On Optimizing Load Balancing of Intrusion Detection and Prevention Systems

Anh Le*, Ehab Al-Shaer*†, Raouf Boutaba*

* David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, ON, N2L 3G1, Canada
† School of Computer Science, Telecommunications and Information Systems
DePaul University, Chicago, IL 60604, USA
a4le@uwaterloo.ca, ehab@cs.depaul.edu, rboutaba@uwaterloo.ca

***Abstract*—In large-scale enterprise networks, multiple network intrusion detection and prevention systems are used to provide high quality protection. A challenging problem is to maintain load balancing of the systems, while minimizing the loss of information due to distributing traffic. Because anomaly-based detection and prevention of some intrusions require a single system to analyze attack-correlated flows, this loss of information might severely reduce the accuracy of the detection and prevention.**

In this paper, we address this problem by first formalizing the load balancing problem as an optimization problem, considering both the load variance and the information loss. We then present our Benefit-based Load Balancing (BLB) algorithm as a solution to the problem. We have implemented a prototype load-balancer with BLB algorithm and evaluated it against a DDoS attack. Our results show that the load-balancer significantly improves the detection accuracy, while being able to keep the load of the systems close within a desired bound.

I. Introduction

Nowadays, as people rely heavily on computer systems to conduct businesses and operate mission critical devices, effects of viruses and worms are much more disastrous. One way to combat the spread of viruses and worms is by using network intrusion detection and prevention systems (NIDPSs). An NIDPS is usually placed at an edge of a network, between its internal and external networks. The NIDPS monitors all packets coming in from the external network and going out of the internal network to detect and prevent intrusions.

Since network traffic speed and volume are increasing with an exponential rate [1], and NIDPSs are becoming more complex, a critical problem with using a single NIDPS is that it could be easily overloaded. When overloaded, the NIDPS eventually has to drop packets. Dropping packets compromises the security offered by the NIDPS, because some intrusions can not be detected if their related packets are dropped.

Using clusters of NIDPSs offers the most affordable and scalable solution to the above problem [1], [2]. When a cluster of NIDPSs is used in a network, keeping load evenly distributed among the NIDPSs is crucial because even load distribution, most importantly, provides protection: there will be less likely an overloaded system. Additionally, it allows for better traffic engineering which improves the network's quality of service.

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

In this paper, we propose a novel approach to distribute traffic to the NIDPSs. First, we formalize the load balancing problem as an optimization problem, considering both the load variance and the information loss. We then present our Benefit-based Load Balancing (BLB) algorithm as a solution to the problem. This algorithm uses on-line clustering technique to distribute flows in real-time such that: (1) Correlated flows are grouped together at a single NIDPS to minimize the information loss, and (2) load of NIDPSs are kept close within a specified bound.

We have implemented a prototype load-balancer with BLB algorithm and evaluated it against a DDoS attack. Our results show that the load-balancer significantly improves the detection accuracy while keeping the load of the systems close within a desired bound. Fig. 1 shows how our load-balancer fits into the network topology.

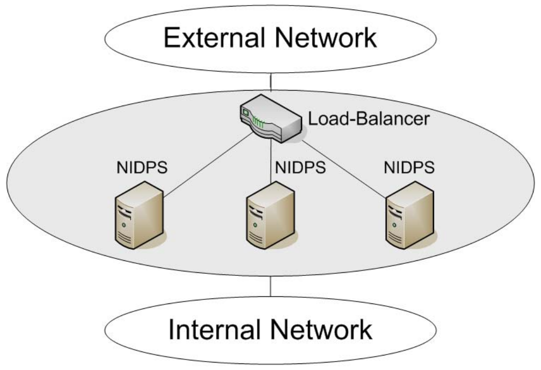

Fig. 1. Placement of the NIDPS Load-Balancer

The rest of this paper is organized as follows: §II describes the problem statement and approach overview. In §III we formalize the problem. §IV provides an approximation algorithm to solve the flow assignment optimization problem. In §V we explain the on-line clustering technique and describe the BLB algorithm. §VI discusses the correlation between flows. In §VII we describe the implementation and evaluation results. §VIII presents related work. Finally, we conclude in §IX.

II. Problem Statement and Approach Overview

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

Approach Overview. First, we introduce *cluster* to better capture correlations of flows, and more importantly, to provide some structures to the flows. Secondly, *benefit* is introduced as a mean to measure the correlation between a new flow and previous flows. Finally, our approach can be summarized as follow: 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.

III. Problem Formalization

Formalization. At time t, let n be the number of NIDPSs and m be the number of clusters. The mapping

Listing 1 Flow Assignment Optimization Problem $\mathbf{P}$

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

between NIDPSs and clusters is one-to-many. For each NIDPS i ($i \in [1,n]$), let $\overrightarrow{G_i}$ be a vector of size m whose j^{th} element ($j \in [1,m]$) is either 1 if NIDPS i owns cluster j or 0 otherwise. Now let f be the new flow. Assigning f to a cluster j gives a benefit B_j. Essentially, this benefit reflects how much correlation there is between f and flows in cluster j. Following, let $\overrightarrow{B}$ be a vector of size m whose j^{th} element is the benefit B_j.

We use L_i to denote the load of NIDPS i in the system at the current time. L_i's could be either indirectly estimated or directly updated by using periodical SNMP queries or time-based traps. Next, let μ be the current average load of all NIDPSs and V be the upper bound for the new variance after the assignment. Subsequently, let L_f be the predicted load of the new flow. Afterward, let F be the maximum number of NIDPSs which f could be assigned to concurrently, and finally, let $\overrightarrow{X}$ be the solution vector of size m whose j^{th} element is either 1 if f is going to be assigned to cluster j or 0 otherwise.

In order to determine which clusters to assign f to, we solve the optimization problem $\mathbf{P}$ specified in listing 1. In the case that there is no solution to $\mathbf{P}$, a new cluster is created with f as its centroid. This new cluster is then assigned to an NIDPS with the lowest load.

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

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

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

1) **Relaxing variance constraint:** Setting V high loosens the load balancing requirement; thus, higher benefit might be achieved. To the extreme, V could be set high enough so that load balancing is completely ignored. In this case, traffic is distributed based on only benefit, resulting in the use of only one NIDPS, which might be a desirable setting when the traffic load is low.
2) **Duplicating flows:** A high value of F reduces the loss of information because flows are duplicated up to F times to be sent to NIDPSs. However, duplication of flows consumes system resources like bandwidth and CPU load; therefore, it must be used selectively.
3) **Threshold-based load distribution:** Load balancing requirement could be replaced by threshold-based requirement, which is easier to satisfy. The later requirement gives more room to obtain higher benefit, hence lower information loss. Threshold-based load distribution could be readily achieved by replacing constraint (4) with a simpler constraint: (4*) $L_i + L_f(\overrightarrow{X} \cdot \overrightarrow{G_i}) < th_l \quad, \forall i \in [1, n]$

IV. Solving Optimization Problem P

P is very hard to solve since its decision version is an NP-complete problem. The proof of NP-completeness can be found in our technical report [3]. Because of the real-time requirement of the flow assignment, we propose a greedy-based approximation algorithm **SolveP** to solve the problem **P** in linear time. Listing 2 shows the algorithm. **SolveP** searches for a cluster, which gives maximum benefit and satisfies the constraints, at a time. This could be done in $O(m)$ time, and it tries to do this up to F times. We also note that when F is set to 1, the result of **SolveP** is the optimal solution to **P**.

Listing 2 SolveP Algorithm

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

V. On-line Clustering Technique

We have customized an on-line clustering technique, introduced by Aggarwal et al. [4], to create a suitable algorithm for our load-balancer. Specifically, we have integrated into the existing technique three concepts: cluster weight, decay of weight, and benefit.

Weights of Clusters. Each cluster has a weight whose value is between 0 and 1 inclusive. A weight of a cluster reflects both the number of flows the cluster has and the distances of the flows to the cluster *centroid* – a flow representing the cluster. At any time t, the weight of cluster j is as follow: $W_{j,t} = \lambda^{(t_j - t)} W_{j,t_j}$. Where $\lambda > 1$ is the *decaying factor*, and t_j is the last time when a new flow or a new packet gets assigned to this cluster.

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

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

The smaller the distance between f and c_j is, i.e. smaller $D(f, c_j)$, the larger weight it adds to the cluster,

Listing 3 Benefit-based Load Balancing Algorithm

```
use k-Means to create n clusters
while there is a new flow $f$
    $C$ = solveP($f$)
    if $C = \emptyset$
        if number of clusters $> m_{max}$
            delete clusters whose weights $< th_w$
        end if
        create a cluster (centroid $f$, weight 1)
        assign it to lowest load NIDPS
    else
        assign $f$ to clusters in $C$
        update those clusters
    end if
end while
```

i.e. larger $e^{-D(f,c_j)}$. In addition, when f is further away from c_j, i.e. $e^{-D(f,c_j)} \sim 0$, the weight it adds to the cluster is negligible. This formula also guarantees that the weight of any cluster is never bigger than 1.

Benefit-based Load Balancing Algorithm. At the beginning, a traditional k-Means algorithm [5] is used on a set of past flows to create n clusters with weight 1, which are then mapped to n NIDPSs. At time t, when there is a new flow f, benefits of adding f to existing clusters are needed to solve $\mathbf{P}$. Benefit of adding f to cluster j is: $B_j = (1 - D(f, c_j))W_{j,t}$. This formula expresses that: (1) The closer f is to c_j, the higher benefit the assignment gives because of larger $1 - D(f, c_j)$, and (2) the heavier the cluster j is, the higher benefit the assignment gives because of larger $W_{j,t}$. Consequently, the assignment giving the highest benefit would give the highest possible correlation, taking into consideration both logical distances and weights.

If $\mathbf{P}$ has a solution then f is added to the appropriate clusters. Otherwise, a new cluster, whose centroid is f and weight is 1, is created. Afterward, this cluster is added to an NIDPS with the lowest load. The number of clusters is closely monitored. If this number exceeds a limit m_{max}, old clusters having weights less than a threshold th_w are deleted. The deleting of old clusters, together with the creating of new ones, occurs in real-time and assures that the existing clusters are representing the current traffic. Listing 3 details this algorithm.

VI. Flows Correlation

Logical Distance Function. Given two flows f_1 and f_2, logical distance between them $D(f_1, f_2)$ is formally defined as follow: $D(f_1, f_2) = \sum_{\forall i \in \mathbf{F}} \alpha_i d_i(f_1, f_2)$. Where $\mathbf{F}$ is either ip, port, or protocol; and α's are the weights of the fields. Depending on attack scenarios, α_{ip} might be bigger or smaller than α_{port}. For example, in a DDoS attack scenario where an attacker uses many hosts to attack many services of a single victim, α_{ip} should be bigger because it is desirable to group flows having the same destination IP address. On the other hand, in a port sweep attack scenario, where an attacker uses many hosts to scan a specific port of many victims, α_{port} should be bigger. Generally, $\alpha_{protocol}$ is always the smallest because it is common to have flows with the same protocol, like TCP or UDP.

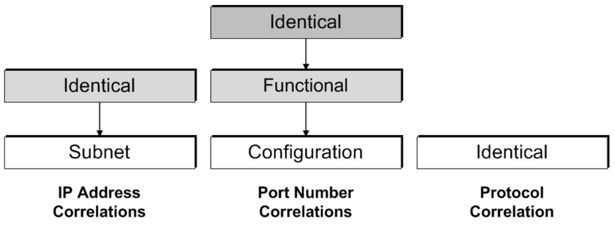

Fig. 2. Matching Order of Correlations given by IP Addresses, Port Numbers and Protocols

Logical Distance by IP Addresses – $d_{ip}()$. We match the correlation given by IP addresses of two flows with the following correlations: identical and subnet correlations. $d_{ip}()$ returns a value between 1 and 0, corresponding to the matching. The correlations are defined as follow:

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

  This correlation is the most important correlation between two flows. For example, in a DDoS attack scenario, different source IP addresses might be used for the attack [6]. However, computers corresponding to those source IP addresses attack the same target. Because they all have the same destination IP address, flows belong to the attack have identical correlation.
- **Subnet Correlation:** If destination IP addresses of two flows belong to the same subnet or vlan, then their correlation matches subnet correlation.

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

Fig. 2 shows the order in which the matching is done. Going from top to bottom, the significances of the correlations decrease, so do the values $d_{ip}()$ returns.

Logical Distance by Port Numbers – $d_{port}()$. Because destination port numbers represent target services, they play a more important role than source port numbers. As a result, we concentrate on investigating correlations between destination port numbers. Similar to the IP address case, in order to determine a value between 1 and 0, which $d_{port}()$ 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 identical correlation.

  This correlation supports the detection and prevention of intrusions targeting a particular service provided by a computer. For example, flows belonging to an attack aiming at a web server all have 80 as their destination port number.
- **Functional Correlation:** If destination port numbers of two flows are functionally correlated then their correlation matches functional correlation.

  For example, flows belonging to an FTP connection have both destination port number 20 and 21. Thus, it is desirable to group these flows together.
- **Configuration Correlation:** If destination port numbers of two flows belong to a set of port numbers provided by administrators, then their correlation matches configuration correlation.

  In practice, 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 by Protocol – $d_{protocol}()$. Either 1 or 0 is returned, depending on the following correlation:

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

VII. Implementation and Evaluation

Implementation. The load-balancer is developed using *libpcap* library [7] – a library for capturing and sending network packets directly from and to network interfaces in real-time. Both BLB and the naive flow-based *round robin* algorithms are implemented in the load-balancer.

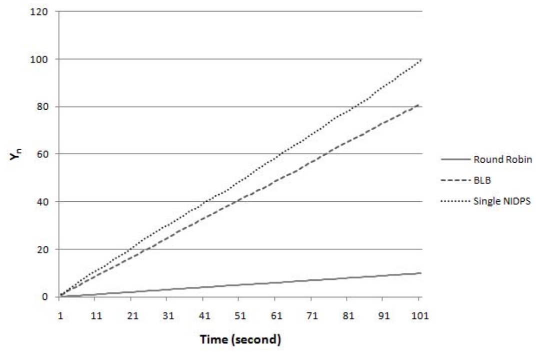

Fig. 3. Values of Y_n over time

In addition, a DDoS detector is developed using CUSUM – a simple and robust algorithm to detect DDoS by T. Peng, et al. [8]. Fundamentally, CUSUM detects the change of the mean value of the percentage of the number of new source IP addresses overtime. A sequence $\{Y_n\}$ is used to characterize the change. If at any time, a value of $\{Y_n\}$ is bigger than a predefined threshold th_y, then an attack is detected.

Evaluation. Because of the limited space, we only present one evaluation result related to the detection accuracy. Other results related to the load balancing performance can be found in our technical report [3]. We carry out an experimentation to evaluate how BLB algorithm increases the detection accuracy of DDoS attacks, in comparison with the naive round robin algorithm.

V is set at 9, i.e. NIDPSs have load within 18% of each other, and F is set at 1, i.e. no duplication of flows. A large scale UDP flood attack, which involves about 10000 distinct attacking hosts and a victim, is generated. Each UDP packet is of fixed size 1 KB, and its source port and destination port are randomly selected. The simulation lasts 100 seconds, during which the victim has to manage a constant rate of traffic about 1 MBps. The victim sees about 100 new source IP addresses per second. Also, the background traffic is insignificant comparing to the attack traffic.

There are 10 NIDPSs in our system. Using the round robin algorithm, each NIDPS detects about 10 new source IP addresses per second. Using our BLB algorithm, there exists an NIDPS which detects a significantly higher number of new IP addresses per second. Fig. 3 shows values of Y_n over time when a single NIDPS, 10 NIDPSs with BLB, and 10 NIDPSs with round robin are used. It could be observed that Y_n values when the round robin algorithm is used is significantly smaller than when the BLB algorithm is used. Thus, there are scenarios when the round robin algorithm fails to detect the attack but the BLB algorithm succeeds. For example, if the threshold th_y is set at 20, the round robin algorithm will fail to detect the DDoS attack; however, the BLB algorithm detects the attack at second 25, which is 5 seconds later than when a single NIDPS is used.

We note that when a single NIDPS is used, the attack is detected earlier because all flows go to this NIDPS, and thus it has a complete picture of the network traffic. Also, hash-based algorithm, used by others [1], [2], would perform similarly to the round robin algorithm in this experimentation since flows having different source IP addresses are not grouped together in both algorithms. In conclusion, our load-balancer with BLB distributes traffic in a way which increases the detection accuracy of the DDoS attack significantly.

VIII. Related Work

Network Intrusion Detection (NIDS) Cluster. In a recent literature by Vallentin et al. [2], the authors present an NIDS cluster as a solution for realizing high-performance and stateful network intrusion detection on commodity hardware. This work is very particular to Bro, an NIDS developed at UC Berkeley, and it also requires communication between NIDSs. Quite different from this approach, ours is independent of NIDS; thus, it could be applied to a cluster of any NIDPSs. Moreover, we examine correlation between flows instead of introducing communication between NIDSs to improve the detection and prevention accuracy.

Hash-based Load-balancer. Schaelicke et al. [1] present a load-balancer with a hash table and multiple hash functions. A hash function is used on flows to hash them into buckets, which are then assigned to NIDSs. The main focus is to handle the case when an NIDS is overloaded. Overload condition is handled by reassigning buckets to different NIDSs or applying multiple hash functions. This approach, however, does not examine correlations between flows and does not provide fine-grained load balancing.

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

IX. Conclusion

In this paper we have proposed a novel Benefit-based Load Balancing algorithm, which thoroughly considers both the load variation of NIDPSs and the loss of information due to flow distribution. As far as we know, none of the previous work has considered this loss of information in their approaches.

We have implemented a prototype load-balancer, which uses the BLB algorithm to distribute traffic flows in real-time such that: (1) Correlated flows are grouped together at a single NIDPS to minimize the information loss, which greatly increases the accuracy of anomaly-based intrusion detection and prevention; and (2) the load of NIDPSs are maintained close within a desired bound, which provides protection and allows for better traffic engineering. The evaluation results show that our load-balancer significantly improves the detection accuracy of DDoS attacks while keeping the load of NIDPSs close within a bound.

References

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

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

[3] Technical report: On optimizing load balancing of intrusion detection and prevention systems. [Online]. Available: http://www.cs.uwaterloo.ca/%7Ea4le/project.html

[4] C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *Proc. of the 29th VLDB*, vol. 29, 2003, pp. 81–92.

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

[6] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, Apr. 2004.

[7] Libpcap. [Online]. Available: http://www.tcpdump.org/

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

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

[10] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," *IEEE Tran. on KDE*, vol. 15, no. 3, pp. 515–528, Mar. 2003.

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

[12] K. Xinidis, I. Charitakis, S. Antonatos, K. G. Anagnostakis, and E. P. Markatos, "An active splitter architecture for intrusion detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 1, pp. 31–44, Mar. 2006.