Dynamic load balancing for network intrusion detection systems based on distributed architectures

Mauro Andreolini, Sara Casolari, Michele Colajanni, Mirco Marchetti
Department of Information Engineering
University of Modena and Reggio Emilia
{mauro.andreolini, sara.casolari, colajanni, mirco.marchetti}@unimore.it

Abstract

Increasing traffic and the necessity of stateful analyses impose strong computational requirements on network intrusion detection systems (NIDS), and motivate the need of distributed architectures with multiple sensors. In a context of high traffic with heavy tailed characteristics, static rules for dispatching traffic slices among distributed sensors cause severe imbalance. Hence, the distributed NIDS architecture must be combined with adequate mechanisms for dynamic load redistribution. In this paper, we propose and compare different policies for the activation/deactivation of the dynamic load balancer. In particular, we consider and compare single vs. double threshold schemes, and load representations based on resource measures vs. load aggregation models. Our experimental results show that the best combination of a double threshold scheme with a linear aggregation of resource measures is able to achieve a really satisfactory balance of the sensor loads together with a sensible reduction of the number of load balancer activations.

1 Introduction

Network Intrusion Detection Systems (NIDS) are a diffuse reality in many complex networks where it is necessary to look for malicious packets and illicit network activities. To obtain a fully reliable analysis, a NIDS needs to track, reassemble and examine each distinct connection at wire-speed. Any architecture based on just one traffic sensor cannot be sufficient to face networks that are characterized by a continuous increase of capacities and traffic. Hence, distributed architectures with multiple sensors appear as the most effective solution for a scalable traffic analysis of present and future high speed networks. These distributed architectures are characterized by a set of multiple slicers that assign portions of network traffic to different NIDS sensors through some traffic shaping policy (an example is in Figure 1). Each NIDS sensor (*sensor* for short) analyzes the received traffic for illicit network activities and, when necessary, generates alerts. The main problem is that inbound traffic received by the distributed NIDS architecture exhibits heavy-tailed characteristics [4, 1] and flash crowds [12]. If the system adopted static traffic partitioning, the amount of traffic reaching each sensor would be extremely variable and would cause severe load unbalance situations, with consequent risks of packet losses. For these reasons, we think necessary to recur to a dynamic redistribution of the load among the sensors. To this purpose, we introduce a *coordinator* that receives periodic information about the state of the sensors and, on the basis of some policy, it can activate a load balancing mechanism to move portions of network traffic from overloaded to less loaded sensors. The load conditions of each sensor are usually evaluated through the periodic sampling of monitored raw data, that we call *resource measures* or *samples*. While a single measure offers an instantaneous view of the load conditions of a sensor, it does not help the system to distinguish real overload conditions from transient peaks. Consequently, in a context of heavy-tailed traffic with unexpected spikes and burst arrivals, it is extremely difficult to define the best activation decision policy and load redistribution algorithm for the load balancing process. The optimal policy would

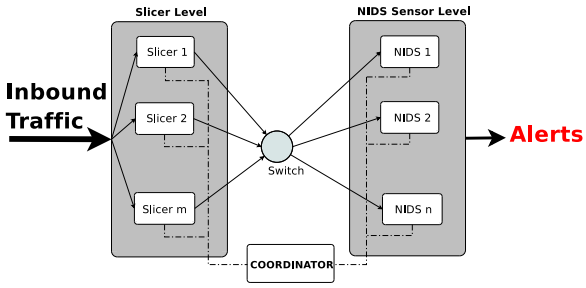


Figure 1. An example of distributed NIDS architecture.

achieve the best sensor load balance with the minimal number of load balancer activations.

We initially show that the activation policies that are based on resource measures lead to unsatisfactory load distribution and excessive load balancer activations. It is interesting to observe that this result is due to the extreme variability of the samples and it is independent of the activation/deactivation strategy that may be based on one or two thresholds. With the goal of reducing the load representation skews and to improve the load balancing process, we propose and compare different models for load aggregation for various measures of the samples. To the best of our knowledge, this is the first paper studying aggregation models and activation policies in the context of dynamic load balancing for distributed NIDS. Any previously proposed solution for NIDS [20, 5] evaluates the sensor load conditions and takes decisions on the basis of monitored samples. We show that even linear aggregations (Simple Moving Average and Exponential Moving Average) that are integrated with a double threshold mechanism for activation/deactivation lead to a consistent improvement of the load balancing process and of the overall performance of a distributed NIDS. Several experimental results achieved through a prototype implementation show a reduction of load balancer activations, a better stability of the sensor loads, and a consequent decrement of the traffic that cannot be examined with fully reliable properties.

The remaining part of the paper is organized as follows. In Section 2, we focus on the static and dynamic traffic distribution mechanisms for NIDS architectures consisting of multiple sensors. In Section 3, we consider and compare load balancer activation policies based on resource measures and on two aggregation models that is, Simple Moving Average and Exponential Moving Average. In Section 4, we evaluate the benefits of load aggregation on the performance of a distributed NIDS architecture where the load balancer is based on a double threshold policy for activation/deactivation. In Section 5, we compare the contributions of this paper with respect to the state of the art. In Section 6, we conclude the paper with some final remarks.

2 Static traffic splitting

The design of a distributed NIDS architecture is a difficult task that requires several choices. For example, the incoming traffic can be distributed statically or dinamically across different sensors. In this section we show that static rules lead to severe imbalance conditions because the traffic is unknown a priori. Hence, it is necessary to recur to mechanisms that can reassign some network traffic flows across sensors.

We demonstrate through experimental results the drawbacks of a static distribution and motivate the use of dynamic load balancing mechanisms that will be described in Section 3. The results are obtained through a prototype implementation of a distributed NIDS as in Figure 1, that is detailed in [6]. It consists of two slicers, three sensors based on the *Snort* intrusion detection system [21], and one coordinator. The distributed NIDS are exercised through the IDEVAL [15] traffic dumps that are considered standard traces containing attacks. Each component of the architecture runs on a different machine that is connected to the others through Gigabit Ethernet adapters. The traffic is generated by multiple machines, where each of them executes the *tcpreplay* trace replaying software [22]. Each experiment typically lasts for 1000 seconds.

Static splitting is a technique commonly adopted in different contexts to distribute incoming traffic statically among different nodes [14]. Splitting can be performed on the basis of protocol information (IP addresses, TCP ports) or more sophisticated application-level payload (signatures). In this paper, we consider splitting based on TCP and IP information. We have instrumented the popular *iptables* packet filter framework [11] with a set of *slicing rules* for frame analysis, frame routing and MAC address rewriting. We partition evenly the IP space of the internal network (27 different hosts) across the 3 sensors. Although each NIDS sensor becomes responsible for the analysis of a subset of 9 IP addresses, the resulting traffic is quite different in terms of packets.

Figure 3 reports the network traffic (in Mbps) received by each sensor during the experiment. The horizontal line at T_{max}=40Mbps denotes the maximum capacity of analysis of each sensor. Higher load causes packet drops that correspond to a portion of traffic that cannot be analyzed for intrusion detection alert. Figure 3 clearly shows that static traffic splitting does not balance at all the sensor load. Sensor 1 operates always beyond its maximum capacity, while sensor 3 receives one fourth of the traffic that it could analyze. The reason behind this severe imbalance is the ex-

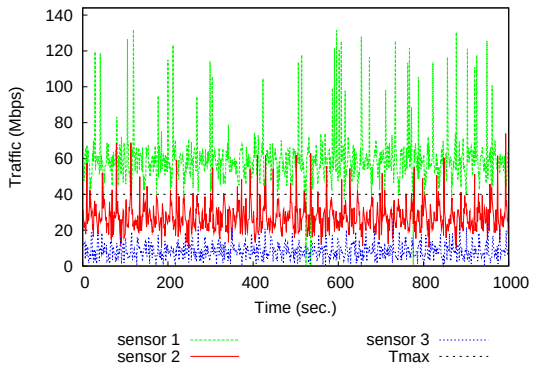


Figure 2. Static traffic splitting based on IP addresses.

Table 1. Workload models characteristics.

	min	avg	max	stddev
packets per connection	1	31.34	130105	671.71
packet size	42	212.73	1514	443.68

treme variability of the numbers and sizes of the packets related to each subset of IP addresses. The considered workload model reflects the typical behavior of a real network traffic. In Table 1 we present some statistics about two important parameters that influence the amount of work performed by each sensor: the number of packets per connection and the size of network packets. We can see that the standard deviation of the number of packets per connection is 20 times higher than its average, which indicates a very high dispersion. Since a sensor analyzes all packets pertaining to a connection, the amount of work corresponding to a subset of IP addresses can be (and usually is) very variable. A similar consequence holds for the packet size. As a sensor analyzes the whole payload of a packet to detect anomalies, different payload sizes imply different amounts of work. Static traffic slicing rules do not (and cannot) take into account connection lengths or packet sizes. Hence, different sensors can be subject to extremely variable amounts of received traffic and consequent work. These considerations motivate the need for the integration of a dynamic load balancer into the distributed NIDS architecture.

3 Dynamic load balancing

Balancing the load of multiple sensors that are subject to highly variable traffic is a difficult problem. Multiple alternatives may characterize a load balancer mechanism, such as: the *activation strategy* that triggers the load balancing process, the *selection policy* that chooses the amount of traffic to be moved from one sensor to another, the *location policy* that decides the new target sensor(s), the *deactivation strategy* that stops the load balancing process and the *sensor load* that may be evaluated in terms of one or multiple different resources, performance metrics and possible aggregations of the resource samples.

3.1 Alternative policies

The simplest activation/deactivation scheme is based on a *single threshold*: a load balancing action is triggered whenever the last load information used by the coordinator overcomes a static threshold. The single threshold model has been widely adopted (just to cite an example in [17]), and its oscillatory risks of continuous activation and deactivation phases are well known, especially in highly variable environments, such as those considered in this paper.

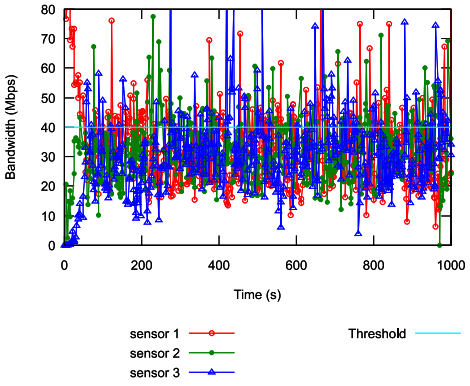
The risks of false alarms can be reduced through several schemes, for example by signaling an alarm only when multiple samples overcome the threshold or enlarging the observation period. In this paper, we consider another popular scheme that is based on a *double threshold*, where the load balancer is activated when the load of at least one sensor is above the *high threshold* that represents the activation threshold. The *low threshold* is used for a twofold purpose because it denotes the deactivation threshold and the NIDS sensors that are suitable for receiving traffic slices from the overloaded sensor(s).

Once the load balancer has been activated, there are many feasible selection and location policies. When the load balancing mechanism has been activated, we choose to move one *traffic slice* of one internal network host from the overloaded sensor to other underloaded sensor(s) through a round-robin algorithm. We find convenient to consider this simple, yet stable and efficient policy that minimizes the computational overhead of the load balancing process.

An open issue is that any load balancing mechanism requires a representation of the sensor load. Choosing a good representation is a problem by itself, because there are many critical hardware and software resources at each sensor node (CPU, disk, network, memory, open file and socket descriptors). Typically, the resource load or status can be measured through several system monitors (e.g., sysstat [10] that yields instantaneous measures of CPU utilization, disk and network throughput at regular time intervals). In this paper, the load of a sensor is measured in terms of received network traffic (in Mbps), which is a common solution in the context of distributed NIDS systems [20, 5]. Initially, we use a representation of the sensor load based on periodic samples and then we pass to consider linear aggregation models applied to the resource load samples.

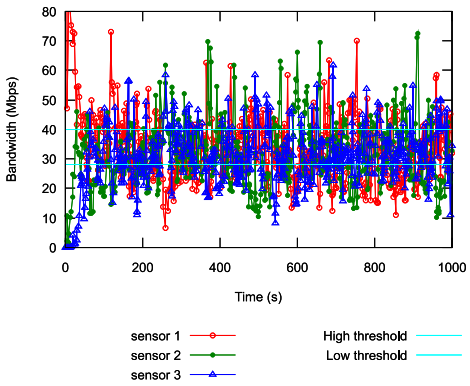
3.2 Policies based on resource samples

MWe initially consider the two activation schemes based on single and double threshold that use the network traffic at each sensor as their measure of the sensor load. The traffic is sampled every second, although different time intervals do not change the main results and conclusions. Figures 3 (a) and (b) report the traffic on each of the three sensors during the experiment. The horizontal line set to 40 Mbps in Figure 3 (a), and to 40 Mbps and 28 Mbps in Figure 3 (b) denote the activation/deactivation thresholds for the dynamic load balancing mechanism. Comparing the results in Figure 2 referring to static traffic splitting with those in Figures 3, we can see that the introduction of a dynamic load balancer improves substantially the sensor load balance. To quantify the benefits of dynamic load balancing, we compute three indexes that are related to the performance and to the efficiency of the distributed NIDS: the load balance metric, the packet loss percentage and the number of sensor



(a)

Activation/deactivation scheme based on single threshold.



(b)

Activation/deactivation scheme based on double threshold.

Figure 3. Dynamic load balancing based on resource samples.

overload signals.

The *load balance metric* [3] (LBM) measures the degree of load balancing across different nodes. Let us define the load of the sensor i (of n sensors) at the j^{th} observation (of the observation period m) as $load_{i,j}$ and $peak\_load_j$ as the highest load on any sensor at the j^{th} observation. The LBM is defined as follows:

$$LBM = \frac{\sum\limits_{1 \le j \le m} peak\_load_j}{(\sum\limits_{1 \le j \le m} \sum\limits_{1 \le i \le n} load_{i,j})/n} \quad (1)$$

Note that the value of the LBM can range from 1 to the number of sensors n. Smaller values of the LBM indicate a better load balance than larger values.

The packet loss percentage is the ratio between not analyzed and received packets for all sensors, that is,

$$\frac{\sum_{j=1}^{n} f_j}{\sum_{j=1}^{n} f_j + r_j} \quad (2)$$

where n is the number of sensors, f_j is the number of packets not analyzed by sensor$_j$ and r_j is the number of packets received by sensor$_j$.

We also consider the number of overload signals generated by each sensor, that lead to the invocation of the dynamic load balancing mechanism.

Table 2 reports the three performance results for the static traffic splitting, dynamic single threshold and dynamic double threshold distribution mechanisms. The LBM values show that the dynamic load balancing schemes improve sensor load significantly. The packet loss percentage drops from 40% to 20% when moving from a static to a dynamic load balancer, and to 13% if we consider a double threshold activation mechanism. This implies an improvement in the efficiency of the network traffic analysis. Furthermore, the number of overload signals decreases by a factor of at least 2.4. This result is important because it denotes that there are fewer critical load conditions. However, there are still many invocations of the dynamic load balancer (357 and 321 in the single and double threshold strategy, respectively). We think that the main motivation for these high numbers is due to an inappropriate representation of the sensor load that is based on direct measures. For this reason, we propose some load aggregation models that should reduce the dispersion of load information samples and yield a cleaner view of sensor status.

Table 2. Evaluation of load distribution mechanisms.

	Static	**Dynamic Single Threshold**	**Dynamic Double Threshold**
LBM	1.88	1.41	1.36
Packet loss percentage	40%	20%	13%
Overload signals	861	357	321

3.3 Policies based on aggregation models

We think that in the context of a distributed NIDS architecture subject to heavy-tailed distributed traffic, the use of direct resource samples as a measure of sensor load information is inappropriate because they tend to signal continuous variations between the need of activation and deactivation. For this reason, we consider some load aggregations that allow us to obtain a more representative view of the load status from monitored raw data. The goal of a load aggregation model is twofold: to filter outlier values and to reduce the variability of resource samples, such as the network traffic measured at each sensor.

Let us suppose that at time t_i, a *load aggregation* model can count on a set of n previously obtained resource samples $S(i,n) = (s_{i-n+1}, \ldots, s_i)$ to compute a representation of the load conditions of a sensor. A continuous application of this model produces a sequence of load values that should yield a cleaner trend of the sensors load conditions.

As load aggregation models we consider the class of *moving average* functions. We think that these linear functions, that are commonly used as trend indicators, are sufficient to smooth out resource mDeasures, reduce the effect of out-of-scale values, and are fairly easy to compute at runtime. In this paper, we count on two classes of moving average functions: the *Simple Moving Average* (SMA) and the *Exponential Moving Average* (EMA) that use uniform and non-uniform weighted distributions of the past samples, respectively. As we are interested to run-time models in a context of highly variable systems, we cannot consider other popular linear auto-regressive models, such as ARMA and ARIMA [7, 23], because in the considered extremely variable workload scenario they would require frequent updates of their parameters. These operations are computationally too expensive and inadequate to support run-time decision systems.

Simple Moving Average (SMA). It is the unweighted mean of the n resource measures of the vector $S(i,n)$, that is evaluated at time t_i ($i > n$), that is,

$$SMA(S(i,n)) = \frac{\sum_{i-(n-1)\leq j\leq i} s_j}{n} \quad (3)$$

An SMA-based representative function evaluates a new SMA(S(i,n)) for each sample s_i during the observation period. The number of considered resource measures is a parameter of the SMA model, hence hereafter we use SMA_n to denote an SMA representative function based on n samples. As SMA models assign an equal weight to every resource measure, they tend to introduce a significant delay in the trend representation, especially when the size of the set $S(i,n)$ increases. The EMA models are often considered with the purpose of limiting this delay effect.

Exponential Moving Average (EMA). It is the weighted mean of the n resource measures of the vector $S(i,n)$, where the weights decrease exponentially. An EMA based function $S(i,n)$, at time t_i, is equal to:

$$EMA(S(i,n)) = \alpha * s_i + (1-\alpha) * EMA(S(i-1,n)) \quad (4)$$

where the parameter $\alpha = 2/(n+1)$ is the *smoothing factor*. The initial EMA(S(i,n)) value is set to the arithmetical mean of the first n measures:

$$EMA(S(i,n)) = \frac{\sum_{0\leq j\leq n} s_j}{n} \quad (5)$$

Similarly to the SMA model, the number of considered resource measures is a parameter of the EMA model, hence EMA_n denotes an EMA function based on n samples.

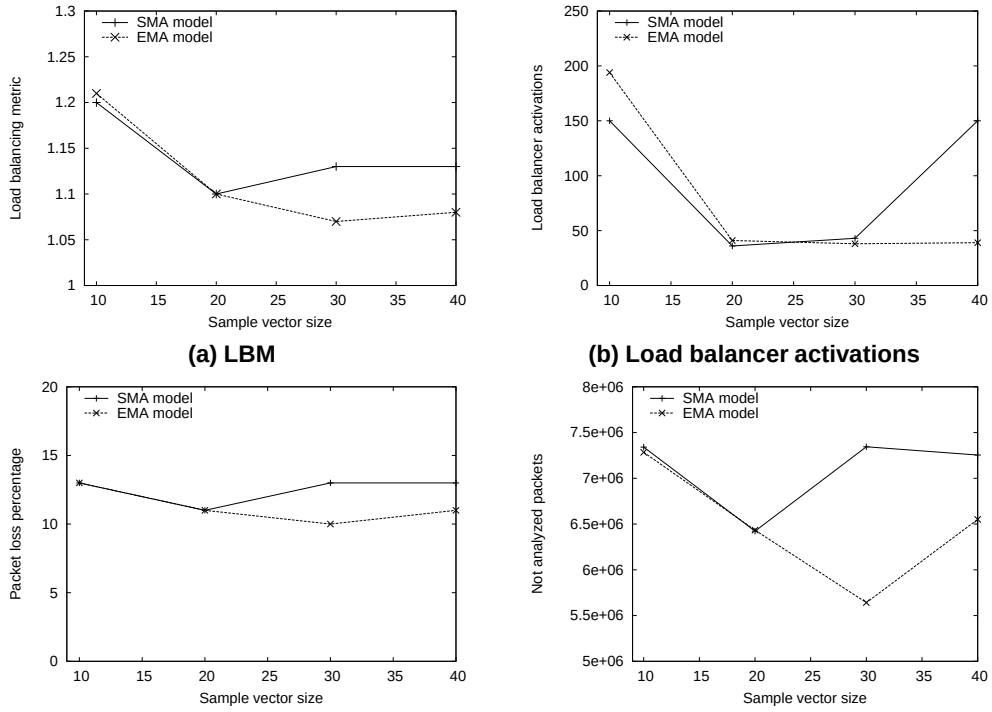
4 Performance results

We will see that, through an appropriate choice of the sample vector sizes, load aggregation based on SMA or EMA models can lead to a significant reduction of the load balancer activations and of the packet loss percentage. All results presented in this section refer to the double threshold activation scheme that has been demonstrated to perform always better than the single threshold scheme in terms of LBM, packet loss percentage and number of load balancer activations. We give a quantitative analysis for both aggregation models by evaluating three important factors: quality (represented by a low value of LBM), overhead (represented by a low number of load balancer activations) and efficacy (represented by a low packet loss percentage and a low number of not analyzed packets).

In Figure 4(a) we report the results of the LBM for the SMA and EMA-based load aggregations and different sizes of the sample vector. The worst balancing performance ($LBM = 1.2$) is obtained for small values of the sample vector history ($n = 10$ in our experiments). If we consider larger sample vectors, we initially observe an improvement of the load balancing process for both load aggregation models ($LBM = 1.1$ at $n = 20$ for the SMA model, $LBM = 1.06$ at $n = 30$ for the EMA model). Further increments of the sample vector history do not improve load balancing.

Let us now consider Figure 4(b) that reports the number of load balancing activations as a function of the sample vector size. Some of the previous considerations are still valid for both the SMA and EMA models. In particular, for small values of n, the load balancing process is activated very often (over 150 times in the $n = 10$ case), while higher values of n do not necessarily reduce the number of activations (for example, we measured 150 activations using the SMA_{40} model). However, if we consider the SMA model, while in Figure 4(a) the LBM remains almost constant for n in $[30, 40]$, the overhead of a load balancer based on the SMA model increases abruptly. The reason of this behavior is straightforward. With increasing values of n, the SMA model introduces delays in the representation of sensor load. This, in turn, implies a reduced ability of the load balancer to react to sudden changes in the traffic distribution, which could lead more easily to sensor overload. The load balancer must be activated more often in order to mitigate this overload.

The impact of load balancing instability on the performance of the sensors is shown in Figure 4(c) and (d), that report the packet loss rate and the number of not analyzed



(a) LBM **(b) Load balancer activations**

(c) Packet loss percentage **(d) Not analyzed packets**

Figure 4. Performance results as a function of the sample vector size.

packets as a function of the sample vector size, respectively. Inefficiencies in the load balancing process can quickly turn to sensor overload and considerable packet loss (up to 13% when using the SMA_{10} model).

All the Figures 4 lead to an important conclusion about the sample vector size: for each considered load aggregation model, there exists a value of n that optimizes quality, overhead and efficacy. The reason of this behavior is simple. For small values of n, load aggregation models present oscillations and lead to false detections and packet losses. For high values of n, the SMA and the EMA models are not able to react quickly enough to load condition changes of the sensors. Hence, in the initial part of the experiment, where one sensor receives all incoming traffic and the other two sensors are idle, the load balancing process is slower. As a consequence, a transient unbalance state may last longer because the initially overloaded sensor passes traffic to the other sensors too slowly. This explains the high packet loss percentages of the most overloaded sensor and the very low packet loss percentage of the initially offloaded sensor.

In the considered scenario, the best dynamic load balancers are based on the SMA_{20} and the EMA_{30} load aggregation models. From these and other not shown results related to different scenarios, we can conclude that the EMA is not only the best load aggregation model, but also the most stable. The stability of EMA is a consequence of its low sensitivity with respect to the sample vector size, provided that n is chosen sufficiently high ($n > 20$ in our scenarios). On the other hand, the SMA load model shows a higher dependency on the sample vector size.

In Table 3 we summarize all performance metrics for the best load balancers based on double threshold activation and aggregated load models, namely SMA_{20} and EMA_{30}. We also include the load balancer based on samples and a static splicing NIDS. The benefits of dynamic load balancing based on aggregation models are clear: the number of sensor overloads reduces considerably, ranging from a factor of 2.6 when passing from a static to a dynamic strategy. If we consider the effects of load aggregation, sensor overload signals drop by a factor of 40 and over. Moreover, the load balancer activations drop by a factor of 7 when considering linear aggregation models instead of resource sample representations. Finally, a dynamic load balancing strategy improves the LBM of at least 30% with respect to static traffic splitting. If we consider load aggregations, the improvement is about 40%. The exponential moving average EMA_{30} shows the best performance in terms of load balance and packet loss. It is worth to note that its results are achieved through the minimum number of load balancer activations.

Table 3. Evaluation of load distribution mechanisms.

	Static strategy	Dynamic load balancing		
	Static traffic splitting	Sample-based model	SMA_{20}-based model	EMA_{30}-based model
LBM	1.88	1.36	1.10	1.07
Packet loss percentage	40%	13%	11%	10%
Not analyzed packets	20971154	7780001	6422771	5641611
Sensor overload signals	861	321	43	40

5 Related work

Detecting significant and permanent load changes of a system resource is at the basis of most run-time decisions for the management of distributed systems. Some examples of applications include load balancers [2], overload and admission controllers [13], request routing mechanisms and replica placement algorithms [18]. The common method to represent resource load values for run-time management systems is based on the periodic collection of samples from server nodes and on the subsequent use of these values. Some low-pass filtering of network throughput samples has been proposed in [19], but the large majority of proposals detect load changes and predict future values on the basis of some functions that work directly on resource measures. Even the proposals that adopt a control theoretical approach to prevent overload or to provide guaranteed levels of performance in Web systems [13] refer to single resource samples (e.g., the CPU utilization or the average Web object response time) as feedback signals. On the other hand, we think that in the proposed scenario of a distributed NIDS, characterized by high instability and variability of network bandwidth samples, real-time management decisions that are based on the direct use of single samples may lead to risky when not completely wrong actions. Our preliminary experimental results show the impossibility of deducing a representative view of a system resource from collected raw measures that are characterized by very large variability.

To demonstrate our hypotheses, we implemented a real small-scale distributed NIDS but the results obtained from this prototype could be extended using larger scale simulations [9, 8].

There are many studies on the characterization of resource loads, albeit related to systems that are subject to quite different workload models with respect to those considered in this paper. Hence, many of the previous results cannot be applied directly to the proposed distributed NIDS system. For example, the authors in [16] evaluate the effects of different load representations on job load balancing through a simulation model that assumes a Poisson job inter-arrival process. Dinda et al. [7] investigate the predictability of the CPU load average in a UNIX machine subject to CPU-bound jobs. The adaptive disk I/O prefetcher proposed in [23] is validated through realistic disk I/O inter-arrival patterns referring to scientific applications. On the other hand, the workload features considered in all these pioneer papers differ substantially from the load models characterizing Web-based servers that show high variability, bursty patterns and heavy tails even at different time scales.

The focus on run-time operations is another key difference of this paper with respect to previous literature. The common method for investigating the efficacy of load representation for run-time management tasks is off-line analysis of samples collected from access or resource usage logs [7]. In this paper, the need for run-time decision supports in a highly variable context has led to evaluate the feasibility of simple yet effective load aggregation models, and possibility of integrating them into our distributed NIDS.

6 Conclusions

A fully reliable and stateful network traffic analysis requires NIDS that are able to handle the ever increasing capacities of present and future network technologies. In this context, distributed architectures with some suitable traffic shaping and load balancing mechanisms appear as the most valuable solution for high speed traffic analysis, due to their intrinsic scalability. However, in a context of high traffic with heavy tailed characteristics, load balancing is a non trivial task that requires adequate solutions at the level of activation mechanisms and load representation. We show that direct resource measures offer instantaneous load views, but they are of little help for distinguishing overload conditions from transient peaks, thus making the activation/deactivation load balancing process extremely unstable. We propose and implement different linear models for sample aggregation that reduce the skewness of the load representation and improve the load balancing process. We show that even simple, linear aggregations can lead to a consistent performance improvement of the entire distributed NIDS architecture. There are two main consequences of an improved sensor load balance: the reduction of activations of the dynamic balancing mechanisms and the reduction of packet losses experienced by the NIDS sensors.

References

[1] M. Arlitt, D. Krishnamurthy, and J. Rolia. Characterizing the scalability of a large web-based shopping system. *IEEE Trans. Internet Technology*, 1(1):44–69, Aug. 2001.

[2] J. Bahi, S. Contassot-Vivier, and R. Couturier. Dynamic load balancing and efficient load estimators for asynchronous iterative algorithms. *IEEE Trans. Parallel and Distributed Systems*, 16(4):289–299, Apr. 2006.

[3] R. B. Bunt, D. L. Eager, G. M. Oster, and C. L. Williamson. Achieving load balance and effective caching in clustered Web servers. In *Proc. of WCW*, San Diego, CA, USA, USA, Apr. 1999.

[4] J. Challenger, P. Dantzig, A. Iyengar, M. Squillante, and L. Zhang. Efficiently serving dynamic data at highly accessed Web sites. *IEEE/ACM Trans. on Networking*, 12(2):233–246, Apr. 2004.

[5] I. Charitakis, K. Anagnostakis, and E. Markatos. An active traffic splitter architecture for intrusion detection. In *Proc. of MASCOTS)*, page 238, Orlando, FL, 2003. IEEE Computer Society.

[6] M. Colajanni and M. Marchetti. A parallel architecture for stateful intrusion detection in high traffic networks. In *In Proc. of the IEEE/IST MonAM*, 2006.

[7] P. Dinda and D. O'Hallaron. Host load prediction using linear models. *Cluster Computing*, 3(4):265–280, Dec. 2000.

[8] A. M. Dobber, G. M. Koole, and R. D. van der Mei. Dynamic load balancing for a grid application. In *In Proc. of HiPC*, 2004.

[9] D. Gao, Y. Shu, S. Liu, and O. Yang. Delay-based adaptive load balancing in mpls networks. In *In Proc. of ICC*, 2002.

[10] S. Godard. Sysstat: System performance tools for the Linux OS, 2004. http://perso.wanadoo.fr/sebastien.godard/.

[11] Iptables, 2005. – http://www.netfilter.org/.

[12] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: characterization and implications for CDNs and Web sites. In *Proc. of WWW)*, Honolulu, HW, May 2002.

[13] A. Kamra, V. Misra, and E. M. Nahum. Yaksha: a self-tuning controller for managing the performance of 3-tiered sites. In *Proc. of IWQOS2004*, Montreal, CA, June 2004.

[14] 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, May 2002. IEEE Press.

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

[16] M. Mitzenmacher. How useful is old information. *IEEE Trans. Parallel and Distributed Systems*, 11(1):6–20, Jan. 2000.

[17] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. M. Nahum. Locality-aware request distribution in cluster-based network servers. In *Proc. of ACM ASPLOS*, San Jose, CA, USA, Oct. 1998.

[18] M. Rabinovich, X. Zhen, and A. Aggarwal. Computing on the edge: a platform for replicating internet applications. In *Proc. of WCW*, Hawthorne, NY, USA, 2003.

[19] A. Sang and S. Li. A predictability analysis of network traffic. In *Proc. of INFOCOM*, 2000.

[20] L. Schaelicke, K. Wheeler, and C. Freeland. SPANIDS: a scalable network intrusion detection loadbalancer. In *In Proc. of CF*, New York, NY, USA, 2005. ACM Press.

[21] Snort home page, http://www.snort.org.

[22] Tcpreplay home page, http://tcpreplay.sourceforge.net.

[23] N. Tran and D. Reed. Automatic ARIMA time series modeling for adaptive I/O prefetchingp. *IEEE Trans. Parallel and Distributed Systems*, 15(4):362–377, Apr. 2004.