PRIDE: A Practical Intrusion Detection System for Resource Constrained Wireless Mesh Networks

Amin Hassanzadeh, Zhaoyan Xu, Radu Stoleru, Guofei Gu, Michalis Polychronakis*

Department of Computer Science and Engineering, Texas A&M University
** Computer Science Department, Stony Brook University, USA*
{amin, z0x0427, stoleru, guofei}@cse.tamu.edu, mikepo@cs.stonybrook.edu

Abstract

As interest in wireless mesh networks grows, security challenges, e.g., intrusion detection, become of paramount importance. Traditional solutions for intrusion detection assign full IDS responsibilities to a few selected nodes. Recent results, however, have shown that a mesh router cannot reliably perform full IDS functions because of limited resources (i.e., processing power and memory). Cooperative IDS solutions, targeting resource constrained wireless networks impose high communication overhead and detection latency. To address these challenges, we propose PRIDE (PRactical Intrusion DEtection system for resource constrained wireless mesh networks), a non-cooperative real-time intrusion detection scheme that optimally distributes IDS functions to nodes along traffic paths, such that intrusion detection rate is maximized, while resource consumption is below a given threshold. We formulate the optimal IDS function distribution as an integer linear program and propose algorithms for solving it effectively and fast (i.e., practical). We evaluate the performance of our proposed solution in a real-world, department-wide, mesh network. An earlier version of this article appeared in ICICS 2013 [1] and the current article is significantly extended with new technical contents.

Keywords: wireless mesh network, intrusion detection, resource constraints, integer linear program, system implementation

1. Introduction

Wireless Mesh Networks (WMN) are self-managing networks that provide Internet, intranet, and other services to mobile and fixed clients using a multi-hop multi-path wireless infrastructure consisting of mesh nodes [2, 3, 4]. They have emerged as a cost-effective broadband network technology for services in large remote areas where no networking infrastructure is available, e.g., rural connectivity in Zambia [5], cost-efficient VoIP in South Africa [6] and disaster response [7, 8]. One example of a WMN is depicted in Figure 1. As shown, a wireless mesh network serves as the backbone communication infrastructure among WiFi networks, ad hoc networks, sensor networks and the Internet. It is important to remark the lack of a vantage point for the network traffic, due to the peer-to-peer nature of communication.

As the interest in WMN grows, security issues, especially intrusion detection, become of paramount importance. Due to the *decentralized nature of WMN*, researchers have proposed distributed solutions for network-wide intrusion detection. Distributed solutions do not rely on a single vantage point (e.g., gateways in traditional IDS in wired networks) as there always could be internal traffic in WMN to be monitored (as shown in Figure 1). The state-of-the-art distributed solutions can be categorized as: i) *monitoring node* solutions; and ii) *cooperative* solutions.

Monitoring node solutions [9, 10, 11, 12, 13, 14, 15] assign the same set of IDS functions to monitoring nodes (note: each monitoring node is responsible for a distinct part of the network). This allows (potentially) all nodes to perform intrusion detection engine by overhearing their neighbors' traffic utilizing the open nature of wireless medium. These solutions, however, have high false negative rates in *resource constrained* networks. This is because some IDS functions cannot be executed on monitoring nodes with limited resources (e.g., processing power and memory). A related work [16] investigates challenges in

applying off-the-shelf IDS (Snort [17] and Bro [18]) on mesh devices and proposes a lightweight (i.e., customized) IDS for WMN. The proposed lightweight IDS requires less memory and decreases the packet drop rate, when compared to off-the-shelf IDS. These achievements, however, are at the price of detecting fewer types of network attacks (smaller detection coverage), since most IDS functions are not implemented.

Cooperative solutions (e.g., hierarchical [19, 20] or group-based [21, 22] cooperation) distribute IDS functions to multiple cooperative nodes, in order to achieve higher detection rate and lower IDS load. These solutions, however, incur high communication overhead and high latency in attack detection. This is because nodes have to exchange their local observations with other nodes running different IDS functions. Considering the relatively high traffic rates in WMN, caused by mesh clients and external hosts as shown in Figure 1, the communication overhead of cooperative IDS [23, 22] degrades the network performance and delays intrusion response.

This research is motivated by the fact that neither monitoring nodes nor cooperative IDS techniques can practically solve the intrusion detection problem in WMN. As we will show in Section 2, the fact that WMN are resource constrained poses significant challenges for intrusion detection.

Our idea is to use the domain knowledge from security administrator about the WMN traffic, and distribute IDS functions more efficiently. More precisely, a security administrator, knowing the routing paths of the traffic in the WMN, would employ a traffic-aware framework that optimally places IDS functions on the nodes along the routing paths. For example, the idea of interference-load aware routing metric [24] in WMN, aims to route the mesh traffic through congestion free areas and provides a traffic-aware framework for the security administrator. The information about the busiest and most frequently used paths in the WMN is obtained from routing algorithms (e.g., OLSR) and network monitoring tools (e.g., tcpdump). Furthermore, it is observed [8] that when deploying WMN for disaster response, the points of interest, i.e., the physical location of data sources (e.g., Search & Rescue Robots) and destinations (e.g., Command and Control Center), and consequently the traffic paths are always known (as shown in Figure 1).

A related idea for traffic-aware IDS deployments in wired networks has been proposed [25], where different IDS responsibilities (i.e., different portions of network traffic) are assigned to each node along the traffic paths while ensuring that no node is overloaded. However, as we will show in Section 2, [25] cannot be directly applied to WMN since it assumes that each node performs all IDS functions - infeasible for resource constrained mesh devices. Our proposed solution has no communication overhead for the detection process, has no detection latency (i.e., it provides real-time intrusion detection, in contrast to cooperative IDS) and it has a higher detection rate, when compared with monitoring node solutions. In our proposed solution, *each node along a routing path, runs a distinct and customized IDS*. This *customized* IDS (technically a subset of IDS functions) allows resource conservation. The combination of *distinct* IDS along the path allows for a complete set of IDS functions to be applied to the entire network traffic. More precisely, our research makes the following contributions:

- It demonstrates that distributing IDS functions along routing paths increases the intrusion detection rate and decreases the average memory load.
- It formulates a novel IDS function distribution problem, called Path Coverage Problem (PCP), with the objective to maximize the detection rate while ensuring that nodes are not overloaded by IDS functions.
- It presents PRIDE, a protocol implemented to solve PCP effectively and fast, based on an Integer Linear Program (ILP).
- It presents results obtained from a real prototype system implementation and an evaluation in a real-world, department-wide, deployed wireless mesh network.

This article is an extension of a previously published research [1] that investigated the problem of deploying intrusion detection systems in resource constrained wireless mesh networks, and proposed a protocol that optimally distributes IDS functions on mesh nodes to conserve resources consumed by each node while achieving a high detection rate.

The rest of this article is organized as follows. We show the inability of mesh devices in running off-the-shelf IDS in Section 2 and present the state of art so-

lutions in Section 3. The system and security models considered in this article are presented in Section 4. We formulate the problem of optimal IDS function distribution in Section 5. PRIDE and the challenges in modeling IDS memory consumptions are presented in Section 6. We evaluate the performance of PRIDE and compare it with the state-of-the-art solutions using a real system implementation in Section 7 and conclude in Section 8.

When compared with its previous version (ICICS 2013 [1]), the current article is significantly extended with new technical contents:

1. We have studied two more realistic attack scenarios, insider and outsider multi-hop attacks against a WMN router/host, and considered them in the system evaluation section;
2. Section 3 is a new section that reviews state-of-the-art solutions proposed for intrusion detection in resource-constrained infrastructure-less wireless networks;
3. Section 6.1 presents Snort memory consumption modeling using extensive experiments. This section investigates the accuracy of our memory consumption model and rule file modularization considered in the ILP problem formulation;
4. Section 7 of this article is significantly extended with more details about our real-world, department-wide mesh network and the experiment settings used in our system evaluation;
5. Section 7.1 of this article is a new section explaining the effect of OLSR routing protocol on the proposed mechanism in real-world system implementation;
6. We have added Section 7.2 to this article for explaining the intrusion detection evaluation tools we have used in our system implementations. This section shows how we modified the Rule-to-Attack tools to generate real-time exploits;
7. Section 7.3 of this article is a new section that presents all details about Snort rule file modularizations, the rule files and exploits we have used in our experiments, and the alerts generated for each attack in our system evaluation;
8. We have also added Section 7.6 that evaluates the PRIDE's performance for attackers who are aware of the PRIDE protocol (i.e., PRIDE-aware attacks). This section evaluates intrusion detection rates of the PRIDE protocol for PRIDE-aware attacks.

2. Motivation and Background

The research presented in this article is motivated by the challenges we faced when we attempted to deploy a common off-the-shelf IDS with a full configuration (i.e., configured to detect the largest set of attacks) on existing WMN router hardware. When loading Snort [17] with its full configuration on a Netgear WNDR3700 router, the router crashes because the RAM is not sufficiently large. In the remaining part of this section we describe in detail the hardware capabilities of our mesh routers, background information on Snort, and experimental results that illustrate how different Snort configurations of increasing complexity and detection capabilities impact the memory load of the router.

The Netgear WNDR3700 router has an Atheros AR7161 processor running at 680MHz, 64MB RAM, 8MB flash memory. It has two wireless cards with Atheros AR9223-bgn and Atheros AR9220-an chipsets, working on 2.4GHz and 5GHz, respectively. The operating system on the router is the most recent release of OpenWrt (i.e., Backfire 10.03.1), a Linux distribution for embedded networking devices, with kernel version 2.6.32.10. We emphasize that our mesh hardware is more powerful (in terms of processing and memory resources) than devices used in some existing real world deployments [6, 5, 3]. Although in this research we focus mainly on Netgear WNDR3700 router hardware, later in this section we present our experience and results with more sophisticated and expensive mesh hardware, e.g., Meshlium Xtreme [26] which has a 500MHz CPU, 256MB RAM, 8/16/32GB disk memory and WiFi, Zigbee, and GPRS wireless interfaces.

The router runs Snort [17], an off-the-shelf intrusion detection system. Snort's detection engine is based on thousands of detection rules (categorized in multiple rule files, corresponding to known network threats) and several preprocessors. All files are listed in "snort.conf", a global configuration file. Upon activating each rule file in "snort.conf" and running Snort, all detection rules present in the rule file are loaded in memory and are used for packet investigation. A full Snort configuration activates all preprocessors and rule files. A customized configuration activates only some preprocessors and rule files (i.e., IDS functions), thus, the network traffic is analyzed by fewer detection functions.

The intrusion detection in Snort is performed by

packet-level rule matching. Each packet is preprocessed, following preprocessing directives for extracting possible plain-text content. The preprocessed packet is then inspected by Snort detection rules, to expose whether it is an intrusion attempt or not. Preprocessors parse network packets and provide abstract data for some high-level detection rules in the rule files. It is important to note that a rule file that contains high-level detection rules has *preprocessor dependency*. This dependency means that the rule file cannot be activated (i.e., Snort generates an error message and stops) unless all the preprocessors required by its rules (usually one or two preprocessors) are also activated. From here on, we refer to a Snort rule file as an "IDS function."

To understand how different Snort configurations impact the memory load on the Netgear WNDR3700 and Meshlium Xtreme, we performed several experiments. Once the firmware boots on a typical mesh router (e.g., Netgear), the memory utilization will be between ∼20% to ∼30%, depending on the size of kernel and services it initially performs. We refer to this memory utilization of firmware kernel as the *base memory load*. We have experimentally observed that it is infeasible to lower the base memory load below ∼15% by customizing the kernel during compilation. Moreover, the it is very likely to have higher base memory loads when activating networking and routing services to WMN routers. Additionally, running Snort causes two types of memory loads to the router: 1) *static*, the initial load imposed by packet capturing modules, preprocessors, detection rules, etc. when Snort is loaded; 2) *dynamic*, the variable load imposed by stateful preprocessors (e.g., *Stream5*) which is a function of the traffic load and some configuration parameters.

We first investigate the static memory load of Snort on the routers when no network traffic is applied. We have observed that a typical memory load on a Netgear WNDR3700 router is ∼30% and on the Meshlium Xtreme router it is ∼60%. This accounts for OS firmware and various services (OLSR routing, DHCP, etc.). Without preprocessors or rule files active, loading Snort on Netgear WNDR3700 increases memory load to 43% ("Snort(S)" in Figure 2). Memory load increases to 46% if preprocessor Stream5 is activated ("S+str5" in Figure 2), and to 48% if preprocessors *http-inspect*, *smtp* and *ftp-telnet* are also activated ("S+4Pre" in Figure 2).

The memory load of a rule file is a function of the number of detection rules in it and the pattern matching algorithm Snort uses (e.g., `Aho-Corasick`). For example, using `ac-bnfa-nq` search method, "dos.rules" which has 20 detection rules and requires the Stream5 preprocessor, increases memory load to 47% ("S+dos" in Figure 2). A very large file such as "spyware-put" ("SpyConf" in Figure 2) which contains ∼1,000 rule files increases the RAM load to 70%. The memory load caused by activating a set of rule files also depends on their sizes. For example, activating 20 small rule files (i.e., 10 rules per file on average) and the Stream5 preprocessor (which the rules require) increases memory load to 49%. Activating two large rule files, "spyware-put.rules" and "backdoor.rules" ("SpyBack" in Figure 2) increases memory load to 98%. We have experimentally verified that adding a few small rule files on top of "spyware-put.rules" and "backdoor.rules" causes the router to crash. *We have observed a similarly overloaded operation for the Meshlium Xtreme router, where a full configuration Snort increases the memory load to 98.5%, leaving almost no room for processes/services beyond stock deployment.* We also emphasize here the rapid increase in the number of Snort rule files (i.e., currently about 70 files) and their sizes as functions of the number of threats. Some rules might not be needed in a particular setting, but conversely, that setting might require many more rules of some other kind (e.g., custom signatures for suspicious or blacklisted domains, which can increase significantly).

Dynamic memory load, imposed by Stream5 when tracking traffic sessions, is the other considerable type of Snort memory load since almost all rule files require this preprocessor. Two configuration parameters of Stream5, `max_tcp` and `memcap`, specify the maximum simultaneous TCP sessions it tracks (similarly, `max_udp`, `max_icmp`, and `max_ip`) and the maximum buffer size for TCP packet storage, respectively. We have experimentally observed that the value of `max_tcp` affects both dynamic and static memory loads. When using the Snort version available on the OpenWrt development tree, the default configuration has max_tcp=8192. Choosing max_tcp=100,000, imposes ∼10% more static load than default "S+Str5" to the routers. Moreover, this value allows more simultaneous TCP sessions to be inspected which also imposes more dynamic load and may cause the router to crash at high traffic rates (note: we observed that for max_tcp≥150,000 the router crashes if a simple HTTP request is sent using the Linux *wget* tool).

Throughout this article, we use the default setting, i.e., max_tcp=8192, and consider the maximum dynamic load this setting imposes on the router. Hence, from here on, the total memory load of Stream5 is assumed to be its static load plus its maximum allowable dynamic load. It is worth mentioning that although hardware improves, also transmission speeds get faster, the amount of traffic that needs to be inspected grows, and the complexity of the applied processing increases. Hence, the fundamental challenge for a resource-limited node to handle *ever-increasing traffic* still remains.

In addition to RAM, processing power (CPU) is also limited on current mesh hardware. Consequently, investigating the impact of Snort IDS on this limited resource might seem worthwhile. Experimentally we have found that network traffic, actually, has a much larger influence on CPU utilization than executing Snort IDS functions. Our experimental results are depicted in Figure 3 where we enabled `tcp_track` and `icmp_track` in Stream5 and used *hping3* to generate TCP and ICMP traffic. As shown, for an extremely high traffic rate, both lightweight and heavy Snort configurations impose more than 95% CPU utilization. Similar with our result, it was shown [16] that even a lightweight IDS exhausted the CPU when traffic rate was extremely high. However, as shown in Figure 3, "S+dos", a lightweight IDS configuration, imposes less processing load than "SpyBack", a heavyweight IDS configuration, when the traffic rate is not high. *Consequently, our main concern in this research is the reduction of RAM utilization as we have experimentally observed that it also improves the CPU utilization in regular traffic rates (as shown in Figure 3).*

3. Related Work

As the first step towards providing dynamic and cost effective network services in environments with no network infrastructure, wireless mesh networks are becoming more popular. As an instance of real WMN implementations the rural wireless mesh network project in Zambia [5] provides telephony and internet access in some remote physical areas. Moreover, the lack of cellular network in disaster areas has convinced researchers [7, 8] to propose mesh networks as a cost-efficient and easy-to-deploy solution in order to provide networking services in disaster situations. In addition to these applications, mesh networks have been deployed in academic and research centres as test-bed for developing and evaluating networking protocols for WMN [27, 3].

As wireless mesh networks become a popular choice for offering wireless services, security challenges grow in importance [28]. Due to the variety of services and protocols used in WMN, these networks are susceptible to different attacks and security threats [29], i.e., not only WMN specific attacks (Sinkhole, Wormhole, Rushing, etc.), but also traditional TCP/IP based attacks (Scanning, Buffer Overflow, IP Spoofing, etc.).

The problem of intrusion detection in wireless mesh networks has received considerable attention from the research community. Some existing solutions address specific attacks (e.g., Man-in-the-Middle and Wormhole attacks [30, 31], Grayhole attack [32, 33, 34], Pollution [35] and Jamming [36] attacks, message fabrication attack [37], and attacks against scheduling [38], QoS [39] or anonymity [40] in WMN). Other solutions are general IDS solutions for mesh networks, which consider memory, processing [16, 41], and energy [14] constraints. In [41], a set of technical challenges associated with IDS solutions in mesh networks are presented. The authors provide interesting evaluation results on the CPU utilization of a Netgear WG302 router and propose an initial design of a modular IDS but do not evaluate their solution. Performance evaluations of the Netgear WG302 mesh router, when running off-the-shelf IDS have also been reported [16]. The authors propose a lightweight open source IDS that outperforms off-the-shelf IDS solutions with respect to CPU utilization and packet drop rates. However, the proposed solution detects only a limited number of network attacks, since many of the IDS functions are not implemented because they require significant resources.

Due to the decentralized nature of wireless mesh networks, researchers [9, 10, 11, 12, 13, 14, 15, 42] propose *monitoring node* solutions. In a *monitoring node* solution, a minimum subset of nodes are strategically selected to perform intrusion detection to *monitor* the entire network. Recently, optimized solutions for assigning the optimal set of channels to a given set of monitoring nodes in a multi-radio multi-channel WMN are proposed [11, 12, 13, 15]. The problem is formulated as an ILP and solved with rounding techniques. For example, in [12], the expected number of active users monitored by the monitoring nodes is defined as the quality of monitoring (QoM) metric. The authors investigate the problem of maximizing

QoM by optimally assigning sniffers to communication channels based on the information obtained from user activities.

In [14], an energy efficient monitoring technique is proposed for intrusion detection in battery powered WMN. Since the number of IDS functions running on the *monitoring node* is limited by the amount of available resources, these solutions only detect a limited number of attacks even though all communication links and network traffic are monitored. Since the number of services provided by WMN is expected to increase (e.g., delay tolerant services [8], VoIP services [6]), fewer resources will be available for intrusion detection. Consequently, the intrusion detection rate is expected to degrade. As an alternative, cooperative IDS [20, 22, 21, 23] have been proposed. They distribute IDS functions to all nodes, thus helping lower resource consumption and increase detection rate. However, since the higher detection rate is obtained through message exchange, it is not an efficient approach in highly loaded WMN. We note here that term *cooperative* refers to any message exchange required for detection process (that PRIDE does not have) not for role assignments (e.g., [22, 23] require message exchange for both detection process and role assignments).

In order to reduce the processing and memory loads, and increase detection rates, non-cooperative traffic-aware solutions have been proposed. They distribute IDS functions to multiple nodes (i.e., each node runs an individual customized IDS) located on network traffic paths. A research in wired networks [25] proposed a scheme where each node along a network path has enough *static* memory space to execute a full Bro IDS (i.e., infeasible for WMN nodes as we showed they simply crash). To save resources on the *dynamic* part of memory (*processing and memory that would be allocated based on traffic*), each node investigates only a portion of the network traffic (e.g., each node is responsible for inspecting only a few packets of a session). This requires changes in TCP/IP header to assign a hash value to each packet and also changes in the IDS for accurate packet selection and investigation. It is worth mentioning that WMN nodes, as shown earlier, do not have enough static memory to run a full IDS, and running a customized IDS will reduce the detection rate regardless to the traffic volume. Our proposed solution requires no changes to the IDS and TCP/IP protocol stack, only to how rule files are organized. Consequently, our proposed solution which avoids executing a full IDS on nodes, may prove more practical.

4. System and Security models

The system we are considering in this research is as specified by the IEEE 802.11s WLAN Mesh Standard [2]. The system consists of: i) *mesh access points (AP)* connecting mesh clients (from now on we will refer to them as "clients") to the mesh network; ii) a *wireless mesh backbone*; and iii) a *gateway*, connecting the mesh network to the Internet. The network traffic is either *external*, i.e., between clients and external hosts (external to the mesh), or *internal*, i.e., between two hosts[1] inside the mesh network. Examples of external and internal traffic are shown in Figure 1. Our system also requires the presence of a base station – a computer which periodically and securely (i.e., secured with neighborhood keys [43]) collects, via a middleware, information about mesh nodes: *processing/memory* loads, traffic information, etc. Based on the collected information, the base station assigns intrusion detection functions to nodes.

4.1. Security and Attacker Model

The IDS we are considering in this research is Snort [17]. We chose Snort because it is a mainstream off-the-shelf IDS that consumes less resources than other IDS, e.g., Bro (as it was experimentally shown [16]). Moreover, Snort is readily available for our mesh hardware, as part of the OpenWrt development tree. To the best of our knowledge, there is no port of Bro to the mesh hardware we have available. Assigning a Snort IDS function to a node is equivalent to activating a rule file in the Snort configuration file on that node. As we showed in Section 2, activating a rule file imposes a specific amount of memory load to the device, thus, a limited number of rule files can be activated when running Snort on the device. We use the default search method of Snort, i.e., `ac-bnfa-nq`, as we experimentally observed that it consumes the minimum memory among all *low memory* search methods, e.g., `lowmem`.

In this article, we consider *multi-hop* attacks where the attacker and the target are connected to the mesh network at different APs (Note: we will evaluate the

[1] A host inside the mesh is either a client or a local server (e.g., a local FTP server) connected to the mesh routers.

performance of our solution at the presence of *single-hop* attacks in Section 7.6). Thus, the attack traffic (malicious packet(s)) is routed across multiple nodes. The attacker can be either an *insider* or *outsider*. An insider attacker is a client, connected to a mesh AP, running attacks against a target (a router or host) several hops away. An outsider attacker is an external host attacking a router or a host in the mesh network (e.g., the attack traffic travels through internet and reaches the target over the mesh backbone).

As two realistic attack scenarios, insider and outsider multi-hop attacks against a WMN router/host, we refer to the attacks we launched against routers in DistressNet WMN [8] [44] during a demo at Disaster City ®, College Station, Texas, in May 2012. **Insider:** a mobile client connected to an AP launched multiple port scan attacks against all routers (i.e., other APs or relay nodes) in the WMN. **Outsider:** an external client connected to the DistressNet network through internet and launched remote root exploits against Dropbear SSH configured on the mesh routers.

5. Preliminaries and Problem Formulation

In this section, we formulate the optimal distribution of IDS functions as an optimization problem and we propose a method to solve it. Although Snort is our target IDS (and present a problem formulation that uses Snort terminology), we believe that other IDS can be analyzed similarly, if their internals and functionality are publicly available. For example, in Di-Sec [45], a security framework recently proposed for wireless sensor networks, the sub-components of M-Core can be modeled as Snort preprocessors while the detection and defense modules play the same roles as Snort rule files.

5.1. IDS Function Distribution

We denote the number of nodes and number of links in the wireless mesh network by N and Q, respectively. Considering the information collected from the nodes, we denote the number of nodes and links actively contributing in traffic routing by n ($n \leq N$) and q ($q \leq Q$), respectively. Thus, we model the wireless mesh network (i.e., after removing *idle* nodes/links) as a reduced graph $G = \{V, E\}$, where V is the set of nodes $\{v_1, v_2, \cdots, v_n\}$, and E is the set of links $\{e_1, e_2, \cdots, e_q\}$. An example of a reduced graph, in Figure 4, $V = \{v_1, v_2, ..., v_9\}$ and $E = \{e_1, e_2, ..., e_8\}$.

We denote the set of routing paths for the network traffic by $P = \{p_1, p_2, \cdots, p_l\}$, where set $P_i = \{v_j \,|\, v_j \text{ is located in } p_i\}$ and $P_i \subseteq V$. In Figure 4 two paths are present: p_1 and p_2. Additionally, we denote by matrix $\mathbb{T}_{l \times n}$ the mapping between nodes and paths, i.e., $t_{ij} = 1$ iff node j is located on path i. For the example shown in Figure 4, the matrix $\mathbb{T}$ is as follows:

$$\mathbb{T} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

We denote the set of all IDS functions by $\mathcal{F} = \{f_k \,|\, f_k \text{ is a set of detection rules}\}$ with size K (i.e., $|\mathcal{F}| = K$). We denote the set of IDS preprocessors by $\mathcal{C} = \{c_r \,|\, \exists\, f_k \in \mathcal{F} \text{ that requires } c_r\}$ of size R (i.e., $|\mathcal{C}| = R$). For the example in Figure 4, $\mathcal{F} = \{f_1, f_2, ..., f_7\}$ and $\mathcal{C} = \{c_1, c_2\}$. The dependency between IDS functions and preprocessors is stored in matrix $\mathbb{D}_{K \times R}$ where $d_{kr} = 1$ means that activation of function f_k requires the activation of preprocessor c_r. For the example shown in Figure 4, the matrix $\mathbb{D}^T$ is as follows:

$$\mathbb{D}^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Let $w : \{\mathcal{F}, \mathcal{C}\} \longrightarrow [0, 1]$ be a cost function that assigns memory load w_k^f and w_r^c to IDS function f_k and IDS preprocessor c_r, respectively. Consequently, vectors $W^f = [w_1^f, w_2^f, \cdots, w_K^f]$ and $W^c = [w_1^c, w_2^c, \cdots, w_R^c]$ represent memory loads for the IDS functions in $\mathcal{F}$ and for the IDS preprocessors in $\mathcal{C}$, respectively (we remark that $w_{Stream5}^c$ is the summation of its static load and its maximum dynamic load). We denote by $B = [b_1, b_2, ..., b_n]$ the base memory load (i.e., without IDS functions loaded) of all nodes.

Finally, we use vector $\Lambda = [\lambda_1, \lambda_2, \cdots, \lambda_n]$ (*also called Memory Threshold*) to represent the maximum allowable memory load after IDS functions are loaded. Memory threshold is an important parameter. It is typically set by a network administrator based on the number of active services in the mesh network and the memory space they require.

Definition 5.1. *An **IDS Function Distribution**, $A = \{(v_j, \mathcal{F}_j, \mathcal{C}_j) | \, v_j \in V, \mathcal{F}_j \subseteq \mathcal{F}, \text{and } \mathcal{C}_j \subseteq \mathcal{C}\}$, is a distribution of IDS functions in the network, such that node v_j only executes IDS functions $\mathcal{F}_j$ and their corresponding preprocessors $\mathcal{C}_j$.*

For example, the *IDS Function Distribution* in Figure 4 is:

$$A = \{(v_1, \{f_2, f_7\}, \{c_1,c_2\}), (v_2, \{f_6\}, \{c_2\}), ... \\ ..., (v_9, \{f_6, f_7\}, \{c_2\})\}.$$

We represent an *IDS Function Distribution* by matrices $\mathbb{X}_{n\times K}$ and $\mathbb{Z}_{n\times R}$, corresponding to IDS functions and preprocessors active on each node, respectively. For $\mathbb{X}$, $x_{jk} = 1$ iff IDS function f_k is activated on node v_j. For $\mathbb{Z}$, $z_{jr} = 1$ iff preprocessor c_r is activated on node v_j. Matrices $\mathbb{X}$ and $\mathbb{Z}$ for the network in Figure 4 are:

$$\mathbb{X} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \mathbb{Z} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

Considering the above mathematical formalism, the dependencies between IDS functions and preprocessors can now be represented more compactly as:

$$z_{jr} = \begin{cases} 1 & if \quad (\mathbb{X} \cdot \mathbb{D})_{jr} \geq 1 \\ 0 & if \quad (\mathbb{X} \cdot \mathbb{D})_{jr} = 0 \end{cases} \tag{1}$$

Equation 1 indicates that preprocessor c_r must be activated on node v_j if there exists at least an IDS function f_k requiring c_r, assigned to it. It is important to note that $z_{jr} = min\{1, \Sigma_{k=1}^{K} x_{jk} d_{kr}\}$ and $z_{jr} \in \{0, 1\}$.

After the *IDS Function Distribution*, the total memory load for node v_j becomes:

$$L_j = b_j + \Sigma_{c_r \in \mathcal{C}_j} w_r^c + \Sigma_{f_k \in \mathcal{F}_j} w_k^f$$

where $w_r^c \in W^c$ and $w_k^f \in W^f$. It is important to mention that an *IDS Function Distribution* in which $L_j > \lambda_j$, i.e., the load L_j is greater than threshold λ_j, for any node v_j, is deemed infeasible.

From a network security administrator point of view, we aim for an *IDS Function Distribution* where all IDS functions are activated on each path. This means that the entire network traffic will be investigated by all IDS functions (albeit at different times), eliminating the possibility of false negatives.

Definition 5.2. *For a given path p_i and its corresponding set of nodes P_i,* ***Coverage Ratio (CR)*** *is defined as $CR_i = |U_i|/K$, where $U_i = \bigcup_{v_j \in P_i} \mathcal{F}_j$ is the set of IDS functions assigned to nodes along the path. Path p_i is called* ***covered*** *if $CR_i = 1$ ($U_i = \mathcal{F}$), i.e., for $\forall f_k \in \mathcal{F}$, $\exists\, v_j$ assigned by $\mathcal{F}_j$ such that $f_k \in \mathcal{F}_j$.*

Considering the effect of *IDS Function Distribution* on the memory load of each node and the desired distribution of IDS functions to the nodes, in order to achieve higher intrusion detection rate, we define Path Coverage Problem (PCP) as follows:

Definition 5.3. ***Path Coverage Problem (PCP)*** *Given $G = \{V, E\}$, a set of paths P in WMN, the dependency matrix $\mathbb{D}$, and vectors W^f and W^c, find a distribution $A = \{(v_j, \mathcal{F}_j, C_j) | \, v_j \in V$ and $\mathcal{F}_j \subseteq \mathcal{F}$ and $C_j \subseteq C\}$, such that $\frac{1}{l} \sum_{p_i \in P} CR_i$ is maximized and $L_j \leq \lambda_j \; \forall v_j \in V$.*

PCP is an NP-hard optimization problem which has the objective of maximizing the average coverage ratio, while guaranteeing that memory loads on nodes are below a memory threshold. The NP-hardness of PCP can be proven from reducing the classic Multiple Knapsack Problem (MKP) to Single path PCP (SPCP) in polynomial time. If SPCP is NP-hard, then PCP is also NP-hard when applying the same proof. Although a lower memory threshold λ_j allows more additional processes to execute on the mesh router, it makes solving PCP much more difficult.

5.2. Optimal IDS Function Distribution

We formulate PCP as an Integer Linear Program (ILP) that can be solved by an ILP solver. The objective function is maximizing the average coverage ratio of all paths. Additionally, preprocessor dependency and memory threshold are considered as ILP constraints. More specifically, the ILP formulation is as follows:

$$\text{maximize} \quad \frac{1}{l}(\mathbf{1}^T \cdot \mathbb{T})(\mathbb{X} \cdot \mathbf{1}) \tag{2}$$

$$\text{subject to:} \quad B^T + \mathbb{Z} \cdot W^{cT} + \mathbb{X} \cdot W^{fT} \leq \Lambda^T \tag{3}$$

$$(\mathbb{T} \cdot \mathbb{X})_{ik} \leq 1 \qquad , \forall i, k \tag{4}$$

$$z_{jr} \geq \frac{(\mathbb{X} \cdot \mathbb{D})_{jr}}{K} \qquad , \forall j, r \tag{5}$$

$$z_{jr} \leq (\mathbb{X} \cdot \mathbb{D})_{jr} \qquad , \forall j, r \tag{6}$$

$$x_{jk}, z_{jr} \in \{0, 1\} \qquad , \forall j, k, r \tag{7}$$

To better understand the mathematical formulation of the objective function, one can expand it as $\frac{1}{l}\Sigma_{i=1}^{l}\Sigma_{j=1}^{n}\Sigma_{k=1}^{K}t_{ij}x_{jk}$ where $t_{ij} = 1$ means node v_j is located on path p_i and $x_{jk} = 1$ means node v_j is assigned by function f_k. In other words, the average CR has to be maximized.

Constraint 3 limits the memory load on every node v_j, i.e., $\Sigma_{r=1}^{R}z_{jr}w_r^c + \Sigma_{k=1}^{K}x_{jk}w_k^f$, to be less than its memory threshold λ_j. Most importantly, (*to ensure that we can formulate PCP as a linear program*), this constraint computes the total memory load as the sum of individual memory loads of preprocessors and rule files. Obviously, one needs to investigate if this linearity assumption always holds (we will discuss this in the next section). Constraint 4 ensures that only one copy of each function is assigned to the nodes along each path. Constraints 5 and 6 ensure that if an IDS function is assigned to a node, its required preprocessors are also assigned to the node. As presented in Equation 1, $z_{jr} = 1$ if at least one of the IDS functions assigned to node v_j requires preprocessor c_r, otherwise $z_{jr} = 0$. The maximum number of functions that require a specific preprocessor is at most K. Hence, Constraint 5 ensures that $0 < z_{jr} \leq 1$ if there is a function assigned to node v_j that requires preprocessor c_r. On the other hand, if none of the functions assigned to node v_j requires preprocessor c_r, then Constraint 6 enforces z_{jr} to be zero. Taking into account Constraint 7, i.e., z_{jr} has to be either 0 or 1, Constraint 5 enforces $z_{jr} = 1$ if preprocessor r is required on node j, otherwise, Constraint 6 enforces $z_{jr} = 0$.

6. PRIDE: Challenges and Solutions

Considering the aforementioned ILP formulation for PCP, we investigated two major challenges that impact the accuracy and time complexity of a solution. First, we experimentally observed that the total memory load of multiple Snort rule files is generally linear (i.e., it is equal to the sum of their individual memory loads), but not always (e.g., for some small rule files and certain rule types). This influences the accuracy of our proposed model for calculating the total memory load on each node (i.e., Challenge 1). Next, one can observe that the complexity of ILP depends on the number of paths in the network, the path lengths, the number of IDS functions, the number of preprocessors, and the memory threshold. For example, considering the number of Snort preprocessors (i.e., more than 20) and the number of Snort rule files (i.e., more than 60), *for single path* p_i, the number of ILP constraints grows to more than $1400 \times |P_i|$, where $|P_i|$ is the path length. Additionally, a lower memory threshold λ_j increases the number of infeasible solutions, thus requiring more iterations for the ILP solver. Hence, the ILP performance degrades as network size increases or memory threshold decreases (i.e., Challenge 2). In this section, we investigate the aforementioned challenges and propose techniques to overcome them. Finally, we present PRIDE protocol that distributes IDS functions to the nodes effectively and fast (i.e., practical).

6.1. Memory Consumption Modeling

Experimentally, we observed that when activating multiple *small* rule files (i.e., containing at most 50 detection rules), Snort memory load is much less than the sum of individual memory loads. However, we observed that when multiple *large* rule files (i.e., containing more than 250 detection rules) were activated, the memory load is closer to the sum of the rule file's individual memory loads. When a rule file is activated, depending on: 1) the number of detection rules it has; 2) the preprocessors it activates (if already not activated); and 3) the Snort search method, a different amount of memory load will be imposed to the node. In this subsection we investigate how the aforementioned three factors impact our assumption about memory load linearity (i.e., constraint 3).

Every Snort detection rule has the following structure:

```
[alert_type] [protocol] [src][src_port] -> [dst]
[dst_port]:[Options][ContentMetaData][Operations].
```

The string patterns of each rule are organized in an automaton, which has a tree-like structure, thus, we expect a sub-linear behavior when activating new rules. Besides the strings, Snort keeps additional information per rule in its internal data structures, and this increases linearly with the number of rules. The metadata for each rule usually consumes more memory than the strings contained in the rule (most strings are small, and many rules do not even have string patterns). In order to investigate the linearity of memory load, we put all detection rules in a single rule file and then measured the memory load for different number of detection rules being enabled. Since in addition to preprocessor dependency, there exists a dependency between detection rules of each Snort rule file, we had to remove all dependencies

Algorithm 1 Dependency Relaxation

```
1: Pr = {set of all preproc. directives}
2: Rr = {set of all Snort detection rules}
3: while ∃pr ∈ Pr do
4:    Kr ← GET_KEYS(pr)
5: end while
6: for ∀r ∈ Rr do
7:    H_r ← GET_KEYS(r)
8:    for ∀h ∈ H_r do
9:       if h ∈ Kr then
10:         RLX(r, h)
11:         for ∃r' ≠ r and r' ▷ r do
12:            RLX(r, h, r')
13:         end for
14:      end if
15:   end for
16: end for
```

(i.e., dependency relaxation) so that we could arbitrarily add/delete rules and change the size of the file. For this, we removed all keywords that appear in the `options`. Algorithm 1 presents how the dependency relaxation is implemented.

Given the set of Snort preprocessing directives and Snort detection rules, Algorithm 1 first creates two sets Pr and Rr (Lines 1, 2). Next, for each preprocessing directive in Pr, the Algorithm extracts a set of *keys* that are keywords dependent to the preprocessing directive (Lines 3-5). The extraction is based on our intimate/expert knowledge about preprocessing directives. Next, for each rule in Rr it extracts all keywords seen in the rule (Line 7). Since one rule may depend on several preprocessing directives, the Algorithm examines each extracted keyword (Line 8) and checks whether it exists in set Kr or not. If it exists, the Algorithm removes the keyword from the rule (Lines 9, 10). The Algorithm also examines if any of the other rules have a dependency on the current rule r and its keyword h (Line 11). If so, the keyword will be also removed from rule r' (Line 12).

After all dependencies are removed, we can arbitrarily enable/disable each detection rule in the single large file. We group the rules in two ways: i) by simply concatenating their files ("regular" case) and ii) by shuffling them into the single file ("shuffled" case) and plot Snort's memory consumption as we increase the number of loaded rules in each case. We performed the experiment for the `ac-bnfa-nq` and `lowmem` search methods. Figures 5(a) and 5(b) depict the results for the `ac-bnfa-nq` and `lowmem` search methods, respectively. Thus, *irrespective of rule order and search method, we observe a linear behavior (consistent to our intuition, as explained above) when adding blocks of 250 rules to the set of active rules.* Although the string patterns from all rules are organized in a single automaton for fast string searching (which alone would result to a sub-linear memory consumption pattern) the observed linear behavior is due to other dominant rule-specific data that increase with the number of rules. Such data includes the descriptive message to be printed in the alert, reference numbers and identification codes, numerous other keywords like `rawbytes`, `byte_test`, and `pcre`, as well as other rule metadata. We use these findings to address the non-linearity of memory load for the variable-size rule files in the following subsection.

6.2. Rule Files Modularization

To reduce the complexity of the problem the ILP solver faces (i.e., Challenge 2), we propose to reduce the number of individual preprocessors and IDS functions, which would result in a decrease in the number of constraints in ILP. Our proposal is to group multiple IDS functions together and consider them as a single function. *From here on, we refer to each group of rule files as a "detection module" and use the term "group" for a group of preprocessors.* If a detection module is assigned to a node, all rule files in that module will be activated. We experimentally observed that grouping rule files not only reduces the problem complexity (Challenge 2), but also decreases the variance in memory load estimation (Challenge 1). When several small rule files are grouped in a single detection module, it acts as a larger rule file (same as a block of 250 rules), and the estimated memory load is more accurate. In addition, considering the *preprocessor dependency* mentioned in Section 5, an efficient rule file grouping reduces the number of preprocessor dependencies. For example, if two rule files require the same preprocessor(s), they can be grouped in the same detection module. Similarly, multiple preprocessors required for the same rule files, can be grouped together. Hence, when activating a new *detection module*, the load imposed by rules' data structure dominates the load imposed by the new activated preprocessor (that can be ignored). This is very similar to the behavior observed in Figures 5(a) and 5(b) in the absence of preprocessors.

Grouping rule files together, however, has a disadvantage when the memory threshold set by the system administrator is very low. For low memory thresholds, we cannot assign larger modules to nodes, which results in low coverage/detection ratio. Consequently, despite the positive aspects of grouping small rule files together, memory threshold forces us to avoid large detection modules. Unfortunately, there already exist large detection modules. For example, the memory space required by the "backdoor" rule file is twice the memory space required by a detection module with 25 small rule files. This illustrates the need to also split extremely large rule files into some smaller ones (i.e., creating several detection modules out of a large rule file).

We thus define "modularization" as the procedure that, for a given set of IDS functions (e.g., Snort rule files), i) *groups* small IDS functions together in order to reduce the problem complexity and load estimation error, and ii) *splits* large IDS functions into several smaller functions so that they can be activated with low memory thresholds.

6.2.1. Rule File Splitting

When splitting a rule file, we consider the dependency between detection rules and the dependency between preprocessors and detection rules. This is to ensure that two dependent rules along with all of their essential preprocessing directives are included in the same split rule file. In order to split a rule file into several detection modules, we first pre-parse each detection rule and specify its preprocessing dependency in advance. For example, a detection rule using TCP traffic match (i.e., protocol:TCP) requires the Stream5 preprocessor directive, which enforces all HTTP-relevant rule files to also contain the same directive. We summarize all these preprocessing dependencies before splitting the rule files.

In addition, rule dependency is expressed by the options' keywords, e.g., "flowbits." To meet the rule dependency requirements, we parse each detection rule and specify whether the rule contains such keywords or not, if it does, it must be relevant. For example, the "flowbits" options can help us maintain the stateful check in a set of Snort detection rules. When some keys are set by "flowbits" in a detection rule, every other detection rule which does not set the "flowbits," is dependent on that detection rule. Similarly, the keyword "rev:VALUE" in a detection rule, that identifies revisions of Snort rules, denotes that it is related to a detection rule whose "sid" is "VALUE." Thus, using these two types of dependency, we split large rule files properly.

6.2.2. Proposed Modularizations

We propose three modularizations with different numbers of detection modules and different sizes. We then compare the execution time of the solver, i.e., Matlab ILP solver, for each modularization. Our modularizations are based on the size of the rule files and also preprocessor dependencies, such that the memory loads of detection modules are roughly same and the rule files in each detection module require the same preprocessors.

In the first modularization, the entire set of Snort rule files is classified into 23 detection modules where 6 different groups of preprocessors are required. The average memory load of the 23 detection modules is 3.98% and the standard deviation is 1.68%. The second modularization consists of 12 detection modules of average memory load 6.76% and standard deviation 2.31%, while the third modularization has only 6 detection modules of average memory load 15.06% and standard deviation 1.88%. The second and the third modularizations require three groups of preprocessors. Details about each modularization, e.g., the rule files in each module, are provided in Section 7.

Figure 6 shows the execution time of the ILP solver when solving the problem for different lengths of a single path. As depicted in Figure 6, 12-module and 6-module configurations are much faster than 23-module configuration, especially for longer paths (i.e., more complex problems). With these two modularizations, the ILP solver finds the optimal solution in less than 2 sec, which is very fast, thus practical in real deployments. The longer execution time for the 6-module configuration, comparing to the 12-module configuration, is because of its larger detection modules that increase the number of infeasible solutions for a given memory threshold (increasing the solver's execution time). We use 6-module and 12-module configurations in our system evaluations.

6.3. PRIDE Protocol

Given a modularization chosen for the IDS configuration, PRIDE periodically collects the local information from the nodes, removes *idle* nodes from the network, i.e., those not contributing in the traffic routing, and optimally distributes IDS functions to the nodes along traffic paths. If the reduced graph

Algorithm 2 PRIDE Protocol

1: $Data\_Collection(V, E, N, Q)$
2: $Relaxation(V, E, n, q)$
3: $Path\_Extract(V, E, P)$
4: $\mathcal{P} = P$
5: $g = 0$
6: **while** $\exists\, p_i \in \mathcal{P}$ **do**
7: $\quad g++$
8: $\quad S_g = \{p_i\}$
9: $\quad \mathcal{P} = \mathcal{P} \backslash \{p_i\}$
10: $\quad$ **while** $\exists\, p_j \in Q$ and $\bigcup_{p_k \in S_g} (P_j \cap P_k) \neq \varnothing$ **do**
11: $\quad\quad S_g = S_g \cup \{p_j\}$
12: $\quad\quad \mathcal{P} = \mathcal{P} \backslash \{p_j\}$
13: $\quad$ **end while**
14: **end while**
15: **for** $\forall\, S_g$ **do**
16: $\quad V_g = \{v_j | v_j \in P_i\, and\, p_i \in S_g\}$
17: **for** $\forall\, V_g$ **do** ILP(V_g, P)

is disconnected, each graph component is considered as a sub-problem and solved separately. Algorithm 2 presents PRIDE protocol.

Given the set of nodes, the protocol first collects information from nodes and then produces the reduced sets V and E by removing idle nodes/links (Lines 1, 2). Next, the set of active routing paths P is extracted in Line 3. Given P, the Algorithm creates the set $\mathcal{P}$ of unvisited paths (Line 4), and then defines variable g as the number of sub-problems (Line 5). For every unvisited path p_i in set $\mathcal{P}$ (Line 6), the Algorithm first creates a new sub-problem S_g (Lines 7, 8) and marks it as a visited path (Line 9). The Algorithm then searches $\mathcal{P}$ to find any unvisited path p_j which is *connected* (two paths are *connected* if they are in the same component of the reduced graph) to at least one path in the current S_g (Line 10). If so, the corresponding path p_j will be added to the current sub-problem S_g and removed from $\mathcal{P}$ (Lines 11, 12). When no more paths in $\mathcal{P}$ can be added to the current S_g, the Algorithm increases g and creates a new sub-problem. This process repeats until there is no unvisited path in $\mathcal{P}$. Next, for every sub-problem S_g, the Algorithm creates the corresponding set V_g as the set of nodes located on the paths of component S_g (Lines 15, 16). Finally, the Algorithm runs ILP on the nodes and paths of each sub-problem S_g (Line 17).

7. System Implementation and Evaluation

In this section, we evaluate the performance of PRIDE in a department-wide mesh network. Our mesh network (as shown in Figure 7) consists of 10 Netgear WNDR3700 routers deployed in a $50 \times 30m^2$ rectangular area (Note: comparing with other testbeds, DistressNet [8] with eight nodes, SMesh [3] with fourteen nodes, and QuRiNet [46] (a large-scale research platform) with thirty nodes, PRIDE uses an average size testbed.). The presence of the walls in this area makes it a suitable environment for a multi-hop mesh network. Additionally, the "tx-power" parameter in the network configuration file of OpenWrt is used to adjust the communication range of the routers. The routers use OLSR version 0.6.1 as the routing protocol (as part of the OpenWrt development tree) and provide mesh connections on their 5GHz wireless interfaces. In Figure 7, each node is labeled with its local subnet IP address. We will refer to the nodes using the third number in the subnet IP address, e.g., 192.168.5.0 is node 5. Some routers work as Mesh Access Points (MAP), e.g., node 3, and provide network access for the clients on the 2.4GHz wireless interface. Node 9 is the network gateway that connects WMN to the Internet. PRIDE periodically (i.e., 5 minutes in the current implementation) collects nodes/traffic information and runs ILP. This interval can be optimally chosen by administrator in dynamic networks.

As briefly explained in Section 4, our system also requires the presence of a base station, e.g., a computer that periodically and securely collects processing/memory loads and traffic information of WMN nodes and performs PRIDE to find the optimal rule assignment, and finally broadcasts the results to the entire network in a secure way. The data collection and security function assignment is implemented through our previous research [43] that uses neighborhood keys to securely, and optimally broadcast a message from the base station to every WMN nodes. In order to minimize the communication overhead of rule assignment mechanism, we use a binary string of size $(n \times K)$ bits, where n is the number of nodes receiving the message and K is the number of detection modules in each modularization. After producing the results, base station generates n block of size K bits, where each block corresponds to a node ID (e.g., block one for Node 1) and each bit in a block corresponds to a detection module. Upon receiving the message, each node reads its correspond-

ing block to figure out what detection modules, and consequently what preprocessors, it has to activate on its intrusion detection engine.

We evaluate the *intrusion detection rate (coverage ratio)* and *average memory load* of nodes. The parameters that we vary are the *Path Lengths (PL)* and *memory threshold (λ)*. In all our experiments, the memory thresholds of all nodes are considered equal (for simplicity) and some of the preprocessors (e.g., perfmonitor) are not used as they are not activated by default or not required by rule files. Since the maximum path length in our mesh network is 4 hops, we consider 2-hop, 3-hop and 4-hop paths ($PL = 2, 3,$ *and* 4). We consider two different paths for each given PL (six paths in total) in our evaluation. For example, in the 2-hop scenario ($PL = 2$), $P_1 = \{5, 10\}$ and $P_2 = \{9, 10\}$, and in the 3-hop scenario, $P_1 = \{5, 10, 8\}$ and $P_2 = \{9, 10, 8\}$. The initial memory load on the routers is $\sim 30\%$ (as caused by DHCP, OLSR, and other services). We vary the Snort memory threshold from 30% to 60% (i.e., $60\% \leq \lambda \leq 90\%$). Since implementing the related traffic-aware solution [25] on the mesh devices is infeasible (as shown in Section 2), we compare PRIDE with monitoring node solutions ([9, 10, 14]). We implement a monitoring node solution [14] to which we refer as "MonSol". MonSol first selects the optimal set of nodes that can monitor all traffic paths and then monitoring nodes load detection modules (based on the indices of detection modules shown in Table 1) up to a given memory threshold. If a monitoring node monitors at least one link of a given path, the entire path is considered as monitored.

7.1. The Effect of Route Changes

We use OLSR routing protocol (version 0.6.1) as part of the OpenWrt development tree available on the URL below: "*http://downloads.openwrt.org/backfire/10.03.1/ar71xx/packages/.*"

OLSR is a proactive link-state routing protocol optimized for ad hoc networks that uses hello messages and also topology control messages to discover and broadcast link state information. We have experimentally observed that even static WMN topologies and routing paths are subject to change in both indoor and outdoor deployments. This is because of a) link-quality variations caused by weather, noise and other radio signals; b) mobility of clients and their requested services that result in changes of WMN routing paths; and c) node failure (e.g., running out of power) or node replacement (e.g., administrative reasons) during network lifetime. Hence, it is very important to ensure that the network information periodically collected by the base station reflects the most recent network topology of the WMN.

Since the aforementioned changes in communication links directly affect the OLSR routing table of WMN routers, we use OLSR routing table of the nodes for network topology extraction. More precisely, each router logs its routing table every five seconds (chosen experimentally) and reports the state of its communication links after five minutes, i.e., those links that are available for the majority of samples in the last five minutes sampling period are considered available in the final report.

7.2. Intrusion Detection Evaluation Tool

We define the intrusion detection rate as the ratio between the number of detected attacks and the number of detectable attacks by all modules. For example, for the 12-module configuration, we ran 12 distinct attacks for each path where each attack can be detected by only one of the detection modules (i.e., because the corresponding detection rule is put in that module), and then measure the number of detected ones. To generate attack traffic, we modify an open source Snort test framework - the Rule to Attack (R2A) tool. R2A is a rule-based tool which parses each Snort detection rule and generates an exploitation packet targeting that rule. We modify the R2A's source code to generate real-time exploits for a given set of detection rules. The exploits, as listed in Table 2, are launched against the multi-hop remote target through wireless mesh links. If the module that can detect the attack is assigned to the nodes along the path, then the attack is detected and relevant alerts are generated.

7.3. Snort Rule Sets and Modularizations

The Snort rule files we used in our experiments and evaluations are shown in Table 1. The second column, namely "Size", specifies the number of detection rules in each rule file. Columns "M6", "M12", and "M23" specify the index of the detection module that each file belongs to, when that modularization is used. For example, in the 6-module configuration, both "community-nntp" and "shellcode" are put in the first detection module, however, in the 12-module configuration, "community-nntp" belongs to the first

detection module and "shelcode" belongs to the second one. As depicted in Table 1, "backdoor" and "spyware-put" are split into 4 and 5 smaller files, respectively.

The amount of memory load caused by each detection module in 6-module and 12-module configurations are shown inthe lower part of Table 1. As depicted, the average memory load of detection modules in the 6-module configuration is much higher than the 12-module configuration. We omit the details about memory loads in the 23-configuration here since it is not used in the system evaluation.

Depending on the rule files and modularizations we use, the attacks chosen for intrusion detection evaluation may change. In order to evaluate the detection rate of PRIDE, we choose one or two rule files from each detection module and give them to the R2A tool as the input file. We also provide the IP address of multi-hop targets for R2A so that the attack exploits (malicious traffic) will be sent to the target through a multi-hop network path. Upon running each attack, the detection modules distributed along the path generate corresponding intrusion detection alarms. Table 2 specifies the rule files chosen from each detection module of the 12-module configuration and also the number of alarms generated by the corresponding detection module. It is worth mentioning that the same files are used for the 6-module configuration. For example, in order to generate exploit(s) against module 1 in the 6-module configuration, all rule files in module 1 and module 2 of Table 2 are used, as they are all grouped in the module 1 of the 6-module configuration (according to Table 1).

7.4. Proof-of-Concept Experiment

When assigning IDS functions to multiple nodes on a path, each node can detect only a subset of attacks depending on the detection modules it executes. As a proof-of-concept experiment, we show that distributing two IDS functions to two nodes generates exactly the same alerts as if both detection modules were assigned to a single node (e.g., MonSol). For that purpose, we used two routers and one laptop connected wirelessly to each router (one laptop was the attacker and the other was the target). We ran a customized Snort on each router (monitoring the mesh traffic) ensuring that every Snort rule file is activated on at least one of the routers. We then generated two R2A exploits such that their corresponding rule files, e.g., "dos.rules" and "exploit.rules", were activated on routers 1 and 2, respectively. When running attacks, the Snort on node 1 generated 4 alerts, while the one on node 2 generated 10 alerts (real-time detection, unlike cooperative IDS). We repeated the experiment where only node 1 was running Snort and both rule files were activated on node 1 (many other rule files were deactivated due to memory constraint). In this experiment, node 1 generated exactly the same 14 alerts upon launching the same exploits. Hence, we have shown that PRIDE can distribute IDS functions to nodes along a path such that network packets are inspected by all IDS functions.

7.5. Effects of Memory Threshold and Path Length

Given the network paths in our test-bed mesh network, we evaluate the intrusion detection rate of PRIDE and the average memory load on nodes, using 6-module and 12-module configurations. For each modularization, we change λ and PL as our evaluation parameters to see their effects on PRIDE performance. Given a memory threshold, we show PRIDE can achieve higher detection rate than MonSol.

Figure 8 shows the effect of memory threshold and path length on intrusion detection rate and average memory load on the nodes when using the 6-module configuration. As depicted in Figure 8(a), maximum detection rate for MonSol is 50% which occurs when $\lambda = 90\%$. However, PRIDE can achieve 100% detection rate even in a lower memory threshold (e.g., at $\lambda = 80\%$ for $PL = 4$ and $PL = 3$). This is because more than one node is assigned with IDS functions and packets are inspected by more detection modules. In this modularization, for a low memory threshold (e.g., $\lambda = 60\%$), only module 3 can be activated on the nodes, and thus, PRIDE cannot achieve a higher detection rate than MonSol. Figure 8(b) depicts the average estimated memory load on the nodes for different memory thresholds. It can be observed that PRIDE usually impose less memory load than MonSol, especially for the longer paths, since the modules are distributed to multiple nodes.

The results for the same evaluations performed on the 12-module configuration are shown in Figure 9. As depicted in Figure 9(a), the intrusion detection rate for the 12-module configuration is higher than the detection rate for the 6-module configuration (for the same memory threshold). This is because the size of the detection modules in the 12-module configuration is smaller than for the 6-module configuration, which allows more modules to fit in the small free

memory spaces. In contrast with the 6-module configuration, where at low memory thresholds the detection rate was similar to MonSol, in the 12-module configuration the detection rate at 60% (a low memory threshold) is higher than for MonSol. This is because more modules are activated on the nodes even at this low memory threshold. The average estimated memory loads for this modularization are shown in Figure 9(b). Similar to the 6-module configuration, it is observed that the 12-module configuration usually impose less memory load than MonSol solution for the longer paths.

When considering PRIDE and MonSol, one can observe that for an adversary it will be significantly harder to compromise multiple IDS nodes (as in PRIDE), than a single monitoring node (as in MonSol). A compromised IDS node in PRIDE implies the loss of few IDS functions while, in MonSol, it means the loss of the entire set of activated functions on the monitoring node (i.e., higher vulnerability). Hence, in addition to achieving higher detection rates and lower memory loads, PRIDE provides a higher degree of IDS attack tolerance than MonSol.

We also compare the estimated memory loads and the actual memory loads of the two configurations in all of the experiments, i.e., different memory thresholds and path lengths. Figures 10(a) and 10(b) show the difference between estimated memory load and actual load measured on the routers when using 6-module and 12-module configurations, respectively. One can see that the difference is below $\sim$5%, thus giving confidence in our ILP formulation and memory consumption modelling. It is also worth mentioning that the estimated values for the 12-module configuration are closer to the real values than the 6-module configuration because the modules are roughly the same size as 250-rule blocks.

Figure 11 shows the ILP solver execution time for $PL = 3$ and $PL = 4$, and for each modularization. As depicted, the execution time of the algorithm ranges from a few seconds to tens of seconds, thus making it practical for real world deployments. As shown, the lower the memory threshold is, the longer the execution time is. This is because lower memory thresholds increase the number of infeasible solutions and the solver requires more iterations to obtain feasible and optimal solutions. As shown in Figure 11, the execution time increases with the path length as well. As mentioned in Section 6, this is because the number of ILP constraints (i.e., the problem complexity) is a direct function of path length.

7.6. PRIDE-aware Attacks

This section evaluates PRIDE's performance for PRIDE-aware attacks. We categorize PRIDE-aware attacks in two levels of severity: 1) the attacker is aware of PRIDE in the WMN but cannot compromise the secure communication between nodes and the base station (Level 1); 2) the attacker is aware of PRIDE and also the content of secure information exchanged between nodes and the base station (Level 2). Obviously, the latter type of attack is more severe and very difficult to defend. In fact, the second attack type assumes that the attacker has broken the secure communication link (i.e., secured with neighborhood keys [43]) between the base station and all routers and has access to all information (i.e., memory loads and traffic paths) and the IDS distributions. Unlike our attacker model presented in Section 4, we assume that a PRIDE-aware attacker can launch an attack against an *intermediate* node in a traffic path (not necessarily the multi-hop *destination* on the path). This type of attack sounds reasonable because a PRIDE-aware attacker aims to compromise some intermediate nodes running specific detection modules, and finally attack the destination.

We concentrate on Level-1 attack because the possibility of running Level-2 attack in WMN depends on the robustness of key distribution and also encryption mechanisms used in wireless networks, which is out of our scope in this article. In Level-1 attack, the attacker can not produce the same IDS distribution (to find the most beneficial node to be compromised) as the base station produces. This is because the ILP solutions depend on information collected from WMN nodes (securely sent to the base station) and the initial random solutions. Thus, we consider an attacker that knows WMN nodes are assigned *some* IDS functions but does not know which node is running which module. The attacker first connects to an AP, then chooses a node (destination or an intermediate) as the target and a *random* type of attack (i.e., an attack among those detectable by 6 or 12 modules), and finally launches the attack. It is obvious that the average detection rate for the PRIDE-aware attacks against destination nodes is always equal to the PRIDE coverage ratio (i.e., as shown in Figures 8(a) and 9(a)). Hence, we only consider attacks against intermediate nodes.

We perform an experiment to evaluate the PRIDE performance (detection rate) when a PRIDE-aware attacker at Level 1 runs attacks against intermediate nodes. Considering the IDS distributions produced for our real-world WMN, with the detection rates shown in Figures 8(a) and 9(a), we ran 1000 random attacks for each modularization. In each of 1000 attacks, the attacker chooses a random intermediate node and a random attack (among those detectable by 6 or 12 modules). If the corresponding detection module is not activated on the nodes along the path (starting from the attacker AP towards the intermediate node) the attack cannot be detected, otherwise it is detectable. Figures 12(a) and 12(b) show the average detection rate for Level-1 PRIDE-aware attacks against intermediate nodes, for 6-module and 12-module configurations, respectively. As depicted, the detection rate increases as the λ increases. We observe that the 12-module configuration has a higher detection rate than the 6-module configuration (for the same reason we explained about average detection rate in PRIDE). One may argue about the detection rate for (PL=4) in the 6-module configuration that decreases at higher memory thresholds. This is because we perform all 1000 random attacks against a specific IDS distribution used in our real test. In that experiment, for higher thresholds, most of the IDS modules were assigned to the destination, thus, intermediate nodes ran less IDS modules and were unable to detect attacks. The same argument applies for (PL=3) in the 12-module configuration.

PRIDE was shown (in Figures 8(a) and 9(a)) to achieve 100% detection rate for WMN with routes of at least 3 hops long and memory thresholds above $\sim$ 70% (i.e., reasonable assumptions for real-world WMN applications). On the other hand, undetectable PRIDE-aware attacks are those targeting intermediate nodes in a multi-hop source-destination path protected with detection modules. In fact, this is similar to launching single-hop or 2-hop attacks while the path considered by the security administrator is at least 3 hops long (similarly for lower memory thresholds than what the security administrator has considered). Hence, taking into consideration that almost all of traffic paths in WMN are multi-hop and PRIDE is a detection mechanism proposed for multi-hop attacks, which is a realistic assumption for WMN, the $\sim$ 60% detection rate shown in our experiments is a considerable detection rate.

An alternative method for detecting Level-1 PRIDE-aware attack is to force the algorithm to assign specific modules (e.g., those including detection rules for IP sweep and port scanning attacks as these are the first step in multi-step compromising attacks) to the nodes close to the source node in a routing path. This may help early nodes in a multi-hop attack to detect the trial and error attacker behavior.

7.7. Distributed Approach

Although we showed that PRIDE can achieve high detection rates by solving an ILP in less than a minute, one may argue about the communication overhead of IDS Function Distribution (i.e., not for the detection process as in cooperative IDS) caused by the message exchange between nodes and the base station. In addition to the message complexity, the time complexity increases as network size grows. Thus, the centralized approach seems not suitable for very large WMN (e.g., 100+ nodes) as its time and message complexity increases. The message complexity of the IDS Function Distribution in the centralized algorithm proposed for PRIDE is $O(n \log n)$. Since this communication overhead is due to the message exchange between mesh nodes and the base station, distributed IDS Function Distribution might be intuitively an alternative. In a distributed mechanism, however, the message complexity is $O(n^2)$ if nodes require to exchange their routing information with each other. Thus, a question that arises here would be "Is it possible for the nodes actively contributed in the traffic routing to randomly choose an IDS configuration and still achieve reasonable intrusion detection (path coverage) rates for WMN paths?" It is obvious that such a distributed approach, that is very fast and incurs no communication overhead for IDS Function Distribution, will provide a locally optimal solution since the decision is made based on the local information available on each node. However, the centralized approach in PRIDE produces the optimal IDS function distribution based on global information about WMN routing paths.

In order to compare the centralized approach in PRIDE with the distributed IDS Function Distribution mechanism, we implemented a distributed mechanism where each active node, based on its memory threshold, randomly chooses the set of detection modules it can perform. The average path coverage rates for 100 random solutions produced for the WMN topology and its corresponding paths shown in Figure 7 are depicted in Figure 13. When compar-

ing to the results shown in Figures 8(a) and 9(a), the centralized approach outperforms the distributed approach in most of situations, especially when there is enough memory for activating multiple detection modules. It is worth mentioning that although the centralized approach achieves higher detection rates at the price of higher communication overhead, it is very important to achieve 100% detection rates (at higher prices) in mission critical scenarios and a sub-optimal solution will not be accepted as an applicable solutions.

8. Conclusions and Future Work

In this article, we have shown that intrusion detection in WMN requires significant resources, and that cooperative and monitoring node-based solutions are not practical for WMN. To address these challenges, we propose a solution for an optimal distribution of IDS functions. We formulate the optimal IDS function distribution as an integer linear program and propose algorithms for solving it effectively and fast (i.e., practical). Our solution maximizes intrusion detection rate, while maintaining the memory load below a threshold set by network administrators. We have investigated the performance of our proposed solution in a real-world, department-wide, deployed WMN.

Looking to the future, we plan to investigate how Stream5 parameters, e.g., "max_tcp," can be chosen optimally for each node, based on the traffic load it faces. This way, two nodes with the same memory space available but different traffic rates, will dedicate their memory spaces to the static and dynamic loads differently.

Acknowledgment

This article is based in part on work supported by Naval Surface Warfare Center, Grant No. N00164-11-1-2007.

References

[1] A. Hassanzadeh, Z. Xu, R. Stoleru, G. Gu, and M. Polychronakis, "PRIDE: Practical intrusion detection in resource constrained wireless mesh networks," in *ICICS*, 2013.

[2] G. R. Hiertz, D. Denteneer, S. Max, R. Taori, J. Cardona, L. Berlemann, and B. Walke, "IEEE 802.11s: the WLAN mesh standard," *IEEE Wireless Communications*, 2010.

[3] Y. Amir, C. Danilov, R. Musăloiu-Elefteri, and N. Rivera, "The SMesh wireless mesh network," *ACM Transactions on Computer Systems*, September 2008.

[4] I. F. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: a survey," *Computer Networks and ISDN Systems*, vol. 47, no. 4, pp. 445–487, 2005.

[5] J. Backens, G. Mweemba, and G. Van Stam, "A rural implementation of a 52 node mixed wireless mesh network in Macha, Zambia," *EInfrastructures and EServices on Developing Countries*, pp. 32 – 39, 2010.

[6] M. Adeyeye and P. Gardner-Stephen, "The Village Telco project: a reliable and practical wireless mesh telephony infrastructure," *EURASIP Journal on Wireless Communications and Networking*, 2011.

[7] N. Aschenbruck, J. Bauer, R. Ernst, C. Fuchs, and J. Kirchhoff, "Poster: Deploying a mesh-based command and control sensing system in a disaster area maneuver," in *ACM SenSys*, 2011.

[8] H. Chenji, A. Hassanzadeh, M. Won, Y. Li, W. Zhang, X. Yang, R. Stoleru, and G. Zhou, "A wireless sensor, adhoc and delay tolerant network system for disaster response," LENSS-09-02, Tech. Rep., 2011.

[9] D. Subhadrabandhu, S. Sarkar, and F. Anjum, "A framework for misuse detection in ad hoc networks-part I," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 274–289, Feb 2006.

[10] ——, "A framework for misuse detection in ad hoc networks- part II," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 2, pp. 290–304, Feb 2006.

[11] D.-H. Shin and S. Bagchi, "Optimal monitoring in multi-channel multi-radio wireless mesh networks," in *ACM MobiHoc*, 2009.

[12] A. Chhetri, H. Nguyen, G. Scalosub, and R. Zheng, "On quality of monitoring for multi-channel wireless infrastructure networks," in *MobiHoc*, 2010.

[13] P. Arora, N. Xia, and R. Zheng, "A gibbs sampler approach for optimal distributed monitoring of multi-channel wireless networks," in *IEEE GLOBECOM*, 2011.

[14] A. Hassanzadeh, R. Stoleru, and B. Shihada, "Energy efficient monitoring for intrusion detection in battery-powered wireless mesh networks," in *ADHOC-NOW*, 2011.

[15] D.-H. Shin, S. Bagchi, and C.-C. Wang, "Distributed online channel assignment toward optimal monitoring in multi-channel wireless networks." in *IEEE INFOCOM*, 2012.

[16] F. Hugelshofer, P. Smith, D. Hutchison, and N. J. Race, "OpenLIDS: a lightweight intrusion detection system for wireless mesh networks," in *MobiCom*, 2009.

[17] *SNORT Users Manual v2.9.2*, The Snort Project, September 2011.

[18] V. Paxson, "Bro: A system for detecting network intruders in real-time," in *Computer Networks*, 1999, pp. 2435–2463.

[19] A. Hassanzadeh and R. Stoleru, "Towards optimal monitoring in cooperative IDS for resource constrained wireless networks," in *IEEE ICCCN*, 2011.

[20] C. Liu and G. Cao, "Distributed monitoring and aggregation in wireless sensor networks," in *IEEE INFOCOM*, 2010.

[21] G. Li, J. He, and Y. Fu, "A distributed intrusion detection scheme for wireless sensor networks," in *ICDCS*, 2008.

[22] I. Krontiris, Z. Benenson, T. Giannetsos, F. C. Freiling, and T. Dimitriou, "Cooperative intrusion detection in wireless sensor networks," in *Wireless Sensor Networks*. Springer-Verlag, 2009.
[23] A. Hassanzadeh and R. Stoleru, "On the optimality of cooperative intrusion detection for resource constrained wireless networks," *Computers & Security*, 2013.
[24] D. Manikantan Shila and T. Anjali, "Load aware traffic engineering for mesh networks," *Computer Communications*, vol. 31, no. 7, pp. 1460–1469, 2008.
[25] V. Sekar, R. Krishnaswamy, A. Gupta, and M. K. Reiter, "Network-wide deployment of intrusion detection and prevention systems," in *ACM CoNEXT*, 2010.
[26] *Meshlium Xtreme Technical Guide*, Libelium Comunicaciones Distribuidas S.L., September 2011.
[27] S. Uludag, T. Imboden, and K. Akkaya, "A taxonomy and evaluation for developing 802.11-based wireless mesh network testbeds," *International Journal of Communication Systems*, vol. 25, no. 8, pp. 963–990, 2012.
[28] N. Ben Salem and J.-P. Hubaux, "Securing wireless mesh networks," *IEEE Wireless Communications*, vol. 13, no. 2, pp. 50 –55, April 2006.
[29] O. E. Muogilim, K.-K. Loo, and R. Comley, "Wireless mesh network security: A traffic engineering management approach," *Journal of Network and Computer Applications*, vol. 34, no. 2, pp. 478–491, March 2011.
[30] S. Glass, V. Muthukkumarasamy, and M. Portmann, "Detecting man-in-the-middle and wormhole attacks in wireless mesh networks," in *AINA*, 2009.
[31] J. Zhou, J. Cao, J. Zhang, C. Zhang, and Y. Yu, "Analysis and countermeasure for wormhole attacks in wireless mesh networks on a real testbed," in *AINA*, 2012.
[32] D. Shila and T. Anjali, "A game theoretic approach to gray hole attacks in wireless mesh networks," in *MILCOM*, 2008.
[33] D. M. Shila, Y. Cheng, and T. Anjali, "Channel-aware detection of gray hole attacks in wireless mesh networks," in *IEEE GLOBECOM*, 2009.
[34] A. Prathapani, L. Santhanam, and D. Agrawal, "Detection of blackhole attack in a wireless mesh network using intelligent honeypot agents," *The Journal of Supercomputing*, vol. 64, no. 3, pp. 777–804, 2013.
[35] J. Dong, R. Curtmola, and C. Nita-Rotaru, "Practical defenses against pollution attacks in intra-flow network coding for wireless mesh networks," in *WiSec*, 2009.
[36] S. Jiang and Y. Xue, "Providing survivability against jamming attack for multi-radio multi-channel wireless mesh networks," *Journal of Network and Computer Applications*, vol. 34, no. 2, pp. 443–454, March 2011.
[37] A. Morais and A. Cavalli, "A distributed and collaborative intrusion detection architecture for wireless mesh networks," *Mobile Networks and Applications - Springer*, 2013.
[38] M. Kim, V. K. S. Iyer, and P. Ning, "Mrfair: Misbehavior-resistant fair scheduling in wireless mesh networks," *Ad Hoc Networks*, pp. 299 – 316, 2012.
[39] S. Szott, "Selfish insider attacks in ieee 802.11s wireless mesh networks," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 227–233, June 2014.
[40] J. Sun, C. Zhang, Y. Zhang, and Y. Fang, "SAT: a security architecture achieving anonymity and traceability in wireless mesh networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 2, pp. 295–307, March 2011.
[41] D. Makaroff, P. Smith, N. Race, and D. Hutchison, "Intrusion detection systems for community wireless mesh networks," in *IEEE MASS*, 2008.
[42] A. Hassanzadeh, A. Altaweel, and R. Stoleru, "Traffic-and-resource-aware intrusion detection in wireless mesh networks," *Ad Hoc Networks*, vol. 21, pp. 18 – 41, 2014.
[43] A. Hassanzadeh, R. Stoleru, and J. Chen, "Efficient flooding in wireless sensor networks secured with neighborhood keys," in *IEEE WiMob*, 2011.
[44] S. George, W. Zhou, H. Chenji, M. Won, Y. O. Lee, A. Pazarloglou, R. Stoleru, and P. Barooah, "DistressNet: a wireless ad hoc and sensor network architecture for situation management in disaster response," *IEEE Communications Magazine*, 2010.
[45] M. Valero, S. S. Jung, A. S. Uluagac, Y. Li, and R. A. Beyah, "Di-sec: A distributed security framework for heterogeneous wireless sensor networks." in *IEEE INFOCOM*, 2012.
[46] D. Wu, D. Gupta, and P. Mohapatra, "QuRiNet: A wide-area wireless mesh testbed for research and experimental evaluations," *Ad Hoc Networks*, vol. 9, no. 7, 2011.

Appendix: Symbols and Parameters

In this section, we represent the definitions of all of symbols and parameters used in the entire article. Table 3 depicts all symbols categorized in different groups based on their relevance.

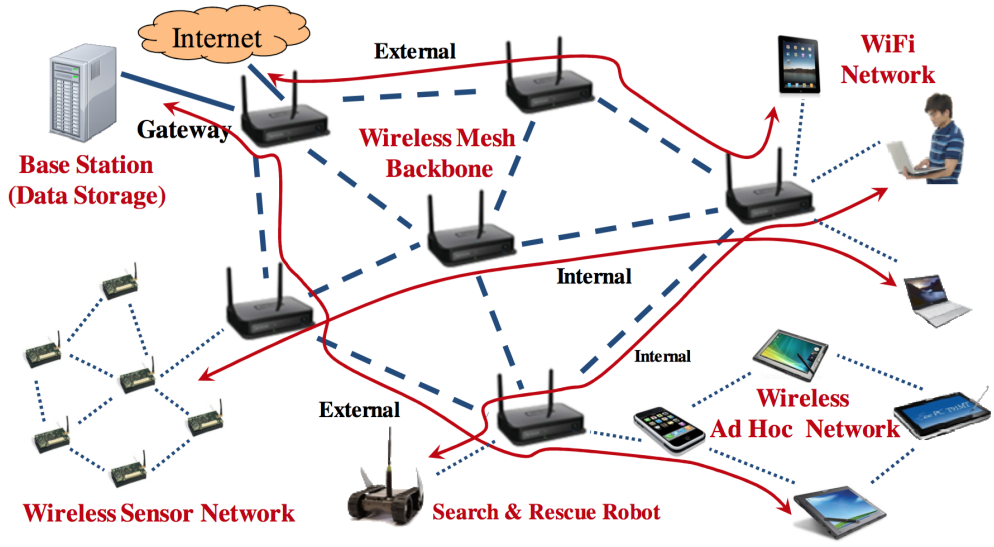

Figure 1: A typical mesh network providing network services for a variety of users.

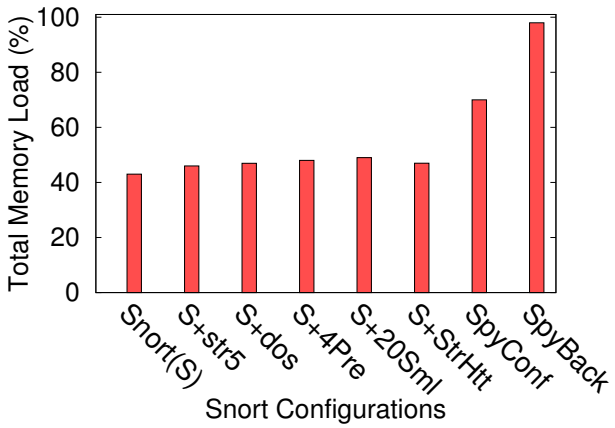

Figure 2: The effect of Snort configuration on memory consumption.

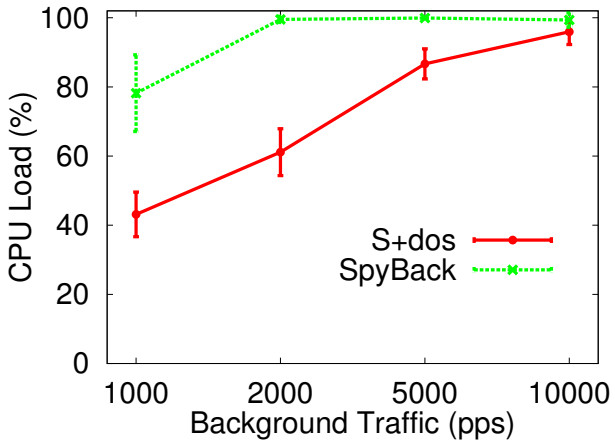

Figure 3: The effect of Snort configuration and traffic rates (packets per second) on CPU utilization.

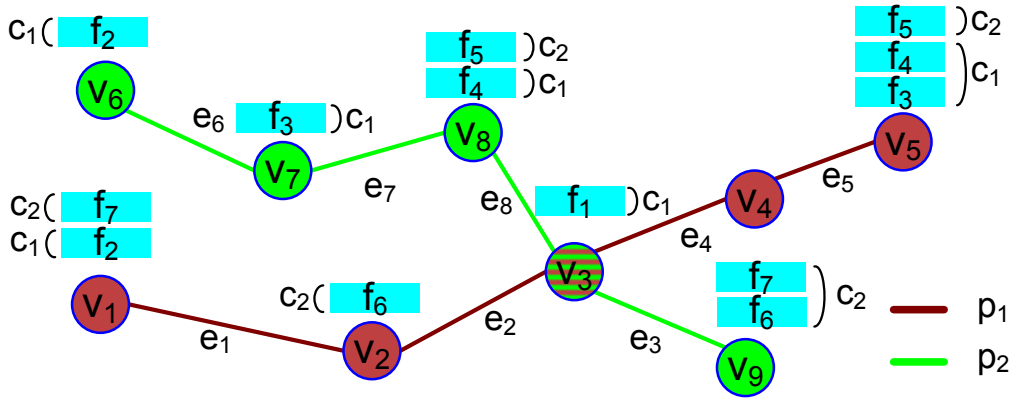

Figure 4: An example graph for a mesh network, consisting of 9 nodes and 8 links. As shown, two paths p_1 and p_2 are present. The nodes run different configurations of Snort, e.g., node v_5 runs Snort functions f_3, f_4 and f_5, which require preprocessors c_1 and c_2.

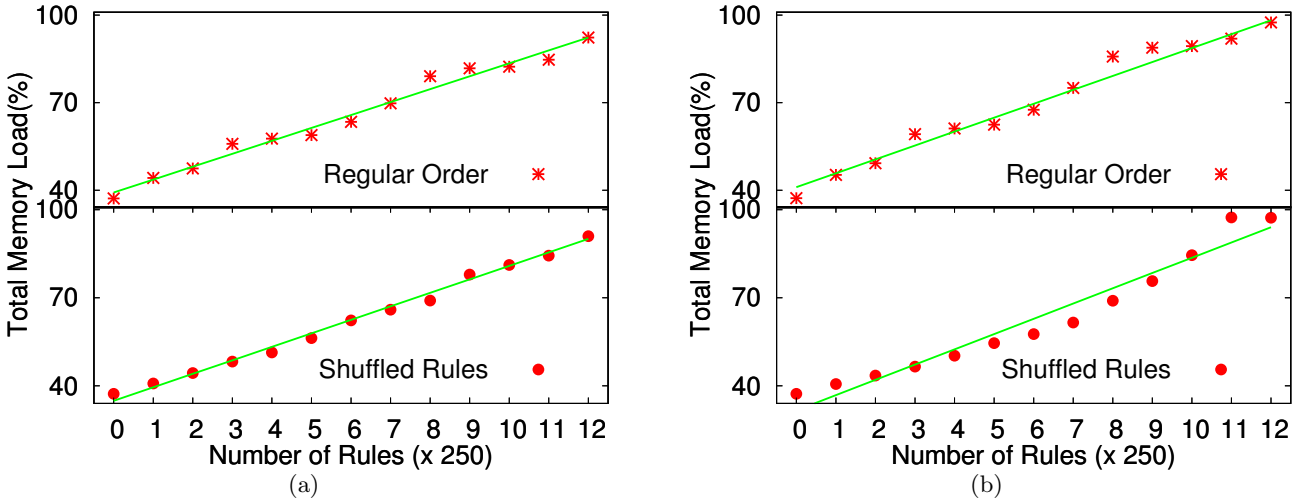

Figure 5: Linearity of memory consumption in different search algorithms as the number of activated rules increases: a) AC-bnfa-nq; b) Lowmem.

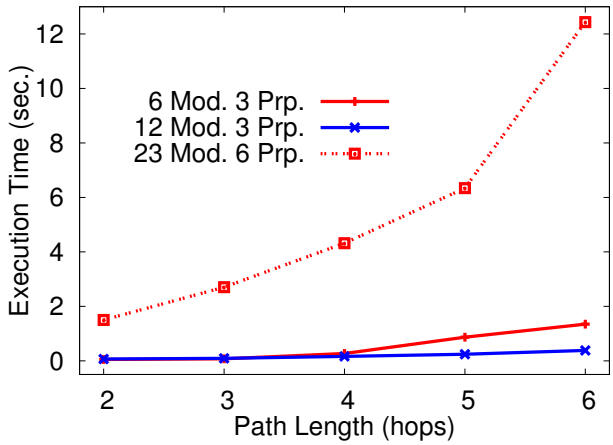

Figure 6: The effect of number of detection modules on the ILP execution time

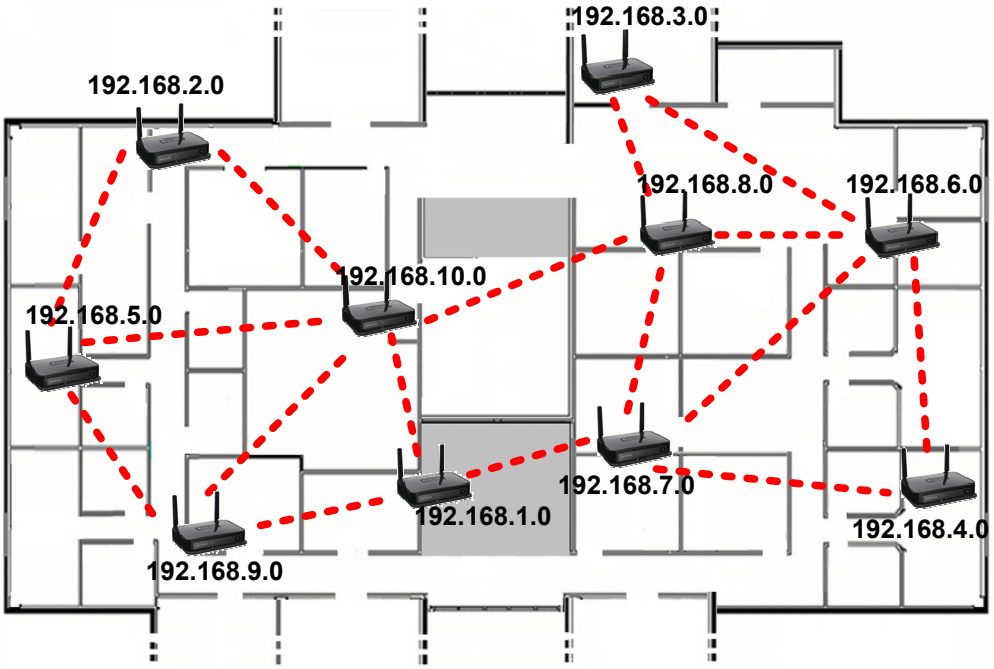

Figure 7: Our department-wide wireless mesh network.

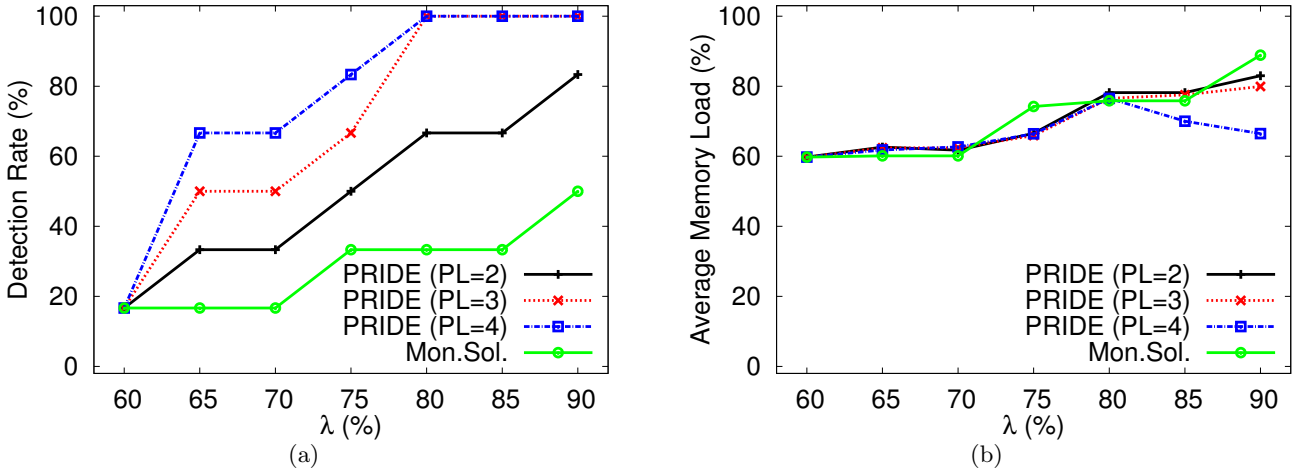

Figure 8: 6-module configuration: effect of λ and PL on a) Intrusion detection rate, and b) Average estimated memory load.

Table 1: Snort rule sets, modularizations, and memory load associated to each module used in our experiments

Rule File	Size	M6	M12	M23	Rule File	Size	M6	M12	M23
community-nntp	1	1	1	2	shellcode	25	1	2	3
community-oracle	1	1	1	2	ddos	30	1	2	3
x11	2	1	1	2	pop3	35	1	2	3
pop2	2	1	1	2	specific-threats	36	2	4	8
community-icmp	2	1	1	2	web-frontpage	38	2	4	9
comm.-inapproprate	3	1	1	2	chat	42	1	2	3
other-ids	3	1	1	2	web-coldfusion	44	2	5	9
community-web-iis	4	2	4	8	community-bot	45	1	2	3
community-policy	4	2	4	8	voip	45	1	2	3
community-exploit	5	1	1	2	imap	60	1	1	3
comm.-web-attacks	5	2	4	8	misc	62	2	4	8
multimedia	5	2	4	9	policy	74	1	2	4
community-game	5	2	4	9	ftp	76	1	3	7
community-dos	6	1	1	2	sql	87	1	3	4
community-smtp	6	1	3	5	icmp-info	93	1	1	1
bad-traffic	6	1	1	2	smtp	94	1	3	5
community-sip	7	1	1	1	web-iis	95	2	5	9
community-web-client	8	2	4	9	web-client	135	2	5	9
community-imap	8	1	1	2	web-php	142	2	5	9
comm.-sql-injection	9	2	4	9	rpc	168	1	3	4
community-virus	10	2	4	9	comm.-web-misc	190	2	5	10
info	10	1	1	2	exploit	208	2	5	10
scan	12	1	1	2	oracle	310	2	6	11
finger	13	1	1	2	web-cgi	357	2	6	11
rservices	13	1	1	2	web-misc	370	3	6	12
comm.-web-cgi	13	2	4	9	netbios	430	3	7	13
nntp	13	1	1	2	comm.-web-php	463	3	7	12
tftp	16	1	3	7	web-activex	587	3	8	14
snmp	16	1	1	2	backdoor-frag1	172	6	8	15
attack-response	17	1	1	2	backdoor-frag2	172	4	9	16
telnet	19	1	3	6	backdoor-frag3	172	4	9	17
dos	20	1	1	2	backdoor-frag4	171	6	10	18
porn	21	1	1	2	spyware-put-frag1	196	4	10	19
dns	22	1	1	2	spyware-put-frag2	196	5	11	20
mysql	22	1	1	2	spyware-put-frag3	195	5	11	21
icmp	22	1	1	1	spyware-put-frag4	195	5	12	22
p2p	23	2	4	8	spyware-put-frag5	195	6	12	23
community-misc	24	2	4	8					

Configuration	ID	Mem. (%)	ID	Mem (%)
6-Module	M1	13.32	M4	17.33
	M2	14.66	M5	14.66
	M3	13.04	M6	17.33
12-Module	M1	3.40	M7	6.99
	M2	5.14	M8	9.57
	M3	3.75	M9	9.03
	M4	4.52	M10	9.53
	M5	5.49	M11	8.77
	M6	6.13	M12	8.81

Table 2: List of Snort rule files used by the R2A tool for generating exploit against each detection module and the number of generated alarms by each module

Mod. ID	Rule File	#Alarms	Mod. ID	Rule File	#Alarms
M1	other-ids	3	M7	community-web-php	2
	dns	17		netbios	15
M2	ddos	16	M8	backdoor-frag1	33
	chat	33	M9	backdoor-frag2	51
M3	rpc	9		backdoor-frag3	36
	telnet	4	M10	backdoor-frag4	31
M4	p2p	18		spyware-put-frag1	30
	misc	26	M11	spyware-put-frag2	32
M5	exploit	10		spyware-put-frag3	31
M6	oracle	4	M12	spyware-put-frag4	44
	web-cgi	3		spyware-put-frag5	29

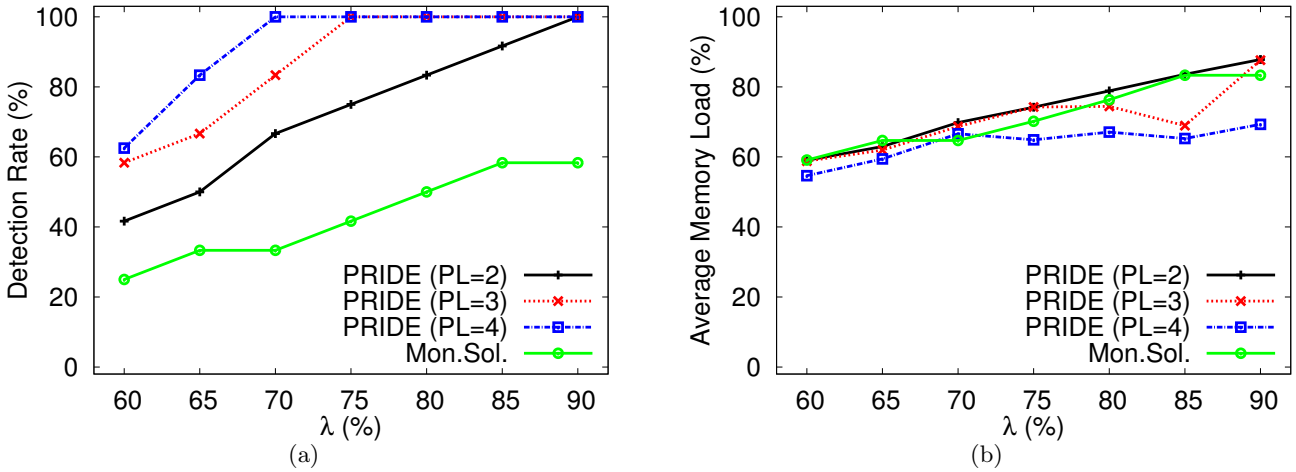

Figure 9: 12-module configuration: effect of λ and PL on a) Intrusion detection rate, and b) Average estimated memory load.

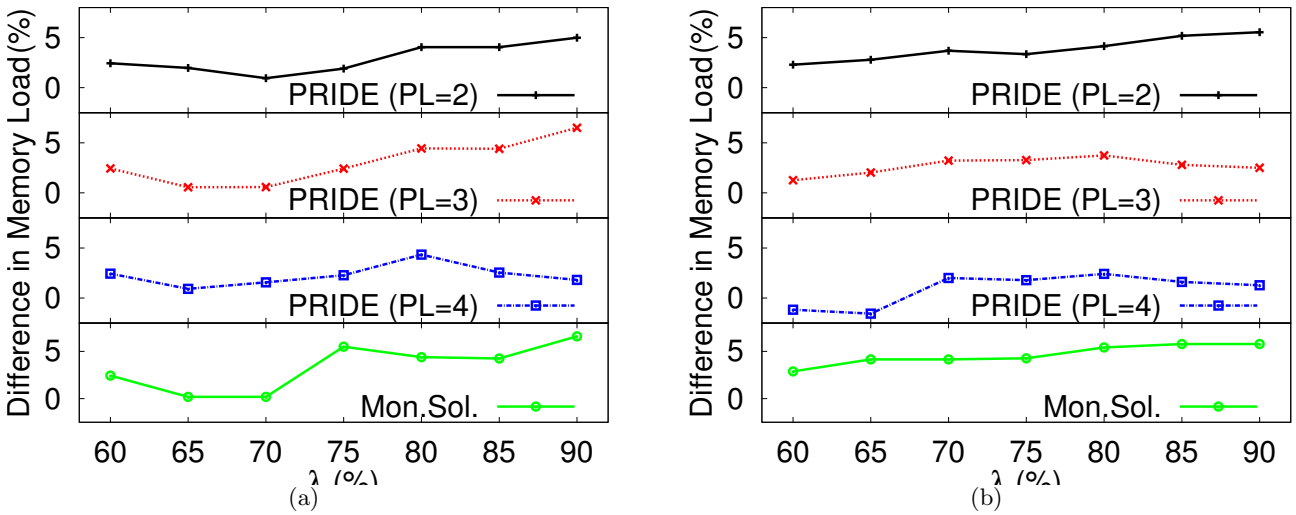

Figure 10: The difference between estimated and actual average memory load: a) 6-Module configuration, and b) 12-Module configuration.

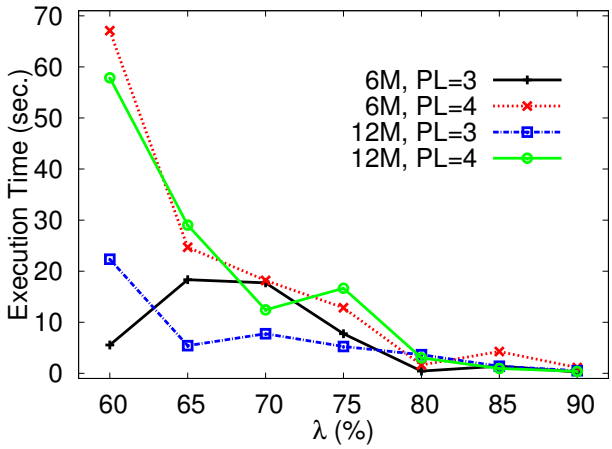

Figure 11: ILP solver execution time for different modularizations, path lengths and memory thresholds.

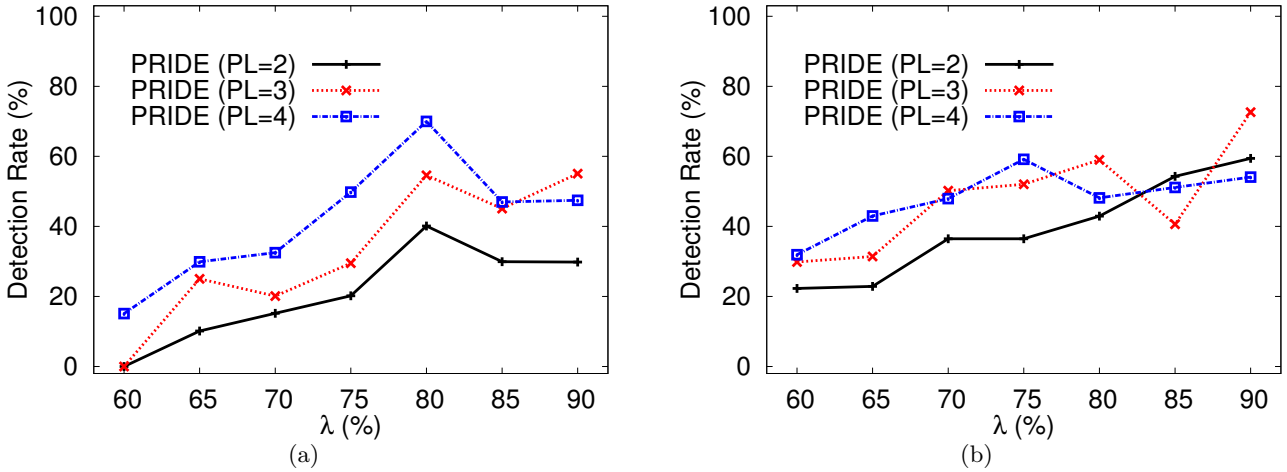

Figure 12: Average detection rate for PRIDE-aware attacks: a) 6-Module configuration, and b) 12-Module configuration.

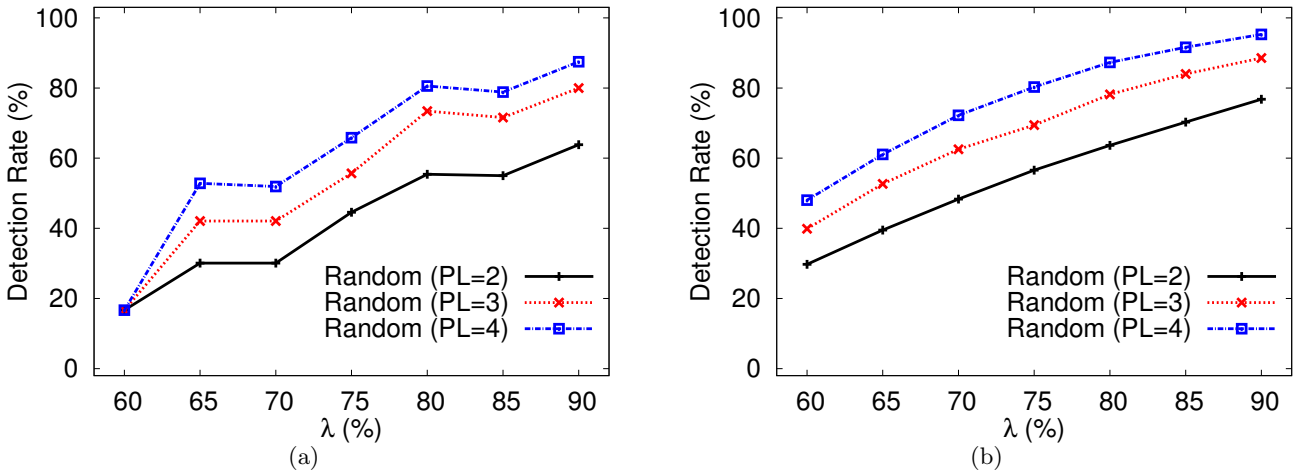

Figure 13: Distributed approach based on random module selection: effect of λ and PL on the intrusion detection rates in a) 6-Module Configuration, and b) 12-Module Configuration.

Table 3: Table of Symbols used throughout this article

Symbol	Description
N	Number of all WMN nodes
Q	Number of all WMN backbone links
G	Reduced graph by removing idle nodes/links
V	Set of nodes in the reduced graph G
E	Set of links in the reduced graph G
P	Set of routing paths in the reduced graph G
n	Number of all nodes contributing in routing
q	Number of all links contributing in routing
l	Number of all active paths in the WMN
p_i	Routing path i in the WMN
v_j	Node j actively contributing in traffic routing
e_k	Link k actively contributing in traffic routing
P_i	Set of all nodes located on the path p_i
$\mathbb{T}_{l\times n}$	Matrix of mapping between nodes and paths
t_{ij}	Entity of matrix $\mathbb{T}$ (v_j is located on p_i)
$\mathcal{F}$	Set of all IDS functions
K	Number of all IDS functions
f_k	An IDS function (set of detection rules)
$\mathcal{C}$	Set of IDS preprocessors
R	Number of all IDS preprocessors
c_r	An IDS preprocessor
$\mathbb{D}_{K\times R}$	Matrix of dependencies between f_ks and c_rs
d_{kr}	Entity of matrix $\mathbb{D}$ (f_k requires c_r)
w	Assigns memory load to all f_ks and c_rs
w_k^f	Memory load caused by IDS function f_k
w_r^c	Memory load caused by preprocessor c_r
W^f	Vector of all w_k^f (load of all functions in $\mathcal{F}$)
W^c	Vector of all w_r^c (load of preprocessors in $\mathcal{C}$)
b_j	Base (initial) memory load of node v_j
B	Vector of all b_j (base load of all nodes in V)
λ_j	Memory threshold in node v_j
Λ	Vector of all memory thresholds λ_j
A	An IDS function and preprocessor distribution
$\mathbb{X}_{n\times K}$	Matrix representation of assigned f_ks to v_js
x_{jk}	Entity of matrix $\mathbb{X}$ (v_j performs f_k)
$\mathbb{Z}_{n\times R}$	Matrix representation of required c_rs to v_js
z_{jr}	Entity of matrix $\mathbb{Z}$ (v_j has to performs c_r)
$\mathcal{F}_j$	Set of IDS functions assigned to v_j
$\mathcal{C}_j$	Set of IDS preprocessors assigned to v_j
L_j	Total memory load on node v_j after running assigned IDS functions and preprocessors
CR_i	Coverage ratio of path p_i