by the slicers may be unrelated to the real hardware addresses of the sensors input interfaces. On the other hand, if the switch is not programmable, then each sensor indicator is given by the MAC address owned by the input interface of the corresponding sensor.

The next layer of the architecture consists of a set of NIDS sensors. In our proposal, NIDS sensors are implemented through a custom version of Snort IDS [15], that allows the migration of state information related to currently analyzed connections. This new feature makes it possible to redirect an already opened connection to a different NIDS sensor without altering the results of the traffic analysis.

The last element is the *coordinator*, which is directly connected to all slicers and sensors. The coordinator is used to run the load balancing algorithm, and it is important to notice that it does not limit the scalability of the architecture. Indeed, it only monitors the load of sensors and enforces actions possibly triggered by the load balancing algorithm, like changes in slicing rules and migration of connection states.

NIDS sensors are directly connected to the switch, and receives only a subset of the traffic flowing through the monitored link. That subset of traffic includes every necessary frame to carry out a stateful traffic analysis so that the proposed architecture can safely distribute network traffic analysis. Due to the parallel nature of the proposed architecture, network packets belonging to the same transport level connection may reach the NIDS sensor in a wrong order. As an example, this could be due to the use of slicers with different computational power. However, Snort pre-processors are able to restore the correct order of the network packets. Hence, traffic flow reordering can be carried out by the NIDS sensors, without the use of specific network components.

Experimental results that demonstrate scalability and efficacy of the proposed architecture are reported in Section V-A.

IV. Configuration of the parallel architecture

A parallel NIDS architecture is a complex infrastructure that must be carefully configured. To this purpose, we need to know (at least approximately) some important characteristics about the incoming traffic, such as maximum throughput and most common protocols. That information allow us to design and implement the set of slicing rules, which have a great impact on the overall efficacy of the architecture. In our version, a well designed set of slicing rules has to satisfy the following main properties.

1) Packets belonging to the same connection have to be routed towards the same NIDS sensor. This is necessary to trace and reassemble all connections, as required by the stateful traffic analysis characterizing the proposed architecture.
2) Network traffic should be equally distributed among the available NIDS sensors. This allows the architecture to achieve good load balancing properties.

Both requirements can be satisfied even through a well designed set of slicing rules. While it is rather simple to write slicing rules that preserve transport level connections, achieving a reasonable load balancing among NIDS sensors is a non-trivial task.

A. Load Balancing

In static architectures, such as [17], the set of slicing rule is designed on the basis of a quantitative analysis of traffic samples, so that every relevant change in traffic pattern can lead to load unbalance and consequent risks of packet losses. On the other hand, the proposed parallel architecture can achieve load balancing by dynamically adapting slicing rules and NIDS sensors to the current NIDS sensors load.

We have deployed a novel mechanism that allows the parallel NIDS to dynamically move an open connection to a different sensor without altering the results of the traffic analysis. To this purpose, the sensors are implemented by adding two new features to the original version of Snort: the first feature exports state information related to analyzed connection and to store them in files; the second reads state information stored in files generated by other instances of Snort, thus creating the correct state for a connection that has not been yet analyzed.

To move a connection to a different sensor, we have first to export its state from the sensor that previously handled that connection, and then we have to move files containing state information to the new sensor. Finally, the new sensor can read a file and import state information for the moved connection.

Both export and import operations can be triggered at run-time by simply sending the proper signals to a running instance of our Snort version.

Figures 2, 3 and 4 describe an example of the problems related to load balancing for NIDS sensors and the steps of the proposed solution. In Figure 2 we have two NIDS sensors that perform a stateful, in-depth analysis on incoming traffic. We assume that the sensor *NIDS 1* analyzes the network connection *Connection 1*, and *State 1* represents the state information related to that connection. The sensor *NIDS 2* analyzes two network connections (*Connection 2* and *Connection 3*) that have state information stored in *State 2* and *State 3*, respectively.

Let us suppose that sensor *NIDS 2* gets overloaded, and that the load balancing algorithm reacts by moving *Connection 2* from *NIDS 2* to *NIDS 1*. Figure 3 shows the effects of the load balancing action through traditional NIDS sensors: *NIDS 1* picks *Connection 2* midstream, but it is unable to create a consistent state and to perform a stateful analysis on that connection.

Instead, our solution is shown in Figure 4 showing how it is possible to migrate the state of *Connection 2* to *NIDS 1*, thus allowing our system to perform a reliable and stateful analysis.

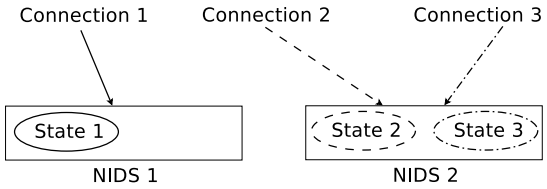

Fig. 2. Initial situation

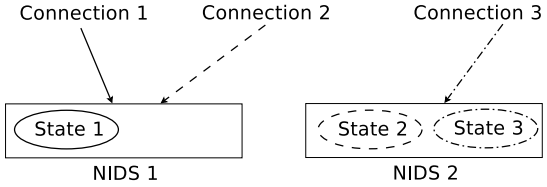

Fig. 3. Connection reassignment without state migration

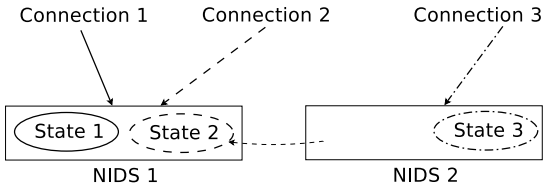

Fig. 4. Connection reassignment with state migration

Thanks to our novel technique, the NIDS receiving an already opened connection, that was previously analyzed by an overloaded sensor, is able to perform a stateful inspection on the last part of the connection. This mechanism allows for a dynamic traffic redistribution between NIDS sensors without altering the analysis results. Moreover, it is important to observe that this mechanism can work with any load balancing policy.

B. Dimensional bindings

An important step for the configuration of the parallel architecture for NIDS is to carry out a dimensional binding for each component. For this analysis, we use the notations in Table I.

TABLE I
CONSTRAINTS OF THE ARCHITECTURE COMPONENTS

Symbol	Meaning
$B^{Link-IN}$	Throughput of incoming traffic
B^{NIC-IN}	Highest bandwidth of the scatterer input interface
$N^{NIC-OUT}$	Number of slicers
$B_i^{NIC-OUT}$	Bandwidth of the scatterer i-th output interface
S_i^{IN}	Highest bandwidth manageable by the i-th slicer
S_i^{OUT}	Bandwidth of the i-th slicer output interface
W_i^{IN}	Bandwidth of the i-th input interface of the switch
W^{T-MAX}	Highest aggregate throughput of the switch
W_i^{OUT}	Bandwidth of the i-th output interface of the switch
I_i^{MAX}	Highest throughput manageable by the i-th NIDS sensor

To get a well dimensioned architecture, where no component is overwhelmed by the incoming traffic, we have to satisfy the following conditions:

$$B^{NIC-IN} \geq B^{Link-IN}$$

The bandwidth manageable by the scatterer input interface has to be bigger than the bandwidth of the traffic that we want to inspect.

$$\sum_{i=0}^{m-1} B_i^{NIC-OUT} \geq B^{Link-IN}$$

The aggregate output bandwidth of the scatterer has to be bigger than the bandwidth of the traffic that we want to inspect.

$$S_i^{IN} \geq B_i^{NIC-OUT}, \ \forall i$$

The computational capacity of the slicer connected to the i-th scatterer output interface has to be large enough to manage all the traffic generated by that interface.

$$W_i^{IN} \geq S_i^{OUT}, \ \forall i$$

The bandwidth of every input interface of the switch has to be bigger than the traffic produced by the directly connected

slicer.

$$W^{T-MAX} \geq \sum_{i=0}^{m-1} S_i^{OUT}$$

The aggregate throughput of the switch has to be bigger than the sum of the traffic volumes generated by the slicers.

$$I_i^{MAX} \geq W_i^{OUT}, \ \forall i$$

The highest bandwidth that the i-th NIDS sensor is able to manage has to be bigger than the throughput generated by the i-th output interface of the switch.

Depending on the slicing rules, a single frame can be required by two or more sensors. In that instance, we have to create a copy of the frame for every sensor that requires it, hence the number of frames generated by the slicers can be bigger than the number of frames captured by the scatterer. We can state that

$$S_i^{OUT} \geq B_i^{NIC-OUT}, \ \forall i$$

Hence, the number of frames to be analyzed by NIDS sensors is equal or greater than the number of frames flowing through the monitored link.

If we consider that slicers share the same configuration rules, and slicers have the same probability to receive a frame that is required by more sensors, then we can state that

$$S_i^{OUT} = B_i^{NIC-OUT} \cdot k_i$$

The constant k_i is given by

$$k_i = \frac{\sum_{t=0}^{T-1} N(f_t)}{T}$$

where T represents the number of frames received by the i-th slicer, and the function $N(f_t)$ denotes the number of copies of the t-th frame produced by the i-th slicer. We can verify that $k_i \geq 1$, and

$$S_i^{IN} = S_i^{OUT} \iff k_i = 1 \iff N(f_t) = 1 \ \forall t$$

The important consequence of this analysis is that the parallel architecture proposed in this paper may achieve an increment in the volume of traffic to be analyzed by NIDS sensors. Frame duplication can be kept reasonably low (eventually nullified) thanks to a careful design of the slicing rules and event space distribution.

V. Experimental results

In this section we describe the most important experimental results that aim to validate the functional properties of the parallel architecture, to demonstrate its scalability and to prove the feasibility of the load balancing mechanism. The prototype architecture has been implemented in C language for GNU/Linux platforms.

A. Architecture validation

For the tests we have configured a virtual network, where virtual hosts have been implemented through *User Mode Linux* [21], and virtual Ethernet links through *OpenVPN* [22]. Virtual networks allow us to test the efficacy of the architecture for an arbitrary number of slicers, reassemblers and NIDS sensors, and limited usage of hardware. The prototype architecture consists of three slicers, and three NIDS sensors. Traffic source activity is emulated through *Tcpreplay* [23], that replays the IDEVAL [24]–[26] traffic towards the input interface of the scatterer at a configurable rate. Each NIDS has been implemented through the open source software Snort [15]. Many different configurations have been tested, and for space limit reasons, here we report the results referring to one significant example, where slicers implement a simple set of slicing rules. The first rule routes to the sensor 1 every TCP packet coming from or, directed to, port 80. The second rule routes to the sensor 2 every TCP packet coming from, or directed to, port 23. The third rule routes to the sensor 3 every packet that has not been routed to sensors 1 or 2.

From this test, we have that the first NIDS (connected to the first sensor) analyzes 526440 frames and generates 495 alerts, the second NIDS analyzes 529439 frames and generates 22 alerts, and the third NIDS analyzes 597498 frames and generates 2686 alerts. The results are summarized in Table II. Despite of the simplicity of the slicer rules, we can observe that the traffic is really well distributed among the three NIDS sensors.

Load sharing is an important property, but we also need to guarantee that the distribution of the traffic analyzes among different components does not affect the stateful analysis properties. To this purpose, we collect reliable data traffic by using just one Snort sensor that is configured with the same rules that were used for the previous experiment. From this system, we obtained a total of 3203 alerts that correspond perfectly to the sum of the alerts obtained by three sensors reported in Table II.

TABLE II
VALIDATION OF THE ARCHITECTURE

	NIDS 1	NIDS 2	NIDS 3	Total
Packets	526440	529439	597498	1653377
Alerts	495	22	2686	3203

This important result proves that the proposed architecture allows us to distribute the computational costs of traffic analysis without altering the results.

B. Scalability of components and system

To verify the scalability of the entire architecture, we carry out several experiments for different number of sensors. In each test we measure the highest network traffic that we are able to analyze correctly, that is, without significant (greater

than 1%) packet loss in NIDS sensors. For the parallel architecture, we use common PC hardware that was connected as in the scheme of Figure 1. Generated traffic reproduces IDEVAL traffic that is transmitted through Tcpreplay and flows through a Gigabit Ethernet. The results are shown in Figure 5, where the X-axis denotes the number of sensors, and the Y-axis the highest analyzed throughput.

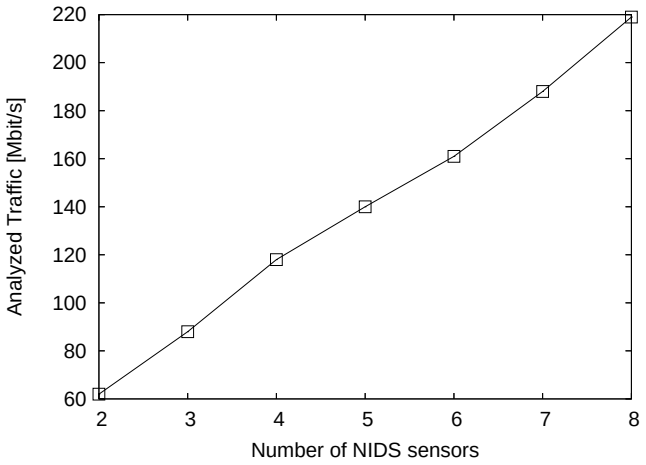

Fig. 5. Scalability of the parallel NIDS architecture

From this figure, it is immediate to observe that the capacity of analysis grows almost linearly for increasing numbers of sensors, thus demonstrating the scalability properties of the proposed architecture.

C. Load Balancing

Another innovation of the proposed architecture is represented by its ability of dynamically migrating the states of analyzed connections between different sensors. This feature can be used to achieve load balancing among NIDS sensors, thus increasing the scalability and the overall effectiveness of the parallel NIDS architecture, although it is mandatory not to alter the analysis results.

To demonstrate the feasibility of our proposal, we configure a simple parallel NIDS architecture, with two slicers and two NIDS sensors, to analyze the IDEVAL network traffic injected in the scatterer by Tcpreplay. In Figure 6, we can see the amount of traffic analyzed by each sensor. The Y-axis reports the throughput analyzed by the sensors in Mbit per second, while the X-axis represents the time of the experiment (in second). We can see that during the first 100 seconds both the sensors have to analyze a manageable amount of traffic (always less than 35 Mbit/s). This result indicates that slicing rules achieve an acceptable load sharing between the two sensors. Around 100 second, the traffic pattern changes, and the traffic analyzed by sensor 1 suddenly increases, while the traffic analyzed by sensor 2 remains almost constant. Figure 6 highlights the consequences of a change in network traffic pattern.

To address this load unbalance, without tampering the stateful inspection carried out by NIDS sensors, we use our load balancing mechanism that allows the migration of state information. The design of the best load balancing algorithm for a parallel NIDS architecture is beyond the scope of this work. Hence, for the purposes of this paper, we used a simple algorithm based on two thresholds and round-robin distribution. A sensor is considered overloaded, if it has to analyze over 40 Mbit/s of traffic for more than 5 seconds (*on threshold*). In this instance, the load balancing algorithm assigns slices of traffic from the overloaded NIDS sensor to the other (not overloaded) sensors in a round-robin way. Meanwhile, state information related to the reassigned slices of traffic are moved to the new NIDS sensor.

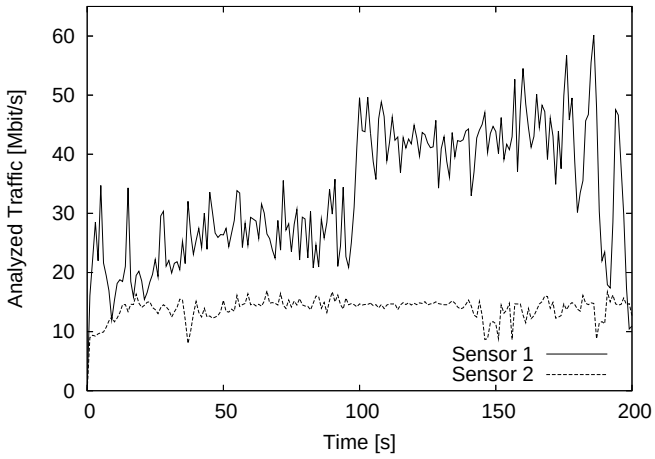

Fig. 6. Throughput for sensors 1 and 2 without load balancing

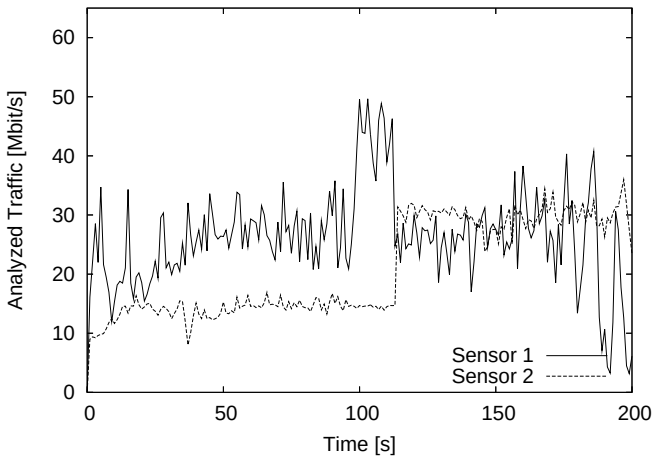

Fig. 7. Throughput for sensors 1 and 2 with load balancing

The results of this simple load balancing algorithm are presented in Figure 7. Network traffic and initial configuration of the parallel NIDS architecture are identical to the scenario represented in Figure 6 for the first 100 seconds. At this point, the increment of traffic to the sensor 1 triggers the load balancing algorithm. It moves some packets to the sensor 2 until the traffic of the sensor 1 goes below the second

threshold equal to 30 Mbit/s (*off threshold*). The success of the mechanism is clearly demonstrated by the second part of the experiment after 100 seconds in Figure 7.

VI. Conclusions

Network Intrusion Detection Systems have to perform a complete analysis of the traffic flowing through networks that can easily reach Gbps capacities. We should also consider that a fully reliable analysis requires the NIDS to track and reassemble each distinct connection. Hence, the throughput of monitored traffic and the number of concurrently open connections may represent a limit to the applicability of NIDS to the modern and future networks. Any NIDS based on a single component cannot scale over certain thresholds, even if it has some parts built in hardware. Hence, parallel architectures appear as the most valuable alternative for achieving a scalable NIDS.

In this paper, we propose a parallel architecture of a NIDS that guarantees stateful analysis, load balancing and high scalability. The proposed architecture represents a significant improvement with respect to previous works. We demonstrate that there is no theoretical limit to increase the number of the parallel architecture components. These performance properties come together with low costs and high flexibility that is guaranteed by a total software implementation.

Future work is directed to propose and compare different load balancing algorithms that may be integrated in the proposed system.

References

[1] L. Schaelicke, T. Slabach, B. Moore, and C. Freeland, "Characterizing the performance of network intrusion detection sensors," in *Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection (RAID 2003)*, ser. Lecture Notes in Computer Science. Berlin–Heidelberg–New York: Springer-Verlag, September 2003.

[2] M. Roesch, "Snort - lightweight intrusion detection for networks," in *LISA '99: Proceedings of the 13th USENIX conference on System administration*. Berkeley, CA, USA: USENIX Association, 1999, pp. 229–238.

[3] "Top layer networks." [Online]. Available: http://tcpreplay.sourceforge.net

[4] "Juniper networks." [Online]. Available: http://www.juniper.net

[5] H. Song, T. Sproull, M. Attig, and J. Lockwood, "Snort offloader: A reconfigurable hardware NIDS filter," in *15th International Conference on Field Programmable Logic and Applications (FPL)*, Tampere, Finland, Aug. 2005.

[6] H. Song and J. W. Lockwood, "Efficient packet classification for network intrusion detection using fpga," in *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*. New York, NY, USA: ACM Press, 2005, pp. 238–245.

[7] L. Bu and J. A. Chandy, "Fpga based network intrusion detection using content addressable memories," *fccm*, vol. 00, pp. 316–317, 2004.

[8] C. R. Clark, W. Lee, D. E. Schimmel, D. Contis, M. Kon, and A. Thomas, "A hardware platform for network intrusion detection and prevention," in *Workshop on Network Processors and Applications at HPCA (NP-3)*, Madrid, Spain, 2004, pp. 136–145.

[9] K. Xinidis, K. G. Anagnostakis, and E. P. Markatos, "Design and implementation of a high-performance network intrusion prevention system." in *SEC*, 2005, pp. 359–374.

[10] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C.-L. Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal, and D. Mansur, "Dids (distributed intrusion detection system) motivation, architecture, and an early prototype," *Internet besieged: countering cyberspace scofflaws*, pp. 211–227, 1998.

[11] S. Snapp, J. Brentano, G. Dias, T. Goan, L. Heberlein, C. Ho, K. Levitt, B. Mukherjee, (with T. Grance D.L. Mansur, K. Pon, and S. Smaha), "A system for distributed intrusion detection," in *COMPCON*, San Francisco, CA, 1991, pp. 170–176.

[12] P. K. Varshney, *Distributed Detection and Data Fusion*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1996.

[13] D. Burroughs, L. Wilson, and G. Cybenko, "Analysis of distributed intrusion detection systems using bayesian methods," in *IEEE International Performance Computing and Communication Conference*, 2002.

[14] "Prelude home page." [Online]. Available: http://www.prelude-ids.org/

[15] "Snort home page." [Online]. Available: www.snort.org

[16] A. Orebaugh, S. Biles, and J. Babbin, *Snort cookbook*. O'reilly, 2005.

[17] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer, "Stateful intrusion detection for high-speed networks," in *Proceedings of the IEEE Symposium on Research on Security and Privacy*. Oakland, CA: IEEE Press, May 2002.

[18] Y. Jianying, Z. Jiantao, W. Pei, and T. Wang, "An application of network address translation on gateway," in *Proceedings of the 2003 International Conference on Neural Networks and Signal Processing*, 2003, pp. 229–238.

[19] L. Schaelicke, K. Wheeler, and C. Freeland, "Spanids: a scalable network intrusion detection loadbalancer," in *CF '05: Proceedings of the 2nd conference on Computing frontiers*. New York, NY, USA: ACM Press, 2005, pp. 315–322.

[20] R. Sommer and V. Paxson, "Exploiting independent state for network intrusion detection," in *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 59–71.

[21] U. M. L. C. Team, "User mode linux howto." [Online]. Available: http://user-mode-linux.sourceforge.net/UserModeLinux-HOWTO.html

[22] "Openvpn home page." [Online]. Available: http://www.openvpn.net

[23] "Tcpreplay home page." [Online]. Available: http://tcpreplay.sourceforge.net

[24] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das, "Analysis and results of the 1999 darpa off-line intrusion detection evaluation," in *RAID '00: Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection*. London, UK: Springer-Verlag, 2000, pp. 162–182.

[25] J. M. Hugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 262–294, 2000.

[26] M. V. Mahoney and P. K. Chan, "An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection." in *RAID*, 2003, pp. 220–237.