On Optimizing Traffic Distribution for Clusters of Network Intrusion Detection and Prevention Systems

by

Anh Le

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of

Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2008

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

To address the overload conditions caused by the increasing network traffic volume, recent literature in the network intrusion detection and prevention field has proposed the use of clusters of network intrusion detection and prevention systems (NIDPSs). We observe that simple traffic distribution schemes are usually used for NIDPS clusters. These schemes have two major drawbacks: (1) the loss of correlation information caused by the traffic distribution because correlated flows are not sent to the same NIDPS and (2) the unbalanced loads of the NIDPSs. The first drawback severely affects the ability to detect intrusions that require analysis of correlated flows. The second drawback greatly increases the chance of overloading an NIDPS even when loads of the others are low.

In this thesis, we address these two drawbacks. In particular, we propose two novel traffic distribution systems: the Correlation-Based Load Balancer and the Correlation-Based Load Manager as two different solutions to the NIDPS traffic distribution problem. On the one hand, the Load Balancer and the Load Manager both consider the current loads of the NIDPSs while distributing traffic to provide fine-grained load balancing and dynamic load distribution, respectively. On the other hand, both systems take into account traffic correlation in their distributions, thereby significantly reducing the loss of correlation information during their distribution of traffic.

We have implemented prototypes of both systems and evaluated them using extensive simulations and real traffic traces. Overall, the evaluation results show that both systems have low overhead in terms of the delays introduced to the packets. More importantly, compared to the naive hash-based distribution, the Load Balancer significantly improves the anomaly-based detection accuracy of DDoS attacks and port scans – the two major attacks that require the analysis of correlated flows – meanwhile, the Load Manager successfully maintains the anomaly-based detection accuracy of these two major attacks of the NIDPSs.

Acknowledgements

I would like to express my deep gratitude to my supervisor, Professor Raouf Boutaba, for his support, guidance, and supervision. He has been very professional and understanding. His encouragement has motivated me tremendously to complete this thesis.

I am also deeply indebted to Professor Ehab Al-Shaer, from DePaul University, for his co-supervision, technical guidance, and professional advice throughout my research. His assistance has helped me enormously to advance this research.

My sincere thanks to Professor David Taylor and Professor Martin Karsten for serving as reviewers of this thesis and for their valuable comments.

Special thanks to my friends who supported me during my study at the University of Waterloo.

To my parents...

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Network Intrusion Detection and Prevention Systems

Nowadays, as people rely heavily on computer systems to conduct business and operate mission critical devices, effects of viruses and worms can easily be disastrous. One way to combat the spread of viruses and worms is to use intrusion detection systems.

Intrusion detection systems (IDSs) detect unauthorized use of and malicious activities on computer systems and networks. There are two major types of IDS: host-based IDS (HIDS) and network-based IDS (NIDS) [9]. While host-based IDSs detect intrusions by monitoring file system modifications, application execution logs, system calls, and so on, network-based IDSs detect intrusions by examining packets that travel on network links.

Compared to network-based IDSs, host-based IDSs have access to more refined resources, such as file system and system calls. On the other hand, network-based IDSs are able to detect intrusions at an earlier stage and they have global views of the networks. As a result, these two systems can complement each other to provide high quality detection.

Intrusion prevention systems (IPSs) extend the capabilities of IDSs by providing real-time protection to the resources. For instance, a network-based IPS is capable of dropping malicious packets while still allowing legitimate traffic to pass through and a host-based IPS can block suspicious accesses to certain files in real-time.

In this research, we are interested in network-based intrusion detection and

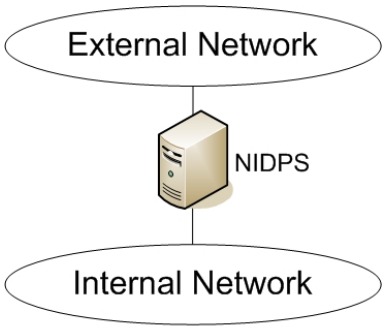

Figure 1.1: Placement of an NIDPS

prevention systems (NIDPSs) which can detect intrusions and prevent them when necessary. NIDPSs are usually placed at the edges of networks – between the internal and the external networks. NIDPSs monitor all packets coming in from the external networks and going out of the internal networks to detect and prevent intrusions. Figure 1.1 shows the placement of an NIDPS in a network. Some of the popular operational NIDPSs are Snort [34], Bro [31], and Cisco IPSs [7].

Based on the approach used for detection, NIDPSs are categorized into two classes: misuse and anomaly-based [9]. Misuse NIDPSs detect intrusions by matching the monitored traffic with previously known intrusive traffic patterns or attack signatures. As a result, misuse NIDPSs are also called signature-based NIDPSs. On the other hand, anomaly-based NIDPSs detect intrusions by detecting deviations of the monitored traffic from the normal traffic patterns or the baselines. These baselines are predefined by the administrators and can include traffic load, typical packet sizes, protocol distribution, and so on.

1.2 Clusters of NIDPSs

Since network traffic volume is increasing with an exponential rate [35], 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 some of the packets involved are dropped. For example, *flow-based analyses*, which are used by most of the NIDPSs, require that all packets belonging to a flow be examined by a single NIDPS.

In order to handle the increasingly high network traffic volume, there are two possible solutions:

1. Upgrading hardware and tuning software of the NIDPSs so that they can handle more traffic.
2. Using clusters of NIDPSs and distributing the traffic across the NIDPSs.

The first solution is costly and not scalable. Since network bandwidth is continuously increasing by a factor of 10 every 4 years [35], continuous upgrades are required for the NIDPSs to operate. This results in high adoption and maintenance costs. Moreover, tuning NIDPSs is a challenging task with many trade-offs [10]. The tuning also makes the NIDPSs overly complicated and the management tasks very difficult.

The second solution, using clusters of NIDPSs, is affordable and scalable. In fact, it has been adopted by many researchers [2, 18, 35, 40, 43]. This solution takes advantage of the availability of low cost computers. It uses several of them together to handle the high traffic load. Moreover, extending the system can be done by adding additional NIDPSs to the cluster. When using clusters of NIDPSs, however, the distribution of the network traffic to the NIDPSs plays a very important role; this is the main focus of our research.

1.3 Motivation

We observe that simple traffic distribution schemes are usually used for NIDPS clusters. These schemes apply simple hash functions on subset of the 5-tuples: source IP address (*src-ip*), source port number (*src-port*), destination IP address (*dst-ip*), destination port number (*dst-port*), and protocol (*proto*) from each of the incoming packets to distribute the packets to the NIDPSs [39, 43].

There are two major problems with these simple distribution schemes:

1. Correlated flows are not sent to the same NIDPS.
2. The loads of the NIDPSs are not balanced.

The first problem is very critical. Since the correlated flows are distributed across many NIDPSs, some *correlation information* – the information derived from

the correlated flows – is lost. This loss of correlation information severely affects the anomaly-based detection and prevention of some intrusions which require analysis of the correlated flows. In the worst case, these intrusions might slip through the NIDPSs undetected. Well-known representatives of these intrusions are distributed denial of service (DDoS) attacks and port scans.

To demonstrate, we conducted a simple simulation. We simulated a port scan which involved an attacker scanning ports 1–1000 of a victim. We used Snort [34] as our port scan detector. In this simulation, there were two settings: a single NIDPS and a cluster of 10 NIDPSs with a hash-based traffic distribution scheme. The implemented hash function was a simple additive hash: (*src-ip* + *src-port* + *dst-ip* + *dst-port*) mod n, where n is the number of NIDPSs. The background traffic was omitted for simplicity.

When a single NIDPS was used to handle all the traffic, it detected the port scan; however, when the cluster was used, the port scan was not detected. This was because the flows of the port scan were distributed fairly evenly to all the NIDPSs, rather than to a single NIDPS.

The second problem is not less critical. The objective of using clusters of NIDPSs is to avoid overload conditions. However, if the loads of the NIDPSs are not balanced, then one of the NIDPSs might become overloaded even when loads of the other NIDPSs are low. Again, overloaded NIDPSs drop packets, which compromises security. Hence, poor load balancing results in an ineffective use of NIDPS clusters and also weakens the security of the clusters.

This research addresses the above two critical problems. We argue that by intelligently analyzing the traffic correlation and actively monitoring the loads of the NIDPSs, we can distribute the traffic to the NIDPSs in better ways – ways that reduce the loss of correlation information and provide load balancing to avoid overload conditions.

Finally, we note that the correlation information considered in our study is of which can be extracted from the five-tuples: *src-ip*, *dst-ip*, *src-port*, *dst-port*, and *proto*. Because extracting and analyzing packets' payload are expensive in terms of processing time, they are not supported by our study, which wishes to distribute the traffic in real-time. As a result, correlation information derived from the packets' payload is out of the scope of this research.

1.4 Contributions

The main contributions of this thesis are the two novel solutions that we developed to intelligently distribute network traffic to the NIDPSs in real-time. The two implemented systems corresponding to the solutions are the "Correlation-Based Load Balancer" and the "Correlation-Based Load Manager" ("Load Balancer" and "Load Manager" for short).

In particular, the proposed Load Balancer delivers the following features:

1. *Fine-grained Load Balancing:* The Load Balancer distributes the traffic in a way such that the difference between NIDPSs' loads is kept within a specified bound. This provides both protection and better traffic engineering to the network.

2. *Anomaly-Based Detection and Prevention Support:* Our Load Balancer is capable of grouping correlated flows in real-time. In particular, we focus on grouping flows which have identical *dst-ip*, *src-ip*, or *dst-port*, which greatly increases the accuracy of anomaly-based detection of DDoS attacks and port scans. Additionally, our Load Balancer preserves flows. Thus, it fully supports flow-based analysis.

3. *Configurable Security:* With our Load Balancer, one might favor security, i.e., reduced loss of correlation information, over performance, i.e., load balancing, when it is desirable to do so.

Meanwhile, the proposed Load Manager offers the following features:

1. *Identical Correlation Preservation:* Flows having the same *dst-ip*, *src-ip*, or *dst-port* are guaranteed to be sent to the same NIDPS. This maintains the anomaly-based detection accuracy of DDoS attacks and port scans. The Load Manager also preserves flows, so flow-based analysis is fully supported.

2. *Dynamic Load Distribution:* The Load Manager considers the current NIDPSs' loads while distributing the traffic and it also provides a mechanism to optimally move flows from one NIDPS to another when it is needed. These help to prevent overload conditions.

3. *Customizable Correlation:* The Load Manager allows for adding custom correlations. This feature extends the capability of the Load Manager to guarantee the grouping of various interested flows, which supports advanced detection techniques.

1.5 Thesis Organization

The remainder of this thesis is organized into 4 chapters. Chapter 2 discusses some of the most influential and recent research literature in the area of traffic distribution for NIDPSs. Chapter 3 and Chapter 4 present the design, implementation, and evaluation of the Load Balancer and the Load Manager, respectively. Finally, Chapter 5 concludes this thesis and discusses some future work.

Chapter 2

Literature Survey

In this chapter, related work in the area of traffic distribution for NIDPSs is presented in chronological order. Some of them are very influential works, which pioneered the research in this area, while others are recent approaches, which represent the current research direction. For each related work, we briefly present the approach; we then summarize the contribution; lastly, we point out how our research inherits from the approach and enhances it.

2.1 Slicing and Reassembling Mechanism

The problem that a single NIDS cannot keep up with the network traffic volume was examined by Kruegel et al. [18] in 2002. The authors proposed the use of clusters of NIDSs and an architecture, which involved a traffic slicing and reassembling mechanism, to distribute packets to the NIDSs. This work pioneered the research in this area.

In this approach, a *network tap* is used to extract link-layer frames and pass them to a *scatterer*. The scatterer uses a round-robin algorithm to partition the frames to send to a set of *slicers*. Then, the slicers send the frames to the *channels* which the NIDSs are attached to. At the channels, the frames are reassembled by the *reassemblers* to make sure that the order in which they arrive at the NIDSs is the same as their original order. The slicers choose the channels based on the attack scenarios that the channels' NIDSs handle. In other words, each NIDS will receive the necessary packets for the attacks that it handles.

The main contribution of this work is the traffic partitioning scheme which realizes the use of multiple NIDSs. This scheme distributes the load across the

NIDSs while making sure that each NIDS receives all the packets required for its detection; however, this work falls short in a few aspects:

- *No support for anomaly-based detection:* The traffic is distributed to the NIDSs based on the predefined signatures of the attack scenarios that they handle; thus, NIDSs are assumed to be signature-based only.
- *No load balancing:* Because the attack scenarios are fixed, the distribution of traffic is static. Although the authors realized the threat of the overload conditions and discussed one possible solution, which requires splitting and reassigning attack scenarios in real-time, the discussion lacked technical details. Also, there was no supportive implementation.
- *Unmanageable duplication:* Frames might be concurrently sent to many channels due to the overlapping attack scenarios – multiple attack scenarios that require the monitoring of the same traffic. Although the authors acknowledged the duplication, there was no discussion about how much this amount could be. This is crucial because the amount of duplication directly increases the chance of overloading an NIDS.

In this thesis, we enhance the above approach in many ways. Besides supporting anomaly-based detection by using dynamic traffic distributions, both of our solutions provide mechanisms to avoid overload conditions in real-time. One of our solutions, the Load Manager, inherits the idea of duplicating traffic to preserve the detection accuracy of the NIDSs; however, we clearly identify the upper bound of the duplication, as well as validate this amount through extensive simulations.

2.2 Hash-Based Distribution with Multiple Hash Functions

One early work that addressed the problem of overload conditions associated with static distributions was presented by Schaelicke et al. [35] in 2005. This work proposed the use of a dynamic distribution based on the NIDSs' loads.

In this approach, the authors 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 mapped to the NIDSs. When an NIDS's load is over a threshold, this NIDS is considered overloaded. To handle

the overload condition, the first proposed solution is to reassign some buckets of the overloaded NIDS to other NIDSs. The second proposed solution is to apply additional hash functions on the flows that are hashed into buckets of the overloaded NIDS by the first hash function, so that these flows are hashed into additional buckets of other NIDSs.

The main contribution of this work is the idea of changing the distribution of traffic when there is an overloaded NIDS. The goal is to remove the overload conditions. This work, however, does not take into account traffic correlation in its traffic distribution. Both of our approaches inherit from this work the idea of accounting for the NIDSs' loads while distributing the traffic. However, unlike this work, both of our approaches take into account the traffic correlation in their distributions.

2.3 Active Traffic Splitter

A state-of-the-art approach to distribute traffic to the NIDPSs was presented by Xinidis et al. [43] in 2006. This work proposed that the traffic distributors should be more active in their roles.

In this work, the authors introduced a traffic splitter with a set of *active operations*. In particular, three operations are implemented in the traffic splitter to improve the NIDPSs' performance:

1. *Early filtering/forwarding:* Incoming packets that do not contain any intrusions are identified early by the traffic splitter. The traffic splitter then filters them out immediately. This reduces the number of packets that the NIDPSs receive, as well as eliminates the process of sending the filtered-out packets to the NIDPSs. The resulting improvement in performance is 8 percent.

2. *Locality buffering:* Incoming packets are buffered and reordered by the traffic splitter, so that they arrive at the NIDPSs in sequences that improve the memory accesses of the NIDPSs, thus reducing their cache misses. The resulting performance improvement is 10–18 percent.

3. *Cumulative acknowledgment:* Early filtering/forwarding requires a coordination between the traffic splitter and the NIDPSs. In particular, the NIDPSs have to tell the traffic splitter which packets to filter/forward. Cumulative acknowledgment utilizes a novel communication scheme between the splitter

and the NIDPSs to reduce the amount of traffic involved in the coordination. This results in 50–90 percent performance improvement.

The major contributions of this work are the three operations implemented in the traffic splitter, which significantly improve the performance of the NIDPSs. Nonetheless, in this work, the traffic splitter uses a simple hash function to distribute the traffic to the NIDPSs. This distribution results in critical drawbacks that we have discussed: the loss of correlation information and an unbalanced load distribution.

We consider this as an important related work because of the idea that this work brings up: the traffic distributors should be more active in their roles. This idea is really an important foundation of our work. In our solutions, the Load Balancer and the Load Manager take into account both the traffic correlation and the current NIDPSs' loads to distribute the traffic intelligently. In other words, they are more active and do more processing than the typical hash-based traffic distributors.

2.4 Double-Threshold Load Balancing

In 2007, Andreolini et al. [2] proposed a dynamic load balancing scheme for the NIDSs. The proposed load balancing scheme uses two thresholds: a high threshold and a low threshold. When the load of one of the NIDSs reaches the high threshold, the load balancing scheme will activate. On the other hand, the low threshold is used for two purposes:

1. *Deactivation:* If the loads of all the NIDSs fall below the low threshold then the load balancing algorithm will terminate.

2. *Identifying target NIDSs:* Any NIDS with load lower than the low threshold is qualified to receive the traffic load, which is moved from another overloaded NIDS.

In addition to the two thresholds, the authors represented load of the NIDSs in terms of the traffic rate (Mbps). Also, the authors used linear aggregation methods to estimate the loads of the NIDSs. Moreover, they showed that the load balancing scheme can achieve a satisfactory balance of load with a low number of activations and deactivations.

The main contribution of this work is the load balancing algorithm which utilizes the two thresholds and the linear aggregation methods; however, similar to the multi-hash-function approach, this load balancing scheme does not take into account traffic correlation in its distribution.

Nevertheless, our work benefits a lot from this work. In particular, we adopt the proposed NIDS load representation and the estimation of load using linear aggregation methods. Being able to estimate the loads of the NIDSs is crucial for us because we cannot do load balancing without such estimates. Lastly, our approaches consider the traffic correlation while theirs does not.

2.5 Hash-Based Distribution with Communication

In 2007, Vallentin et al. [40] introduced a state-of-the-art cluster of NIDSs. On the one hand, this cluster uses a simple additive hash to distribute the traffic, which results in the already discussed drawbacks. On the other hand, the cluster implements a novel inter-NIDS communication scheme which is worthy of discussion.

In this scheme, a cluster has global variables whose values are synchronized among all the NIDSs via communication. In particular, when a global variable is updated at one of the NIDSs, this NIDS will send update packets to all other NIDSs to synchronize the variable. The communication could be seen as a means to compensate for the loss of correlation information due to distributing traffic.

The main contribution of this work is the inter-NIDS communication scheme. However, the communication itself brings in a substantial amount of overhead in terms of traffic generated to synchronize the global variables. Moreover, it must be noted that the inter-NIDS communication scheme requires uniform and open-source NIDSs. In other words, all NIDSs of the cluster must be the same and modifiable to support the communication. In particular, the authors use Bro [31], an open-source NIDS developed at UC Berkeley, as their NIDS of choice since Bro supports low level communication. In the case where there are various NIDSs, or proprietary NIDSs, such as Cisco NIDSs [7], it is difficult to establish the communication scheme.

Our approaches could be considered orthogonal to the above approach. We choose to reduce the loss of correlation information upfront, at the time the traffic is distributed. We also note that since our approach is independent of the NIDPSs,

it could be applied to a cluster of any NIDPSs. Lastly, our approaches provide load balancing while theirs does not.

2.6 Summary

In summary, all of the related work discussed above use simple distribution schemes to distribute traffic to the NIDPSs. Two of them use the static slicing distribution scheme [2, 18], and the others use the hash-based distribution schemes [35, 40, 43]. Some of these schemes do not take into account the current loads of the NIDPSs when distributing the traffic; as a result, they provide no load balancing [18, 39, 43].

More importantly, none of the related work considers the loss of correlation information when distributing the traffic. This loss of correlation information severely reduce the anomaly-based detection accuracy of the NIDPSs. One notable approach, which uses an inter-NIDS communication scheme to compensate for the loss of correlation information, was recently proposed by Vallentin et al. [40].

Our approaches, first and foremost, appropriately distribute the traffic to reduce the loss of correlation information. Furthermore, we leverage the real-time loads of the NIDPSs to provide load balancing. The load balancing provided by our approaches should be considered as alternatives to the other approaches [2, 35].

Chapter 3

Correlation-Based Load Balancer

3.1 Introduction

In this chapter, we present the Correlation-Based Load Balancer – our first novel solution to the problem of traffic distribution for the NIDPSs. The objective of the Load Balancer is to provide fine-grained load balancing while minimizing the loss of correlation information.

To address the traffic distribution problem, we first formalize it as an optimization problem, considering both the NIDPSs' load variance and the loss of correlation information. 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:

- The difference between NIDPSs' loads is kept within a specified bound.
- Correlated flows are grouped together at a single NIDPS to reduce the loss of correlation information.

We have implemented a prototype Load Balancer which uses the BLB algorithm. We also evaluated the Load Balancer against various DDoS attacks and port scans. The evaluation results show that compared to the naive hash-based distribution, our Load Balancer significantly improves the anomaly-based detection accuracy of these attacks while keeping the difference between loads of the NIDPSs small. Figure 3.1 shows how our Load Balancer fits into a network topology.

As briefly discussed in Section 1.4, our Load Balancer delivers the following features:

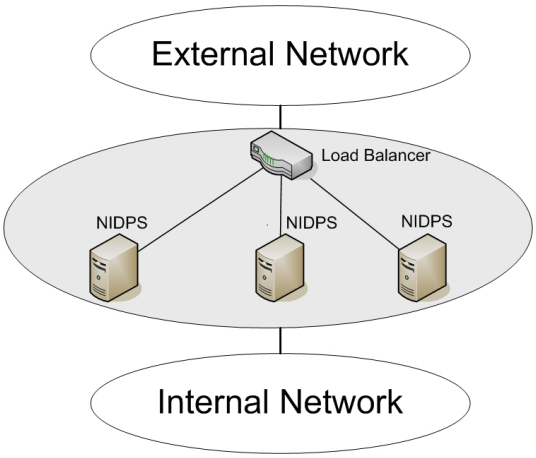

Figure 3.1: Placement of the Load Balancer

1. *Fine-Grained Load Balancing:* Our Load Balancer monitors loads of the NIDPSs and distributes the traffic in a way such that the difference between NIDPSs' loads is kept within a specified bound. This provides both protection and better traffic engineering to the network.

2. *Anomaly-Based Detection and Prevention Support:* Our Load Balancer is capable of grouping correlated flows in real-time. In particular, we focus on grouping flows which have identical *dst-ip*, *src-ip*, or *dst-port*, which greatly increases the accuracy of anomaly-based detection of DDoS attacks and port scans. Additionally, our Load Balancer preserves flows, i.e., packets belonging to the same flow are sent to the same NIDPS. Thus, it fully supports flow-based analysis.

3. *Configurable Security:* With our Load Balancer, one might favor security, i.e., reduced loss of correlation information, over performance, i.e., 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 ways to favor security: (1) relaxing the variance constraint, (2) duplicating the traffic to send to multiple NIDPSs, and (3) operating with a threshold-based constraint instead of a load-balancing-based constraint.

The rest of this chapter is organized as follows: Section 3.2 contains the problem statement and an overview of our approach. In Section 3.3, we formalize the flow assignment problem as an optimization problem and provide an approximation for

it. Section 3.4 explains the on-line clustering technique and describes the BLB algorithm. Section 3.5 discusses the correlations between flows. In Sections 3.6 and 3.7, we describe the implementation and evaluation results, respectively. Finally, we conclude this chapter in Section 3.8.

3.2 Problem Statement and Approach Overview

3.2.1 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 difference between loads of the NIDPSs within a specified bound, and minimizes the loss of correlation information due to distributing flows, which in turn improves the anomaly-based detection accuracy of the NIDPSs.

3.2.2 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 will be missed.

First, we introduce *clusters* to structure the flows. A cluster contains flows which are closely correlated with each other. Clusters are constructed and deleted on-the-fly depending on both the variety and the rate of the traffic. When a new flow arrives, it can join some existing clusters or form a new cluster of its own. We discuss on-line cluster management in detail in Section 3.4. 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*, *src-port*, *dst-port*, and *proto*. As previously mentioned in Section 1.3, we only consider the correlations derived from these five-tuples, and other correlations derived from packets' payload are out of the scope of this study. We discuss the correlations in more detail in Section 3.5.

Benefits play very important role in our approach since we use them to assign new flows to clusters. 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 strongly correlated with it.

Load balancing is achieved by closely monitoring the loads 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.

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 difference between NIDPSs' loads within a bound.

3.3 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 required.

3.3.1 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 1 if NIDPS i owns cluster j and 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 handles 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) [30] to the samples to alleviate the negative effect of spikes in traffic. This linear aggregation method was shown to perform very well in the context of NIDPS load balancing [2], as we discussed in Section 2.4.

In addition, we note that representing an NIDPS's load is a nontrivial task since an NIDPS may have numerous hardware and software resources, for example, CPU, disk, and memory. Therefore, one might challenge our load representation; however, the representation we use is the most common approach in the context of network intrusion detection and prevention [2, 35, 43].

In the following, let μ be the average load of all the NIDPSs and V be the upper bound for the variance after the assignment. 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 UDP flows separately.

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, assigned to two different NIDPSs, one monitoring flows with *dst-ip* 10.0.0.1 and the other monitoring flows with *dst-port* 80. 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 1 if f is going to be assigned to cluster j and 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 3.2.

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 be concurrently assigned to at most F NIDPSs. Constraint (3) requires that f be assigned to at most one cluster of each NIDPS. Finally, constraint (4) requires that the variance of the NIDPSs' loads after the assignment 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 ($\%^2$ load), and load is assumed to be normally distributed among the NIDPSs, then 99.73% of the NIDPSs will have their loads

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

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 3.2: Flow Assignment Optimization Problem

Algorithm 1 HeuristicFlowAssignment(f)

```
1:  solution_set = ∅
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 ∪ cluster
10:     nidps = NIDPS which has cluster
11:     update load of nidps
12:     nidps_set = nidps_set \ nidps
13: end for
14: return solution_set
```

within $3\sqrt{V} = 9$ (% load) of the average load μ, or within 18 (% load) of each other.

3.3.2 Heuristic Flow Assignment Algorithm

In order to solve FAOP, one has to examine all the cluster subsets, whose sizes are less than or equal to F. As a result, any algorithm which optimally solves FAOP would take at least polynomial (greater than linear) time. Due to the real-time requirement of the flow assignment, we propose a greedy-based approximation algorithm, the 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 the same 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.

3.3.3 Configurable Security

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

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

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

   Compared to this threshold-based approach, the Load Balancer's approach, which keeps the difference between NIDPSs' loads within a bound, provides better load balancing. In particular, when using this threshold-based approach, clusters are more likely to have an overloaded NIDPS even when the other NIDPSs' loads are low; this is because all traffic is sent to a NIDPS to achieve the best benefit until this NIDPS's load is high, rather than distributed to all of the available NIDPSs.

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 load, system resources, and performance requirements.

3.4 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 manage clusters using traditional clustering techniques like K-Means [16] or K-Medoids [16]. 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: benefit, cluster weight, and decay of weight.

3.4.1 Benefit Calculation

When a new flow arrives, we calculate the benefits of adding this flow to the existing clusters to make assignment decisions. The benefits vary depending on two factors: the *correlations* between the flow and the clusters; and the *weights* of the clusters.

First, the correlations between the new flow and the existing clusters are determined based on the correlations between the flow and the *centroids* of those clusters, where a centroid of a cluster is a flow representing the cluster. The correlations between flows are described in detail subsequently.

Secondly, weights of clusters represent the activeness of the clusters and the number of flows that the clusters have. The less active a cluster is or the fewer flows the cluster has, the less weight it has. In order to reflect those properties, weights of clusters decay over time and are updated every time the clusters receive a new flow. More details about the decay and update of weights are provided subsequently.

Let $D(f, c_j)$ be the logical distance between a new flow f and the centroid c_j of cluster j, where the logical distance between two flows is a value reflecting how correlated the two flows are (the logical distance notion is discussed in detail in Section 3.5), and let W_j be the current weight of cluster j. The benefit of adding f to cluster j is calculated as follows:

$$B_j = (1 - D(f, c_j))W_j \,.$$

This formula is constructed to provide the following properties:

- The closer f is to c_j, i.e., the more correlated f and c_j are, the higher benefit the assignment gives because of the larger $1 - D(f, c_j)$

- The heavier the cluster j is, i.e., the more active the cluster is or the more flows the cluster has, the higher benefit the assignment gives because of the larger W_j.

3.4.2 Cluster Weight

Each cluster has a weight, whose value is between 0 and 1 inclusive. When a cluster is first constructed, it has weight 1. For each cluster j, we keep track of these variables: the number of flows it has, s_j, its construction time or the last time it received a new flow, $t_{j,f}$, its weight at this time, $W_{j,t_{j,f}}$, and the last time it received a packet, $t_{j,p}$. For instance, at time t_0 when cluster j is constructed, $s_j = 1$, $t_{j,f} = t_{j,p} = t_0$, and $W_{j,t_{j,f}} = 1$.

The weight of a cluster decays when the cluster is inactive, i.e., when it does not receive network traffic. In particular, if a cluster j has not received any packet for a period of time $t - t_{j,p}$, where t is the current time, then its weight at time t is calculated as follows:

$$W_{j,t} = \lambda^{(t_{j,p}-t)} W_{j,t_{j,f}} \,, \tag{3.1}$$

where $\lambda > 1$ is the *decaying factor*. We introduce the decay of weight as a means to better group recent flows, as well as to manage the number of clusters in the system.

In particular, with the decay of weight, when a new flow arrives, it is more likely to be assigned to more active and recent clusters. This is because these clusters have higher weights due to less decay. For example, let $t_{a,p}$ and $t_{b,p}$ be the times clusters a and b received their last packets, respectively. If $t_{a,p} > t_{b,p}$, i.e., cluster a received the last packet more recently than cluster b, then at a later time t, cluster a has decayed less than cluster b due to a smaller decaying duration: $t - t_{a,p} < t - t_{b,p}$.

Furthermore, if clusters are inactive for a long period of time, they have very low weights due to the decay. As a result, we can delete these clusters by setting a minimum weight threshold T_{weight} and deleting any cluster whose weight is less than this threshold. The deletion of old clusters is necessary to give room for more recent clusters and to make sure that the maintained clusters in the system represent the recent traffic.

When a new flow f is added to cluster j at time t, all the variables get updated as follows:

$$W_{j,t} = \lambda^{(t_{j,p}-t)} W_{j,t_{j,f}} + [1 - \lambda^{(t_{j,p}-t)} W_{j,t_{j,f}}] \frac{1 - D(f, c_j)}{s_j + 1}\,, \tag{3.2}$$
$$s_j = s_j + 1\,,$$
$$t_{j,f} = t_{j,p} = t\,,$$
$$W_{j,t_{j,f}} = W_{j,t}\,.$$

If at time t, there were no new flow added to cluster j then its weight would equal $\lambda^{(t_{j,p}-t)} W_{j,t_{j,f}}$, according to Equation 3.1. However, since f is added to it, its weight increases by the amount $[1 - \lambda^{(t_{j,p}-t)} W_{j,t_{j,f}}] \frac{1-D(f,c_j)}{s_j+1}$, according to Equation 3.2. Consequently, adding a new flow to a cluster essentially increases the cluster's weight.

The increment amount depends on both the correlation between the new flow and the cluster – the $D(f, c_j)$, as well as the number of flows that the cluster already has – the s_j. In particular, the more correlated with the cluster the new flow is, i.e., the smaller $D(f, c_j)$, the more weight it adds to the cluster. In contrast, the more flows that the cluster already has, i.e., the larger s_j, the less weight the new flow adds to the cluster. It must be noted that as a cluster becomes large, the increment of weight becomes insignificant due to the large s_j. As a result, this increment eventually can not compensate for the loss of weight due to the decay to keep the cluster alive.

In summary, the increment of weight ensures that clusters with more flows and clusters with flows which are more correlated have heavier weights than the others. Ultimately, this ensures that any new flow is more likely to be assigned to more dense clusters due to better benefits.

3.4.3 Benefit-Based Load Balancing Algorithm

The Benefit-Based Load Balancing (BLB) algorithm, which is used to distribute traffic intelligently to the NIDPSs, is the heart of the Load Balancer. This algorithm ties together all the details presented in this chapter. We summarize the BLB algorithm in Algorithm 2 and describe it below.

When there is a new flow f, we use the HFA algorithm to solve the FAOP to get a candidate set of clusters (Line 1). 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

Algorithm 2 BENEFIT-BASEDLOADBALANCING(f)

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

with the lowest load (Line 2–4). In the other case, f is added to the candidate clusters and these clusters are updated appropriately (Line 5–7).

3.5 Flow Correlations

The correlations between flows are 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.

3.5.1 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:

$$D(f_1, f_2) = \alpha_{ip}\, \delta_{ip}(f_1, f_2) + \alpha_{port}\, \delta_{port}(f_1, f_2) + \alpha_{protocol}\, \delta_{protocol}(f_1, f_2)\,,$$

where $\delta_{ip}(f_1, f_2)$, $\delta_{port}(f_1, f_2)$, $\delta_{protocol}(f_1, f_2)$ are the logical distances given by the IP addresses, port numbers, and protocols of the two flows respectively; and the α's are their weights.

The $\delta_{ip}(f_1, f_2)$ reflects the correlations between the source IP addresses, as well as the destination IP addresses of the two flows f_1 and f_2. On the other hand, $\delta_{port}(f_1, f_2)$ mainly concerns with the correlations between the destination ports of the two flows. This is because the source ports play an insignificant role in anomaly-based detection.

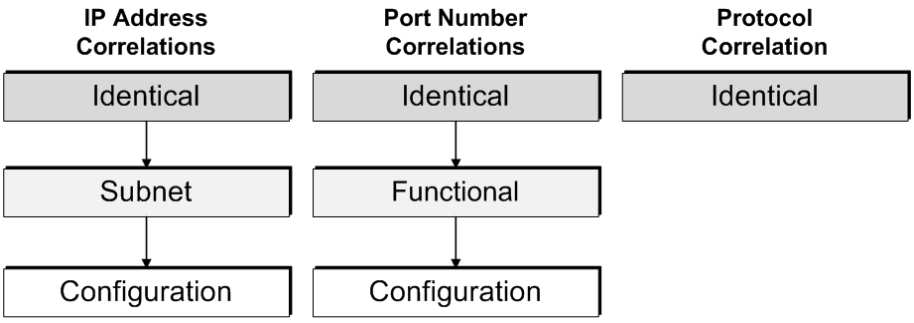

Figure 3.3: Matching Order of Correlations given by IP Addresses, Port Numbers, and Protocols

Through detection scenarios that we conducted, we observe that the IP addresses' and port numbers' correlations play more important roles than the protocols' correlation. As a result, weights of the IP addresses and port numbers should be heavier than that of the protocols. We suggest 0.4 - 0.5 as a range for both α_{ip} and α_{port}, and $\alpha_{protocol} = 1 - (\alpha_{ip} + \alpha_{port})$.

3.5.2 Logical Distance Components

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

We match the correlation given by IP 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 [27]. 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 source IP addresses or destination IP addresses of two flows belong to the same subnet or virtual LAN, 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.

  For instance, it is sometimes 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 specified set of addresses in general.

Figure 3.3 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.

Logical Distance given 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 of the destination port numbers instead of the 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. It

is also possible to use the configuration correlation, which is discussed below, 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.

Logical Distance given 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.

3.6 Implementation

We have implemented a prototype Load Balancer, which can distribute the traffic to the NIDPSs in real-time. In addition, in order to evaluate the Load Balancer, we implemented a DDoS detector. We describe our implementations in detail below.

3.6.1 Load Balancer

The prototype Load Balancer was developed using the *libpcap* library [22] – 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 IP 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. As discussed in Chapter 2, various hash-based algorithms were used by others [31, 35, 43] to distribute the traffic and they all shared a common property: applying a simple hash function on a subset of the five-tuples. Hence, 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 an Intel Dual Core 2.0 GHz CPU, 2 GB RAM, 2×1 Gbps NICs. 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 the 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).

3.6.2 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. [32]. Fundamentally, this algorithm detects the change of the mean value of the percentage of new source IP addresses over time. 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.

3.7 Evaluation

3.7.1 Performance

In order to evaluate how well our Load Balancer distributes the traffic, we needed high volume traffic traces. Consequently, we chose two weeks of 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) [29].

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 2$^{\text{nd}}$, 2003. This hour was one of the busiest

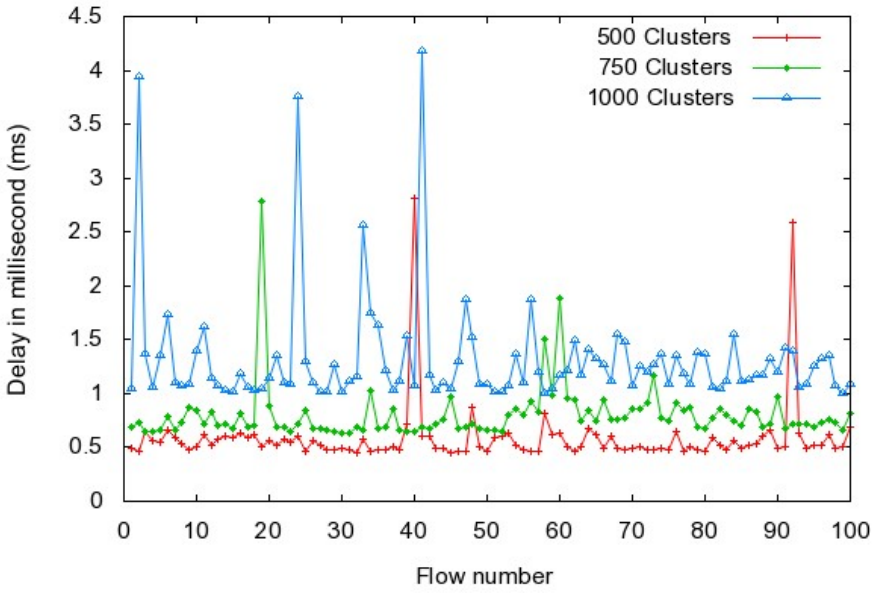

Figure 3.4: Effect of the Number of Clusters on the System Overhead

hours of the network. During this hour, there were 200 new flows per second and 14 Mbps of traffic on average [37].

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 T_{weight} was varied to get the desired number of maintained clusters. There were 10 NIDPSs, each of which had a capacity of 3 Mbps; V was 25; F was 1; α_{ip} and α_{port} were 0.5 and $\alpha_{protocol}$ was 0; $T_{benefit}$ was 0.1; 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 to which cluster(s) to assign the flow. This is the primary system overhead associated with the flow. We measured this overhead by the delay introduced to the first packet of the flow, which was the time to run the BLB algorithm. The measurement was done as follows:

For every packet read from the trace file, the Load Balancer checked if it belonged to previously assigned flows. If it did not, BLB algorithm was then used to assign this packet; in this case, the difference between the timestamps taken after and before the execution of the BLB algorithm was recorded. This difference was the overhead introduced to this packet, which starts a new flow, by the Load

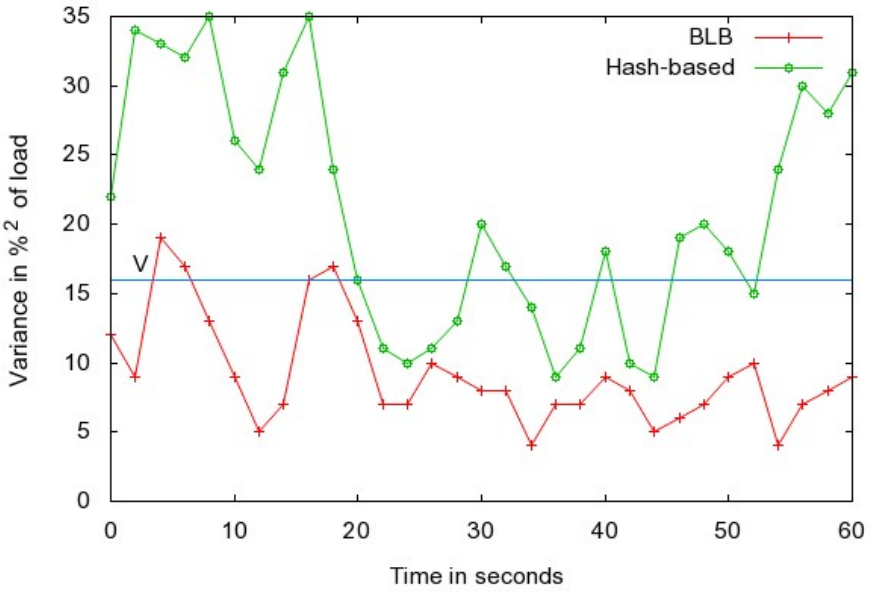

Figure 3.5: Effect of the Algorithms on the Variance

Balancer. The timestamps were taken using the Java function *System.nanoTime()*. We also note that the packets were not actually sent to the NIDPSs since it was not necessary.

Figure 3.4 plots the delays of the first packets of a sampling of 100 consecutive flows when the system maintained 500, 750, and 1000 clusters. The average delays of the full trace were 0.58, 0.80, and 1.31 milliseconds respectively (not shown in the figure). This result indicates that the higher the number of clusters, the higher the delays. This is because it takes more time to calculate the benefits when there are more clusters. Also, the spikes happened when there was a significant number of clusters that satisfied the variance constraint, which increased the execution time of the BLB algorithm. In summary, the delay per flow introduced by our system is on the order of a millisecond, which is tolerable.

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 500 clusters and used both the BLB and the hash-based algorithms. There were 10 NIDPSs, each of which had a capacity of 3 Mbps; λ was 1.1; $T_{benefit}$ was 0.1; α_{ip} and α_{port} were 0.5 and $\alpha_{protocol}$ was 0; F was 1; and V was 16. We note that because the trace only contained packet headers, we uniformly generated packet

payload having sizes between 400 and 600 bytes. This is to preserve the overall traffic rate of 14 Mbps of the trace file.

Throughout the experiment, we recorded the loads of the NIDPSs to calculate the variance. For each NIDPS, we take a sample of the traffic going to it every 1 second to estimate its load. For the Single Exponential Moving Average [30], the number of samples for the initial calculation is $s = 30$. In other words, the first estimated load is the mean of 30 initial samples of load. Also, the smoothing factor is $\alpha = \frac{2}{s+1}$. The smoothing factor is essentially the weight of the current load in the estimation:

$$\text{estimated\_load} = \alpha \cdot \text{current\_load} + (1 - \alpha) \cdot \text{last\_load}\,.$$

Figure 3.5 plots the variance associated with the two algorithms at every 2 seconds during a sampling of 60 seconds. We observed that the variance of the hash-based algorithm was noticeably higher than that of the BLB algorithm. This indicated that the loads of the NIDPSs when the hash-based algorithm was used were substantially unbalanced. Also, our BLB algorithm often had variance less than V; however, there were occasions when the variance was bigger than V. These were points of time at which there was no solution to the FAOP.

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

3.7.2 Security

Three simulations were conducted to evaluate how the BLB algorithm supports the detections of DDoS attacks and port scans compared to the hash-based algorithm. For the BLB algorithm, the Load Balancer maintained 500 clusters; λ was 1.1; $T_{benefit}$ was 0.1; α_{ip} and α_{port} were 0.5 and $\alpha_{protocol}$ was 0; NIDPS capacity was 10 Mbps; V was 25; and F was 1. The Load Balancer was used to distribute traffic to 10 NIDPSs.

In the simulations, the protected internal network was a Class B network – a network which has 65534 addresses, each of which has a leading bit string "10". 10% of the address space was occupied. The generated background traffic was about 15 Mbps with 100 flows per second on average. The source and destination addresses were randomly generated such that 80% of the flows were from or to 20%

of the machines in the internal network. Packets had sizes uniformly distributed from 500 to 1500 bytes. Flows' durations were uniformly distributed from 1 to 30 seconds.

DDoS Attack

We simulated a large scale UDP flood attack [36], 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 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.

Figure 3.6 plots the highest fractions of new source IP addresses observable by one of the NIDPSs in the following three settings: (1) a single NIDPS without the Load Balancer, (2) 10 NIDPSs with the Load Balancer using the BLB algorithm, and (3) 10 NIDPSs with the Load Balancer using the hash-based algorithm. We note that when the single NIDPS was used, we assumed that it could handle all the traffic without dropping packets. The corresponding values of Y_n are shown in Figure 3.7.

It can 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 substantially 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 failed to detect the attack but the BLB algorithm succeeded.

For example, if the threshold T_y was set to 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 without the Load Balancer was used. In the case when the single NIDPS was used, the attack was detected earlier because all flows of the attack went to this NIDPS.

This evaluation, however, has several limitations. First, the success of the detection strongly depends on the arbitrarily chosen T_y. Secondly, given the importance of T_y on the success of the detection, this evaluation lacks a thorough investigation of T_y. As a result, in order to characterize the benefit of the BLB algorithm more

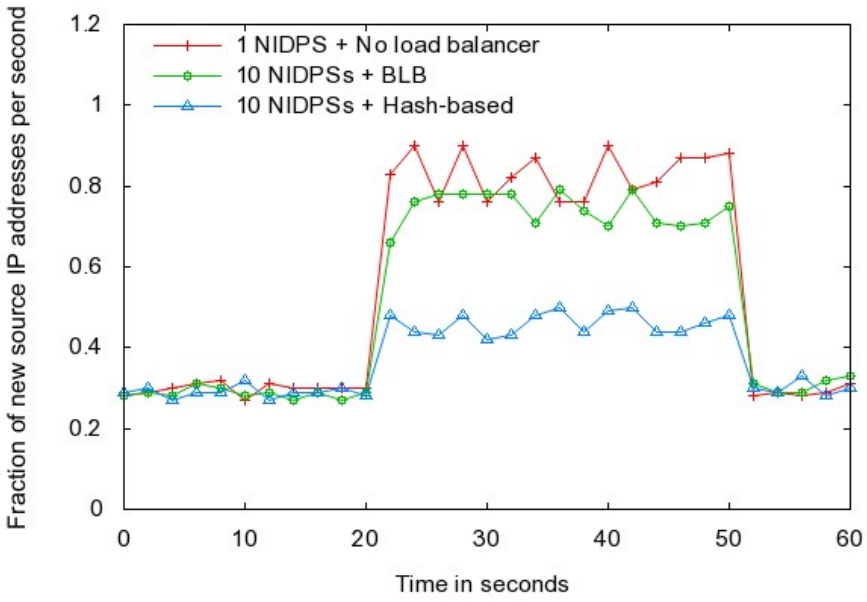

Figure 3.6: Effect of the Algorithms on the Fraction of New Source IP Addresses per Second

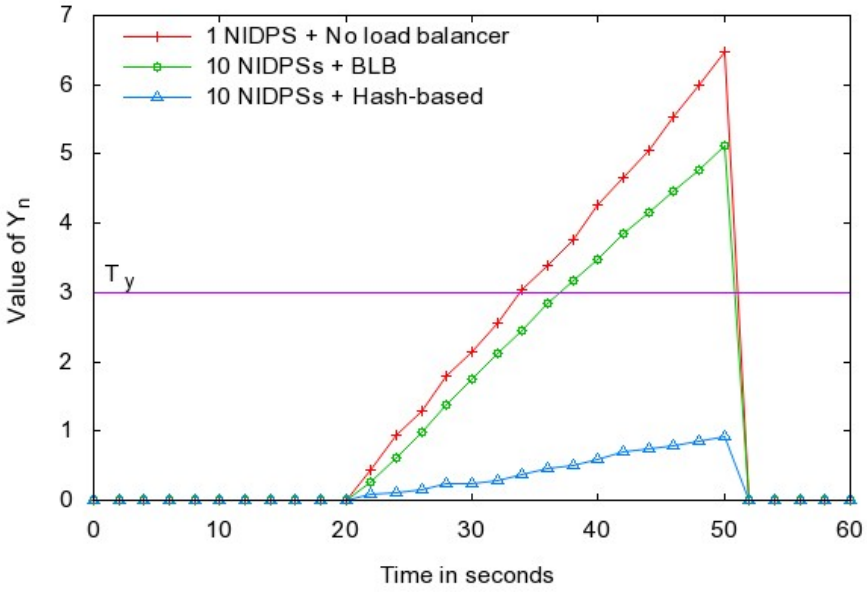

Figure 3.7: Effect of the Algorithms on the Value of Y_n

accurately, we plan to answer the following question in future work:

- Given a good sampling of traffic and a reasonable distance between a T_y of the normal traffic and a T_y of the attack traffic, what is the probability of a successful detection for a well chosen T_y?

In conclusion, this evaluation result shows that the BLB algorithm distributes the traffic in a way which increases the detection accuracy of the DDoS attack significantly compared to the hash-based algorithm. Also, this evaluation has several limitations that are the subjects of future work.

Port Scans

For this part, we used Snort [34] 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 [6]:

- *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.
- *Decoy Portscan:* A high number of scanning hosts with a few spoofed hosts, scanning a small number of hosts, for a small number of ports.
- *Distributed Portscan:* Similar to Decoy Portscan but with a high number of ports.

We note that sophisticated scans, such as decoy portscans and distributed portscans, might not be well supported by the Load Balancer because of the diversity of these scans' flows. In order to support the detection of these scans, it might be beneficial to monitor the number of connection attempts to unused address space of the protected networks. The Load Balancer can support the grouping of flows of these connection attempts by specifying the unused address space in its configuration correlations.

In practice, sophisticated scans currently have low detection accuracy (high false positive rate) [6]. In other words, current NIDPSs do not support them well. As a

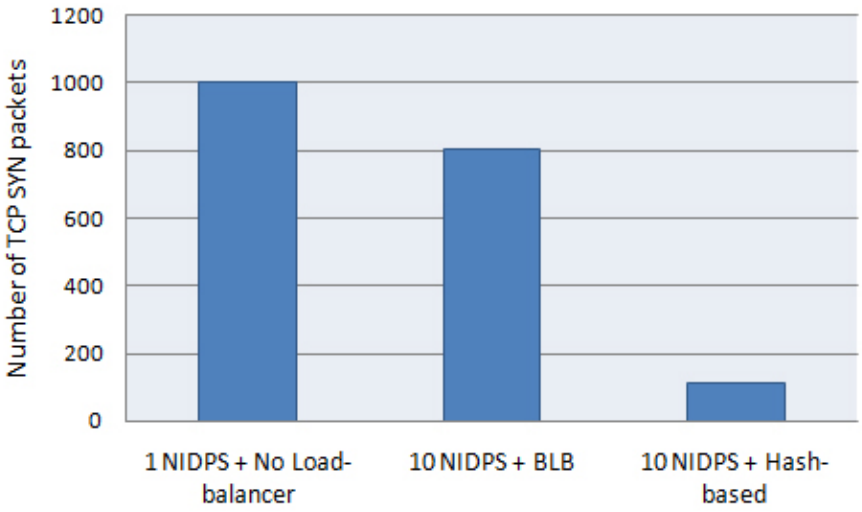

Figure 3.8: The Highest Number of SYN Packets Observable by One of the NIDPSs During the Portscan

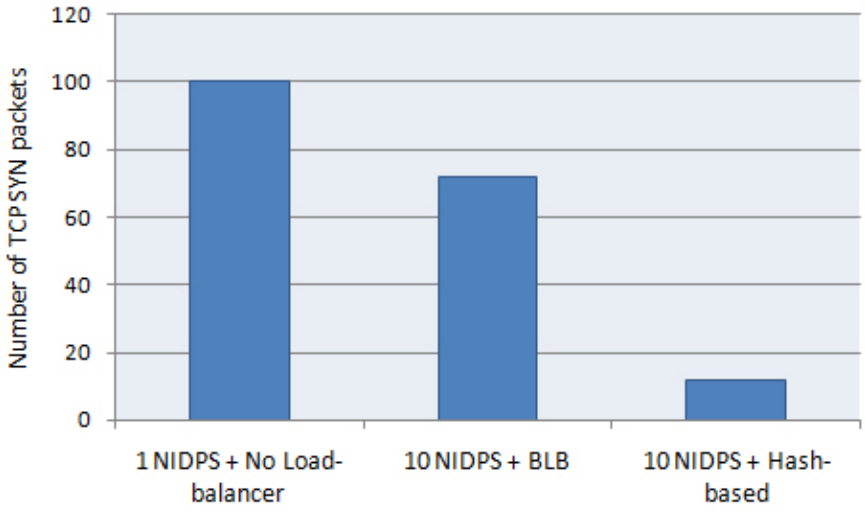

Figure 3.9: The Highest Number of SYN Packets Observable by One of the NIDPSs During the Portsweep

result, in this evaluation, we focus on evaluating how our Load Balancer supports the detection of ordinary portscans and portsweeps.

For both of the following simulations, the background traffic was the same as in the previous DDoS simulation. Similarly, when a single NIDPS without the 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. Lastly, the scans were designed so that a single NIDPS without the Load Balancer always successfully detected them.

First we used nmap [28] 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 of the NIDPSs was about 100. This was not enough for the sfPortScan to trigger a portscan alert. However, when the BLB algorithm was used, one NIDPS observed up to 700 SYN packets; thus, it generated a portscan alert. These numbers are plotted in Figure 3.8.

Secondly, we used nmap to carry out a portsweep. We used one host to scan port 80 of 100 victims. When a single NIDPS without the 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 the sfPortscan to trigger a portsweep alert. On the other hand, when the BLB algorithm was used, one NIDPS observed as many as 80 SYN packets; thus, it generated a portsweep alert. These numbers are plotted in Figure 3.9.

One might argue that the sfPortscan's mechanism to detect port scans is naive and simple and that a better mechanism should be used in our evaluation. Nonetheless, this simplicity represents commonly used techniques for detecting port scans and for anomaly-based detection in general. To the best of our knowledge, the mechanism used by two other popular operational NIDPSs: Bro [31, 40] and Cisco IPS [7], 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. [15].

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 shows that our Load Balancer using the BLB algorithm can substantially improve the detection accuracy of portscans and portsweeps compared to the hash-based algorithm.

3.7.3 Number of Clusters

We conducted a simulation to evaluate the effect of the number of maintained clusters on the security. In particular, we want to evaluate how the number of maintained clusters affects the grouping of scan flows.

In this simulation, the network model and the background traffic were similar to the ones in Section 3.7.2. Recall that the background traffic had about 100 flows per second and was about 15 Mbps. There were 10 NIDPSs, each of which had capacity of 10 Mbps. The Load Balancer distributed traffic using the BLB algorithm. λ was 1.1; $T_{benefit}$ was 0.1; α_{ip} and α_{port} were 0.5 and $\alpha_{protocol}$ was 0; and F was 1.

We set V to 25. V was set high enough so that the variance of the NIDPSs during the simulation was always less than V. This was to eliminate the negative effect of the variance of the NIDPSs on the distribution of the scan packets because if the variance constraint could not be satisfied then any new flow would be assigned to the NIDPS with the lowest load regardless of the benefit.

For the simulation, we varied the weight threshold T_{weight} to get the desired number of maintained clusters: 500, 1000, and 1500. We first let the Load Balancer distribute just the background traffic for 60 seconds. We then generated a TCP SYN scan, which consisted of 1024 packets and scanned port 1–1024 of a victim. There were two settings used for the scan duration: 5 and 60 seconds. During the scan, the Load Balancer distributed the scan packets in addition to the background traffic. We recorded the assignment of the scan packets.

Fast Scans

Figure 3.10 plots the distribution of the scan packets when the number of maintained clusters was 500, 1000, and 1500; and when the scan duration was 5 seconds. It can be observed from this figure that in all three cases, the percentage of the highest number of the scan packets that one NIDPS received was about 80%. We conclude that the number of maintained clusters did not affect the grouping of

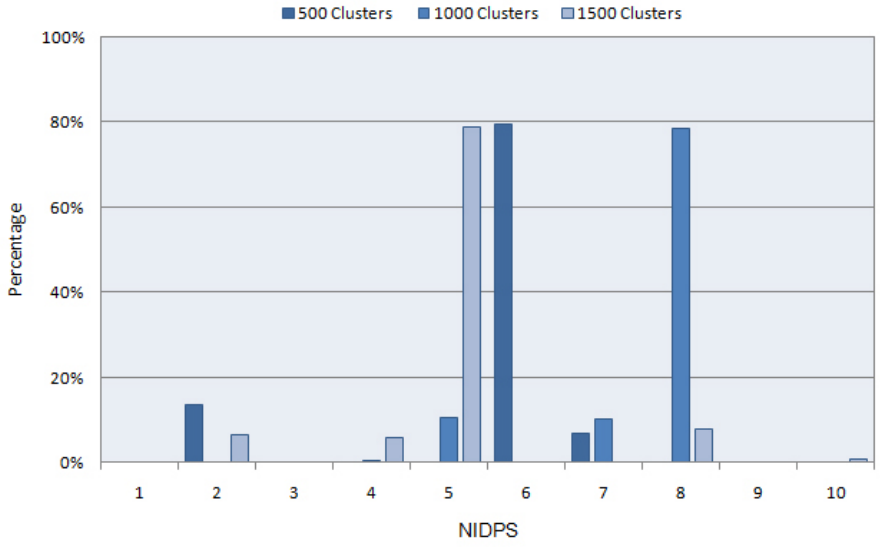

Figure 3.10: The Distribution of the Scan Packets of the 5-Second Scan

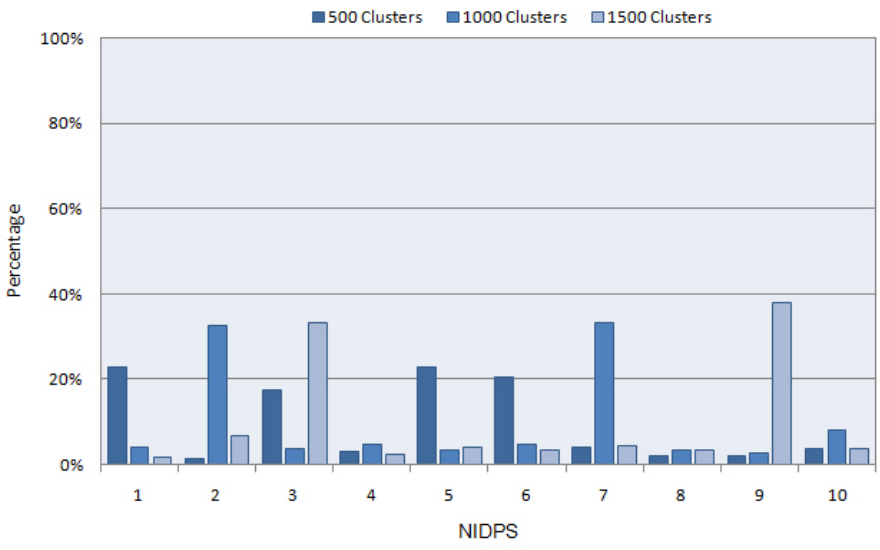

Figure 3.11: The Distribution of the Scan Packets of the 60-Second Scan

flows of this 5-second scan. In general, this evaluation result implies that the Load Balancer works well with fast scans.

Slow Scans

Figure 3.11 plots the distribution of the scan packets when the scan duration was 60 seconds. It can be observed from this figure that packets of the scan were scattered. The spread of the scan packets was due to new clusters that were created during the scan. These new clusters might give higher benefit than the clusters which received part of the scans; thus, some scan packets were distributed to the new clusters. The overall result was the spread of scan packets.

Furthermore, Figure 3.11 also indicates that the percentage of the highest number of scan packets that one NIDPS received slightly increases when the number of maintained clusters increases. This was because the more clusters maintained in the systems, the less likely new clusters would be created. Thus, the new clusters created during the scan would have less effect on the distribution of the scan packets. Nevertheless, the small improvement in grouping might not justify the large overhead that the large number of clusters creates, especially, since the system overhead increases linearly with the number of maintained clusters as discussed in Section 3.7.1.

In summary, this evaluation result shows that the Load Balancer does not work well with slow scans. In particular, it could not group flows of the 60-second scan well even when the load balancing constraint was waived. This indicates the limitation of the proposed cluster dynamics. In order to support the grouping of flows of slow scans well, the proposed cluster dynamics must be revised. This remains as future work.

3.7.4 Traffic Duplication

All the previous experiments were conducted to gain a fundamental understanding of the performance and the security of the Load Balancer. As such, we started with a basic configuration, which involves setting $F = 1$. This essentially does not allow any packet duplication.

A value of F higher than 1 will at least double the system overhead – the amount of time required to make the assignment decision. This is because Algorithm 1, the Heuristic Flow Assignment algorithm, now has to execute the "for" loop at least

two times and this loop is the most time consuming task of the whole process of making assignment decisions.

On the other hand, there is very little that we can tell about the variance of the systems when changing the value of F. This is because the loads of the NIDPSs are now completely different from those when setting $F = 1$ due to additional traffic from the duplication.

Finally, a value of F higher than 1 might improve the detection accuracy of the port scans and DDoS attacks since it might send more attack flows to the same NIDPS compared to the situation with $F = 1$. With $F = 1$, the Load Balancer is already able to send about 70% of the attack flows to the same NIDPS. A higher value of F, therefore, might improve the grouping to above 70%.

In summary, the effect of F on the variance and the detection accuracy needs additional experiments. This remains as future work, as indicated in Section 5.2

3.8 Summary

In this chapter, we detail the Correlation-Based Load Balancer – our first solution to the traffic distribution 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 traffic. Our algorithm performs real-time optimization, thus it accommodates intrusion detection systems as well as intrusion prevention systems.

We have implemented a prototype Load Balancer which uses the BLB algorithm. Our Load Balancer intends to achieve the following properties:

- The difference between the NIDPSs' loads are kept within a specified bound.
- Correlated flows are grouped together at a single system to reduce the loss of correlation information. In particular, we focus on grouping flows which have identical *dst-ip*, *src-ip*, or *dst-port*.

Extensive simulations with real traffic traces and major attacks showed that our Load Balancer using 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. More importantly, compared to the naive hash-based distribution, it significantly improves the accuracy of anomaly-based detection of DDoS attacks and port scans. Nevertheless, the evaluation results also point out a limitation of the proposed Load Balancer: The Load Balancer does not work well with slow scans. This is an open problem and we plan to address it in future work.

Chapter 4

Correlation-Based Load Manager

4.1 Introduction

We first started with the Load Balancer. At the time, our main focus was to spread the traffic load evenly to the NIDPSs, i.e., to keep the difference between the loads of the NIDPSs within a bound, and the traffic correlation took a back seat. We did not focus much on the security of the cluster.

After experimenting with this idea, we realized that we should have focused on security as long as there is no overloaded NIDPS. This is because security – detecting and preventing attacks – is the most important goal of the NIDPSs. As a result, we developed the Load Manager with the security as its main focus.

In particular, from the evaluation of the Load Balancer, we learned that providing load balancing is a challenging task. Sometimes the Load Balancer could not achieve its load balancing goal, i.e., keeping the difference between the NIDPSs' loads within a bound. This was illustrated by the points of time at which the variance of the NIDPSs was above the upper bound for variance in the evaluation done in Section 3.7.1.

Furthermore, the Load Balancer tries its best to group attack-correlated flows together; however, it does not guarantee the grouping of all of them. Thus, some of the attack-correlated flows will not be sent to the desired NIDPSs. Consequently, it is possible that attacks go through the NIDPSs undetected. For example, in the evaluation done in Section 3.7.2, if the threshold T_y was set at 6 then the DDoS attack would not be detected even when the BLB algorithm was used.

The resulting Load Manager can prevent overload conditions and group flows of the two major attacks of interest: DDoS attacks and port scans. We consider

the Load Manager as an improvement over the Load Balancer because of its better security. The only advantage that the Load Balancer has over the Load Manager is that the Load Balancer can keep the loads of the NIDPSs more balanced.

In this chapter, we present the Correlation-based Load Manager – our second solution to the traffic distribution problem. Unlike the Load Balancer's objective, the objective of the Load Manager is to group flows having the same *dst-ip*, *src-ip*, and *dst-port* together while preventing overload conditions.

As briefly introduced in Section 1.4, our Load Manager delivers the following features:

1. *Identical Correlation Preservation:* Flows having the same *dst-ip*, *src-ip*, and *dst-port* are guaranteed to be sent to the same NIDPS. In other words, the Load Manager preserves the identical correlations given by addresses and port numbers. This maintains the anomaly-based detection accuracy of DDoS attacks a port scans.

   For example, using the Load Manager, all flows of a DDoS attack targeting a victim are guaranteed to be sent to the same NIDPS. Therefore, if the NIDPS is capable of detecting the attack in the first place, then the attack is detected.

   We note that the grouping of flows is achieved at the cost of duplicating traffic; however, unlike the approach proposed by Kruegel et al. [18], which was discussed in Section 2.1, the amount of duplication generated by our Load Manager is bounded.

   Moreover, our Load Manager preserves flows. In other words, packets belonging to the same flow are sent to the same NIDPS. Thus, it fully supports flow-based analysis.

2. *Dynamic Load Distribution:* The Load Manager considers the current NIDPSs' loads while distributing traffic. In particular, the majority of the new traffic is assigned to the NIDPS with the lowest load. This reduces the chance of overloading any NIDPS when the loads of the other NIDPSs are low. Furthermore, the Load Manager provides a mechanism to optimally move flows from one NIDPS to another when needed. The moving helps to prevent overload conditions.

3. *Customizable Correlation:* Besides the default correlations which we initially support, the Load Manager allows the addition of custom correlations. This feature extends the capability of the Load Manager to guarantee the grouping

of various flows of interest. As a result, advanced detection techniques, which require the grouping of diverse flows, can be used.

We have implemented a prototype Load Manager. We evaluated both the performance and the security of the Load Manager using simulation. The evaluation results show that the Load Manager has a low overhead in terms of the delays it introduces to packets. Moreover, it causes traffic duplication noticeably less than the established upper bound. Most importantly, the results show that the Load Manager successfully assigns attack-correlated flows of a variety of DDoS attacks and port scans to the same NIDPS, thereby maintaining successful detection of the attacks.

The rest of this chapter is organized as follows: In Section 4.2, we describe the overall architecture of the Load Manager. Section 4.3 and Section 4.4 discuss the design of the two main components of the Load Manager: the Flow Distributor and the Flow Manager. In Section 4.5, we describe our implementation of the system. Evaluation results are presented in Section 4.6. Finally, we conclude this chapter in Section 4.7.

4.2 Overall Architecture

Figure 4.1 details the overall architecture of the Load Manager. The Load Manager has two main components: the Flow Distributor and the Flow Manager.

The Flow Distributor's first task is to distribute packets of previously assigned flows to the corresponding NIDPSs. Its second task is to distribute new flows to the NIDPSs based on both the current loads of the NIDPSs and the traffic correlation. The new-flow distribution is done in such a way that identical correlations are preserved. When there is an overloaded NIDPS, the Flow Manager's task is to optimally select a set of flows from this NIDPS to move to the lowest load NIDPS. How these two components work is briefly described below.

When a new packet arrives, the Flow Distributor checks to see if this packet belongs to a previously assigned flow. If it does then the packet is sent to the corresponding NIDPS(s). Otherwise, the Flow Distributor checks the correlations between the new flow, which starts with this packet, and the NIDPSs. Afterward, the packet is duplicated to be sent to all the NIDPSs that the flow is correlated with.

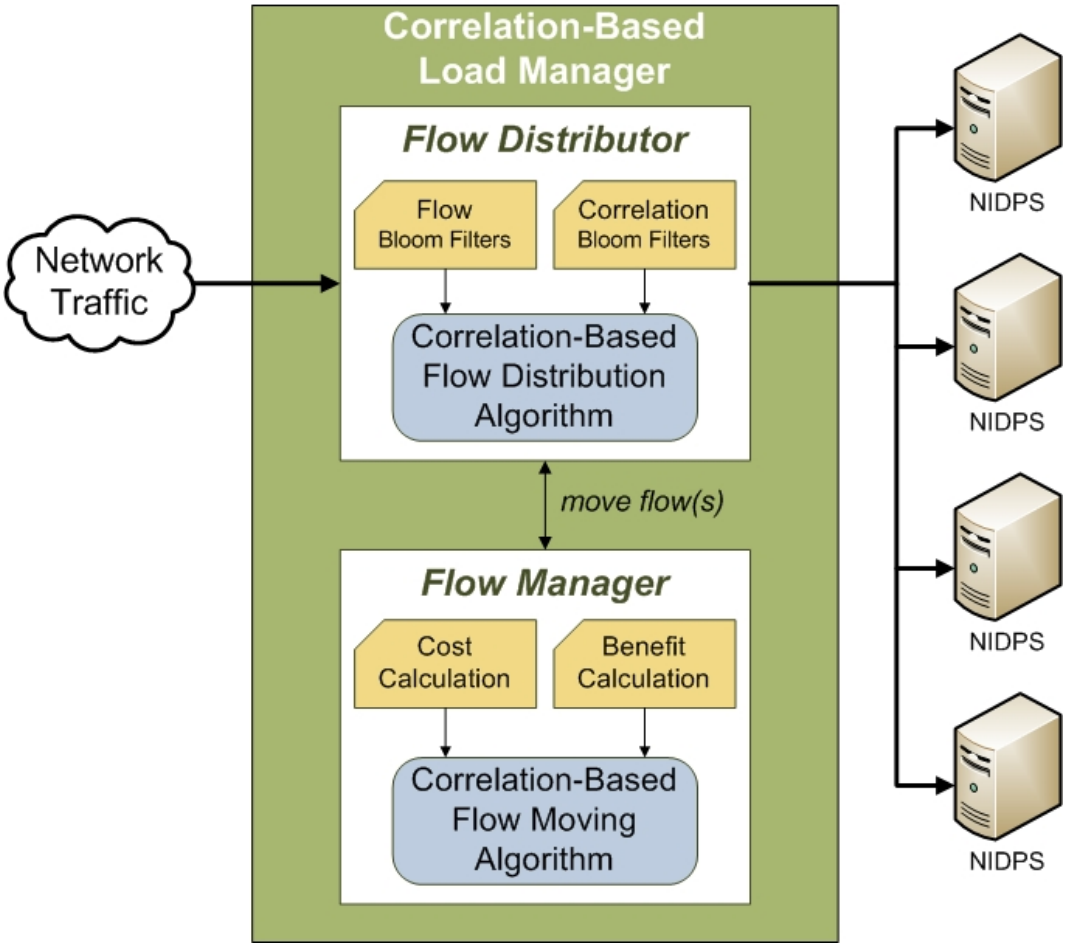

Figure 4.1: Correlation-Based Load Manager Architecture

The Flow Distributor must be in operation all the time; meanwhile, the Flow Manager only operates when an NIDPS becomes overloaded. When so, the Flow Manager selects from this NIDPS a set of flows sufficient to reduce the load below the predefined load value and with the minimum correlation with the rest of the flows in the NIDPS to move to the lowest load NIDPS. By moving flows, the Flow Manager prevents overload conditions. We also note that there are several alternative ways that could be used to choose the NIDPSs that receive the moving flows. For example, as discussed in Section 2.4, a low load threshold can be used to determine the eligible NIDPSs to receive the moving flows. Then, the moving flows can be distributed evenly to them, rather than distributed only to the lowest load NIDPS.

Table 4.1: Intrusions whose detections require the analysis of correlated flows

ID	Name	Description
1	DoS Machine	Denial of service (DoS) attack to a target machine
2	DDoS Machine	Distributed DoS attack to a target machine
3	DoS Application	DoS attack to a target service
4	DDoS Application	Distributed DoS attack to a target service
5	Scan	Horizontal, vertical, and mixed scans
6	Distributed Scan	Distributed horizontal, vertical, and mixed scans

4.3 Flow Distributor

In this section, we present the design of the Flow Distributor. First, we differentiate intrusions whose detection requires the analysis of correlated flows from the ones whose detection only requires the analysis of individual flows. We then discuss our representation of NIDPS load. Afterward, we describe the Bloom filter – the data structure that we utilize to make distribution decisions. Finally, we present our Correlation-Based Flow Distribution algorithm, which is used to distribute packets of previously assigned flows as well as new flows.

4.3.1 Applicable Intrusions

There are intrusions, such as viruses and worms, whose detection relies on analyzing individual flows. As such, the detection of these intrusions is not affected by any flow-based traffic distribution, which distributes packets belonging to the same flow to the same NIDPS. Therefore, this type of intrusion is not the target of our research. Instead, we focus on intrusions whose detection requires the analysis of correlated flows.

Table 4.1 lists the most common intrusions whose detection requires the analysis of correlated flows. We do not claim that this is a comprehensive list. In fact, one can add additional intrusions to this table; however, the intrusions in this list are very well-known in the research literature [15, 21, 27, 35, 40]; therefore, we consider it a reasonable start. We note that the classification of the intrusions was adopted from the classifications proposed by Mirkovic and Reiher [27], and by Weaver et al. [41].

4.3.2 NIDPS Load

Although this topic is briefly discussed in Section 3.3, we visit it again because it is important to the design of the Load Manager, and because we gain an additional advantage with our choice of NIDPS load representation.

As there are numerous hardware and software resources an NIDPS might have, for example, CPU, disk, memory, and open files, representing an NIDPS load is a nontrivial task. For the purpose of our study, load of each NIDPS is represented as the amount of traffic it handles per second. In particular, bits per second is used to represent the NIDPS load. For example, an NIDPS load could be 40 Mbps. Although one might argue about our choice of load representation, this representation is the most common solution to the load representation problem in the context of network intrusion detection and prevention [2, 35, 43].

It is worth noting that packets per second would be a more accurate load measure for the NIDPSs than bits per sec if the Load Manager worked with NIDPSs which only do anomaly-based detection based on the packet headers. When designing the Load Manager, however, we want to have it work with general NIDPSs, which do both signature-based and anomaly-based detection. In fact, the Load Manager preserves flows to fully support signature-based detection. Because our target is general NIDPSs, which spend 80–90% of their time analyzing packet payload, we choose bits per second as a load measure for the NIDPSs.

An advantage that we gain by representing NIDPS load in this way is the direct relation between the NIDPS load and the flow load. In particular, we can easily determine the percentage contributed by a flow to the load of an NIDPS. This is because a flow load is naturally represented in terms of its traffic rate. This enables the Flow Manager to select appropriate flows to move when an NIDPS becomes overloaded.

Lastly, to estimate load of an NIDPS, we take periodic samples of the raw traffic going to that NIDPS. Then, we apply the standard Single Exponential Moving Average method [30] on the samples to estimate the load. This linear aggregation method helps us to alleviate the negative effect of traffic spikes in our estimation. In fact, it has been shown to perform very well in the context of NIDS load balancing [2], as discussed in Section 2.4.

4.3.3 Bloom Filters

In order to determine if a packet belongs to a previously assigned flow, and more importantly, to determine the correlations between a new flow and the NIDPSs, we use Bloom filters.

Background

A Bloom filter, originally introduced by Bloom [4], is a space-efficient probabilistic data structure that is used to test whether a set contains a certain element. In the context of this study, it is used to test whether an NIDPS has a certain flow, and whether an NIDPS monitors a certain *src-ip*, a certain *dst-ip*, or a certain *dst-port*.

In short, a Bloom filter is an array of m bits and has k different hash functions. Each of these functions maps an element to one of the m bits. Adding an element involves hashing it using these functions and setting all the mapped bits to 1. Likewise, querying an element involves checking if all the mapped bits are 1. If one of the bits is 0 then the element is not in the set. The followings are two fundamental properties of a Bloom filter:

1. False positives are possible, but false negatives are not.
2. Elements can be added to the set but not removed.

The primary reason why we use Bloom filters in our design is their space and time advantages. A Bloom filter has a significant space advantage over other data structures, such as hash tables, linked lists, and arrays. While these data structures require storing the elements themselves, a Bloom filter with a low error rate (about 1%) requires storing only several bits per element, independent of the element size. As a result, a Bloom filter of several KB can handle thousands of flows with low error rate. Furthermore, adding and querying an element takes a constant time $O(k)$, regardless of the number of elements the set already has. This, therefore, supports the real-time requirement of adding and querying flows.

Utilizing Bloom Filters

In order to assign packets in a way that preserve flows and correlation information, we construct two types of Bloom filter: the Flow Bloom (FB) filters and the Correlation Bloom (CB) filters. The FB filters help to preserve flows, while the CB filters help to preserve the correlation information.

Table 4.2: Correlation Bloom filters

ID	Name	Description
1	*dst-ip* CB filter	Bloom filter containing a set of *dst-ip*'s
2	*dst-port* CB filter	Bloom filter containing a set of *dst-port*'s
3	*src-ip* CB filter	Bloom filter containing a set of *src-ip*'s

First, each NIDPS has one FB filter. Whenever a new flow is assigned to an NIDPS, this NIDPS's FB filter is updated by adding the flow. At a later time, if a packet of this flow arrives, the Flow Distributor checks the FB filters of all the NIDPSs to determine which NIDPSs contain the flow. Then the packet is sent to the corresponding NIDPSs.

Secondly, for each NIDPS, three CB filters are constructed. These three filters contain sets of *dst-ip*'s, *dst-port*'s, and *src-ip*'s that the NIDPS is monitoring. These filters are termed "*dst-ip* CB filter", "*dst-port* CB filter", and "*src-ip* CB filter" respectively and are listed in Table 4.2.

When a new flow comes, if this new flow's *src-ip*, *dst-ip*, or *dst-port* belongs to one of the three corresponding CB filters of an NIDPS, then this flow is said to be *correlated* with this NIDPS and is sent to this NIDPS. The source of the traffic duplication comes from the scenarios when a flow is correlated to multiple NIDPSs. In these scenarios, the flow is sent to all of the NIDPSs that it is correlated with.

We note that only CB filters for *dst-ip*, *dst-port*, and *src-port* are constructed because these sufficiently support the grouping of flows of the default intrusions listed in Table 4.1. Nevertheless, additional CB filters might be added for the purposes of detecting other attacks and advanced detection techniques. We discuss this in detail in Section 4.3.5.

4.3.4 Correlation-Based Flow Distribution Algorithm

Here, we describe the Correlation-Based Flow Distribution (CFD) algorithm. First, we note that the CFD algorithm is flow-based, which means it sends all packets belonging to the same flow to the same NIDPS. Secondly, the CFD preserves the correlation information, which means no correlation information is lost during the distribution. Finally, the CFD makes extensive use of the FB and CB filters to distribute the traffic appropriately. The essence of this algorithm lies in the querying and updating of the filters. We provide the pseudocode of the CFD algorithm in Algorithm 3, and we describe it below.

Algorithm 3 CORRELATION-BASEDFLOWDISTRIBUTION(p)

```
1: determine if p belongs to a previously assigned flow by checking the FB filters
2: if p belongs to a previously assigned flow then
3:     distribute p to the NIDPSs that the flow was assigned to
4:     return
5: end if
6: determine the correlated NIDPSs of flow f of p by checking the CB filters
7: if f is correlated with some NIDPSs then
8:     distribute f to those NIDPSs
9:     update FB filters of those NIDPSs to contain f
10:    update CB filters of the correlated NIDPS with the lowest load to contain
       new fields of f
11: else if f is not correlated with any NIDPS then
12:    distribute f to the lowest load NIDPS
13:    update FB filter of this NIDPS to contain f
14:    update CB filters of this NIDPS to contain new fields of f
15: end if
16: return
```

When a packet comes, it could belong to an existing flow, which was assigned to an NIDPS, or it could be the first packet of a new flow. To determine this, the packet is checked against the FB filters (Line 1). If it belongs to a previously assigned flow, the FB filters also tell which NIDPSs that the flow was assigned to. The packet is then sent to those NIDPSs (Line 2–5). This case does not involve updating the FB and CB filters.

On the other hand, if the packet is the first packet of a new flow, then we examine the correlations that this flow has with all the NIDPSs. These correlations are identified by checking the CB filters (Line 6). If this new flow is correlated with some NIDPSs then it will be sent to those NIDPSs. Furthermore, the FB filters of those NIDPSs will be updated by adding this flow. In addition, if this flow introduces any new *dst-ip*, *dst-port*, and *src-ip* values, these values will be added to the corresponding CB filters of the correlated NIDPS with the lowest load (Line 7–10).

In the other case, if this new flow is not correlated with any of the NIDPSs, then it is assigned to the NIDPS with the lowest load. The FB filter of this NIDPS is updated by adding the flow, and the CB filters of this NIDPS are updated by adding the fields of the flow (Line 11–15).

We note that the CFD algorithm always takes into account the current loads of the NIDPSs while adding a new flow to an FB filter or a new field to a CB filter. In particular, it always adds a new flow or a new field to the FB filter or the CB filter of the NIDPS with the lowest load. This minimizes the probability of overloading any already heavily loaded NIDPS.

4.3.5 Customizable Correlation

By default, our Flow Distributor preserves the identical correlations given by *dst-ip*, *dst-port*, and *src-ip*. In particular, flows having the same value of any one of these fields are guaranteed to be sent to the same NIDPS. Furthermore, our Flow Distributor can support additional correlations by adding appropriate CB filters. We next describe how the Flow Distributor supports a new correlation.

Let δ be the new correlation that we want to support. That is if flows have the correlation δ then they must be sent to the same NIDPS. For example, δ could specify a group of various *dst-ip*'s, so that flows having *dst-ip*'s belonging to this group will be sent to the same NIDPS. We construct a CB filter for δ by adding the appropriate field values to an empty Bloom filter. For instance, adding the above *dst-ip*'s to a new Bloom filter. Let Δ be the CB filter of δ.

Following, we adjust the CFD algorithm to accommodate the new CB filter Δ. The new algorithm is termed Δ-CFD and is detailed in Algorithm 4. The changes made to the original CFD algorithm to produce this algorithm are as follows: At the beginning, Δ is not assigned to any particular NIDPS. When a new flow f comes, right before checking for the default CB filters, we check if f belongs to Δ. If so then we assign the CB filter Δ to the current lowest load NIDPS, and we distribute f to this NIDPS (Line 8–10). After the assignment of Δ, future flows which belong to Δ will be sent to the NIDPS which owns Δ (Line 11–12).

In the case there are multiple Δ's, i.e. when there are multiple custom correlations, the Δ's would be assigned to the NIDPSs similarly: each of the Δ's will be assigned to the lowest load NIDPS when its first matching flow arrives. After that, when a new flow comes, the already assigned Δ's have to be checked before the CB filters are checked.

By supporting additional correlations, the Flow Distributor allows for the grouping of diverse flows, which enables advanced detection techniques, such as detecting port scans by analyzing connection attempts to the unused address space and detecting various attacks derived from known attack graphs.

Algorithm 4 Δ-CORRELATION-BASEDFLOWDISTRIBUTION(p)

```
1: determine if p belongs to a previously assigned flow by checking the FB filters
2: if p belongs to a previously assigned flow then
3:     distribute p to the NIDPSs that the flow was assigned to
4:     return
5: end if
6: determine if flow f of p belongs to Δ
7: if f belongs to Δ then
8:     if Δ was not assigned then
9:         assign Δ to the NIDPS with the lowest load
10:        distribute f to this NIDPS
11:    else
12:        distribute f to the NIDPS that owns Δ
13:    end if
14: end if
15: determine the correlated NIDPSs of f by checking the CB filters
16: if f is correlated with some NIDPSs then
17:    distribute f to those NIDPSs
18:    update FB filters of those NIDPSs to contain f
19:    update CB filters of the correlated NIDPS with the lowest load to contain
       new fields of f
20: else if f is not correlated with any NIDPS then
21:    distribute f to the lowest load NIDPS
22:    update FB filter of this NIDPS to contain f
23:    update CB filters of this NIDPS to contain new fields of f
24: end if
25: return
```

4.3.6 Duplication Upper Bound

Here, we show that the amount of packet duplication created by the Flow Distributor is bounded. This duplication is needed to send flows having the same *dst-ip*, *src-ip*, or *dst-port* to the same NIDPS.

Lemma 1. *Assuming that the Bloom filters have 0% error rate, the maximum amount of packet duplication introduced by the Flow Distributor is 200%.*

Proof.

Let f be a new flow. First, when f arrives, if it is correlated with multiple NIDPSs then it is duplicated to be assigned to all of them. As a result, the amount of duplication is dependent on the number of correlated NIDPSs.

Secondly, because each new value *dst-ip*, *dst-port*, or *src-ip* of any previous flow is only added once to the corresponding CB filters of an NIDPS by the CFD algorithm, the CB filters of the NIDPSs are mutually exclusive. In other words, given a value of *dst-ip*, *dst-port*, or *src-ip*, it cannot belong to more than 1 CB filter. As a result, each of the *dst-ip*, *dst-port*, and *src-ip* of f can belong to a maximum of 1 CB filter. Thus, f could be correlated with a maximum of 3 CB filters.

Finally, given any 3 CB filters, they can belong to a maximum of 3 NIDPSs (each NIDPS owns 1 CB filter). Hence f is correlated with a maximum of 3 NIDPSs. In this case, 2 additional copies of f are produced to be sent to the 2 additional NIDPSs. Thus, the maximum amount of packet duplication is 200%. □

As discussed in Section 4.3.3, Bloom filters have false positives, which may introduce additional amount of packet duplication; however, if the Bloom filters are constructed with reasonable sizes – a Bloom filter with 1% error rate requires only several bits per element that it stores – their error rates will be small. Thus, the amount of packet duplication due to Bloom filters' errors is negligible.

It must also be noted that the duplication upper bound does not apply when the Flow Distributor implements additional correlations. This is because in order to support the additional correlations, the Flow Distributor has to duplicate more traffic.

4.4 Flow Manager

In this section, we present the design of our Flow Manager. When an NIDPS *becomes overloaded*, i.e., its load is higher than a predefined threshold T_{load}, the

Flow Manager selects from this NIDPS an optimal set of flows, which has the least *cost* and a satisfactory *benefit*, to move to the lowest load NIDPS. We first describe the need to move flows, its expense, and our approach to it. Then, we describe the cost and the benefit of moving flows. We then formalize the problem of selecting flows to move as an optimization problem, as well as provide an approximation to it. Finally, we present the Correlation-Based Flow Moving algorithm, which is the heart of the Flow Manager.

4.4.1 Moving Flows

Why Moving Flows?

Although the Flow Distributor usually distributes flows to the NIDPSs with the lowest load, there remains the risk of an NIDPS becoming overloaded due to the burstiness of network traffic. When an NIDPS is overloaded, it drops packets. As a result, intrusions whose packets are dropped might slip through the NIDPS undetected. This seriously compromises the security provided by the cluster. We argue that although moving flows involves the loss of some detection states – as we discuss subsequently – it is still more desirable than random packet drops.

In fact, the idea of dynamically moving flows has been included in several recent approaches [2, 35]. In their approaches, there is usually a threshold and when the load of an NIDPS reaches that threshold, some of the flows going to that NIDPS are moved to other NIDPSs. We have adopted the threshold-based approach; however, we consider moving flows as our last resort to guard against overload conditions, not the main method for providing load balancing, in contrast to the work of others [2, 35].

The Expense of Moving Flows

The primary expense of moving flows is the loss of detection states. Since modern NIDPSs, such as Snort [34], Bro [31], and Cisco IPS [7], are *stateful*, they keep track of the states of the connections to accurately detect and prevent intrusions. Moving flows, however, renders the states at both the source and the target NIDPSs incomplete. This affects the detection accuracy for intrusions whose effects can be seen only by monitoring the moving flows.

We note that the loss of detection states when moving flows is very difficult to avoid. In particular, in order to preserve these detection states, additional complex

hardware and software are required. We discuss one possible way to preserve these detection states in Section 5.2. Nevertheless, the loss of detection states caused by moving flows is still preferable to random packet drops. This is because random packet drops, in the worst case, might cause the loss of all detection states of the overloaded NIDPS.

Another notable expense when moving flows is the loss of correlation information. Since the correlations among flows in an NIDPS plays an important role in the anomaly-based detection of the NIDPS, this loss of correlation might decrease detection accuracy. For instance, if some early attack flows of an intrusion are moved in the middle of the intrusion, then the NIDPS will lose the correlations between those flows and the later attack flows. As a result, this intrusion might avoid the NIDPS's detection. Unfortunately, this expense was overlooked by the approaches taken by others [2, 35].

Our Approach

Our objective is to develop a flow moving mechanism that provides a desired reduction of load of the overloaded NIDPS, while minimizing the loss of correlation information. Before going into detail in subsequent sections, we briefly describe our approach here.

When there is an NIDPS whose load is over a predefined threshold, the Load Manager activates. It first collects the active flows of the overloaded NIDPS. These flows serve as the candidates for the move. It then executes an algorithm, which we describe subsequently, to select a set of flows to move. This set of flows has the desired amount of load and the minimal amount of correlation with the other flows in the NIDPS.

Finally, we note that the process of moving flows is computationally expensive since it involves collecting active flows, executing an algorithm, and so on. As a result, in order to ensure the Load Manager's real-time assignment of packets, this process must be run as a separate thread or process, utilizing a different core or processor. The purpose of this is to ensure that the Flow Manager does not interfere with the Flow Distributor except when updating the Bloom filters.

4.4.2 Cost – Correlations among Flows

Within the same NIDPS, a flow has a certain degree of correlation with other flows. This correlation is translated directly into a cost when the flow is moved to another

Table 4.3: Correlations and the intrusions that they entail

ID	Correlation	Intrusion ID
1	*none*	none
2	{*dst-ip*}	1,2,5,6
3	{*dst-port*}	3,4,5,6
4	{*src-ip*}	1,3,5
5	{*dst-ip*,*dst-port*}	1,2,3,4,5,6
6	{*dst-ip*,*src-ip*}	1,2,3,5,6
7	{*dst-port*,*src-ip*}	1,3,4,5,6
8	{*dst-ip*,*dst-port*,*dst-ip*}	1,2,3,4,5,6

NIDPS. Intuitively, if a flow has a very little correlation with other flows within the same NIDPS, then moving this flow away costs very little and vice versa. Therefore, in order to determine the cost of moving a flow, we fundamentally have to identify the degree of correlation it has with other flows.

We construct a correlation table, Table 4.3, based on the intrusions listed in Table 4.1 and the information given by the flow tuples. Table 4.3 lists all the correlations given by the 3-tuples: *dst-ip*, *dst-port*, and *src-ip*; and the intrusions each correlation entails. For example, if two flows have the same *dst-ip*, then they might belong to a DoS machine (1), a DDoS machine (2), a vertical scan (5), or a distributed vertical scan (6). Therefore, separating flows having the same *dst-ip* would negatively affect the detection accuracy of these intrusions.

We next describe how a cost for moving each flow in an NIDPS is calculated. At a particular point in time, for each flow in the NIDPS, we identify its correlations with every other flow in the same NIDPS. We note that if two flows have the same *dst-ip* and *dst-port*, then their correlation is {*dst-ip*,*dst-port*} but not {*dst-ip*} or {*dst-port*} alone and so on. Afterward, for each correlation, a score is given depending on the intrusions that the correlation entails, which is based on Table 4.3. Lastly, we total the scores to get the cost.

For example, Table 4.4 illustrates how the cost of moving a flow f_1 in an NIDPS having four flows f_1, f_2, f_3, and f_4 is calculated. Regarding the scoring, for simplicity, we give every intrusion a weight of one, so all intrusions have the same weight. One could give different weights to different intrusions, depending on their severities. For instance, if one gives intrusion 1 weight 2 and other intrusions weight 1, then the corresponding scores are 5, 4, and 6 instead of 4, 3, and 5. Thus, the cost is 15 instead of 12.

Table 4.4: Calculation of the cost of moving flow f_1 in an NIDPS having four flows f_1, f_2, f_3, and f_4

Flows	**Correlation**	**Intrusions Entailed**	**Score**
f_1 and f_2	{*dst-ip*}	1,2,5,6	4
f_1 and f_3	{*src-ip*}	1,3,5	3
f_1 and f_4	{*dst-ip*,*src-ip*}	1,2,3,5,6	5
		Cost	12

Formally, the cost of moving a flow in an NIDPS is calculated as follow:

$$c_i = \sum_{j=1, i \neq j}^{n} \sum_{k=1}^{m} w_k a_{i,j,k} \, .$$

In this equation, c_i is the cost of moving flow i, n is the number of flows in the NIDPS, m is the number of intrusions, w_k is the weight of intrusion k, and $a_{i,j,k}$ equals 1 if the correlation between flow i and flow j entails the intrusion k and 0 otherwise.

In summary, the cost of moving a flow in an NIDPS is a direct translation of how much it correlates with other flows in the same NIDPS. Our calculation of cost is tightly connected with the frequently experienced intrusions, whose detection requires analysis of correlated flows. Lastly, administrators could customize the list of intrusions, as well as the weight of each intrusion, to achieve the most suitable cost calculation for their domains.

4.4.3 Benefit – Reduced Load

While the cost of moving previously assigned flows from one NIDPS to another is the loss of correlation information, the benefit of it is the reduced load of the former NIDPS. This amount of load is discussed here.

First, given a set of flows, we need to be able to calculate its load in order to determine the benefit of moving it. The load of a flow set equals the sum of the load of its flows and the load of each flow is estimated similarly to the NIDPS load, as discussed in Section 4.3.2.

Secondly, suitable values of the benefit (reduced load) should be specified depending on the overall system load and the burstiness of the traffic. When the overall system load is high, the load of every NIDPS in the system may be high.

In this case, giving the benefit a low value is preferable since there is little room to move the load around. In fact, if the benefit value is set high, then a substantial amount of load may be moved from the overloaded NIDPS to another NIDPS. In this case, the latter NIDPS is very likely to become overloaded since its load is already high. On the contrary, when the overall system load is low, a high value of benefit becomes more viable as there is more room to move the load; however, there is a high cost associated with the high benefit value since more flows are moved.

With regard to the burstiness of the traffic, it is preferable to give the benefit a high value if the traffic is bursty. This would give the overloading NIDPS more room to deal with the traffic spikes. Otherwise, a smaller benefit value is better suited for more uniform traffic.

Finally, if the benefit value is too low, the overloading NIDPS would be more likely to become overloaded again in a short time. This is because only a small amount of load is relocated. Since moving flows is costly in terms of computation, this setting should be avoided.

4.4.4 Optimal Flow Selection Problem

The optimal flow selection problem (OFSP) is to find in an NIDPS a set of flows, which has the minimum cost of moving, i.e., the least amount of correlation with the other flows, and has a total benefit higher than or equal to a specified benefit value.

Let n be the number of flows the NIDPS has. For each flow i, denote its benefit b_i and its cost c_i. Let B be the predefined benefit value. Then, the OFSP can be formulated as a Binary Integer Programming problem:

$$\begin{aligned} \text{Minimize } & \sum_{i=1}^{n} x_i \, c_i \,, \\ \text{subject to } & \sum_{i=1}^{n} x_i \, b_i \geq B \,, \\ & x_i = 0 \text{ or } 1 \,, \quad i = 1, \cdots, n \,, \end{aligned}$$

where x_i is a binary integer: $x_i = 1$ if flow i is selected, and $x_i = 0$ otherwise. The OFSP can be solved by simply converting it to a standard 0-1 Knapsack problem (KP) through the change of variable:

$$y_i = 1 - x_i,\ i \in [1, n]\,.$$

After the change of variable, the following standard KP needs to be solved for y_i:

$$\begin{aligned} \text{Maximize } & \sum_{i=1}^{n} y_i\, c_i\,, \\ \text{subject to } & \sum_{i=1}^{n} y_i\, b_i \leq \sum_{i=1}^{n} b_i - B\,, \\ & y_i = 0 \text{ or } 1\,, \quad i = 1, \cdots, n\,. \end{aligned}$$

Here c_i and b_i could be thought of as value and weight of item i correspondingly. Also, the capacity of the knapsack is $\sum_{i=1}^{n} b_i - B$. Once the y_i's are solved, finding x_i's, the solution to the OFSP, is straightforward.

4.4.5 Heuristic Flow Selection Algorithm

Solving OFSP primarily involves solving the standard KP. The KP is a very well-known NP-hard problem, which cannot be solved in polynomial time. For the purpose of this study, because of the real-time requirement of the flow selection, we choose the popular greedy algorithm described by Martello and Toth [25] to find the y's. Afterward, we derive the x's from the y's. The details of our approximation – the Heuristic Flow Selection algorithm – can be found in Appendix A.

It is worth noting that because the KP has been studied extensively in the combinatorial optimization research community, there are other algorithms to find better approximate solutions, as well as the optimal solution [25]. However, these algorithms usually require more computational time. Nevertheless, there is room for further improvement.

4.4.6 Correlation-Based Flow Moving Algorithm

After the flows are selected, in order to complete the move, Bloom filters of the source NIDPS and the target NIDPS have to be updated. In particular, the FB and CB filters of the source NIDPS have to be reconstructed to exclude the selected flows and their fields since Bloom filters do not support removals. Meanwhile, for

Algorithm 5 CORRELATION-BASEDFLOWMOVING(S, B)

Require: An NIDPS S, a predefined benefit B

1: build the set of active flows F_{active} of S
2: F_{move} = HEURISTICFLOWSELECTION(B, F_{active})
3: T = NIDPS with the lowest load
4: Reconstruct FB filter of S to exclude flows in F_{move}
5: Reconstruct CB filters of S to include only fields which are in both the original filters and the remaining flows
6: Update FB filter of T to include flows in F_{move}
7: Update CB filter of T to include fields in the flows being moved
8: **return**

the target NIDPS, its FB filter is updated by adding the selected flows, and its CB filters are updated by adding the fields which are removed from the source NIDPS's CB filters. The complete Flow Manager's moving algorithm – Correlation-Based Flow Moving (CFM) algorithm - is presented in Algorithm 5.

4.5 Implementation

We have implemented a prototype Load Manager using the *libpcap* [22]. Besides our CFD algorithm, for the purpose of comparison, the Load Manager also has a hash-based distribution algorithm. In particular, we implemented a simple additive hash: (*src-ip* + *dst-ip* + *src-port* + *dst-port*) mod n, where n is the number of NIDPSs.

Bloom filters are constructed to be capable of handling up to 1 million elements ($q = 10^6$) with a low error rate at 1%. Specifically, each Bloom filter is of size $p = 10^7$ (bit) – about 1.2 MB – and has the optimal number of hash functions: $k = 0.7\frac{p}{q} = 7$. The optimal value k is as presented in the work of Putze et al. [33]. We construct the 7 hash functions based on an efficient and effective method introduced by Kirsch and Mitzenmacher [17]. In particular, the 7 hash functions are constructed as follows:

$$g_i = (h_1 + i\, h_2) \bmod p\,,\ i = 1 \cdots 7\,,$$
$$\text{where } h_1 = (\textit{src-ip} + \textit{src-port} + \textit{dst-ip} + \textit{dst-port}) \bmod p\,,$$
$$\text{and } h_2 = (\textit{src-ip} + \textit{src-port} - \textit{dst-ip} - \textit{dst-port}) \bmod p\,.$$

For each NIDPS, we take a sample of the traffic going to it every 1 second to estimate its load. For the Single Exponential Moving Average [30], the number of samples for the initial calculation is $s = 30$. In other words, the first estimated load is the mean of 5 initial samples of load. Also, the smoothing factor is $\alpha = \frac{2}{s+1}$. The smoothing factor is essentially the weight of the current load in the estimation:

$$\text{estimated\_load} = \alpha \cdot \text{current\_load} + (1 - \alpha) \cdot \text{last\_load}\,.$$

Flow load is estimated similarly but with $s = 5$.

Intrusions listed in Table 4.1 are initially supported and they all have weight 1 in the implementation. We also note that the Flow Manager is implemented to run as a separate thread beside the Flow Distributor. This thread only interferes with the distribution process when it updates the filters. We have done this to ensure real-time traffic distribution.

Finally, for the simulations, the prototype Load Manager is run on a system having a 2.0 GHz Intel Duo Core CPU, 2 GB RAM, and 2×1 Gbps NICs; and it is used to distribute traffic to 8 NIDPSs.

4.6 Evaluation

4.6.1 Performance

In order to evaluate the performance of the Load Manager, we used the same trace file that was previously used to evaluate the performance of the Load Balancer in Section 3.7. In particular, this is an hour trace captured from 12:00 to 13:00 on Tuesday, December 2$^{\text{nd}}$, 2003, by NLANR. This hour was one of the busiest hours of the network. During this hour, there were 200 new flows per second and 14 Mbps traffic on average [37].

We also note that in the two performance evaluations conducted, which evaluated the system overhead and the packet duplication amount, there was no flow movement. The NIDPSs' capacity was set high enough so that no flow movement was needed.

System Overhead

We conducted the first simulation to evaluate the system overhead in terms of delays that the Load Manager introduces to the packets. In this simulation, the

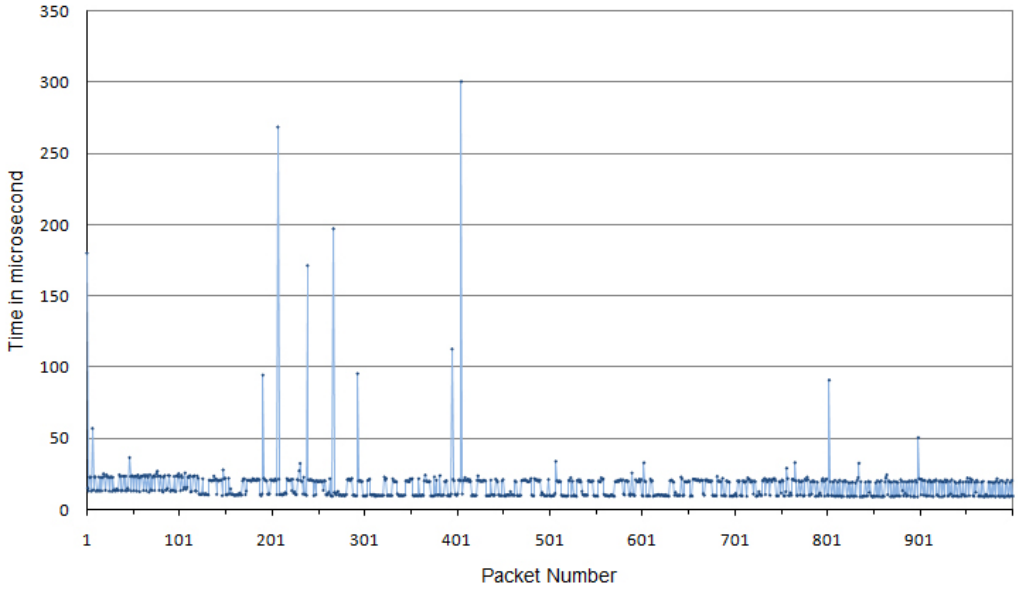

Figure 4.2: System Overhead – Packet Assignment Time

Load Manager used the trace file as its input. It read the packets off the trace file and distributed them using the CFD algorithm. We recorded the time required to run the CFD algorithm for all the packets. The recorded time was the difference between the two timestamps, one before and one after the execution of the CFD algorithm. The timestamps were taken using the Java function *System.nanoTime()*. We also note that the packets were not actually sent to the NIDPSs since it was not necessary.

Recall that when a new packet arrives, we check the FB and CB filters to assign the packet correspondingly. In addition, in some scenarios, FB and CB filters need to be updated. The overhead associated with each packet is the time required to query and update the filters. More specifically, it is the time required to calculate the different hash functions of the Bloom filters for querying and updating them.

Figure 4.2 plots the delays that the Load Manager introduces to a sampling of 1000 consecutive packets. It can be observed from this graph that there are 3 distinct classes of delays. The lowest class has about 10 microseconds (μs) delay, the middle class has about 20 μs delay, and the highest class has above 20 μs delay. By analyzing the CFD algorithm, we learned that these classes corresponded to the following three different scenarios:

1. *The new packet belongs to some previously assigned flows:* In this scenario, the Load Manager introduces the lowest amount of delay to the packet because

only the FB filters of the NIDPSs are queried and there is no update.

2. *The new packet starts a new flow but is not correlated with an NIDPS:* In this scenario, additional CB filters are queried, and the FB and CB filters of the NIDPS with the lowest load are updated.

3. *The new packet starts a new flow and is correlated with some NIDPSs:* In this scenario, additional CB filters are queried, FB filters of all correlated NIDPSs and CB filters of the correlated NIDPS with the lowest load are updated.

Overall, the delays introduced to the packets by the Load Manager are very low since they are on the order of μs. In fact, the average delay per packet for the whole trace is 17 μs (not shown on the graph). As a result, we conclude that the Load Manager has a very low overhead in terms of the delays introduced to the packets. With our prototype implementation and our hardware, the Load Manager can provide real-time traffic distribution to 8 NIDPSs for up to 500 Mbps traffic.

Duplication Amount

As discussed previously, the preservation of correlation is achieved at the cost of duplicating traffic. Although we proved in Section 4.3.6 that the amount of packet duplication is bounded by 200% – assuming that Bloom filters have no errors and that there are no additional correlations – it is valuable to learn what this amount could be in practice. Here, we conduct a simulation to measure the amount of packet duplication introduced by the Load Manager over time.

Similar to the previous simulation, in this simulation, the Load Manager read packets off the trace file then assigned them using the CFD algorithm. We measured both the number of packets taken in, P_{in} (packet), and the number of packets sent out, P_{out} (packet), by the Load Manager. Then, we used them to calculate the packet duplication amount, P_{dup} (%), as follows:

$$P_{dup} = \frac{P_{out} - P_{in}}{P_{in}} \cdot 100\%$$

Figure 4.3 plots the packet duplication amount reported at every 5-minute interval of the simulation. It can be observed from the graph that the packet duplication amount for this trace changes gently over time and has value about 115%. This amount of packet duplication, however, is a lot less than the upper bound 200%.

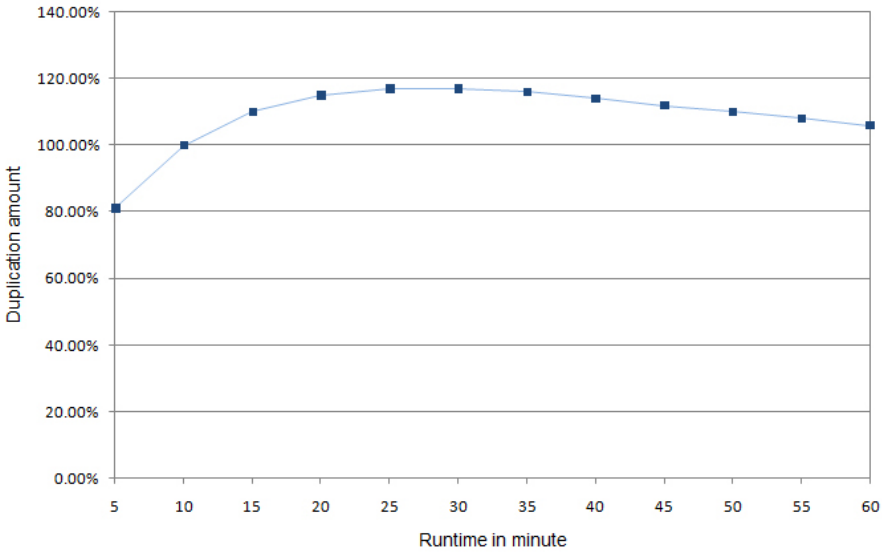

Figure 4.3: Duplication Amount over Time

We note that the amount of packet duplication might vary from trace to trace. In particular, it depends on the diversity of the traffic as well as the rate of the new connections; however, the result of this simulation gives us a valuable observation of what the amount of packet duplication could be in practice. In this particular case, the amount of packet duplication is significantly less than the established packet duplication upper bound and is about 115%.

The packet duplication is the cost of grouping flows having the same *dst-ip*, *src-ip*, or *dst-port* together. This cost, however, can be compensated by adding additional NIDPSs to a cluster because the amount of duplication does not depend on the number of NIDPSs. In particular, the more NIDPSs a cluster has, the more tolerable the duplication because the additional load caused by the duplication is distributed to more NIDPSs.

4.6.2 Security

In order to evaluate the security of the Load Manager, we conducted a variety of simulations. These simulations involved both background traffic and attack traffic. Here, we first describe the network model and the background traffic used in the simulations. We then present the evaluation results related to various DoS attacks and port scans.

Network Model and Background Traffic

The internal network that we simulated was a Class B network, and 10% of the address space was occupied. Thus, the protected internal network had about 6500 active IP addresses.

Variable background traffic was generated as follows: Source and destination addresses were randomly generated such that 65% of the flows were going to or coming from 10% of the occupied IP addresses of the internal network; the flows are randomly generated such that 80% of the load came from 20% of the flows; the traffic rate was about 100 Mbps; there were about 100 new flows per second; packets had sizes uniformly distributed from 500 to 1500 bytes; and flows' durations were uniformly distributed from 1 to 30 seconds.

DDoS Attacks

We carried out 9 simulations to evaluate how the Load Manager supports the grouping of attack flows for 9 different (D)DoS attacks. The generated attacks were all UDP flood attacks, which involved a number of attackers sending large numbers of UDP packets to random ports on a single victim. Nonetheless, the attacks varied in the number of machines used for the attacks, the attack duration, and the rate of the attack traffic. Meanwhile, we used the same background traffic described above for all 9 simulations. For each simulation, both the CFD algorithm and the hash-based algorithm were used, and we recorded the assignment of the attack packets. We also note that in these simulations, the NIDPSs' capacity was set high enough so that there was no flow movement.

Table 4.5 lists the generated (D)DoS attacks and the NIDPSs which received 100% of the attack packets when the CFD algorithm was used. The results of the simulations showed that for each of the attacks, when the CFD algorithm was used, there was always at least one NIDPS which received all the attack packets.

Figure 4.4 and Figure 4.5 detail the distribution of the attack packets of two of the 9 generated (D)DoS attacks when both of the algorithms were used. The plotted attacks are 2 and 8. It can be observed from the graphs that when the hash-based algorithm was used, the attack packets were fairly evenly distributed to all of the NIDPSs. Thus, there was no NIDPS which could receive all of the attack packets.

In conclusion, the evaluation results validate that the Load Manager successfully grouped the flows of the (D)DoS attacks together. For each of the simulated attacks,

Table 4.5: Various simulated (D)DoS attacks

DDoS ID	No. of Attackers	No. of Victims	Duration (sec)	Rate (pkt/sec)	NIDPSs receiving 100% atk-packet
1	1	1	5	5000	1, 5
2	1	1	30	10000	1, 5
3	1	1	60	20000	1, 5
4	500	1	5	5000	1
5	500	1	30	10000	1
6	500	1	60	20000	1
7	1000	1	5	5000	1
8	1000	1	30	10000	1
9	1000	1	60	20000	1

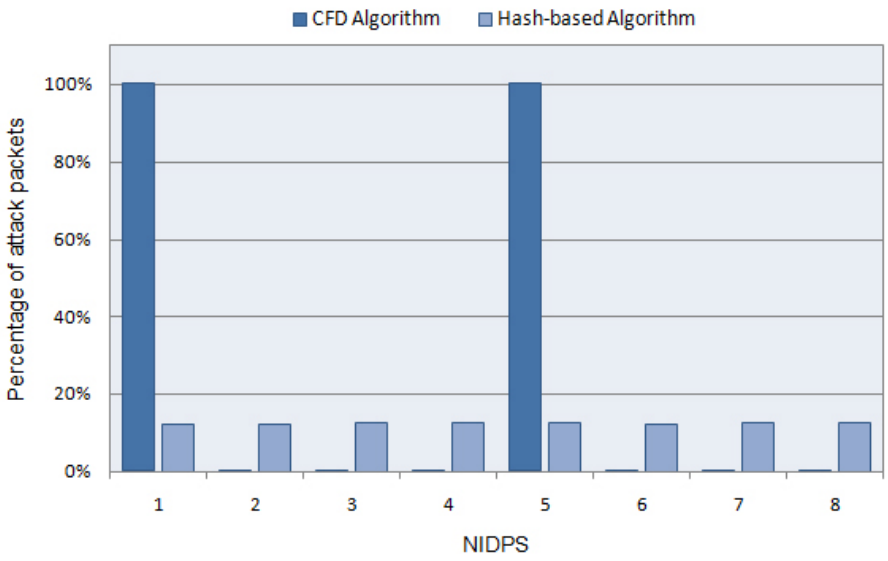

Figure 4.4: The Distribution of Attack Packets of the DDoS 2

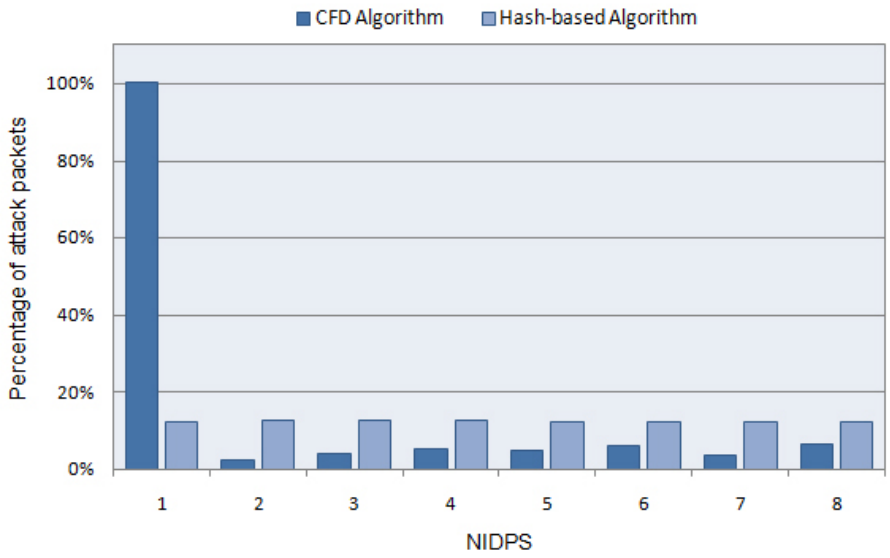

Figure 4.5: The Distribution of Packets of DDoS 8

when the CFD algorithm was used, the Load Manager guaranteed that there was at least one NIDPS which received all packets of the attack. Thus, this NIDPS had all the available correlation information to detect and prevent this attack.

Port Scans

We conducted 8 simulations with different port scans to evaluate how the Load Manager supports the grouping of the scans' flows. Similar to the DDoS evaluation, we use the same background traffic in all the simulations. Furthermore, all of the scans were TCP SYN scans. However, the scans varied in the number of scanning hosts, the scan duration, the number of target machines, and the number of target ports. For each simulation, both the CFD algorithm and the hash-based algorithm were used, the target machines and the target ports were randomly selected, and we recorded the assignment of every scan packet. We also note that in these simulations, the NIDPSs' capacity was set high enough so that there was no flow movement.

Table 4.6 lists the 8 scans that we simulated, together with the NIDPSs which received 100% of the scan packets when the CFD algorithm was used. The last column of the table shows that, for each scan, there was at least one NIDPS which received 100% of the scan packets.

Figure 4.6 further shows the distribution of the packets of scan 7 when both the CFD algorithm and the hash-based algorithm were used. Similar to the DDoS

Table 4.6: Various simulated port scans

Scan ID	Number of Scan Hosts	Duration (sec)	Number of Target Machines	Number of Target Ports	NIDPSs receiving 100% atk-packet
1	1	5	1	1024	1,5
2	1	60	1	1024	1
3	100	5	1	1024	1
4	100	60	1	1024	4
5	1	5	1000	1	1
6	1	60	1000	1	1
7	100	5	1000	1	5
8	100	60	1000	1	4

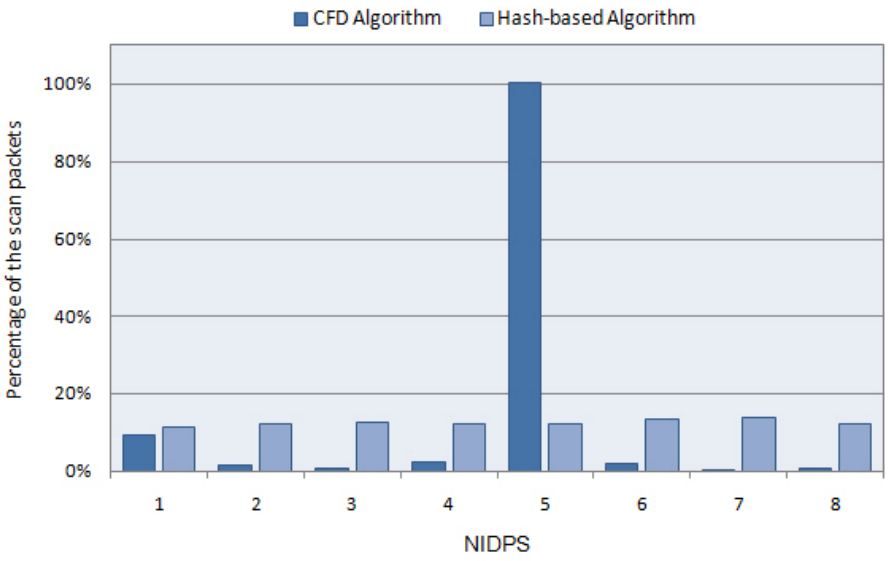

Figure 4.6: The Distribution of Packets of Scan 7

evaluation result, when the hash-based algorithm was used, the packets were scattered; thus, each NIDPS only received a small number of scan packets. On the other hand, when the CFD algorithm was used, NIDPS 5 received 100% of the scan packets.

It is worth noting that with regard to sophisticated scans, which use many scanning hosts to scan many ports of many victims over a long duration, the Load Manager might not be able to group all flows of these scans together due to the diversity of the scans' flows. Nevertheless, these scans might be detected by monitoring the number of connection attempts to unused address space of the protected networks. In this case, custom correlations might help to group flows of these connection attempts together, thereby supporting the detection of such scans.

In summary, the results of this evaluation showed that the Load Manager successfully grouped flows of the port scans together. For each of the simulated scans, when the CFD algorithm was used, the Load Manager guaranteed that there was at least one NIDPS which received 100% of the scan packets. Therefore, if this NIDPS was capable of detecting the scan then it would detect the scan successfully.

4.6.3 Flow Movement

Effect of Flow Movement on NIDPSs' Loads

We conducted two simulations to evaluate the effect of flow movement on loads of the NIDPSs when the cluster is lightly loaded. In both simulations, variable background traffic as described in Section 4.6.2 was generated. Recall that the background traffic was about 100 Mbps. The Load Manager received the traffic and assigned it to the NIDPSs using the CFD algorithm. Both the simulations were run five times with different seeds for the random traffic generator. During the simulations, we recorded the NIDPSs' load.

In the first simulation, there was no flow movement. The NIDPSs' capacity was set high enough so that no flow movement was needed. In particular, both the NIDPSs' capacity and the Flow Manager's activation threshold were set to 100 Mbps. In the second simulation, there were flow movements because loads of some NIDPSs were larger than the Flow Manager's activation threshold. Specifically, the NIDPSs' capacity was set to 50 Mbps and the threshold was set to 30 Mbps. With regard to the flow movement, the benefit (reduced load) was set to 10 Mbps, and the Flow Manager checked the NIDPSs' loads every 10 seconds to move flows if necessary.

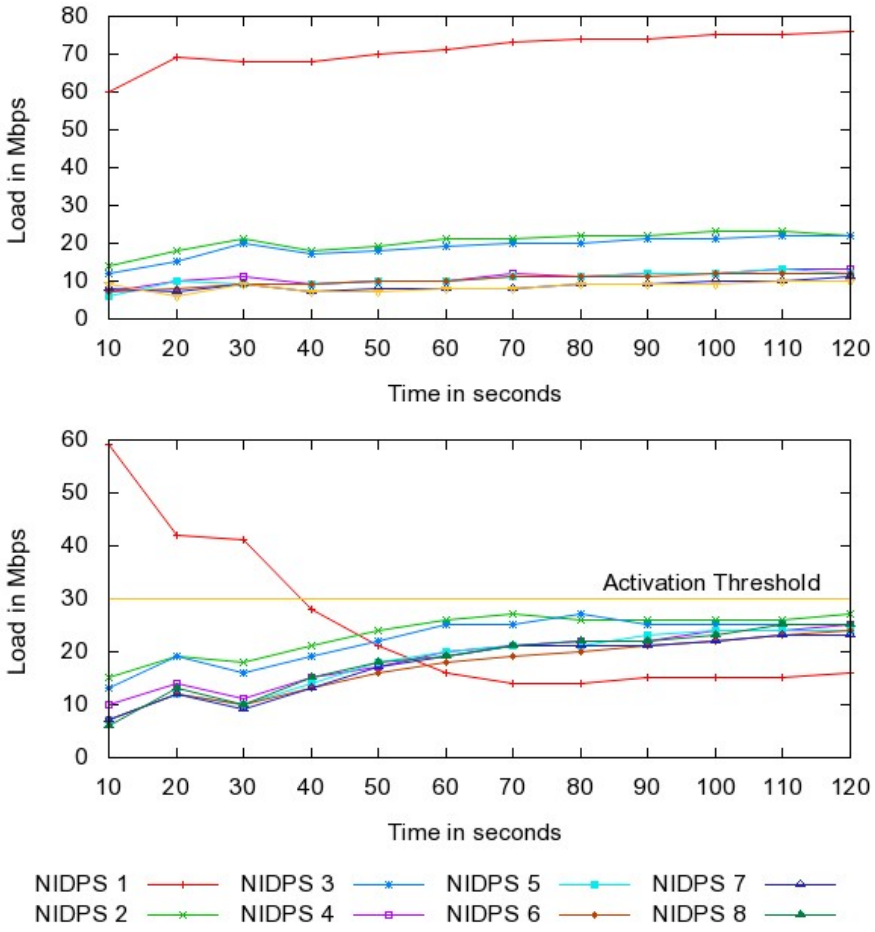

Figure 4.7: Loads of the NIDPSs over Time without Flow Movement (top) and with Flow Movement (bottom)

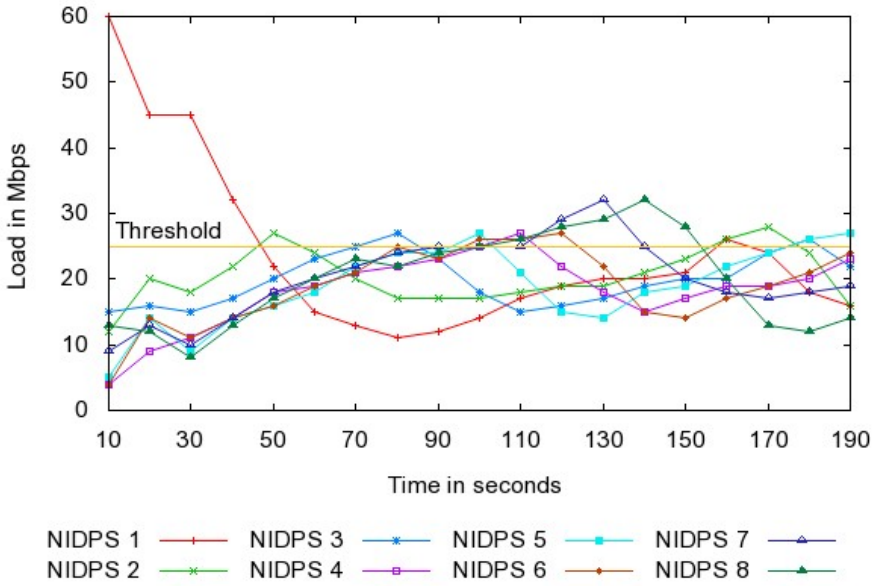

Figure 4.8: Loads of the NIDPSs over Time with Flow Movement

Figure 4.7 plots average values of loads of the NIDPSs in both simulations. We note that the x-axis does not begin at zero but begins at time 10 seconds. During the first 10 seconds, the system establishes an initial load. On the one hand, it can be observed from the top graph that when there was no flow movement, the loads of the NIDPSs were significantly unbalanced. In particular, the load of the first NIDPS was noticeably higher than the rest. The first NIDPS could have dropped packets due to being overloaded if the NIDPSs' capacity had been lower than 70 Mbps. On the other hand, it can be observed from the bottom graph that when there were flow movements, the loads of the NIDPS were more balanced. In particular, the loads of the NIDPSs were kept below the specified threshold of 30 Mbps.

In conclusion, flow movements help to balance NIDPSs' load. In particular, if overall traffic load is not high and an activation threshold is reasonably chosen then the flow movements can help to keep the NIDPSs' loads below this threshold.

Effect of Flow Movement on Port Scans

We conducted 8 simulations to evaluate the effect of flow movement on the detection accuracy of port scans when overall load of the cluster was high.

In these simulations, the same background traffic as the previous evaluation was generated. There were flow movements during the simulations. The Flow Manager's

activation threshold was set to 25 Mbps, which was less than the threshold used in the previous evaluation. The reason was that we wanted flow movements to occur more often during the simulations. With regard to the moving, the benefit (reduced load) was set to 10 Mbps, and the Flow Manager checked the NIDPSs' loads every 10 seconds to move flows if necessary.

Figure 4.8 plots the average values of the NIDPSs' loads of five runs with different seeds for the random traffic generator. As in Figure 4.7, the x-axis begins at time 10 seconds. It can be observed from this graph that with the activation threshold set to 25 Mbps, flow movements occurred almost every 10 seconds. Also, since some NIDPSs usually had loads larger than the threshold, the cluster was considered heavily loaded.

Within this setting, we generated 8 different port scans, which were identical to the ones presented in Section 4.6.2. The scans all started at second 60. For each simulation, the CFD algorithm was used to distribute the traffic and we recorded the assignment of every scan packet.

The result of this evaluation was that for each of the generated scans, there was always at least one NIDPS which received 100% of the scan flows. This result indicated that the flow movements did not decrease the anomaly-based detection accuracy of port scans.

If flows had been randomly chosen to move from an overloaded NIDPS to another, flows belonging to the same scan could have been distributed to different NIDPSs, resulting in a decrease of detection accuracy. This evaluation result, however, indicated that the Flow Manager, which always selects flows having the least amount of correlation to move, did not select flows from the same scan to move.

In summary, the above two evaluation results show that flow movement helps to balance the loads of the NIDPSs when the cluster is lightly loaded and it does not negatively affect the grouping of scan flows. Nonetheless, whether the flow movement can help to balance the loads of the NIDPSs when the cluster is heavily loaded needs further investigation. We plan to examine this in future work.

4.7 Summary

In this chapter, we present the design, implementation, and evaluation of the Load Manager. As our second solution to the traffic distribution problem, the Load

Manager's first objective is to group flows having the same *dst-ip*, *src-ip*, and *dst-port* together, which preserves the anomaly-based detection accuracy of DDoS attacks and port scans, and its second objective is to prevent the NIDPSs from being overloaded.

The Load Manager consists of two main components: the Flow Distributor and the Flow Manager. The Correlation-Based Flow Distribution algorithm is used by the Flow Distributor to intelligently distribute the traffic to the NIDPSs in a way such that: (1) the identical correlations given by addresses and port numbers are preserved and (2) the risk of overloading any NIDPS is reduced. On the other hand, the Flow Manager uses the Correlation-Based Flow Moving algorithm to move flows to prevent overload conditions of the NIDPSs. Together, the Flow Distributor and the Flow Manager provide the key attributes of the Load Manager: identical correlations preservation, customizable correlation, and dynamic load distribution.

We have implemented a prototype Load Manager and evaluated it using simulation. The evaluation results show that the Load Manager has low overhead in terms of the delays introduced to the packets and that the amount of traffic duplication is significantly lower than the established upper bound. Most importantly, the evaluation results validate that the Load Manager successfully sends flows of a variety of DDoS attacks and port scans to the same NIDPS, thereby preserving the anomaly-based detection accuracy of the NIDPSs.

Chapter 5

Conclusion and Future Work

To address the overload conditions brought up by the increasing network traffic volume, recent literature in the network intrusion detection and prevention field has proposed the use of clusters of NIDPSs [2, 18, 35, 40, 43]. When using clusters of NIDPSs, distributing the network traffic to the NIDPSs plays a very important role in both performance and security of the clusters. In particular, on the one hand, uneven load distribution greatly increases the chance of overloading an NIDPS even when the loads of the other NIDPSs are low. On the other hand, since simple traffic distributions cause high loss of correlation information, some intrusions might slip through the NIDPSs undetected. These intrusions are those that require the analysis of correlated flows, such as DDoS attacks and port scans.

Clusters proposed in recent research often use simple traffic distribution schemes. Some of these schemes do not provide load balancing [18, 43, 40], which results in a high risk of overloading the NIDPSs. More importantly, none of the previously proposed approaches takes into account the traffic correlation in their traffic distributions. As a result, the security of the cluster might be compromised because of the substantial loss of correlation information.

In this thesis, we propose two novel systems: the Correlation-Based Load Balancer and the Correlation-Based Load Manager as two different solutions to the NIDPS traffic distribution problem. The Load Balancer and the Load Manager both consider the current loads of the NIDPSs when distributing traffic to provide fine-grained load balancing and dynamic load distribution, respectively. More importantly, both systems consider traffic correlation in their distributions, thereby significantly reducing the loss of correlation information during their distribution of traffic.

We have implemented prototypes of both systems and evaluated them using extensive simulations and real traffic traces. Overall, the evaluation results show that both systems have low overhead in terms of the delays introduced to the packets. More importantly, compared to the naive hash-based distribution, the Load Balancer significantly improves the anomaly-based detection accuracy of DDoS attacks and port scans – the two major attacks that require the analysis of correlated flows – meanwhile, the Load Manager successfully maintains the anomaly-based detection accuracy of these two major attacks of the NIDPSs.

5.1 Summary of Contributions

This thesis contributes to the network intrusion detection and prevention research community two novel traffic distribution systems for clusters of NIDPSs:

1. The **Correlation-Based Load Balancer** is flow-based and delivers the following features:

   - *Fine-grained Load Balancing:* The difference between loads of the NIDPSs is kept within a specified bound.
   - *Anomaly-based Detection and Prevention Support:* The loss of correlation information is minimized, so the accuracy of the anomaly-based detection is significantly improved.
   - *Configurable Security:* Various configurations are available to favor security, i.e., reduced loss of correlation information, over performance, i.e., load balancing.

2. The **Correlation-Based Load Manager** is also flow-based and offers the following features:

   - *Identical Correlation Preservation:* Flows having the same *dst-ip*, *src-ip*, or *dst-port* are guaranteed to be sent to the same NIDPS. This maintains the anomaly-based detection accuracy of DDoS attacks and port scans.
   - *Dynamic Load Distribution:* The chance of overloading any NIDPS when the loads of the other NIDPSs are low is greatly reduced. Overload conditions are prevented.
   - *Customizable Correlation:* Custom correlations can be added, which supports the grouping of various flows of interest. Thus, advanced detection techniques requiring the grouping of diverse flows can be used.

5.2 Future Directions

There are several valuable research topics that follow from our work:

- *Duplication amount:* When a flow is distributed using the BLB algorithm of the Load Balancer, it might be duplicated up to F times. It would be interesting to learn which values of F would give the best trade-off between the loss of correlation information and the load balancing. These values of F might have some important connections with the Load Manager's duplication amount upper bound.

- *Number of maintained clusters:* The clusters play a very important role in the BLB algorithm of the Load Balancer since they characterize the received traffic. It would be interesting to identify the lowest number of clusters which can still characterize the traffic well enough to support the detection of the major intrusions. Because a high number of clusters causes a high system overhead, the smaller the number of clusters, the better the performance of the Load Balancer.

- *Detection-state preservation:* As we discussed in Section 4.4.1, the primary expense of moving flows is the loss of some detection states. With the emerging virtual machine (VM) technology, it is possible to live-migrate a VM from one physical machine to another. It would be interesting to see how the VM technology can help to move flows from an overloaded NIDPS to another NIDPS while preserving the detection states.

- *Complex correlations:* With the space and time advantage of Bloom filters, the Load Manager is capable of supporting potentially complex correlations, as long as they are compressible into Bloom filters. It would be interesting to develop a systematic way of compressing correlations into Bloom filters.

Appendix A

Heuristic Flow Selection Algorithm

This section provides details of the Heuristic Flow Selection algorithm, which is discussed in Section 4.4.5. This algorithm is presented in Algorithm 6. The inputs of this algorithm are the specified benefit B and the set of active flows F of the overloaded NIDPS. The flows are assumed to be ordered according to decreasing values of the cost per unit benefit, i.e.:

$$\frac{c_1}{b_1} \geq \frac{c_2}{b_2} \geq \cdots \geq \frac{c_n}{b_n}.$$

In this algorithm, we first initialize 3 variables B^*, C, and i_m (Line 1–7): B^* is the capacity of the knapsack, C is the current cost (or value), and i_m is the index of the highest-cost flow. Next, we use the greedy algorithm proposed by Martello and Toth [25] to find the x_i's (Line 8–20). We note that although the original algorithm would find the y_i's, here we directly derive the values of the x_i's instead. Finally, to avoid the worst case scenario, we compare the resulting total cost (total item value) with the cost of the highest-cost flow (value of the highest value item) (Line 21–26). The highest-cost flow will be solely selected if it has a higher cost. Similar to the greedy algorithm, our algorithm has the worst-case performance ratio $\frac{1}{2}$, i.e., in the worst case scenario, total value of the items of the solution of this algorithm is $\frac{1}{2}$ of the optimal solution's. Also, the time complexity is $O(n)$, plus $O(n \log n)$ to sort the flows in advance.

Algorithm 6 HEURISTICFLOWSELECTION(B, F)

Require: $\frac{c_1}{b_1} \geq \cdots \geq \frac{c_n}{b_n}$, b_i and c_i are the benefit and cost of flow f_i in F.

Ensure: A set containing the selected flows

1: $B^* = 0$
2: $C = 0$
3: $i_m = 1$
4: **for** $i = 1$ to n **do**
5: $\quad B^* = B^* + b_i$
6: **end for**
7: $B^* = B^* - B$
8: $B_{remain} = B^*$
9: **for** $i = 1$ to n **do**
10: $\quad$ **if** $b_i > B_{remain}$ **then**
11: $\quad\quad x_i = 1$ *//select flow*
12: $\quad$ **else**
13: $\quad\quad x_i = 0$ *//do not select flow*
14: $\quad\quad B_{remain} = B_{remain} - b_i$
15: $\quad\quad C = C + c_i$
16: $\quad$ **end if**
17: $\quad$ **if** $c_i > c_{i_m}$ and $b_{i_m} \leq B^*$ **then**
18: $\quad\quad i_m = i$
19: $\quad$ **end if**
20: **end for**
21: **if** $c_{i_m} > C$ **then**
22: $\quad$ **for** $i = 1$ to n **do**
23: $\quad\quad x_i = 1$ *//select flow*
24: $\quad$ **end for**
25: $\quad x_{i_m} = 0$ *//do not select flow*
26: **end if**
27: $M = \emptyset$ *//return set*
28: **for** $i = 1$ to n **do**
29: $\quad$ **if** $x_i = 1$ **then**
30: $\quad\quad M = M \cup \{f_i\}$
31: $\quad$ **end if**
32: **end for**
33: **return** M

References

[1] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A framework for clustering evolving data streams. In *VLDB '03: Proceedings of the 29th International Conference on Very Large Data Bases*, pages 81–92, Berlin, Germany, 2003. VLDB Endowment. 21

[2] Mauro Andreolini, Sara Casolari, Michele Colajanni, and Mirco 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, Cambridge, MA, USA, July 2007. 3, 10, 12, 17, 47, 54, 55, 74

[3] Attack list, http://www.ll.mit.edu/IST/ideval/docs/1998/attacks.html.

[4] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970. 48

[5] Brian Caswell and Jay Beale. *Snort 2.1 Intrusion Detection*. Syngress, 2 edition, June 2004.

[6] Brian Caswell, Jay Beale, and Andrew Baker. *Snort IDS and IPS Toolkit*. Syngress, 2007. 34

[7] Cisco NIPS, http://tinyurl.com/2nbxbx. 2, 11, 36, 54

[8] Evan Cooke, Farnam Jahanian, and Danny Mcpherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, pages 39–44, MIT, Stata Center, Cambridge, MA, USA, June 2005.

[9] H. Debar, M. Dacier, and A. Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999. 1, 2

[10] Holger Dreger, Anja Feldmann, Vern Paxson, and Robin 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. 3

[11] Robert Durst, Terrence Champion, Brian Witten, Eric Miller, and Luigi Spagnuolo. Testing and evaluating computer intrusion detection systems. *Commun. ACM*, 42(7):53–61, 1999.

[12] Jose M. Gonzalez, Vern Paxson, and Nicholas 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.

[13] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering data streams: Theory and practice. *IEEE Trans. on Knowl. and Data Eng.*, 15(3):515–528, 2003.

[14] H. Hamed, A. El-Atawy, and E. Al-Shaer. Adaptive statistical optimization techniques for firewall packet filtering. In *INFOCOM '06: Proceedings of the 25th IEEE International Conference on Computer Communications*, pages 1–12, Barcelona, Catalunya, Spain, April 2006.

[15] Jaeyeon Jung, Vern Paxson, Arthur W. Berger, and Hari Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *SP '04: Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 211–225, Oakland, California, USA, May 2004. 36, 46

[16] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, March 1990. 21

[17] Adam Kirsch and Michael Mitzenmacher. Less hashing, same performance: building a better Bloom filter. In *ESA'06: Proceedings of the 14th conference on Annual European Symposium*, pages 456–467, London, UK, 2006. Springer-Verlag. 60

[18] Christopher Kruegel, Fredrik Valeur, Giovanni Vigna, and Richard Kemmerer. Stateful intrusion detection for high-speed networks. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 285–294, Washington, DC, USA, 2002. IEEE Computer Society. 3, 7, 12, 43, 74

[19] Anh Le, Ehab Al-Shaer, and Raouf Boutaba. On optimizing load balancing of intrusion detection and prevention systems. In *Proceedings of the IEEE INFOCOM Computer Communications Workshops, 2008*, pages 1–6, Phoenix, AZ, USA, April 2008.

[20] Anh Le, Ehab Al-Shaer, and Raouf Boutaba. On optimizing load balancing of intrusion detection and prevention systems. In *SecureComm '08: To appear in the Proceedings of the 4th ACM International Conference on Security and Privacy in Communication Networks*, Istanbul, Turkey, Sep 2008.

[21] Zhichun Li, Yan Chen, and Aaron Beach. Towards scalable and robust distributed intrusion alert fusion with good load balancing. In *LSAD '06: Proceedings of the 2006 SIGCOMM Workshop on Large-Scale Attack Defense*, pages 115–122, New York, NY, USA, 2006. ACM. 46

[22] Libpcap, http://www.tcpdump.org/. 27, 60

[23] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. Analysis and results of the 1999 DARPA off-line intrusion detection evaluation. In *Recent Advances in Intrusion Detection*, pages 162–182, 2000.

[24] R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, and M.A. 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, Hilton Head, South Carolina, USA, 2000.

[25] Silvano Martello and Paolo Toth. *Knapsack problems: Algorithms and computer implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990. 59, 77

[26] John 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.

[27] Jelena Mirkovic and Peter Reiher. A taxonomy of DDoS attack and DDoS defense mechanisms. *SIGCOMM Comput. Commun. Rev.*, 34(2):39–53, 2004. 25, 46

[28] Network Mapper, http://insecure.org/nmap/. 36

[29] Network traces, http://pma.nlanr.net/Special/auck8.html. 28

[30] NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc431.htm. 17, 31, 47, 61

[31] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23–24):2435–2463, 1999. 2, 11, 27, 36, 54

[32] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Proactively detecting distributed denial of service attacks using source IP address monitoring. In *Proceedings of the 3rd International IFIP-TC6 Networking Conference*, pages 771–782, Athens, Greece, 2004. 28

[33] Felix Putze, Peter Sanders, and Johannes Singler. Cache-, hash- and space-efficient Bloom filters. In *WEA '07: Proceedings of the 6th Workshop on Experimental Algorithms*, volume 4525 of *Lecture Notes in Computer Science*, pages 108–121. Springer, 2007. 60

[34] Martin 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. 2, 4, 34, 54

[35] Lambert Schaelicke, Kyle Wheeler, and Curt Freeland. SPANIDS: A scalable network intrusion detection loadbalancer. In *CF '05: Proceedings of the 2nd Conference on Computing Frontiers*, pages 315–322, New York, NY, USA, 2005. ACM. 2, 3, 8, 12, 17, 27, 46, 47, 54, 55, 74

[36] Stephen Specht and Ruby Lee. Distributed denial of service: Taxonomies of attacks, tools, and countermeasures. In *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems, 2004 International Workshop on Security in Parallel and Distributed Systems*, pages 543–550, San Francisco, CA, USA, September 2004. 32

[37] Statistics of the trace captured from 12:00 to 13:00, 12/02/2003, http://pma.nlanr.net/Special/auck8/20031202-120000.html. 29, 61

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

[39] Matthias Vallentin. Transparent load-balancing for network intrusion detection systems, November 2006. Bachelor's Thesis, Technical University Munich, Germany. 3, 12

[40] Matthias Vallentin, Robin Sommer, Jason Lee, Craig Leres, Vern Paxson, and Brian Tierney. The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. In *Proceedings of the Symposium on RAID '07*, pages 107–126, Queensland, Australia, September 2007. 3, 11, 12, 36, 46, 74

[41] Nicholas Weaver, Vern Paxson, Stuart Staniford, and Robert 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. 46

[42] Patrick Wheeler and Errin Fulp. A taxonomy of parallel techniques for intrusion detection. In *ACM-SE 45: Proceedings of the 45th Annual Southeast Regional Conference*, pages 278–282, New York, NY, USA, 2007. ACM.

[43] Konstantinos Xinidis, Ioannis Charitakis, Spiros Antonatos, Kostas G. Anagnostakis, and Evangelos P. Markatos. An active splitter architecture for intrusion detection and prevention. *IEEE Trans. Dependable Secur. Comput.*, 3(1):31, 2006. 3, 9, 12, 17, 27, 47, 74

Publications

Conference and Workshop Papers

1. Anh Le, Ehab Al-Shaer, and Raouf Boutaba, "*On optimizing load balancing of intrusion detection and prevention systems*", In: Proceedings of the IEEE INFOCOM Computer Communications Workshops 2008, pages 1–6, Phoenix, AZ, USA, April 13–18, 2008.

2. Anh Le, Ehab Al-Shaer, and Raouf Boutaba, "*Correlation-based load balancing for network intrusion detection and prevention systems*", To appear in: Proceedings of the 4th ACM International Conference on Security and Privacy in Communication Networks, Istanbul, Turkey, September 22–25, 2008.

3. Anh Le, Ehab Al-Shaer, and Raouf Boutaba, "*On optimizing traffic distribution for clusters of network intrusion detection and prevention systems*" (in preparation).