Correlation-Based Load Balancing for Network Intrusion Detection and Prevention Systems

Anh Le
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, N2L 3G1, Canada
a4le@uwaterloo.ca

Raouf Boutaba
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, N2L 3G1, Canada
rboutaba@uwaterloo.ca

Ehab Al-Shaer
School of Computer Science, Telecommunications and Information Systems
DePaul University
Chicago, IL 60604, USA
ehab@cs.depaul.edu

ABSTRACT

In large-scale enterprise networks, multiple network intrusion detection and prevention systems are used to provide high quality protections. In this context, keeping load evenly distributed among the systems is crucial. This is because even load distributions provide protection to the networks and improve the networks' quality of service.

A challenging problem, however, is to maintain the load balancing of the systems while minimizing the loss of correlation information due to distributing traffic. Since anomaly-based detection and prevention of some intrusions, such as distributed denial of service (DDoS) attacks and port scans, require a single system to analyze correlated flows of the attacks, this loss of correlation information might severely affect the accuracy of the detections and preventions.

In this paper, we address this challenging problem by first formalizing the load balancing problem as an optimization problem, considering both the systems' load variance and the correlation information loss. We then present our Benefit-based Load Balancing (BLB) algorithm as a solution to the optimization problem.

We have implemented a prototype load-balancer which uses the BLB algorithm. We evaluated the load-balancer against various port scans and DDoS attacks. The evaluation results show that our load-balancer significantly improves the detection accuracy of these attacks while keeping the systems' load close within a desired bound.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General[Security and Protection]

General Terms

Security, Design

SecureComm 2008, September 22-25, 2008, Istanbul, Turkey
Copyright 2008 ACM 978-1-60558-241-2 ...$5.00.

Keywords

Intrusion detection, intrusion prevention, load balancing

1. INTRODUCTION

Nowadays as people rely heavily on computer systems to conduct businesses and operate mission critical devices, effects of viruses and worms are much more disastrous. One way to combat the spread of viruses and worms is by using intrusion detection and prevention systems. Intrusion detection systems detect unauthorized uses and malicious activities on resources like computer hosts and networks. Likewise, intrusion prevention systems extend this by providing real-time protection of the resources, thus they are capable of preventing the intrusions in real-time.

In this paper, we are interested in network intrusion detection and prevention systems (NIDPSs), which can both detect and prevent intrusions. An NIDPS is usually placed at an edge of a network, between its internal network and external network. The NIDPS monitors all packets which are coming in from the external network and going out of the internal network to detect and prevent intrusions.

Since network traffic speed and volume are increasing with an exponential rate [22], and NIDPSs are becoming more complex, a critical problem with using a single NIDPS in a network is that it could be easily overloaded. When overloaded, the NIDPS becomes a bottleneck of the network. The consequence is that packets going in and out of the network suffer long delays; eventually, the NIDPS has to drop some packets. Dropping packets compromises the security offered by the NIDPS because some intrusions cannot be detected if their related packets are dropped. For example, flow-based analyses, which are used by most of the NIDPSs, require that packets belonging to the same flow to be examined by the same NIDPS.

Using clusters of NIDPSs offers the most affordable and scalable solution to the above problem [22, 25]. When a cluster of NIDPSs is used in a network, keeping the load evenly distributed among the NIDPSs is crucial. This is because an even load distribution provides protection to the network – the network will be less likely to have an overloaded system. Moreover, this even load distribution allows for better traffic engineering which improves the network's quality of service.

A challenging problem with using clusters of NIDPSs, however, is to maintain the load balancing of the systems while minimizing the loss of correlation information due to distributing traffic. Because anomaly-based detection and

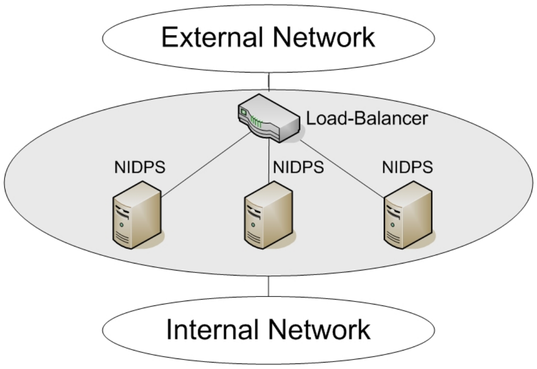

Figure 1: Placement of the NIDPS Load-Balancer

prevention of some intrusions, such as distributed denial of service (DDoS) attacks and port scans, require a single system to analyze correlated flows of the attacks, this loss of correlation information might severely affect the accuracy of the detection and prevention.

In this paper, we propose a novel approach to distribute traffic to the NIDPSs. We first formalize the load balancing problem as an optimization problem, considering both the systems' load variance and the correlation information loss. We then present our Benefit-based Load Balancing (BLB) algorithm as a solution to the optimization problem. This algorithm uses a novel on-line clustering technique to distribute flows in real-time to achieve the following:

- Correlated flows are grouped together at a single NIDPS to reduce the correlation information loss.
- NIDPSs' load are kept close within a specified bound.

The correlation information considered in our study is the one which can be extracted from the five tuples: source IP address (*src_ip*), destination IP address (*dst_ip*), source port (*src_port*), destination port (*dst_port*), and protocol (*proto*). Because extracting and analyzing packets' payload are expensive in terms of processing time, they are not supported by our load-balancer which wishes to distribute the traffic in real-time. As a result, correlation information derived from the packets' payload are out of the scope of this study.

We have implemented a prototype load-balancer which uses the BLB algorithm. We also evaluated the load-balancer against various port scans and DDoS attacks. The evaluation results show that our load-balancer significantly improves the detection accuracy of those attacks while keeping the load of the systems close within a desired bound. Figure 1 shows how our load-balancer fits into a network topology.

1.1 Contribution

Our main contribution in this paper is the design of a load-balancer which delivers the following features:

- *Anomaly-based Detection and Prevention Support.* Our load-balancer is capable of grouping correlated flows in real-time. This greatly supports the anomaly-based detections and preventions of attacks like port scans and DDoS attacks. Additionally, our load-balancer preserves flows, i.e. packets belonging to the same flow are examined by the same NIDPS. Thus it fully supports *flow-based analyses.*
- *Fine-grained Load Balancing.* Our load-balancer monitors load of the NIDPSs and distributes the traffic in a way such that the systems' load are kept close within a specified bound. This provides both protection and better traffic engineering to the network.
- *Configurable Security.* With our load-balancer, one might favor the security, i.e. the low correlation information loss, over the performance, i.e. the load balancing, when it is desirable to do so. For example, when the load of the whole system is low, one might want to use only one NIDPS to analyze all the traffic instead of distributing the traffic across multiple systems. Our load-balancer provides several means to favor security: (1) relaxing the variance constraint, (2) duplicating traffic to send to multiple NIDPSs, and (3) operating with threshold-based constraint instead of load-balancing-based constraint.

The rest of this paper is organized as follows: Section 2 describes the problem statement and our approach overview. In Section 3 we formalize the flow assignment problem as an optimization problem and provide an approximation for it. Section 4 explains the on-line clustering technique and describes the BLB algorithm. Section 5 discusses the correlations between flows. In Section 6 and 7 we describe the implementation and evaluation results respectively. Section 8 presents related work. Finally, we conclude in Section 9.

2. PROBLEM STATEMENT AND APPROACH OVERVIEW

2.1 Problem Statement

Given a cluster of NIDPSs, we want to develop a load-balancer which provides a desired level of load balancing, i.e. keeps the load of the NIDPSs close within a specified bound, and minimizes the correlation information loss due to distributing flows, which in turn improves the detection and prevention accuracy of the NIDPSs.

2.2 Approach Overview

The intuition of our approach is as follows: Since each NIDPS only receives a portion of the network traffic, we want to make sure that this portion contains sufficient information for the NIDPS to detect and prevent intrusions. In particular, we want to send attack-correlated flows to the same NIDPS, so that no attack would be missed.

First, we introduce *clusters* to structure the flows. A cluster contains flows which are closely correlated to each other. Clusters are constructed and deleted on-the-fly depending on both the variety and the rate of the traffic's flows. When a new flow comes, it can join some existing clusters or form a new cluster of its own. We discuss about the on-line cluster management in detail in Section 4. Also, an NIDPS could contain several clusters of flows. This means that an NIDPS could be monitoring multiple groups of correlated flows at the same time to detect and prevent possible intrusions.

Next, the notion of *benefits* is introduced as a means to measure the correlations between a new flow and groups of previously assigned flows, or clusters. The correlations between flows are derived from their five tuples: *src_ip*, *dst_ip*, etc. We discuss about the correlations in more detail in Section 5. Benefits play a very important role in our approach since we base on them to assign new flows. For example, if there are two existing clusters and a new flow comes, we will assign this new flow to the cluster which gives a better benefit. In other words, the new flow is assigned to the cluster which is more correlated to it.

Following, the load balancing is achieved by closely monitoring the load of all the NIDPSs and assigning flows to them correspondingly. A load balancing level is described using a *variance*. Specifically, a small value of variance indicates a high level of load balancing and vice versa.

At last, we summarize our approach as follows: Flows in NIDPSs are organized as clusters, and a desired level of load balancing is specified as a variance constraint. When a new flow comes, we find candidate NIDPSs which satisfy the variance constraint. Then, among clusters of these NIDPSs, we assign the new flow to the ones which give the best benefits. By assigning flows this way, we achieve the highest amount of correlation information possible while keeping the NIDPSs' load close within a bound.

3. PROBLEM FORMALIZATION

In this section, we first describe how the problem of assigning new flows is formalized as an optimization problem. We then show that the problem is NP-hard, thus it cannot be solved in polynomial time. We subsequently present an approximation for the optimization problem. Finally, we discuss how our formalization could be fine-tuned to favor security over performance when it is needed.

3.1 Flow Assignment Optimization Problem

Here we formalize the problem of assigning new flows as an optimization problem. At time t, let n be the number of NIDPSs and m be the number of clusters. The mapping between the NIDPSs and the clusters is one-to-many. For each NIDPS i ($i \in [1, n]$), let $\overrightarrow{G_i}$ be a vector of size m whose j^{th} element ($j \in [1, m]$) is either 1 if NIDPS i owns cluster j or 0 otherwise.

Now let f be the new flow. Assigning f to a cluster j gives a benefit B_j. Essentially, this benefit reflects how much f and the flows in cluster j are correlated. Let $\overrightarrow{B}$ be a vector of size m whose j^{th} element is the benefit B_j.

Next, let L_i denote the current load of NIDPS i in the system. An NIDPS's load is expressed as the amount of traffic it can handle per second (Mbps). We estimate the load by taking periodic samples of the traffic going to each NIDPS and apply the standard Single Exponential Moving Average (SEMA) [18] to the samples to alleviate the negative effect of the traffic's spikes. This linear aggregation method was shown to perform very well in the context of NIDPS load balancing [2].

We note that representing an NIDPS's load is a nontrivial task since there are numerous hardware and software resources an NIDPS might have, for example, CPU, disk, memory, etc. Therefore, one might rightfully challenge our load representation; however, the representation we use is the most common approach in the context of network intrusion detection and prevention [22, 2, 28].

Following, let μ be the average load of all the NIDPSs and V be the upper bound for the variance after the assignment. Subsequently, let L_f be the predicted load of the new flow. Here, SEMA is utilized on sampling flows to make predictions for incoming TCP and UCP flows separately.

Maximize:

(1) $\overrightarrow{X} \cdot \overrightarrow{B}$

Constraints:

(2) $\overrightarrow{X} \cdot \overrightarrow{I} \leq F$

(3) $\overrightarrow{X} \cdot \overrightarrow{G_i} \leq 1, \forall i \in [1, n]$

(4) $\frac{1}{n} \sum_{i=1}^{n} \Big[(L_i + L_f(\overrightarrow{X} \cdot \overrightarrow{G_i}) - (\mu + L_f \frac{\overrightarrow{X} \cdot \overrightarrow{I}}{n}))\Big]^2 \leq V$

Where:

$\overrightarrow{X}$	: Solution vector of size m
$\overrightarrow{B}$	: Benefit vector of size m
$\overrightarrow{G_i}$	: Cluster-ownership vector of size m of NIDPS i
$\overrightarrow{I}$	: Vector of 1's of size m
F	: Maximum number of NIDPSs to assign f
L_i	: Load of NIDPS i
μ	: Average load of all NIDPSs
L_f	: Predicted load of f
V	: Upper bound for the new variance

Figure 2: Flow Assignment Optimization Problem

Afterward, let F be the maximum number of NIDPSs which f could be assigned to concurrently. Assigning f to multiple NIDPSs could give more benefit because f might be correlated to multiple clusters maintained by different NIDPSs. For example, assume that there are two clusters of two different NIDPSs monitoring flows with *dst_ip* 10.0.0.1 and flows with *dst_port* 80 respectively. If f has both *dst_ip* 10.0.0.1 and *dst_port* 80 then it is desirable to assign this flow to both of the NIDPSs.

Finally, let $\overrightarrow{X}$ be the solution vector of size m. The j^{th} element of $\overrightarrow{X}$ is either 1 if f is going to be assigned to cluster j or 0 otherwise. In order to determine which clusters to assign f to, we have to solve the Flow Assignment Optimization Problem (FAOP) specified in Figure 2.

Our optimization problem is a Non-linear Binary Integer Programming problem. Expression (1) states that we want to maximize the total benefit. Constraint (2) requires that f could be concurrently assigned to at most F NIDPSs. Constraint (3) requires that f could be assigned to at most one cluster of each NIDPS. Finally, constraint (4) requires that the variance of the NIDPSs' load after the assignment must be less than or equal to the desired variance V. A small value of V means a high level of load balancing is expected while a high value of V indicates otherwise.

For instance, if V is set at 9 ($\%^2$), and load are assumed to be normally distributed among the NIDPSs, then 99.73% of the NIDPSs will have their load within $3\sqrt{V} = 9$ (%) of the average load μ, or within 18 (%) of each other.

3.2 Heuristic Flow Assignment Algorithm

The FAOP is very hard to solve since its decision version (D-FAOP) is an NP-complete problem. The D-FAOP asks if there is a solution vector $\overrightarrow{X}$ which gives a benefit better than a predefined benefit. The sketch of the proof of NP-completeness of the D-FAOP is provided below:

Proof. Given any solution, the solution can be verified in polynomial time. All one has to do is to calculate the sum of the benefits and the new variance. Both of these

Algorithm 1 HeuristicFlowAssignment(f)

```
 1: $solution\_set = \emptyset$
 2: $nidps\_set$ = all NIDPSs
 3: for i from 1 to F do
 4:    $cluster\_set$ = all clusters of $nidps\_set$
 5:    find $cluster$ in $cluster\_set$
 -        which satisfies variance constraint
 -        and has the biggest benefit
 6:    if no $cluster$ found then
 7:       quit for loop
 8:    end if
 9:    $solution\_set = solution\_set \cup cluster$
10:    $nidps$ = NIDPS which has $cluster$
11:    update load of $nidps$
12:    $nidps\_set = nidps\_set \setminus nidps$
13: end for
14: return $solution\_set$
```

take $O(m)$ time, where m is the number of the clusters. Thus the D-FAOP is in NP.

Now if we relax constraint (4) of the FAOP by replacing it with this simpler constraint:

$$\sum_{i=1}^{n} L_i(\overrightarrow{X} \cdot \overrightarrow{G_i}) \leq V \,,$$

then our problem is identical to the well-known *0-1 Knapsack Problem*. In this case, the NIDPSs are selected instead of items. For each NIDPS, there exists a cluster giving the best benefit, and this benefit represents the item's value. The NIDPSs' load represent the items' weights, and V represents the maximum weight of the knapsack. Since the decision version of the 0-1 Knapsack Problem is known to be NP-hard, the D-FAOP is also NP-hard.

Finally, the D-FAOP is in NP and is also NP-hard, thus it is NP-complete. □

Due to the real-time requirement of the flow assignment, we propose a greedy-based approximation algorithm, Heuristic Flow Assignment (HFA) algorithm, to solve the FAOP in linear time. The HFA algorithm is detailed in Algorithm 1. This algorithm searches for a cluster giving the maximum benefit and satisfying the constraints at a time. This can be done in $O(m)$ time, and it tries to do this up to F times (Line 3–13). We also note that when F equals 1, the result of the HFA algorithm is the optimal solution to the FAOP.

3.3 Configurable Security

When it is desirable to favor the security, i.e. the low correlation information loss, over the performance, i.e. load balancing, our formalization provides three possible configurations:

1. *Relaxing variance constraint :* Setting V high loosens the load balancing requirement, thus a higher benefit might be achieved. To the extreme, V could be set high enough so that the load balancing is completely ignored. In this case, the traffic is distributed based on solely the benefit, resulting in the use of only one NIDPS. This might be a desirable setting when the traffic's load is low.

2. *Duplicating flows :* A high value of F reduces the loss of correlation information because flows are duplicated up to F times to be sent to the NIDPSs. However, the duplication of flows consumes system resources like bandwidth and CPU load; therefore, it must be used selectively.

3. *Threshold-based load distribution :* The load balancing requirement could be replaced by a threshold-based requirement, which requires the NIDPSs' load to be kept below a certain threshold T_{load}. This requirement is easier to satisfy and gives more room to obtain higher benefits. Threshold-based load distribution could be readily achieved by replacing constraint (4) with a simpler constraint:

$$(4^*) \quad L_i + L_f(\overrightarrow{X} \cdot \overrightarrow{G_i}) < T_{load} \quad ,\forall i \in [1, n]\,.$$

In summary, F and V could be set high, along with using the threshold-based load distribution, to reduce the loss of correlation information and ultimately increase the detection and prevention accuracy. However, because these configurations compromise the performance, they should be used selectively depending on the current traffic's load, system resources, and performance requirements.

4. ON-LINE CLUSTERING TECHNIQUE

Managing clusters is a central activity of our load-balancer. Because of the real-time requirement, it is not possible to use the traditional clustering techniques like K-Means [9] or K-Medoids [9] for the management. Thus we have customized an on-line clustering technique introduced by Aggarwal et al. [1] to create a suitable one for our load-balancer. Specifically, we have integrated into the existing technique several new concepts: cluster weight, decay of weight, and benefit.

4.1 Cluster Weight and Decay of Weight

Each cluster has a weight whose value is between 0 and 1 inclusive. The weight of a cluster reflects both the number of flows the cluster has and the distances of the flows to the cluster *centroid* – a flow representing the cluster. At any time t, the weight of cluster j is calculated as follow:

$$W_{j,t} = \lambda^{(t_j - t)} W_{j,t_j} \,,$$

where $\lambda > 1$ is the *decaying factor*, and t_j is the last time when a new flow or a new packet gets assigned to this cluster.

Adding a new flow f to an existing cluster j changes t_j and the cluster's weight. Let s_j be the number of flows cluster j already has. Let $D(f, c_j)$ be the logical distance between f and the centroid c_j, where the logical distance between two flows is a value reflecting how much correlated the two flows are (the logical distance notion is discussed in detail in Section 5). If f is added to cluster j at time t then the cluster's weight is:

$$W_{j,t} = \lambda^{(t_j - t)} W_{j,t_j} + (1 - \lambda^{(t_j - t)} W_{j,t_j}) \frac{1 - D(f, c_j)}{s_j + 1} \,.$$

The smaller the distance between f and c_j is, i.e. smaller $D(f, c_j)$, the larger weight it adds to the cluster, i.e. larger $1 - D(f, c_j)$. In addition, when f is further away from c_j,

Algorithm 2 Benefit-based Load Balancing Algorithm

```
1:  while new flow $f$ do
2:     $C$ = HeuristicFlowAssignment($f$)
3:     if $C = \emptyset$ or Benefit(C) $< T_{benefit}$ then
4:        create a cluster (centroid $f$, weight 1)
5:        assign it to the lowest load NIDPS
6:     else
7:        assign $f$ to the clusters in $C$
8:        update those clusters
9:     end if
10: end while
```

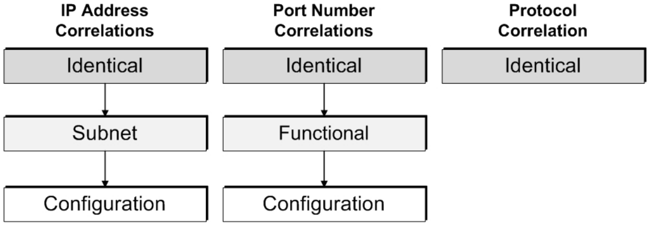

Figure 3: Matching Order of Correlations given by IP Addresses, Port Numbers, and Protocols

i.e. $1 - D(f, c_j) \sim 0$, the weight it adds to the cluster is negligible. This formula also guarantees that the weight of any cluster is never bigger than 1.

4.2 Benefit-based Load Balancing Algorithm

The Benefit-based Load Balancing algorithm, which is used to distribute traffic intelligently to the NIDPSs, is at the heart of this paper. This algorithm tights together all the details presented in this paper. We summarize the BLB algorithm in Algorithm 2 and describe it below.

At time t, when there is a new flow f, benefits of adding f to existing clusters are needed to solve the FAOP (Line 2). Benefit of adding f to cluster j is calculated as follow:

$$B_j = (1 - D(f, c_j))W_{j,t} \,.$$

This formula expresses that the closer f is to c_j, the higher benefit the assignment gives because of the larger $1-D(f, c_j)$, and the heavier the cluster j is, the higher benefit the assignment gives because of the larger $W_{j,t}$. Consequently, the assignment giving the highest benefit would give the highest possible amount of correlation, taking into consideration both of the logical distances and the weights.

If the FAOP has no solution, or this solution gives a benefit below a predefined threshold $T_{benefit}$, then a new cluster, whose centroid is f and weight equals 1, is created. This cluster is added to the NIDPS with the lowest load (Line 3–5). In the other case, f is added to the corresponding clusters and these clusters are updated appropriately (Line 6–8).

The clusters' weights are closely monitored by a separate thread. This thread periodically delete old clusters which have weights less than a threshold T_{weight}. The deletion of old clusters, together with the creation of new ones, occurs in real-time to assure that the existing clusters are representing the most current traffic.

5. FLOWS CORRELATIONS

The correlations between flows are very essential to our benefit calculation. In this section, we explain how the correlations are measured by a *logical distance.* We first present a general formula for the logical distance. Afterward, we thoroughly describe the components of the formula.

5.1 Logical Distance Formula

Given two flows f_1 and f_2, the logical distance between the two flows, which indicates how closely correlated they are, is formally defined as follows:

$$D(f_1, f_2) = \sum_{\forall i \in \mathbf{F}} \alpha_i \delta_i(f_1, f_2) \,,$$

where **F** is either ip, port, or protocol; and α's are the weights of the fields.

Weights of the fields vary depending on the attack scenarios. For example, in a DDoS attack scenario where an attacker uses many hosts to attack many services of a single victim, α_{ip} should be bigger because it is desirable to group together flows having the same destination IP address. On the other hand, in a port sweep attack scenario where an attacker uses many hosts to scan a specific port of many victims, α_{port} should be bigger. Generally, $\alpha_{protocol}$ is always the smallest one because it is very common to have flows with the same protocol, like TCP or UDP.

5.2 Logical Distance Components

5.2.1 Logical Distance by IP Addresses – $\delta_{ip}()$

We match the correlation given by addresses of two flows with the following correlations: identical, subnet, and configuration correlations. $\delta_{ip}()$ returns a value between 0 and 1 corresponding to the matching. The correlations are defined as follows:

- *Identical Correlation :* If source IP addresses or destination IP addresses of two flows are identical then their correlation matches the identical correlation.

  This correlation is the most important correlation between two flows. For example, in a DDoS attack scenario, numerous source IP addresses might be used for the attack [15]. However, computers corresponding to those source IP addresses attack the same target. Because flows of the attack all have the same destination IP address, they have the identical correlation.

- *Subnet Correlation :* If destination IP addresses of two flows belong to the same subnet or vlan, then their correlation matches the subnet correlation.

  In practice, attackers often try to find vulnerabilities in different computers in a target network, thus attack-correlated flows are sent to the same network. Identifying this type of correlation helps to group these flows together to detect and prevent the intrusions.

- *Configuration Correlation :* If source IP addresses or destination IP addresses of two flows belong to a predefined set of addresses then their correlation matches the configuration correlation.

Sometimes, it is of interest to group flows going to the unused address space together to detect suspicious activities. This correlation allows the grouping of these flows in particular, and the grouping of flows of any interested set of addresses in general.

Figure 3 shows the order in which the matching is done. Going from top to bottom, the significances of the correlations decrease, so the values returned by $\delta_{ip}()$ increase.

5.2.2 Logical Distance by Port Numbers – $\delta_{port}()$

Because destination port numbers represent target services, they play a more important role than source port numbers. As a result, we concentrate on investigating the correlations between destination port numbers instead of source port numbers. Similar to the IP address case, in order to determine a value between 0 and 1 which $\delta_{port}()$ returns, we match the correlation between two destination port numbers with one of the following correlations:

- *Identical Correlation :* If destination port numbers of two flows are identical then their correlation matches the identical correlation.

  This correlation supports the detection and prevention of intrusions targeting a particular service provided by a number of computers. For example, flows belonging to an attack aiming at multiple web servers all have 80 as their destination port number.

- *Functional Correlation :* If destination port numbers of two flows are functionally correlated then their correlation matches the functional correlation.

  For example, flows belonging to an FTP connection have both destination port numbers 20 and 21. Thus it is desirable to group these flows together.

- *Configuration Correlation :* If destination port numbers of two flows belong to a predefined set of port numbers then their correlation matches the configuration correlation.

  In practice, the administrators might want to group together flows belonging to different services, for instance, telnet and web, to detect certain attacks. This correlation enables them to do so.

5.2.3 Logical Distance by Protocol – $\delta_{protocol}()$

Either 0 or 1 is returned by $\delta_{protocol}()$, depending on the following correlation:

- *Identical Correlation :* If protocols of two flows are the same then they have the identical correlation.

6. IMPLEMENTATION

We implemented a prototype load-balancer. Moreover, in order to evaluate the load-balancer, we implemented a DDoS detector. We describe our implementations in detail below.

6.1 Load-balancer

The prototype load-balancer was developed using *libpcap* library [11] – a library for capturing and sending network packets directly from and to network interfaces in real-time. The BLB algorithm was implemented as the default load balancing algorithm. The identical correlations given by addresses, port numbers, and protocol were initially supported.

Besides the BLB algorithm, for comparison purposes, we also integrated a *hash-based* algorithm into our load-balancer. Various hash-based algorithms were used by others [25, 22, 28] to distribute traffic, and they all shared a common property: applying a simple hash function on a subset of the five tuples. So we implemented the hash-based algorithm using a simple additive hash: (*src_ip* + *dst_ip* + *src_port* + *dst_port*) mod n, where n is the number of NIDPSs.

For the simulations, our load-balancer was run on a system with Intel Dual Core 2.0 GHz CPU, 2 GB RAM, 2×1 Gbps NICs. Furthermore, the load-balancer operates as follows: first it gets a new packet from a specified source, which could be a network interface or a trace file; it then executes either the BLB or hash-based algorithm to identify which NIDPS(s) to send this packet to; finally, it sends the packet to the corresponding network interface(s) or writes the packet to the corresponding trace file(s).

6.2 DDoS Detector

For evaluation purposes, a DDoS detector was developed using the Cumulative Sum algorithm – a simple and robust algorithm to detect DDoS proposed by T. Peng et al. [20]. Fundamentally, this algorithm detects the change of the mean value of the percentage of the new source IP addresses overtime. A sequence $\{Y_n\}$ is used to characterize the change. If at any time, a value of $\{Y_n\}$ is bigger than a predefined threshold T_y, then an attack is detected.

7. EVALUATION

In this section, we present our evaluation results. We have evaluated both the performance and the security of our load-balancer using simulations with real traffic traces. We describe them below in the corresponding order.

7.1 Performance

In order to evaluate how well our load-balancer distributes the traffic, we needed high volume traffic traces. Consequently, we chose two weeks GPS-synchronized IP header traces, which were captured in December 2003 at the University of Auckland by the National Laboratory for Applied Network Research (NLANR) [17].

We note that because of privacy issues, the traces were sanitized by NLANR. The IP addresses were mapped into the network space 10.X.X.X in a non-reversible way. However, the mapping was one-to-one, which meant IP addresses identical in the traces were identical in the real world. Thus we could still identify the identical correlation given by IP addresses. Identical correlations given by port numbers and protocols were unaffected by this sanitization.

The trace used in both of the below simulations was an hour trace captured from 12:00 to 13:00 on Tuesday, December 2nd, 2003. This hour was one of the busiest hours of the network. During this hour, there were 200 new flows per second on average [23].

7.1.1 Effect of the Number of Clusters on the System Overhead

Here we examine the effect of the number of clusters on the system overhead. The number of clusters maintained in the load-balancer is dependent on the threshold T_{weight} and the decay rate λ. In this simulation, λ was fixed at 1.1, and

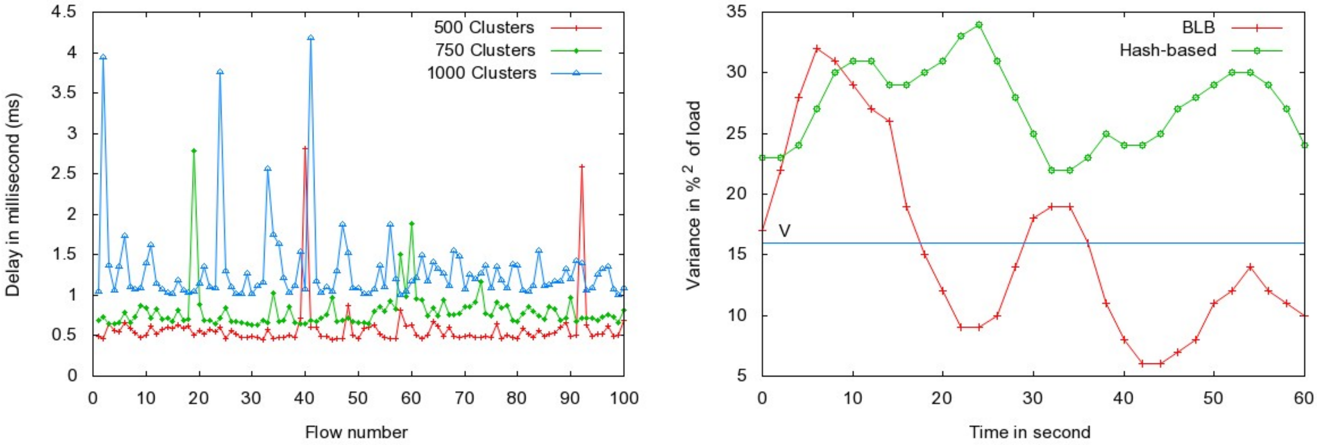

Figure 4: Effect of the Number of Clusters on the System Overhead (left), and Effect of the Algorithms on the Variance (right)

T_{weight} was varied to get the desired number of maintained clusters. There were 10 NIDPSs, V was 25, F was 1, and the load-balancer used the BLB algorithm.

For each flow, when its first packet arrives at the load-balancer, the load-balancer has to perform a calculation to determine which cluster(s) to assign the flow. This is the primary system overhead associated with the flow. We measure this overhead by the delay introduced to the first packet of the flow, which is mostly the time to run the BLB algorithm.

Figure 4 (left) plots the delays of the first packets of 100 consecutive sampling flows when the system maintained 500, 750, and 1000 clusters. The average delays (not shown in the figure) were 0.58, 0.80, and 1.31 milliseconds respectively. This result indicates that the higher the number of clusters is, the higher the delays are. This is because it takes more time to calculate the benefits when there are more clusters. Nevertheless, the delay per flow introduced by our system is in the order of millisecond, which is further divided by the number of packets per flow to get the average delay per packet. Hence, even when the system has to maintain 1000 clusters, the average overhead per packet is tolerable.

7.1.2 Effect of the Algorithms on the Variance

We carried out this experiment to examine how different algorithms affect the NIDPSs' load variance. In this experiment, the load-balancer maintained 750 clusters and used both the BLB and the hash-based algorithms. The variance upper bound V was set at 16, and F was set at 1. We note that because the trace contained only packet headers, we used uniformly random generated data as packet payloads. Nevertheless, we still preserved the overall traffic rate.

Figure 4 (right) plots the variances associated with the two algorithms at every 2 seconds during a sampling of 60 seconds. We observed that the variances of the hash-based algorithm were noticeably higher than those of the BLB algorithm. This indicated that the load of the NIDPSs when the hash-based algorithm was used were substantially unbalanced. Also, our BLB algorithm very often had the variances less than V. The exceptions were points of time at which there was no solution to the FAOP. These points could be due to the traffic's spikes.

In summary, the result of this simulation showed that our BLB algorithm had a solid performance in terms of keeping the variances low in comparison to the hash-based algorithm. Most importantly, the BLB algorithm was very often able to keep the variance below the specified upper bound V.

7.2 Security

Three simulations were conducted to evaluate how the BLB algorithm supports the detections of DDoS attacks and port scans comparing to the hash-based algorithm. In the simulations, the internal (protected) network was a Class B network, and 10% of the address space was occupied. For the BLB algorithm, the load-balancer maintained 500 clusters, V was set at 25, and F was set at 1. Finally, the load-balancer was used to distribute traffic to 10 NIDPSs.

7.2.1 DDoS Attack

We simulated a large scale UDP flood attack, which involved 9000 distinct attacking hosts and a victim. Each UDP packet was of fixed size 1 KB, and its source port and destination port were randomly selected. The simulation lasted 60 seconds, during which there were both background traffic and the attack traffic. The background traffic was of 15 Mbps with 100 flows per second on average, and 80% of the flows was from or to 20% of the machines in the internal network. The attack started at second 20 and lasted for 30 seconds. During the attack, the victim saw about 300 new source IP addresses of the attack traffic per second.

In figure 5, the left graph plots the highest fractions of new source IP addresses observable by one of the NIDPSs in the following 3 settings: (1) a single NIDPS with no load-balancer, (2) 10 NIDPSs with a BLB load-balancer, and (3) 10 NIDPSs with a hash-based load-balancer. The corresponding values of Y_n are shown in the right graph.

It could be observed that during the attack, when the BLB algorithm was used, the fractions of new source IP addresses observable by one of the NIDPSs were significantly higher than those when the hash-based algorithm was used. This was because a much higher number of attack flows went to the same NIDPS when the BLB was used. The higher fractions over time resulted in the higher values of Y_n. Thus there would be scenarios when the hash-based algorithm fails to detect the attack but the BLB algorithm succeeds.

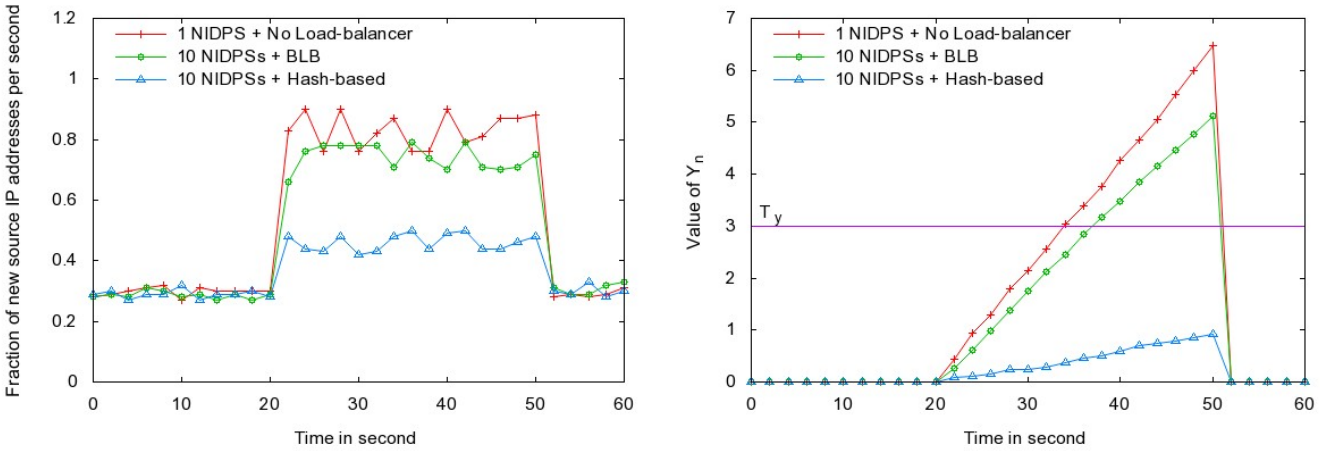

Figure 5: Effect of the Algorithms on the Fraction of New Source IP Addresses per Second (left), and on the Value of Y_n (right)

For example, if the threshold T_y was set at 3 then the hash-based algorithm would fail to detect the DDoS attack. This is because Y_n was always below T_y. However, the BLB algorithm detected the attack at second 38, which was 4 seconds later than when a single NIDPS with no load-balancer was used. We note that when a single NIDPS was used, we assumed that it could handle all the traffic without dropping packets. In this case, the attack was detected earlier because all flows of the attack went to this NIDPS.

In conclusion, this evaluation showed that the BLB algorithm distributed the traffic in a way which increased the detection accuracy of the DDoS attack significantly.

7.2.2 Port Scans

For this part, we used Snort [21] as our scan detector. From version 2.6, the Snort preprocessor *sfPortscan* takes care of detecting port scans. By analyzing the anomaly of the traffic, sfPortscan can detect the following scans [3]:

- *Portscan.* A small number of scanning hosts, scanning one victim, for a lot of ports.
- *Portsweep.* A small number of scanning hosts, scanning many victims, for a small number of ports.
- Also, *decoy portscan* and *distributed portscan.*

Because complex scans such as decoy or distributed portscans currently have low detection accuracy (high false positive rate) [3], we focus on evaluating how our load-balancer supports the detection of portscans and portsweeps.

For both of the following simulations, the background traffic was the same as the one in the previous DDoS simulation. Similarly, when a single NIDPS with no load-balancer was used, we assumed that no packets were dropped. Also, the sensitivity level of sfPortscan was set at medium, and TCP SYN scans were used. Using this scan type, SYN packets were sent from scanning hosts to victims, trying to establish TCP connections. If SYN/ACK packets were received, then the corresponding ports were open. Lastly, the scans were designed so that a single NIDPS with no load-balancer always successfully detected them.

First we used nmap [16] to carry out a portscan. We used one host to scan ports 1–1000 of one victim. When a single NIDPS was used, it detected this portscan. When there were 10 NIDPSs and the hash-based algorithm was used, no attack was detected; however, when the BLB algorithm was used, the attack was detected.

When the hash-based algorithm was used, the highest number of SYN packets observable by one NIDPS was about 100. This was not enough for sfPortscan to trigger a portscan alert. However, when the BLB algorithm was used, there existed an NIDPS which observed up to 700 SYN packets, thus triggering a portscan alert. These numbers are plotted in figure 6 (left).

Secondly, we used nmap to carry out a portsweep. We used one host to scan port 80 of 100 victims. When a single NIDPS with no load-balancer was used, it detected this portsweep. When there were 10 NIDPSs and the hash-based algorithm was used, no attack was detected. In this case, we noticed that each NIDPS observed about 10 SYN packets, targeting port 80 of 10 different victims. This number was not high enough for sfPortscan to trigger a portsweep alert. On the other hand, when the BLB algorithm was used, there existed an NIDPS which observed as much as 80 SYN packets, thus triggering a portsweep alert. These numbers are plotted in figure 6 (right).

One might argue that the sfPortscan's mechanism to detect port scans is naive and simple, and that better mechanism should be used in our evaluation. Nonetheless, this simplicity represents what are popularly used today in terms of detecting port scans in specific, and in terms of anomaly-based detection in general. To the best of our knowledge, the mechanism used by two other popular operational NIDPSs: Bro [25, 19] and Cisco IPS [4], are similarly simple in terms of detecting port scans. As future work, we plan to evaluate our load-balancer with advanced port scan detection techniques, such as the one proposed by Jung et al. [8].

In summary, both the portscan and the portsweep went undetected when the hash-based algorithm was used to distribute the traffic. In contrast, when the BLB algorithm was used, the cluster of NIDPSs successfully detected both of them. Consequently, this showed that our load-balancer with the BLB algorithm could help to substantially improve the detection accuracy of portscans and portsweeps.

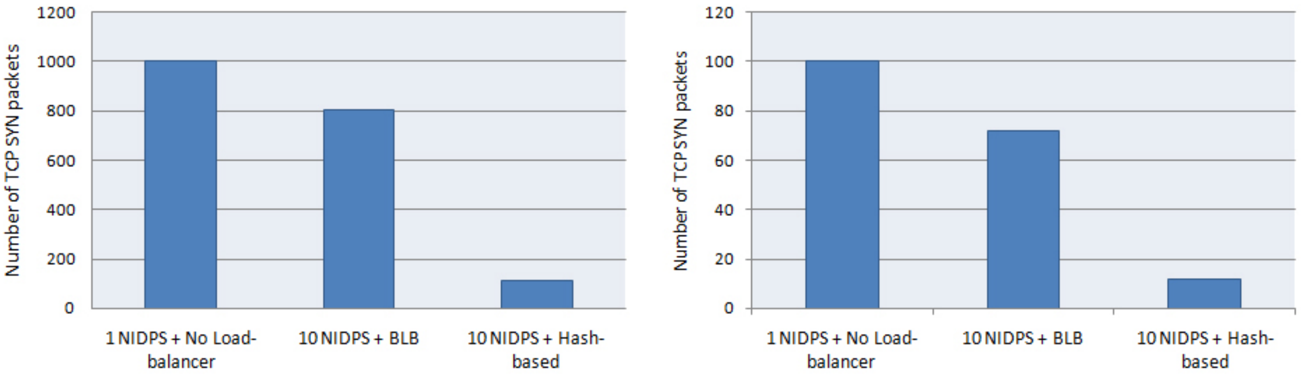

Figure 6: The highest number of SYN packets observable by one of the NIDPSs during the portscan attack (left) and the portsweep attack (right)

8. RELATED WORK

Hash-based Distribution with Communication

In a recent literature by Vallentin et al. [25], the authors presented a cluster of network intrusion detection systems (NIDSs) as a solution for realizing high-performance and stateful network intrusion detection on commodity hardware. This cluster uses an additive hash on the five tuples to distribute the flows. Also, an inter-NIDS communication scheme is introduced as a means to recover the correlation information loss to improve the cluster's detection accuracy.

While the above approach requires NIDSs with low level communication capabilities, such as Bro [19], an NIDS developed at UC Berkeley, our approach is independent of the NIDSs. As a result, it is applicable to clusters of any NIDSs. Moreover, to improve the detection accuracy, we leverage the correlations between flows when distributing them instead of introducing the communication a later time.

Hash-based Distribution with Multiple Hash Functions

In another literature, Schaelicke et al. [22] presented a load-balancer with a single hash table and multiple hash functions. During under-load conditions, one hash function is used on flows to hash them into buckets, which are previously assigned to the NIDSs. When an NIDS is overloaded, the overload condition is handled by reassigning the buckets of the overloaded NIDS to different NIDSes or by applying additional hash functions to assign the incoming flows to buckets of the other NIDSs.

Overall, this approach tries to remove the overload conditions when they occur but does not prevent them. Also, it does not consider the loss of correlation information due to distributing traffic. On the contrary, our approach prevents the overload conditions by load balancing the NIDSs; moreover, we take into account the loss of correlation information when distributing the traffic.

Slicing and Reassembling Mechanism

Kruegel et al. [10] proposed a slicing and reassembling mechanism to distribute the traffic. Packets' frames are scattered and then reassembled at groups of NIDSs that might need them to detect some attacks. An NIDS determines if it needs a packet based on the attack scenarios that it handles.

One major problem with the above approach is the unmanageable duplication of traffic due to correlated attack scenarios – attack scenarios that need some mutual traffic. Also, this approach does not provide load balancing and does not examine the correlations between flows.

In summary, none of the related work gives a satisfactory solution to balance the load of the NIDPSs. More importantly, none of them examines the correlations between flows in their distributions of traffic. Correlated flows are not optimally assigned to increase the accuracy of the intrusion detection and prevention. Our approach gives solutions to both of the above problems.

9. CONCLUSION

The problem of load balancing in the context of network intrusion detection and prevention is challenging. To address this problem, we first formalize the traffic distribution problem as an optimization problem. We then present a novel Benefit-based Load Balancing algorithm as a solution to it. This algorithm thoroughly considers both the load variation of the NIDPSs and the loss of correlation information due to distributing flows. As far as we know, none of the previous work has considered this loss of correlation information in their approaches. Our algorithm performs real-time optimization, thus it accommodates intrusion detection as well as intrusion prevention systems.

We have implemented a prototype load-balancer which uses the BLB algorithm. Our load-balancer distributes traffic flows in real-time and achieves the following properties: (1) correlated flows are grouped together at a single system to reduce the correlation information loss, and (2) load of the NIDPSs are kept close within a desired bound. While the former property greatly increases the accuracy of anomaly-based intrusion detections and preventions, the later property provides protection to the networks and allows for better traffic engineering.

Finally, through extensive simulations with real traffic traces and major attacks, we showed that our load-balancer with the BLB distribution algorithm could achieve high performance and provide enhanced security. In particular, it has low overhead and can keep the load variances below the desired levels of load balancing. Moreover, it significantly improves the accuracy of anomaly-based detections of DDoS attacks and port scans.

10. REFERENCES

[1] C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proc. of the 29th VLDB*, volume 29, pages 81–92, 2003.

[2] M. Andreolini, S. Casolari, M. Colajanni, and M. Marchetti. Dynamic load balancing for network intrusion detection systems based on distributed architectures. In *NCA '07: Proceedings of the Sixth IEEE Symposium on Network Computing and Applications*, pages 153–160, July 2007.

[3] B. Caswell, J. Beale, and A. Baker. *Snort IDS and IPS Toolkit*. Syngress, Mar. 2007.

[4] Cisco NIPS, http://tinyurl.com/2nbxbx.

[5] E. Cooke, F. Jahanian, and D. Mcpherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, pages 39–44, June 2005.

[6] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer. Operational experiences with high-volume network intrusion detection. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 2–11, New York, NY, USA, 2004. ACM.

[7] J. M. Gonzalez, V. Paxson, and N. Weaver. Shunting: a hardware/software architecture for flexible, high-performance network intrusion prevention. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 139–149, New York, NY, USA, 2007. ACM.

[8] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *SP '04: Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 211–225, May 2004.

[9] L. Kaufman and P. Rousseeuw. *Finding Groups in Data. An Introduction to Cluster Analysis*. Wiley, Mar. 1990.

[10] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer. Stateful intrusion detection for high-speed networks. In *Proc. of the IEEE SSP'02*, pages 285–293, 2002.

[11] Libpcap, http://www.tcpdump.org/.

[12] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman. Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation. In *DISCEX '00: Proceedings of DARPA Information Survivability Conference and Exposition, 2000*, pages 12–26, 2000.

[13] 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 *Recent Advances in Intrusion Detection*, pages 162–182, 2000.

[14] J. McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, 2000.

[15] J. Mirkovic and P. Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, Apr. 2004.

[16] Network Mapper, http://insecure.org/nmap/.

[17] Network traces, http://pma.nlanr.net/Special/auck8.html.

[18] NIST/SEMATECH e-Handbook of Statistical Methods, http://tinyurl.com/645fex.

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

[20] T. Peng, C. Leckie, , and R. Kotagiri. Proactively detecting distributed denial of service attacks using source ip address monitoring. In *Proc. of the Third International IFIP-TC6 Networking Conference*, pages 771–782, 2004.

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

[22] L. Schaelicke, K. Wheeler, and C. Freeland. SPANIDS: A scalable network intrusion detection loadbalancer. In *Proc. of the 2nd conference on Computing Frontiers*, pages 315–322, 2005.

[23] Statistics of the trace captured from 12:00 to 13:00, 12/02/2003, http://tinyurl.com/59yj46.

[24] Tcpreplay, http://tcpreplay.synfin.net/trac/.

[25] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney. The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. In *Proc. of the Symp. on RAID'07*, Queensland, Australia, Sept. 2007.

[26] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *WORM '03: Proceedings of the 2003 ACM workshop on Rapid malcode*, pages 11–18, New York, NY, USA, 2003. ACM.

[27] P. Wheeler and E. Fulp. Taxonomy of parallel techniques for intrusion detection. In *Proc. of ACM 45th Southeast Regional Conference*, pages 278–282, 2007.

[28] K. Xinidis, I. Charitakis, S. Antonatos, K. G. Anagnostakis, and E. P. Markatos. An active splitter architecture for intrusion detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, 3(1):31–44, Mar. 2006.