Dottorato di Ricerca in
Computer Engineering and science
Scuola di Dottorato in
Information and Communication Technologies

XXI CICLO

Sede Amministrativa di Modena
Università degli studi di MODENA e REGGIO EMILIA

TESI PER IL CONSEGUIMENTO DEL TITOLO DI DOTTORE DI RICERCA

Distributed architectures and algorithms for network security

Candidato:
Ing. Mirco Marchetti

Relatore:
Prof. Michele Colajanni

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As our society becomes more dependent on networked information systems, it is also becoming increasingly vulnerable to their misuse. The large amount of sensitive information transmitted over computer networks and stored on databases, as well as the high revenues that industries can generate through Internet-based applications, translate in valuable targets for attackers.

Several technologies for increasing network security and dependability already exist. Nowadays, firewalls represent a fundamental first-line defense in any network security infrastructure. The traffic flowing through a network as well as the host activities can be monitored by network and host Intrusion Detection Systems, that are able to detect a wide range of attacks and illicit activities. Besides the inherent difficulties in keeping these technologies updated with respect to increasingly sophisticated attack strategies, the evolution of network-based information systems cause additional issues. In this thesis we consider three main factors that pose critical challenges to network security systems:

- the constant growth of network traffic throughput makes network monitoring increasingly demanding;
- node mobility highlights the deficiencies of security solutions and practices that have been mainly designed to cope with fixed nodes;
- the success of massive and widespread attacks, carried out through distributed hostile infrastructures, demonstrates the limits of traditional network defenses that do not adopt cooperative schemes.

This thesis presents novel distributed architectures, algorithms and protocols that can improve network defenses by keeping them up with the complexity of modern networks. All the proposed solutions are characterized by some sort of cooperation among locally or geographically distributed components. This approach

aims to overcome the lack of global knowledge affecting most of the existing security technologies, that are characterized by centralized or isolated solutions.

The limits in the amount of traffic manageable by traditional Network Intrusion Detection Systems, that is bounded by the computational capacity of one machine, represents the first issue addressed in this thesis. The research is motivated by the observation that a signature-based NIDS implemented through COTS hardware components is not able to perform stateful analysis of traffic flowing in large capacity networks such as Gbit network links [29, 69, 105]. In this context a parallel architecture for network intrusion detection, able to achieve higher throughput by adding more parallel traffic analyzers, represents an ideal solution for stateful analysis in present and future networks [57, 106, 112, 129]. A common drawback of the existing parallel architectures is their inability to perform stateful signature-based analysis together with dynamic load balancing. We propose an innovative *state migration* framework [26], that allows the parallel traffic analyzers to cooperate by exchanging state information. To the best of our knowledge, no other technology exists that allows to exchange state information among stateful and signature-based NIDS. Moreover, we propose a new dynamic load balancing algorithm [5, 69], that is able to leverage our state migration framework to dynamically distribute the load among the traffic analyzers without jeopardizing the analysis reliability. Both these contributions are included in ParNIDS [29, 69], the first prototype of parallel architecture for stateful network intrusion detection that can perform stateful pattern matching and dynamic load balancing among its parallel sensors.

Node mobility is another characteristic of (mainly future) networks that can cause novel security issues. The Mobile IP protocol [87, 89, 90] allows a node to transparently roam across different networks, while being always reachable through its home IP address. In this thesis, we highlight for the first time a novel attack strategy, defined *mobility-based evasion*, that combines traditional NIDS evasion techniques [97] with node roaming. We present several realistic scenarios in which an attacker can take advantage of the NIDS inability to statefully analyze mobile connections by executing "stealth" attacks, undetectable by state-of-the-art NIDS. Moreover, we describe the incompatibility issues between *triangular routing* (default routing scheme of the Mobile IP protocol) and *egress filtering* [127], that is the best practice [54] for modern modem, router and firewall configuration. To nullify the detrimental effects of node mobility on stateful analysis we propose an innovative cooperation scheme, that for the first time enables geographically distributed network intrusion detection systems to cooperate and to perform stateful analysis on mobile connections. The main idea behind this new technique is to make the NIDS state information related to a mobile node to "follow" the node in the destination network.

To overcome the incompatibility between triangular routing and egress filter-

ing we propose *Secure Triangular Routing* (STR) [68], a novel routing scheme that relies on cooperation among Mobile IP agents and firewalls to enable triangular routing together with egress filtering. With respect to other optimized routing schemes for the Mobile IP protocol [91, 128, 136] STR does not require any modifications in the correspondent nodes and/or in their networks and increases the compatibility with existing standards.

Massive attacks generated by self replicating malware towards hosts and networks distributed across the Internet represent the third security issue considered in this thesis. A typical case study is represented by infection attempts carried out by Internet worms when trying to compromise new hosts, possibly with the goal of expanding a *botnet*. In this context, several networks are targeted by the same attacks that aim to exploit widespread software vulnerabilities. To address these issues we propose two novel cooperative architectures for malware detection and analysis.

The first cooperative architecture [27] is based on a *hierarchical* alert aggregation tree, in which only the new threats are propagated to the root that coordinates malware analysis. With respect to traditional malware detection and analysis solutions, the proposed architecture guarantees efficiency because it avoids duplicate analysis of the same threat, and it is able to generate preemptive alerts and even selective alerts, because analysis results are disseminated to all the networks participating in the cooperative effort that can be actually affected by a certain threat. It is also important to observe that all activities are carried out automatically without the need for any human intervention. While similar principles represent the basis of several distributed tools for fighting spam, this is the first proposal of a cooperative architecture that provides automated and runtime analysis of new threats and a dissemination of analysis results that allow a participating network to be alerted before being attacked.

The second cooperative architecture for malware detection and analysis [70] is based on a *peer-to-peer* design that represents an evolution of the hierarchical infrastructure. While fulfilling all the goals of the hierarchical architecture [27], the peer-to-peer scheme solves several problems that affect the hierarchical design by providing higher scalability, self-adaptive properties and higher dependability since it lacks single points of failure.

A common trait of all the research activities presented in this thesis is the experimental evaluation of the proposed architectures and algorithms. The state migration framework, the new stateful and load balancing algorithm for parallel and stateful intrusion detection, the ParNIDS architecture, the distributed NIDS cooperation technique for countering mobility-based evasion, the STR routing scheme, and both the cooperative architectures for malware detection and analysis have been designed and implemented through Open Source software, such as Snort [111], Prelude [96], Nepenthes [79], Iptables [50], Ebtables [39], Dynam-

ics [38], Past [37] and Scribe [103]. All prototype functions have been validated, and their performance has been tested to demonstrate their ability to operate at runtime in realistic scenarios.

As we think that security is strictly related to availability, we have considered also *Byzantine Fault Tolerance* (BFT) issues that may affect both centralized information systems and distributed cooperative services. Two new protocols and systems have been proposed: Aardvark [4,24] and FlightPath [59]. These research activities, carried out in collaboration with the Department of Computer Science of the University of Texas at Austin, are only outlined here and not reported in this thesis, because the main focus is on distributed architectures and algorithms for network security. Additional details can be found in [4, 24, 59].

Aardvark advocates a new approach to building BFT systems. Other proposals [1, 19, 31, 55] achieve very high performance in fault-free scenarios, thus demonstrating the practicality of BFT systems. However, their performance is drastically reduced when faults do happen, demonstrating their inability to really tolerate byzantine faults. Indeed, the common approach in designing BFT systems is to introduce fragile optimizations to increase peak performance, at the expense of the throughput in faulty scenarios. In [4] we redefine the principles on which BFT systems are built by following an approach inspired by the Maximin [85] strategy in game theory: we deliberately give up peak performance in order to maximize the worst-case throughput. The new BFT system is able to maintain a throughput of thousands operations per second even in faulty cases.

FlightPath [59] proposes a new approach in the design of cooperative services able to tolerate both selfish and Byzantine peers. Several cooperative services exist [2,60] that rely on Nash Equilibrium [78] to design incentives and punishments schemes in which rational peers have no better strategy than to follow the protocol. However, the price for designing a protocol that is provably a Nash Equilibrium is paid in terms of low performance and low flexibility. In [59] we demonstrate that the concept of approximate equilibria can be used to design rigorous protocols, in which the gain that a selfish peer can achieve is provably bounded, without sacrificing performance and flexibility.

This thesis is composed by five chapters.

In Chapter 2 we address the security issues related to the stateful analysis of high speed network links, we present the novel state migration framework integrated with the load balancing algorithm, and the overall ParNIDS architecture.

In Chapter 3, that deals with security issues related to mobility, we define mobility-based NIDS evasion, present the NIDS cooperation scheme that allows stateful analysis of mobile connections, and describe the Secure Triangular Routing scheme for Mobile IP.

In Chapter 4, we propose two novel architectures (hierarchical and peer-to-peer) for cooperative malware detection and analysis, that aim to provide better

defenses against massive attacks generated by self-replicating malware.

Conclusions and future works are drawn in Chapter 5.

Chapter 2

Architectures and algorithms for parallel intrusion detection

2.1 Introduction

Network traffic analyzers, such as *Network Intrusion Detection Systems* (NIDS) are one of the most valuable components in the design of modern secure information systems and network infrastructures. A NIDS is able to analyze network traffic with the goal of looking for malicious network packets and illicit network activities.

While the first generation of *stateless* NIDS is vulnerable to a multitude of insertion and evasion attacks [97], the most recent NIDS architectures perform a *stateful* traffic analysis that cannot be easily eluded. To achieve high reliability, a traffic analyzer has to perform an in-depth and stateful analysis on each network packet. This implies that, for example, a traffic analyzer needs to track each distinct connection, and reassemble and analyze every Ethernet frame at wire-speed. Stateful traffic analysis comes at the price of an increased computational complexity that may prevent a single NIDS sensor to perform a sound, stateful analysis on all the traffic flowing through high speed network links. Moreover, stateful analysis relies on management of large quantities of contextual data (the so called *state*) that poses additional burden on NIDS sensors in terms of computational load and memory usage [36, 105].

In a scenario where the number of connected devices, the capacity of network links and the diffusion of network-related applications and services are continuously growing, traffic analysis is becoming a complex and expensive task. The number of concurrent connections and the transmission rate of the traffic generated in modern network installations may easily overwhelm the amount of memory and computational power of a stateful traffic analyzer. Depending on hardware

configuration and traffic pattern, a traffic analyzer built using common hardware may not be able to reliably monitor 100 Mbps network link [105], not to mention high speed network connections (Gbps and above). Even configurations that are specifically designed for dealing with high speed analysis (e.g., fast logging in binary format) [101] are not able to solve the problem of stateful and fully reliable analysis on large capacity networks. (See Section 2.10 for the performance characterization of a common NIDS sensor).

Traffic analyzers able to cope with high speed networks (like Gbps Ethernet links) are typically implemented through custom hardware components. Examples of some commercial and research products are reported in Section 2.2. However, these hardware oriented appliances suffer some relevant limits: they are expensive and not flexible, and even worse they do not represent a long-term solution to scalability issues. Although hardware appliances are able to reach high performance, they will likely be overwhelmed in the near future, because the network bandwidth grows more rapidly than the Moore Law [100].

To allow the analysis of high traffic volumes, while avoiding the drawbacks of custom hardware implementations, parallel NIDS architectures have been proposed [29,57,106,129]. A parallel architecture is not a problem when the analysis is stateless because each parallel sensor can analyze a fraction of the overall network traffic independently, without the need to cooperate with the other sensors. On the other hand, it still represents an open problem to apply stateful algorithms when the traffic is spread among multiple sensors. In a similar context, each distributed or parallel sensor is reached only by a subset of the overall traffic, and this partial view prevents the possibility of a fully reliable analysis.

In this chapter we present an innovative parallel NIDS architecture (ParNIDS) that is based on an original state management and migration framework that makes multiple NIDS sensors able to exchange internal state information. Thanks to state migration it is possible to propagate internal state information among cooperative NIDS sensors, thus providing each sensor with all the information that is necessary to perform a stateful traffic analysis. To the best of our knowledge, this is the first NIDS cooperation mechanism that allows a true exchange of structured internal state information and a consequent execution of advanced traffic analysis algorithms, such as packet content inspection (a detailed explanation of the differences between the proposed framework and other works proposed in literature is given in Section 2.2). The viability of the proposed approach has been demonstrated through a prototype, based on the the integration of the proposed framework with the open source software Snort. (Snort represents the most widespread NIDS software as well as the de-facto standard in the field of signature-based traffic analysis [25,120].)

The *ParNIDS* architecture aims to achieve high performance and scalability through multiple conventional traffic analyzers that operate in parallel for the

analysis of different traffic portions. ParNIDS is characterized by several novel features:

- it is able to analyze high speed traffic without the need for custom hardware components, hence being alternative to the common solutions for high performance traffic analysis;
- it is inherently scalable, because higher performance can be reached by adding further parallel components to the architecture;
- it includes an innovative mechanism for dynamic and fine-grained traffic distribution;
- it includes a novel load balancing mechanism that is based on the migration of the NIDS internal state information among the analyzers of the parallel architecture [26];
- it is able to perform stateful analysis on dynamically reassigned connections.

The viability of the proposed solutions and the performance evaluation is carried out through an operative prototype of the ParNIDS architecture.

The rest of this chapter is organized as follows.

Section 2.2 compares our proposals with respect to other works in the fields of high speed traffic analysis and parallel architectures for intrusion detection.

Section 2.3 reviews the concept of internal state with respect to the main analysis algorithms applied by commonly deployed NIDS.

Section 2.4 evidences the most relevant issues and requirements that are related to NIDS state migration, and contains a detailed design of the proposed state migration framework.

Section 2.5 contains a detailed description of the ParNIDS parallel architecture.

Section 2.6 described the algorithms and techniques allowing the ParNIDS architecture to perform a reliable and stateful analysis of the incoming network traffic while balancing the load among the cooperative sensors.

Section 2.7 details the realization of the ParNIDS prototype. In particular, Section 2.7.1 deals with the modification introduced in Snort in order to enable cooperation, while Section 2.7.2 focuses on the implementation of the ParNIDS architecture.

All the experimental results that demonstrate the viability and the performance of the proposed state migration framework are contained in Section 2.8, while a thorough experimental evaluation of the complete ParNIDS architecture is included in Sections 2.9 and 2.10.

2.2 Related work

The vast majority of high speed NIDSs (both commercial products and research prototypes) is based on custom hardware components, such as *Application Specific Integrated Circuits* (ASIC) [53,119], *Field Programmable Gate Arrays* (FPGA) [10, 113, 114] and *Network Processors* (NP) [23, 130]. Although they are able to achieve high performance, they do not represent a long term solution to ever growing scalability issues. Moreover, they are characterized by high costs and low flexibility.

The problem of traffic analysis in large network installations has been addressed by *distributed* NIDS, that have been extensively investigated in literature [11, 109, 110, 123] and that should not be confused with the class of parallel NIDS architectures here considered. Generally speaking, a distributed NIDS is composed by an arbitrary number of traffic analyzers that are deployed over different subnets. There are various examples of commercial and open-source software solutions implementing a distributed NIDS [84, 96, 111]. The proposed *parallel* architecture follows a completely different approach. A parallel NIDS can be seen as a single logical element composed by multiple traffic analyzers that operate concurrently on a portion of the traffic flowing through the same link. It is also important to observe that the proposed architecture represents a completely software-based solution where every element is built on standard, flexible and inexpensive hardware.

To the best of our knowledge, the proposed solution is innovative because no other parallel architecture for traffic analysis is able to guarantee both stateful signature-based traffic analysis and dynamic load balancing functionality.

The first valuable implementation of a parallel NIDS architecture is described in [57]. Its dispatching process has two limits. The first is represented by the low precision of the packet classification, that may be carried out only on the basis of the IP addresses. This is a relevant problem in many modern network configurations that are behind proxy firewalls and NAT. When an entire, possibly large, computer networks is hidden behind one IP address, its traffic may easily overwhelm the capacity of one traffic analyzer that has received all packets coming from that address. The second limit is related to the lack of a mechanism to dynamically adapt the load distribution to the current traffic patterns, because traffic dispatching is based on a set of static rules. This deficiency is a key problem because network traffic patterns may fluctuate rapidly, with a consequent risk of leaving some analyzers almost unused while other overloaded analyzers are losing packets.

The main contribution of other parallel architectures in [3, 21, 106, 129] is represented by a hardware-based load balancer, that is able to dynamically dispatch traffic according to the load of traffic analyzers. However, all of them balance

the load through a redistribution of the packets without considering the related state information. As a consequence, the network traffic involved in re-balancing cannot undergo a stateful analysis but only a stateless one, which can be easily evaded by malicious packets [97]. ParNIDS differentiates from all the previous systems, because it takes advantage of an innovative *load balancing* and *state migration* [26] mechanisms that can dynamically reassign an already established connection to an arbitrary analyzer. These mechanisms allows traffic analyzers to perform a stateful, in-depth analysis, that guarantees maximum detection accuracy and scalability through load balancing properties. The dispatching algorithm is able to classify network packets on the basis of many information, such as protocol class, IP address, and port number. This fine grain dispatching allows the parallel architecture to achieve an effective load sharing among multiple traffic analyzers.

A cooperation technique for Bro NIDS sensors based on the exchange of some relevant information (e.g., the number of detected portscans) is presented in [112]. Their proposal is strictly related to Bro implementation and configuration issues. Indeed, it relies on the concept of "synchronized variables", that is, a set of variables that are identified at configuration time and that share the same value among all the cooperating sensors. Every change in the value of a synchronized variable causes a sensor to send an update message to all the cooperative sensors through a push-based model.

On the other hand, our system is based on the migration of internal state information related to one or more of the analyzed traffic flows. The proposed framework does not rely on any novel state management paradigm (such as synchronized variables), thus being generally applicable to any NIDS. Moreover it makes it possible to import and export state information on-demand, only when necessary (in conjunction with load balancing) and allows to select at runtime (rather than at configuration time) which state information are required.

Another relevant difference is represented by the communication scheme employed to exchange state information. In [112] each sensor sends an update message to all the other cooperative sensors, without expecting for replies or acknowledgments. In our framework, external state import and export operations rely on a RPC mechanism, and are always implemented as a one-to-one communication. This choice ensures easy management of possible communication errors, better scalability and adaptability to heterogeneous state management policies.

The architecture proposed in this chapter moves the system bottleneck from the traffic analyzers to the scatterer, that remains the only not replicated component of the architecture. Some possibilities for moving ahead this bottleneck are outlined in the paper, but removing this system limitation possibly at the level of the operating system kernel remains a topic for future research. Another interesting investigation is represented by the possibility of extending the results of

this paper to architectures that do not consist of homogeneous parallel analyzers. This would require the definition of a standard external state representation that is suitable for traffic and state migration among different kinds of traffic analyzers.

2.3 NIDS internal state

We classify the network traffic analysis procedures that can be carried out by existing NIDS systems in three major sets.

- **Statistical anomaly analysis**. This type of analysis is based on the computation of several statistical metrics on large packet sets. Intrusions are detected when the measured statistics diverge from the traffic model that is considered normal and benign. Examples of common attacks that are detectable by this method are ping sweeps, port scans, SYN floods, worm propagation attempts.

- **Signature pattern matching**. This method is the most common detection system that is used to detect known attack patterns by inspecting the packet payloads. A stateful NIDS that is based on signature pattern matching comes with a custom language used for signature definition. It looks for known attack signatures in analyzed network packets, where a signature is represented by a sequence of bytes that characterize one or more intrusions. The underlying concept is that it is (ideally) impossible to perform an attack without sending to the victim host a sequence of network packets including the signature. This method can effectively detect remote buffer overflow attacks (for example, by intercepting the NOOP sleds or some common shellcode variants), remote exploits, and all other intrusions that can be characterized by a known signature.

- **Protocol anomaly detection**. The idea is that it is possible to identify anomalies in a data stream behavior by describing all protocol details through a Finite State Machine. A stateful inspection of that protocol may reveal attempts to divert from the legitimate usage patterns, that are interpreted as attempted attacks. Brute force cracking and FTP bounce attack are typical examples of illicit activities that rely on application protocol misuse and that are detectable through a protocol anomaly detection.

The internal state of a NIDS is a representation of the network traffic features that are relevant to the specific detection algorithms. Statistical anomaly analysis and protocol anomaly detection algorithms intrinsically rely on information stored

Type of analysis	State data profile
Statistical anomaly analysis	Aggregated statistical information related to all the network traffic that has been previously collected on the monitored network segment.
Signature pattern matching	Reordered and reassembled information streams built from previously collected network packets. A separate data stream should be stored for each currently active connection.
Protocol anomaly detection	Protocol FSM state for all active connections.

Table 2.1: Classification of NIDS internal state information with respect to the analysis algorithm

during previous detection activities. Indeed, both the value of an aggregated, statistical feature and the current state in a protocol FSM include information related to the traffic analyzed in the past. On the other hand, it is possible to perform a *stateless* signature pattern matching by analyzing each packet on its own, without any information related to previous network activities. The problem is that stateless signature pattern matching can be easily evaded through well known techniques, such as the fragmentation of the attack signature in different network packets [97]. When a signature is fragmented, there is not a single network packet containing all the information necessary to detect the intrusion attempt. Hence, a NIDS sensor analyzing each packet separately is not able to identify a fragmented attack. Other examples of stateless NIDS evasion are represented by overlapping and out-of-order packets, as well as packets with headers specifically crafted to avoid detection.

To effectively combat evasion techniques, modern pattern matching NIDS sensors perform a stateful traffic analysis. This requires that all received packets are temporarily stored, reordered and reassembled. Stream reassembly allows a NIDS sensor to inspect a complete and ordered information stream, in which all the possible attack signatures can be easily detected.

From the above analysis, it emerges that all the intrusion detection approaches rely on some sort of internal state information that represents the current state of the network. However, different detection algorithms adopt different logical abstractions for the generation of heterogeneous state information. Table 2.1 summarizes the internal state representation for each of the three methods.

The generation and continuous update of a complete internal state is commonly carried out by traditional centralized NIDS where one sensor analyzes all the network traffic flowing through a monitored link. However, when the analyzed traffic is processed by multiple sensors operating on the same link (*parallel*

NIDS) or on different links (*distributed NIDS*), the global network state is fragmented because each sensor has only a limited view of the global state. In these contexts, it is impossible to detect attacks that are perpetrated through traffic portions analyzed by different sensors. The ParNIDS architecture addresses this issue through a novel sensor cooperation mechanism that supports migration of relevant state information from one sensor to another. Such framework enhances detection accuracy and allows us to deploy NIDS sensors among any kind of network topology.

2.4 NIDS state migration

2.4.1 NIDS state migration requirements

The aim of the proposed framework is to enable cooperative sensors to exchange valuable state information, thus enabling a distributed and/or parallel NIDS to detect illicit activities that would not be discovered otherwise. This scenario poses several new challenges, that we address through the careful design choices outlined below.

- It is important for NIDS sensors to perform traffic analysis in real-time, hence the impact of state migration operation on the sensors has to be as low as possible. Packet losses due to state migration overhead are not acceptable.

- Both state import and export operations have to keep consistent the internal state of sensors involved in cooperation. These operations should not alter the analysis reliability and cause residual dependencies on imported state information.

- State migration has to be robust to transmission delays. Moreover, it must be possible to perform state migration even out of order with respect to the contextual network traffic.

- The framework should be easily adaptable to various applications, hence it must provide state migration mechanisms without imposing limitations about state management, dispatching policies and initiation schemes.

- To achieve high performance and adaptability, it should be possible to export and import only partial state information (e.g., all the information related to a particular host or transport level connection).

- It should be easy to extend the communication protocol for the transmission of external state representations, even with the goal of allowing the migration of new state information among heterogeneous network components.

The inherent difficulties concerning the designing and implementation of a framework complying to the previous requirements are worth the efforts of achieving a truly cooperative NIDS system. A similar architecture can address scalability issues related to the analysis of large amounts of traffic flowing through high capacity networks and to the inspection of packets flowing through complex network topologies consisting of multiple segments.

In network-related contexts, state migration has been successfully used to realize level 4 [30] and level 7 [6] front-end switches for Web clusters [14]. In this case, the transmitted state information consists of a description of a TCP connection current state. TCP state migration is usually carried out through a custom migration protocol, that usually requires the exchange of one network packet, and where migration activities are initiated by the Web switch. The NIDS state migration proposed in this paper is by far more complex than a TCP connection migration. The proposal in some sense is more similar to process migration schemes [71] that have been extensively analyzed in distributed (operating) systems and mobile agents [93].

Another proposal for NIDS sensor cooperation [112] represents a preliminary and partial solution that is related to a specific NIDS software (see Section 2.2 for further details).

2.4.2 NIDS state migration framework

A typical NIDS [94] consists of three main components[1]:

- A packet input layer feeds the NIDS sensors with the data to be analyzed. It can be based on a packet capture from a promiscuous network interface, a queue coming from kernel space (such as in a Snort in-line setup [20]), or a software library reading a previously acquired network dump. Following the CIDF architecture, we denote this layer as the *Event generator* or E-box.

- One or multiple detection engines perform the network traffic analysis (this component corresponds to the CIDF *Event analyzer* or A-box).

- An output layer handles alerting, logging and packet dumping (that is, the CIDF *Event database* or D-box).

[1]We omit the fourth component defined by the CIDF specification, Response units "R-boxe"), because intrusion detection does not necessary imply a reaction to hostile activities. R-boxes are supposed to *kill processes, reset connections, alter file permissions, etc.*

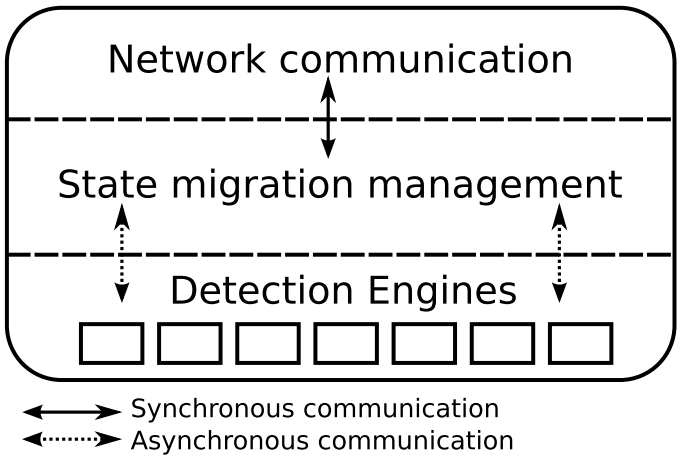


Figure 2.1: NIDS state migration framework

Cooperation at the packet input layer is impossible, because input data depend on network setup and its contents are independent variables. On the other hand, cooperation at the output layer is a well established practice, with many products allowing alert aggregation and correlation (e.g., Prelude [96]).

However, NIDS cooperation becomes a real challenge at the detection layer, especially when real-time intrusion analysis is necessary. Transferring state data between stateful detection engines permits a great improvement of the analysis accuracy. In particular, this kind of cooperation allows a detection engine to discover an intrusion by importing external state data that are generated by other detection engines, thus detecting activities that could not be discovered otherwise.

The proposed state migration framework consists of three main components that are shown in Figure 2.1:

- The network communication layer transfers state data through the network.
- The state migration management layer interprets cooperative sensors communications.
- The detection engines layer feeds each engine with imported data and extracts states.

The network communication layer of the cooperative sensors acts as a Remote Procedure Call (RPC) server that listens for remote procedure calls. Cooperative sensors or heterogeneous external elements, such as intermediate components for state migration management, can easily interact with NIDS sensors implementing state migration by calling state *import* or *export* remote procedures. This design choice has several advantages:

- NIDS sensors implementing state export and import mechanisms are not required to be aware of the cooperative sensors logic and of the message distribution policy.
- Each NIDS is always ready to perform required import and export operations at run-time, without any need for ad-hoc configurations.
- State export and import are implemented as a one-to-one communication, hence the computational overhead related to a state exchange operation is independent of the number of cooperative sensors.
- Values returned by remote procedures allow the caller to handle possible import and export errors.

The proposed framework does not pose any limitation on the deployment of the cooperative elements. However, performance and security issues suggest to rely on a dedicated network for sensors management, rather than transmitting internal states and other control information through the same link of the monitored traffic.

The state migration management layer handles the state import and export requests received by the network communication layer and interacts with the detection engine interfaces.

To ensure maximum flexibility, the state migration framework should allow the transfer of only parts of a detection engine state data. To this purpose, we must characterize the network traffic part which is mapped in a certain block of the state data structure. Our choice is to follow the same criteria that are adopted by the detection engine. Detection performed on protocols belonging to the OSI layer 3 (network level) are discriminated on the basis of the network addresses. Layer 4 (transport level) and higher layer detections are discriminated on the basis of transport level streams that are identified by transport level protocols, source network address, source port, destination network address and destination port. As the transport level identifiers are a superset of those needed by the network level, they can be easily integrated into a single structure type.

To guarantee adequate performance to the detection mechanism, the state migration management layer should be decoupled from the detection engines layer. However, it is essential to use a synchronization construct to lock the detection engines internal state representation, thus allowing to perform state import and export operation on a consistent internal state. Concurrent detection may be slowed because of the state locking, hence the locking duration has to be sufficiently short to prevent network packets loss. Experimental results, described in Section 2.8.2 demonstrate that this constraint can be easily satisfied by our reference implementation.

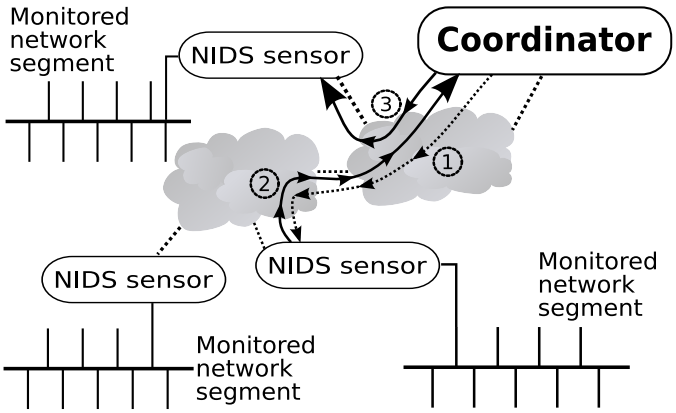


Figure 2.2: Example deployment scheme

The ParNIDS architecture provides cooperative sensors with all the mechanisms required for state exchange. Moreover, to ensure the highest flexibility, the sensor mechanisms are completely decoupled from the state migration policies that depend on the specific deployment scheme.

To describe by example how state information can be exchanged among cooperative NIDS sensors, we refer to a realistic deployment scheme that is composed by three sub-networks as in Figure 2.2. In this scenario we are monitoring three different network segments through three geographically distributed NIDS sensors, in compliance with the typical distributed intrusion detection approach. Moreover, a centralized element called *coordinator* is used to manage and control state migration activities.

A complete state migration transaction is performed in two logical steps: *state export* and *state import*. In the state export phase, the coordinator sends to one of the cooperative sensors a request (the dotted line labeled “1” in Figure 2.2) for the generation of the external state representation that is related, for example, to a particular host. The sensor finds all the relevant information in its internal state and generates a reply to the coordinator (the solid line labeled “2” in Figure 2.2) containing an external representation of the requested state information. The state import phase starts when the coordinator sends to a cooperative sensor a set of external state representations (previously generated by other sensors) that should be imported (the solid line labeled “3” in Figure 2.2). The destination NIDS sensor merges the external state representation with its internal state information, thus increasing its ability to detect illicit network activities.

Although the scheme shown in Figure 2.2 refers to a distributed NIDS architecture, we recall that the proposed framework is highly modular and adaptable to heterogeneous applications and implementations. For example, the same framework can be utilized to create a parallel NIDS architecture, where load balanc-

ing adjustments of network traffic are followed by the transfer of detection state data [29]. In this parallel context, the coordinator element can be easily embedded in the parallel NIDS load balancer.

Another example may refer to the case of wireless mobile networks, where state migration can be effectively used to force an intrusion detection state to *follow* a mobile user. In this context, state migration for all the streams related to a mobile host has to be performed whenever the mobile host changes its access point, as a part of the handover procedure.

Purely distributed deployment schemes without a central coordination element can also be created by embedding a coordinator in every cooperative NIDS sensor. A similar scheme would provide each sensor with an effective way to retrieve the state information deemed as necessary by the analysis logic. As an example, a sensor that receives only part of a connection could ask for other relevant state information to cooperative sensors before analyzing an incomplete stream.

Finally, state migration can be effectively used to transfer all the state information from one sensor to a new sensor that has just been installed on the same link. This possibility augments detection avaliability, because it enables the substitution of NIDS sensors without interruptions in stateful traffic analysis.

The design and implementation of the proposed state migration framework is described in Section 2.7.1.

2.5 ParNIDS: a stateful and parallel NIDS architecture

ParNIDS architecture includes various components that are represented in Figure 2.3.

Starting from the top, the first element of the architecture is the *traffic source*, that is installed on the monitored link. The traffic source is directly connected to the input interface of the *scatterer*. The parallel architecture has one traffic source and one scatterer. The scatterer has several output interfaces that are connected to a variable number of *slicers*. This architecture uses common traffic analyzers to perform stateful analysis of network traffic. The number of slicers and traffic analyzers can be augmented or diminished to achieve the necessary analysis capacity. Slicers and traffic analyzers are connected through a *switch*. An *alert aggregator* is used to concentrate and correlate alerts produced by the traffic analyzers. The *coordinator*, that has to be connected to all slicers and analyzers, manages the dynamic load balancing process.

The traffic source must feed the scatterer with a copy of the network traffic flowing through the monitored link. In Figure 2.3 the traffic source is installed

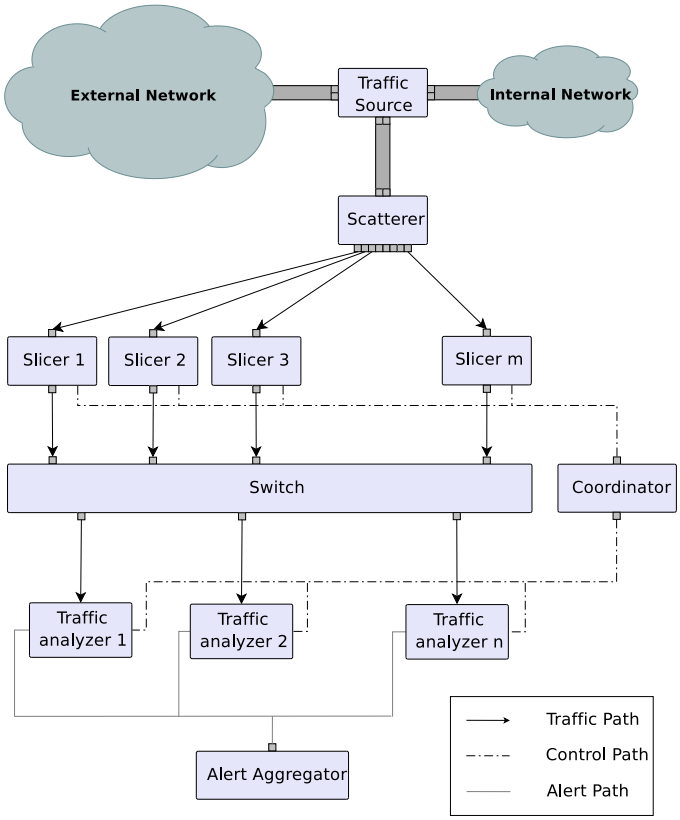


Figure 2.3: ParNIDS: Parallel architecture for stateful traffic analysis.

at the perimeter of a network, thus allowing for a complete analysis of both inbound and outbound traffic. The traffic source is a logical component, that can be implemented with different devices, such as network taps, hubs or switches.

The scatterer is a crucial component, because it is the only element that has to perform operations on network traffic at wire speed. The scalability of the overall architecture is limited by the number of slicers that the scatterer is able to handle. The scatterer has a twofold purpose: to grab every network packet (that is, every Ethernet frame in a LAN environment) transmitted by the traffic source; to distribute packets to the slicers in the fastest way possible. To let the scatterer manage high traffic volumes, it is important to keep the computational complexity of the scattering operations as low as possible. Hence, at this point, it is not possible to use sophisticated load dispatching algorithms or perform even simple traffic analysis. A very simple, yet effective policy is a round-robin dispatching policy that distributes Ethernet frames evenly among all the slicers. In an architecture with K slicers, each slicer will receive $1/K$-th of the total number of frames.

As each slicer receives only a subset of the traffic flowing through the monitored link, at this step it is possible to perform some simple frame analysis before forwarding the packets to the NIDS traffic analyzers. Unlike the scatterer, slicers cannot dispatch packets through a stateless round-robin algorithm. To perform a stateful traffic analysis, all the network packets belonging to the same connection must be received by the same analyzer, and any stateless traffic distribution policy (of which round robin is an example) is unable to guarantee this property.

For this reason, the slicers use a set of properly designed *slicing rules* to identify which traffic analyzer should receive each frame. Each slicer is able to classify packets upon various information, such as level 3 and level 4 protocols, source and destination IP addresses and port numbers. (It is also possible to implement more complex classification criteria, such as those taking into account the content of the payload.)

Frames processed by the slicers are forwarded to the *switch* that applies the routing decisions taken by the slicers in compliance to the slicing rules. Each frame received by a slicer is forwarded towards the port associated with the MAC destination address of the Ethernet header. To ensure that each frame is analyzed by the correct traffic analyzer, this field of the Ethernet header has already been rewritten by the slicers with an identifier of the destination traffic analyzer.

The overall traffic analysis is performed by a configurable number of traffic analyzers that operate in parallel on different slices of the network traffic. With a sufficient number of parallel analyzers, each traffic slice can be correctly managed by a single traffic analyzer without packet loss. The proposed architecture is sensor-agnostic, and can be used with heterogeneous analyzers, such as hardware appliances, software implementations or both. However, if we want to take advantage of the load balancing mechanism without losing the ability to perform

a stateful analysis, it is necessary to deploy traffic analyzers that are able to exchange all useful state information (see Section 2.6).

The *coordinator* monitors the traffic load of the analyzers and performs load balancing operations whenever the load of one or more traffic analyzers approaches the highest analyzable throughput. To this purpose, the coordinator must be connected to all traffic analyzers and all slicers. It is convenient to connect the coordinator and the other elements through ad-hoc network links, out of band with respect to the analyzed traffic. As the coordinator generates only a small amount of network traffic, even low speed network links can be used. Connections with the traffic analyzers allows the coordinator to gather their load state information, that represents the input of the load balancing algorithm. If at least one traffic analyzer is overloaded, a new traffic distribution policy is computed and translated into a new set of slicing rules. These new rules are then deployed on the slicers, with consequent changes of the traffic distribution. The coordinator also manages the *state migration* [26] process, that guarantees a stateful analysis of dynamically redistributed traffic flows (see Section 2.6).

The *alert aggregator* concentrates and correlates all the analysis results. This component provides the network administrator with a single user interface and hides the architecture complexity. The alert aggregator must be connected with all the traffic analyzers. The alert aggregator messages can be transmitted through low speed network links, out of band with respect to the analyzed traffic.

2.6 Integrating load balancing and state migration

An important novelty of the ParNIDS architecture is the load balancing framework that preserves stateful traffic analysis by dynamically redistributing traffic slices and state information among different traffic analyzers.

Dynamic load balancing represents a great improvement with respect to static architectures [57], in which even slight changes in traffic patterns can cause load unbalance and consequent risks of packet loss in one or more analyzers. Our load balancing mechanism allows the architecture to autonomously react to traffic pattern modifications, without any need for human intervention.

Another improvement over other parallel architectures performing stateless analysis on redistributed traffic (such as [3,21,106,129]) is the integration between load balancing and state migration mechanisms, that are necessary to perform stateful traffic analysis even on dynamically reassigned connections.

The *coordinator* in Figure 2.3 triggers and manages all the operations necessary to perform load balancing and state migration. This complex task is performed by the cooperation of four logical components:

- load collector;

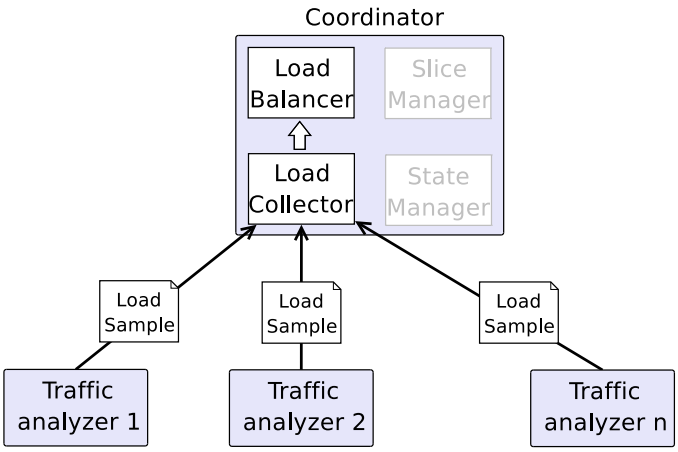


Figure 2.4: The *load collector* gathers load samples from each traffic analyzer and computes the load tracker vector, which is passed to the load balancer.

- load balancer;
- slice manager;
- state manager.

2.6.1 Load Collector

The *load collector* gathers load measures from each analyzer and computes a complete load report. This aggregated information is periodically sent to the load balancer as described in Figure 2.4.

Every T seconds, where T represents a configurable time interval for getting resource information samples, the load collector gathers load information from N traffic analyzers. Let $\overrightarrow{S_t} = (s_t^1, s_t^2, ..., s_t^N)$ denote the vector of load samples gathered at the t-th sampling interval, and s_t^i the load sampled from the i-th analyzer. In the current version, the load collector can use one of the following five load metrics:

1. bits per second (bps) sent by an arbitrary Ethernet NIC;
2. bits per second (bps) received by an Ethernet NIC;
3. CPU utilization;

4. load average samples, for example the number of processes that are currently using the CPU or that are waiting for the CPU[2];

5. load average aggregations and statistics, such as Simple Moving Average (SMA) and Exponential Moving Average (EMA) [5].

The aggregated statistics are motivated by the observation that, in realistic traffic scenarios, pure load samples are not an effective load state representation due to sudden load spikes that may trigger unnecessary load balancing operations. To reduce these risks, the load collector may use some function that computes an aggregated load measure. The considered aggregation functions are the Simple Moving Average (SMA) and the Exponential Moving Average (EMA) that can be computed on the basis of a configurable number of sampled load measures, and are commonly used as load trend indicators [62]. If $\overrightarrow{V_e^i}(t) = (s_t^i, s_{t-1}^i, ..., s_{t-e-1}^i)$ denotes the vector of the last e load samples of the i-th analyzer, for $t > e$, we have that

$$EMA[\overrightarrow{V_e^i}(t)] = \alpha * s_t^i + (1 - \alpha) * EMA[\overrightarrow{V_e^i}(t-1)]$$

where the constant $\alpha = \frac{2}{e+1}$ is the *smoothing factor*. Initially, for $t = e$, the EMA is initialized as the unweighted average of the first e samples.

2.6.2 Load Balancer

The *load balancer* receives the load report from the load collector and applies some *load balancing policy* to evaluate whether it is necessary to activate some traffic redistribution.

In the current version, we find convenient to refer to a typical double threshold mechanism, that defines a Th_{high} and Th_{low} thresholds for activating and deactivating traffic migration, respectively. If at time t the load l_t^i is observed as $l_t^i \geq Th_{high}$, the i-th traffic analyzer is considered *overloaded*. The load balancer reacts by redistributing in a round-robin way some traffic slices among the NIDS analyzers that are not overloaded (that is, whose load $l_t^j < Th_{low}$). The list of the traffic slices that have to be reassigned is sent to the slice manager and to the state manager. To avoid instability that is typical of single threshold mechanisms [13], traffic migration from the i-th traffic analyzer is interrupted by the load balancer only when $l_t^i < Th_{low}$. If the parallel architecture leverages analyzers with different analysis capacity, it is also possible to define a Th_{high}^i and Th_{low}^i for each i-th traffic analyzer.

[2]Linux systems provide three load average measures computed on different time intervals (1, 5 and 15 minutes). We consider the load average computed on the last minute.

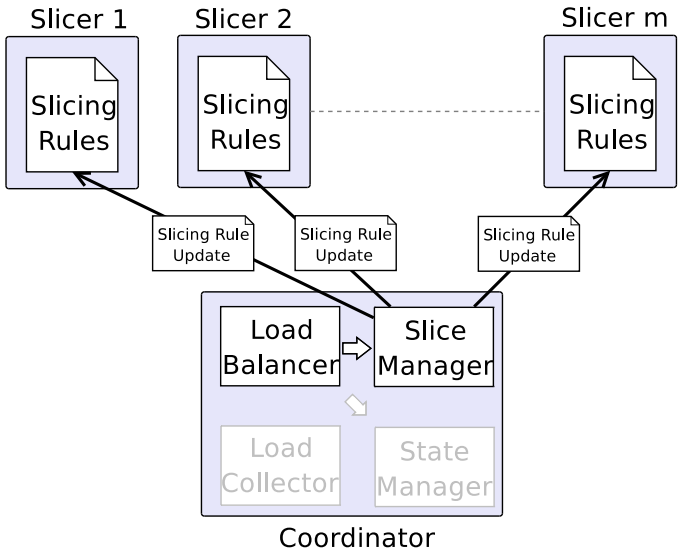


Figure 2.5: The *slice manager* receives balancing instructions from the load balancer and sends slicing rule updates to all the connected slicers.

2.6.3 Slice Manager

The *slice manager* must update the slicing rules used by the slicers to classify and distribute network traffic.

The slice manager keeps a local copy of the slicing rules that are currently deployed at the slicers. Slice manager activities are represented in Figure 2.5. After having received the list of reassignments planned by the load balancer, the slice manager computes an updated set of slicing rules, that are uploaded to all the slicers in parallel.

The load balancing and state migration framework proposed in this paper do not require tight synchronization constraints (see also Section 2.8.1). The only timing requirement is that the slicing rules update has to be completed *before* state migration takes place. This goal is accomplished by activating the state migration only after the slicing rules update.

2.6.4 State Manager

The *state manager* coordinates all the state migration activities. A typical state migration event is shown in Figure 2.6. After the deployment of the new slicing rules, the load balancer activates the state manager by sending all the information related to the changes in the traffic dispatching policy. As an example, let assume that the new set of slicing rules reacts to an overload by moving a subset of traffic

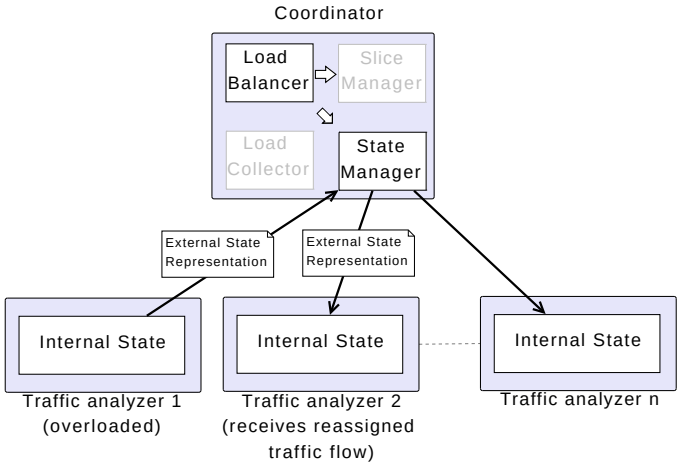


Figure 2.6: The *state manager* receives the external state representation from the overloaded traffic analyzer and relays the necessary state information to the analyzer that received the reassigned traffic flows.

flows from the first to the second traffic analyzer. To allow the second traffic analyzer to perform stateful analysis on those traffic flows, the state manager has to perform two main operations:

- external state representations gathering;
- analyzer internal state update.

In the first step, the state manager asks the traffic analyzer 1 to extract and to serialize all the state information related to all the traffic flows that have been dynamically reassigned (*state export*). Traffic analyzer 1 replies by sending to the state manager an external state representation.

In the second step, the state manager relays the external state representation to the traffic analyzer(s) to which the traffic flows have been reassigned (traffic analyzer 2, in the example depicted in Figure 2.6). The destination traffic analyzer(s) merges the received state information with its internal state (*state import*).

Traffic analyzer's state management has to guarantee a stateful traffic analysis even though internal state update occurs *after* network packets belonging to reassigned connections (that need the new state to be properly analyzed) have been received. To accomplish this task, the state migration framework has been designed to handle the merged state information as a special case of out-of-order network traffic. That design choice allows us to leverage network traffic reordering and traffic flow reassembling mechanisms already implemented in a stateful traffic analyzer to guarantee a proper state management and a stateful traffic analysis [26].

2.7 Realization of the ParNIDS architecture

2.7.1 Enabling state migration in Snort

As explained in Section 2.6, the ParNIDS architecture relies on NIDS sensors able to cooperate through the exchange of internal state information, by following the paradigm described in Section 2.4.2.

To this purpose, we extend the source code of the open source NIDS Snort [111]. We chose this particular software for several reasons:

- it is widely adopted (almost a *de facto* standard), hence the proposed framework can be used in real world environments;
- Snort allows us to measure the performance in a well known environment where inner working, performance and computational requirements have been extensively analyzed [25, 120];
- Snort employs pattern matching analysis and may require the exchange of possibly complex state information (nothing similar has been achieved by other cooperation architectures).

While the described implementation is tailored to a specific software, we remark the general applicability of the proposed framework.

Snort architecture is split across several layers: data flows from the capture interface through the decoding and pre-processing stage; then, they are analyzed by the detection engine and, when necessary, the output layer emits alerts.

Each layer consists of many plugins. Most plugins perform stateful analysis for a specific protocol, hence their state information may be shared with analogous plugins to achieve the cooperation benefits. Each detection engine utilizes a state data type whose contents depend on its analysis policy.

The core detection engine consists of an optimized pattern matcher which applies the rules described in a set of configuration files. While pattern matching is hardly stateful, most of the Snort pre-processors perform the stateful analysis of a specific protocol.

The pre-processors included in Snort (as of version 2.6.1.3) are:

- OSI layer 3
  - `spp_frag2.c`, IP fragmentation reassembly
  - `spp_frag3.c`, IP fragmentation reassembly
- OSI layer 4

  - `spp_flow.c`, TCP/UDP flow tracking
  - `spp_sfportscan.c`, portscan detection
  - `spp_stream4.c`, TCP/UDP stream reassembly
  - `spp_stream5.c`, TCP/UDP stream reassembly
- OSI layer 7
  - `spp_bo.c`, *Back Orifice* detection and communication decoding
  - `spp_dcerpc.c`, Distributed Computing Environment / Remote Procedure Call decoding
  - `spp_dns.c`, DNS pre-processor (analyzes RDATA)
  - `spp_ftptelnet.c`, FTP/Telnet normalization[3] and protocol checks
  - `spp_httpinspect.c`, protocol anomaly detection and request normalization
  - `spp_rpc_decode.c`, Remote Procedure Call request normalization (layer 7 reassembly)
  - `spp_smtp.c`, SMTP protocol checking and normalization
  - `spp_ssh.c`, performs various checks for known attacks and traffic flow anomalies
  - `spp_telnet_negotiation.c`, normalizes ftp and telnet traffic which would normally disrupt pattern matching

The design of a state migration framework can take advantage of the Snort modular architecture. We require each pre-processing plugin to be able to export its own state data and import external state data. In our reference implementation, we implemented the necessary state migration capabilities into the `stream4` pre-processor because it has a general purpose and its operations influence heavily the rule-based detection. Indeed, stream4 is used to generate an ordered data stream out of disordered, fragmented and overlapping network packets belonging to the same level 4 connection, thus enabling stateful pattern matching of TCP connections and UDP streams. Allowing the stream4 state to be transferred between Snort instances greatly enhances detection efficacy in a cooperative NIDS environment, because most of the available Snort rules require to inspect the contents of a packet stream, thus relying on TCP and UDP stream reassembly and tracking.

We remark that our framework is specifically designed for real-time traffic analysis, hence one of the main design issues is to avoid interferences among the

[3]By *normalization* Snort means the adaptation of network traffic needed in order to apply pattern matching, for example by removing control characters that would otherwise cause errors

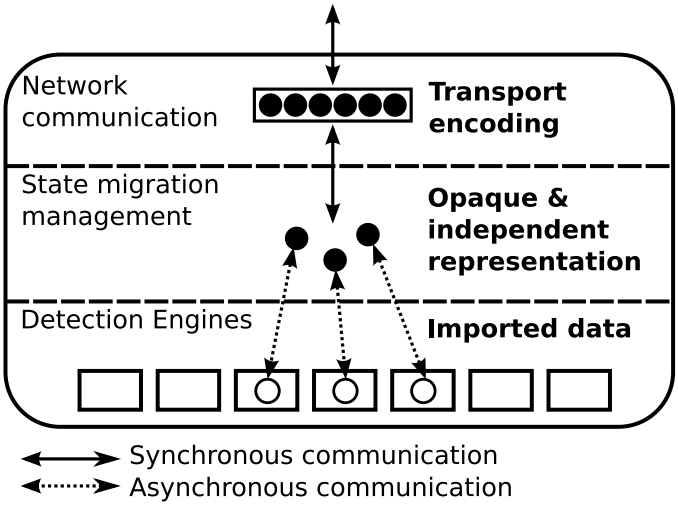


Figure 2.7: State representation in different framework layers during migration.

network traffic analysis and the state import and export operations. For example, it is not possible to block the analysis of network packets while performing a complete network communication among cooperative sensors. This issue has been effectively addressed through a multithread design, that allows state import and export operations and traffic analysis to run concurrently.

The network communication framework has been implemented through XML-RPC [126] by means of the XMLRPC-C library. Snort pre-processor state data can be easily represented in XML, because it resides in C language structures. The XML-RPC specification also includes the array type which becomes convenient when we have to transfer many similar structures.

Exchanging semi-structured data is useful for the communication of NIDS sensors which leverage different detection models, because we do not impose a strict format for the state representation (data representation can be easily extended). *Binary XML* would be preferable as an exchange format because of its space efficiency, but it is not as standardized as XML yet.

Since XML-RPC uses HTTP as its transfer protocol, the communication follows the client/server model. In the proposed model, all Snort instances have the server role and wait for incoming RPC calls. Communication starts when a client calls one of the two available methods: `state.export` or `state.import`.

The export method takes a list of traffic identifiers as its arguments. The method output contains an array of structures, each one consisting of a label that identifies a specific pre-processor and an opaque data structure that is created by the pre-processor using the traffic identifiers as its input (see Figure 2.7). Thanks to this modular approach, a NIDS can implement state migration gradually, by

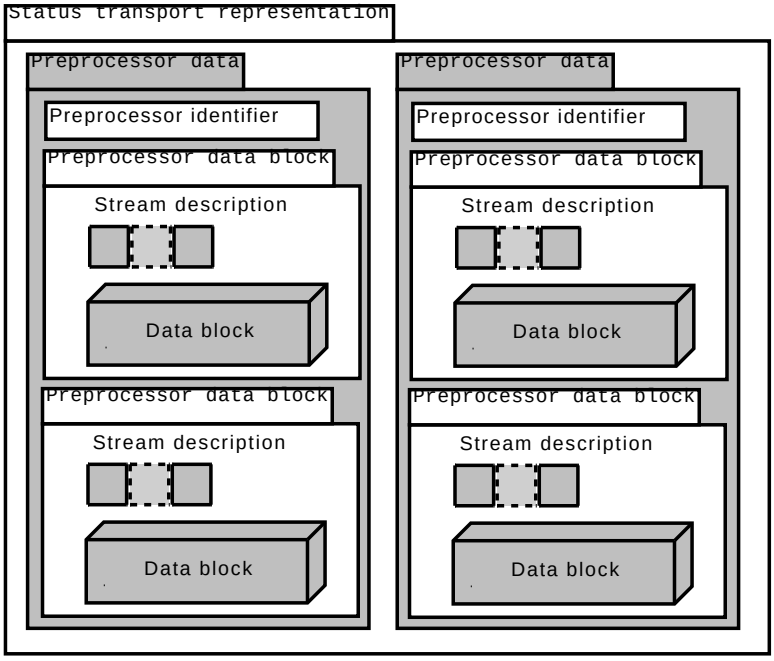


Figure 2.8: Data format for external state representation

enabling one engine at a time to export its data.

The format of the exported data is outlined in Figure 2.8. The transferred state (*Status transport representation*) is composed by several blocks (*Preprocessor data*), each one containing data about a specific pre-processor. These blocks are further divided into sections (*Preprocessors data blocks*) which contain state data extracted from a specific set of streams. A stream description is attached to every state data block in order to identify the streams it is related to. Pre-processors only need to handle the data blocks that are specifically requested from them or headed to them, while the migration framework provides XML wrapping and routing. The import method takes as its input an array of structures that are analogous to those produced by the export method. The migration-enabled Snort will feed each data structure to the matching pre-processor.

Figure 2.9 outlines the main steps that a NIDS sensor has to perform to export a state representation:

1. The state export process begins when the XML-RPC server receives a call to the export procedure. The XML-RPC server is implemented as a concurrent thread that runs cuncurrently with the normal sensor operations, without blocking or delaying traffic analysis.

2. A handler thread is generated to handle each state export request, thus al-

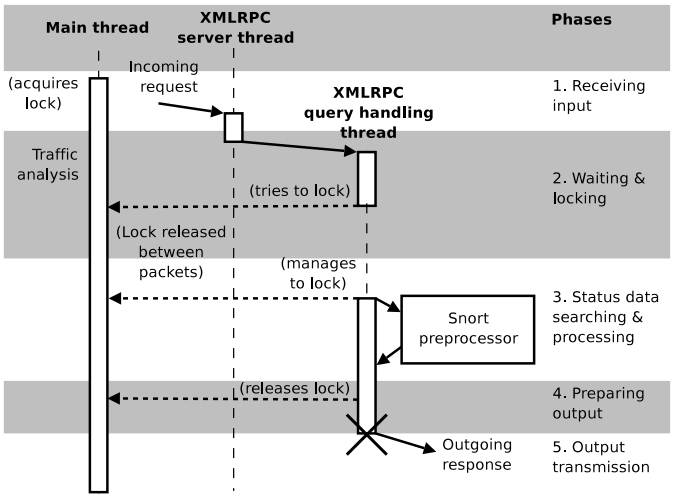


Figure 2.9: Thread interaction during different migration phases.

lowing the server to receive other export or import requests. The handler thread decodes the export request, and tries to obtain a lock on the sensor internal state information. A lock is necessary to ensure that the state information to be exported is consistent.

3. When the sensor is not performing packet analysis (that is, between the analysis of two packets), the main thread releases the lock, allowing the handler thread to analyze the internal state maintained by the Snort preprocessors and to look up the required state information. (We recall that all the network packets received by the sensor in this phase are not dropped, but temporarily stored in a buffer.)

4. When all the required state information has been found, the handler thread releases the lock over the internal state information and allows the main thread to resume the normal traffic analysis. After that, the handler thread prepares a well formed external state representation.

5. The external state representation is sent to the remote caller, thus completing the state export process.

The state import process is analogous to the export process with the main difference that during phase 3 (in which the handler thread holds the lock on the internal state representation) each pre-processor merges the imported state information with the current internal state.

2.7.2 ParNIDS implementation

All the experimental results shown in Sections 2.9 and 2.10 are obtained through an implementation of the ParNIDS architecture. Scatterer, slicers and analyzers have been implemented through open source software and execute on standard hardware components.

2.7.3 Traffic source

Traffic source activities have been emulated through the Tcpreplay software [118], that is able to replay network traffic dumps at configurable rates. Several traffic sources can be used to produce very high network traffic volumes.

2.7.4 Scatterer

The scatterer implementation is based on the open source software Ebtables [39], that makes it possible to perform low level frame modification and routing in kernel space. This guarantees better performance than any other user space implementation. The scatterer requires the realization of an ad-hoc Ebtables module, that is used to distribute Ethernet frames in a round-robin way among a configurable set of Network Interface Cards (NICs).

2.7.5 Slicers

Each slicer has been implemented through a combination of Iptables [50] and Ebtables [39]. Slicing rules are translated into sequences of standard Iptables and Ebtables rules that are able to perform frame analysis, frame routing and MAC address rewriting. Slicers are able to classify packets upon several features, such as level 3 and level 4 protocols, source and destination IP addresses and port numbers. This fine grain distribution allows the most effective load balancing among the traffic analyzers.

2.7.6 Traffic analyzers

NIDS traffic analyzers are implemented through our custom version of Snort [111] NIDS, whose implementation has been described in Section 2.7.1.

Communications between each analyzer and the coordinator are established through Remote Procedure Calls (RPC) that are implemented through the *xml-rpc* libraries [132]. The communication thread includes a simple and lightweight XMLRPC server that receives and answers to coordinator requests.

2.7.7 Coordinator

The coordinator consists of the four software modules described in Section 2.6:

- load collector;
- load balancer;
- slice manager;
- state manager.

The load collector is implemented in Perl, and includes a simple server that collects load measures gathered from the traffic analyzers. Load balancer, slice manager and state manager are implemented through C software modules running in user-space. The state manager also includes a lightweight XMLRPC client to communicate with the XMLRPC server embedded in each traffic analyzer.

2.7.8 Alert aggregator

The alert aggregator collects all the alerts raised by traffic analyzers in a database. The storage back-end has been implemented through the MySQL software [77], while the correlation engine and the graphical front-end are based on the BASE software [8].

2.8 Experimental evaluation of the state migration framework

2.8.1 Validation of the state migration framework

The first set of experiments aims to demonstrate the functional validity of the proposed state migration framework. In particular, we demonstrate the ability of the prototype to detect an attack as a result of cooperative analysis based on state migration. We remark that state migration capabilities allow a set of cooperative sensors to properly recognize complex attacks, with payloads fragmented in multiple, disordered segments. This feature is fundamental to perform stateful traffic analysis even on dynamically migrated connections.

We generate a traffic dump that includes one of the attack signatures known by Snort. For the sake of clarity, we choose a simple ASCII-based signature which represents a NOOP sled that is commonly used in shellcode-based attacks. Its signature, *aaaaaaaaaaaaaaaaaaaa* (21 “a”s) is classified with the signature-id 1394 and is included in the standard Snort rule set. (The same experiments can

be easily reproduced with any of the Snort signatures.) It is important to notice that TCP stream reassembly is a complex task, which employs identifiers such as sequence numbers to ensure that the correct order of packets is preserved. For example, by using a NOOP sled of 21 identical one-byte instructions it would seem that the reassembly order does not matter. However, TCP reassembly does not rely on stream contents, but on sequence numbers and this makes every "a" in the payload logically different. We split the original traffic trace in two parts and obtain two independent traffic dumps that are sent to two different traffic analyzers implemented with our modified version of Snort (described in Section 2.7.1).

In the first experiment, we do not activate the state migration mechanism. As expected, none of the two traffic analyzers raises any alert, because both of them see only a part of the attack, with the consequence that no signature-based alert is triggered.

We validate the state migration mechanism through two experiments: an ideal scenario with synchronous state migration, and a realistic scenario with asynchronous state migration.

In the first experiment the following activities are performed:

1. replay of the first traffic dump toward the first traffic analyzer,
2. export of the internal state of the first traffic analyzer,
3. import of the external state representation by the second traffic analyzer,
4. replay of the second traffic dump toward the second traffic analyzer.

After receiving the second traffic dump, the second traffic analyzer detects the attack, associates it with the right rule, and raises the following correct alert in the Snort syntax (the alert has been made anonymous by removing the IP addresses):

```
[**] [1:1394:5] SHELLCODE x86 NOOP [**]
[Classification: Executable code
      was detected] [Priority: 1]
02/08-12:09:12.844646 xxx.xxx.xx.xxx:xxxx
      -> xxx.xxx.xx.xxx:xx
TCP TTL:240 TOS:0x10 ID:0 IpLen:20
      DgmLen:61
***AP*** Seq: 0xE602E44D  Ack: 0x7C3A96B
      Win: 0xC  TcpLen: 20
```

This experiment demonstrates that the second traffic analyzer is able to effectively use state information generated by another traffic analyzers to perform stateful traffic analysis. However, we observe that in this case traffic dump replays,

state import and state export actions are synchronized to guarantee that they are carried out in their correct logical order. In real world scenarios, it is impossible to wait for the completion of the state migration operations before sending reassigned traffic to the new destination traffic analyzer. Waiting is not a viable solution because it would require sophisticated synchronization mechanisms between traffic slicers and traffic analyzers. For example, it would be necessary to temporarily store network traffic in sufficiently large buffers during state migration, but this would be (at the very least) unpractical in most high speed networks.

In the second experiment, we demonstrate the state migration effectiveness in a realistic scenario, in which state migration is performed when traffic analyzers have already received the whole attack. The steps of the experiment are as follows:

1. replay of the first traffic dump toward the first traffic analyzer,
2. replay of the second traffic dump toward the second traffic analyzer,
3. export of the internal state of the first traffic analyzer,
4. import of the external state representation by the second traffic analyzer.

Although the second traffic analyzer now receives the external state representation only after having analyzed the last part of the attack, it generates the same Snort alert that was raised in the previous experiment. This result is important because it demonstrates that the proposed traffic and state migration mechanism is able to rebuild an ordered information stream, in the same way as it builds an ordered traffic flow out of disordered network packets.

We can conclude that when a Snort sensor which inspects just a portion of the attack receives a representation of the other portions through state migration, the attack is identified correctly. The order between the packet analysis from live capture and state import does not matter: an alert is correctly issued whichever of the two events happens first. This result guarantees the robustness and the utmost flexibility of the state migration framework, which fulfills all requirements for a stateful intrusion detection even in systems consisting of distributed cooperative components. As reported in Section 2.2, we remark that to the best of our knowledge no related framework allows cooperative NIDS to perform a fully stateful pattern matching analysis on traffic received by different sensors.

2.8.2 Performance evaluation of the state migration framework

State migration is a complex process that requires several activities. However, a framework for NIDS cooperation should be suitable to real-time traffic analysis, hence it is important to evaluate the impact of the implemented state migration framework on the overall NIDS performance.

The first experiment aims to measure the third phase shown in Figure 2.9, during which the sensor cannot perform traffic analysis. The test machines are based on an Intel Core Duo 64bit CPU with a 2.40GHz clock frequency and 2GB of memory. The operating system uses the Linux kernel version 2.6.20 and libpcap version 0.8. During the test, we replay the IDEVAL [63] traffic dump to the NIDS sensor at different rates. We send to the XML-RPC server a request for the migration of the stream4 pre-processor internal state related to a single host and measure the duration of phase 3 for different values of the incoming traffic.

Experimental results are summarized in Figure 2.10. The x-axis shows the throughput of the input traffic analyzed by the sensor in Mb/s, while the y-axis represents the duration of this phase in seconds. The points labeled with "Phase 3 duration" represent the duration of phase 3 measured for different traffic rates. From Figure 2.10 it is clear that the duration of phase 3 grows exponentially with respect to the analyzed throughput until 137 Mb/s (the exponential regression fits the measured data). For higher traffic volumes, the duration of this phase becomes independent of the analyzed traffic throughput. This result depends on the inner operations of the stream4 pre-processor, that uses a hash table to store reassembled network packets for each active connection. When it is necessary to find the state information for all the connections of one host, it is necessary to traverse all the stored information sequentially, hence the time grows as a function of the amount of stored states.

We measure also the number of trackers that are stored by the stream4 pre-processor (each tracker represents a network connection whose packets are stored and reassembled). The results are summarized in Figure 2.11, where the x-axis represents the analyzed traffic, and the y-axis the number of trackers. It is noticeable that the number of trackers grows exponentially with respect to the analyzed throughput. Saturation is reached at 137 Mb/s, because the number of trackers reaches 8192, which corresponds to the limit of the default Snort configuration.

In Figure 2.12 we show the duration of phase 3 as a function of the number of trackers. The x-axis denotes the number of trackers and the y-axis the time taken to search for the state information. We can see that the time for the export operation scales linearly with the number of trackers until saturation occurs at 8192 stored trackers, as clearly shown by the linear regression.

Using the standard Snort configuration, that limits the number of stored trackers to 8192, the state export operation requires to lock the internal state (thus blocking the network traffic analysis) for at most 0.014 seconds, independently of the analyzed traffic throughput. On the other hand, the analyzed traffic throughput can be used to determine the dimension of the buffer used to temporarily store the network packets received while the sensor internal state is locked.

With an analyzed traffic throughput of 200 Mb/s (the highest generated in our experimental setup) and an average packet size of 213 bytes, during phase 3 the

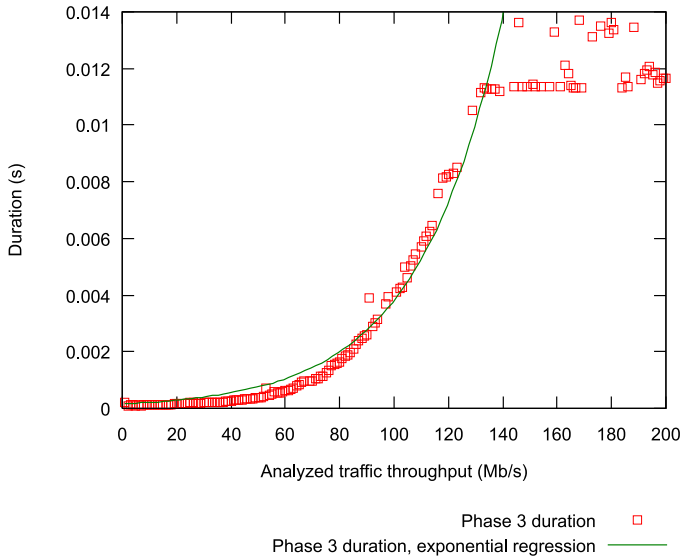


Figure 2.10: Phase 3 duration as a function of the analyzed throughput.

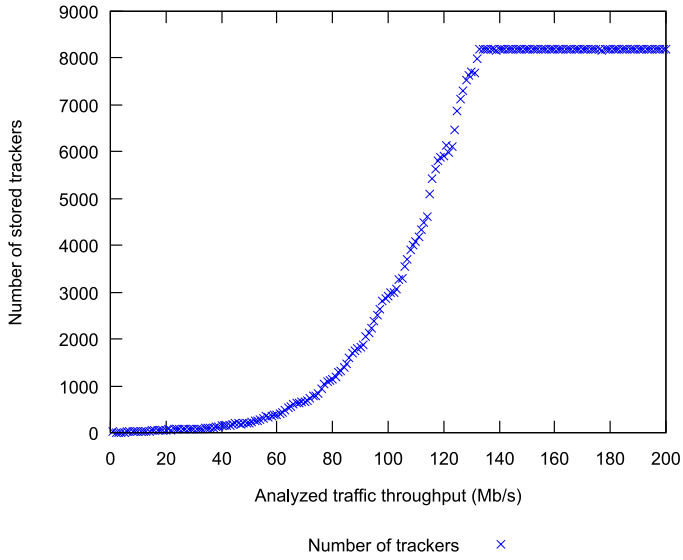


Figure 2.11: Number of stream4 trackers as a function of the analyzed throughput.

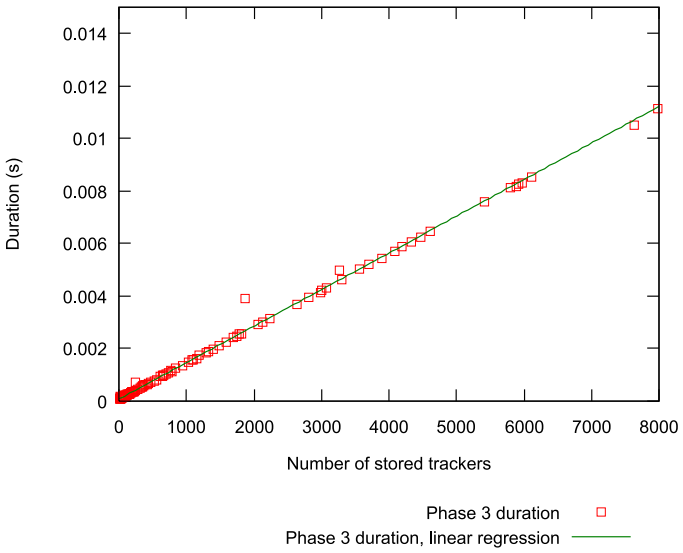


Figure 2.12: Phase 3 duration as a function of the stream trackers number.

sensor will buffer an average of 1644 packets, corresponding to a required buffer size of 350 KB. The packet capture buffer size has to be dimensioned appropriately to avoid dropping packets. Our experiments show that the needed buffer size is very low, compared to the hardware capabilities of a typical real-time NIDS sensor.

The following experiment aims to measure the time required by a sensor to perform all the operations of phases 2, 3 and 4 (all the state migration operations, excluding the network communication). While the duration of the phase 3 is independent of the amount of state information that is actually imported or exported, the number of transferred state information represents a key parameter for the duration of other state migration phases, in which all the involved state information has to be serialized (for state export) or de-serialized and merged with the internal state information (for state import).

Figure 2.13 shows the time required by the implemented sensor to perform phases 2, 3 and 4 during the export and import of a variable number of streams. The x-axis shows the number of transferred (imported or exported) sessions, while the y-axis represents the duration time in seconds. Each migrated session is related to one TCP stream and contains all the information generated by the stream4 preprocessor for that connection (mainly, a normalized form of the already received data stream). As expected, the time taken for the operation grows linearly with the number of transferred sessions.

In the last set of experiments, we measure the time required for a full state migration operation, including the network communication. This experiment is

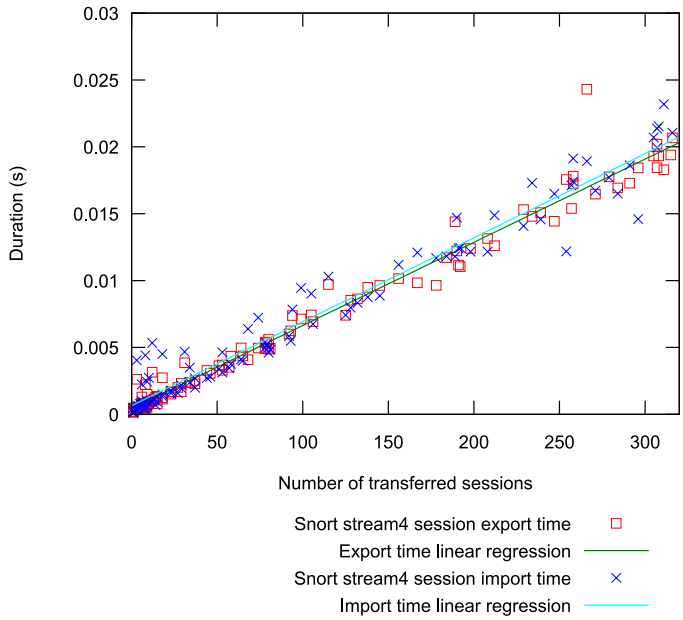


Figure 2.13: State import and export durations as a function of migrated sessions number.

carried out by exporting and importing a single stream at variable analyzed traffic throughput. Results are summarized in Figure 2.14, where the x-axis represents the analyzed traffic in Mb/s, and the y-axis the duration in seconds. The points labeled with "export time" show the durations of the phases 2, 3 and 4, while the points labeled with "overall export time" include the network communication overhead (phases from 1 to 5).

The results show that if the traffic rate is lower than 100 Mb/s, the network communication adds an almost constant overhead of 0.3 seconds, which is (at least) one order of magnitude higher than the time required by the phases 2, 3 and 4. The measurements which include network communication are noticeably noisier at higher throughput, because the internal state serialization has to wait for the main Snort thread to unlock the mutex guarding the stream4 internal data. The time needed for this operation may vary, because the two threads are competing. This also means that the lock is not acquired by the state exporting thread during the network communication. Hence, we guarantee that the main Snort thread is stopped for as little time as possible to avoid an overfill of the packet capture buffer. Similar results have been obtained for the state import, where the network communication adds a constant overhead of about 0.3 seconds.

We can conclude that the most critical phase, in which the analysis is blocked, is also the shortest, and even a small buffer is adequate to avoid packet drop. In the overall state migration process, most time is spent in network communications be-

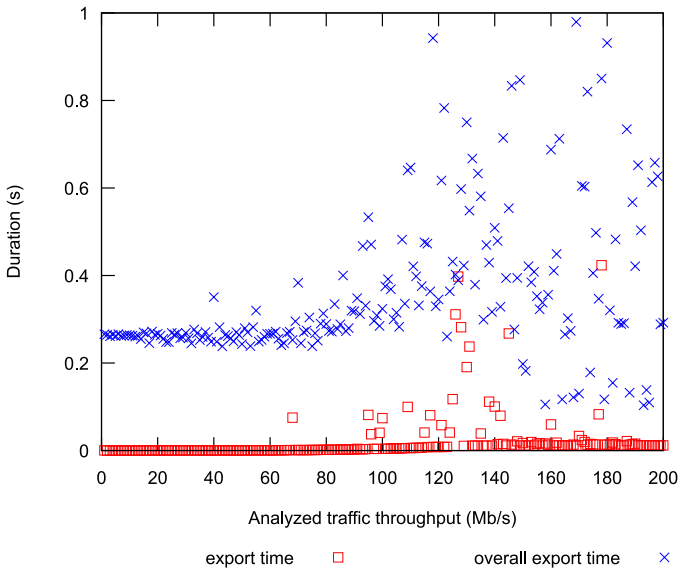


Figure 2.14: Impact of network communication on the state exporting time.

tween the coordinator and the sensor. However, this is not a critical issue because communications are performed in parallel with respect to traffic analysis. Moreover, we remark that cooperative sensors do not need any form of synchronization, and that delays of some seconds are quite acceptable.

2.9 Validation of the ParNIDS architecture

To demonstrate that the parallel architecture does not affect traffic analysis reliability, we compare a traditional centralized NIDS against a ParNIDS prototype consisting of three slicers and three traffic analyzers. The components are installed on machines equipped with one Intel Xeon 2.4 GHz CPU, 1 GByte RAM, a 32 bit and 33 MHz PCI bus and a Gbit Ethernet NICs. Slicers and traffic analyzers are connected through an HP Procurve 2824 switch. As this experiment does not aim to verify the highest manageable traffic, we use a traffic generator running on one machine. It replays the IDEVAL [49,63,66] traffic dumps towards the scatterer input interface. This traffic dump has been used by the Defense Advanced Research Projects Agency (DARPA) for the IDEVAL project. We consider and test many different configurations. Here we report the results referring to two significant examples.

The first experiment refers to a simple set of port-based slicing rules that are applied to the parallel architecture. The first rule routes to the first traffic analyzer every TCP packet coming from or, directed to, port 80. The second rule

	Analyzer 1	Analyzer 2	Analyzer 3	Total (parallel)	Total (centralized)
Packets	526440	529439	697498	1753377	1753377
Alerts	495	22	2686	3203	3203

Table 2.2: Packet analyzed and alerts generated by each traffic analyzer (port-based traffic distribution).

	Analyzer 1	Analyzer 2	Analyzer 3	Total (parallel)	Total (centralized)
Packets	860828	600953	426718	1888498	1753377
Alerts	2726	298	179	3203	3203

Table 2.3: Packet analyzed and alerts generated by each traffic analyzer (IP-based traffic distribution).

routes to the second traffic analyzer every TCP packet coming from, or directed to, port 23. The third rule routes to the third traffic analyzer any packet that has not been routed to the traffic analyzers 1 or 2. Slicing rules are simple, but they allow a fair packet sharing among the three traffic analyzers. Table 2.2 presents the experimental results in terms of analyzed packets and generated alerts for each traffic analyzer. The first analyzer of the ParNIDS architecture receives and examines 526440 frames and generates 495 alerts, the second analyzer analyzes 529439 frames and generates 22 alerts, the third analyzer analyzes 697498 frames and generates 2686 alerts. The centralized NIDS architecture generates a total of 3203 alerts that correspond perfectly to the sum of the alerts obtained by the three parallel analyzers.

In the second experiment, we apply a set of slicing rules based on IP addresses. The internal network considered by the DARPA IDEVAL project includes several hosts that are divided in six class C subnetworks. We configure the slicing rules in order to route all the packets with source or destination IP address belonging to the subnetwork 172.16.112.0/24 to the traffic analyzer 1. Traffic analyzer 2 receives the packets with source or destination address belonging to the subnetwork 172.16.114.0/24. The other packets with source or destination address belonging to the subnetworks 172.16.113.0/24, 172.16.115.0/24, 172.16.116.0/24 and 172.16.117.0/24 are routed to the traffic analyzer 3. The results are summarized in Table 2.3. The first analyzer examines 860828 frames and generates 2726 alerts, the second analyzer examines 600953 frames and generates 298 alerts, and the third analyzer examines 426718 frames and generates 179 alerts. The sum of the alerts generated by the ParNIDS architecture corresponds exactly to the number of alerts raised by the centralized NIDS architecture. It is interesting to

observe that this result is obtained although the total amount of packets examined by the parallel traffic analyzers is slightly higher (about 7%) than the number of generated packets. This increment in the amount of analyzed traffic is caused by the application of the NIDS to the particular configuration of the experimental network. As slicing rules are based on IP addresses, internal packets from one subnetwork to another may be examined by two parallel analyzers. In a more realistic scenario of a NIDS placed on the border (as in Figure 2.3), there is no packet replication due to a parallel architecture.

We can conclude that it is possible to distribute the traffic among parallel NIDS analyzers and to preserve stateful and reliable results. This holds true for any set of slicing rules that are *flow preserving*, that is, when all the packets related to the same traffic flow (e.g., a TCP connection or a UDP communication) have to be routed to the same traffic analyzer.

2.10 Performance evaluation of the ParNIDS architecture

Besides functional correctness, high performance, scalability and load balancing are the raison d'être of the ParNIDS architecture, hence several experiments concerned these attributes.

2.10.1 Performance of one sensor

Before presenting experimental results demonstrating the architecture scalability and the performance of the proposed load balancing scheme, it is important to analyze the performance of one sensor. The highest throughput that one sensor is able to analyze will be used to determine its saturation throughput.

In the following experiment we installed our enriched version of Snort on a machine equipped with one Intel Xeon 2.4 GHz CPU, 1 GByte RAM, a 32 bit and 33 MHz PCI bus and four Gbit Ethernet NICs. One machine with the same hardware configuration is used to relay the IDEVAL traffic dump with a configurable throughput. The highest sustainable load of each analyzer is shown in Figure 2.15. The x-axis represent the throughput of the analyzed traffic, while the y-axis is used to measure the packet rate. The three lines represent the packet rate received by the traffic analyzer, the rate at which the analyzer is dropping packets, and the packet received error rate, respectively. The most important result is that a traffic analyzer with the tested configuration is able to perform a stateful and fully reliable analysis for an incoming throughput up to 40 Mbps corresponding to a packet rate of about 25.000 packets per second. Higher traffic volumes cause

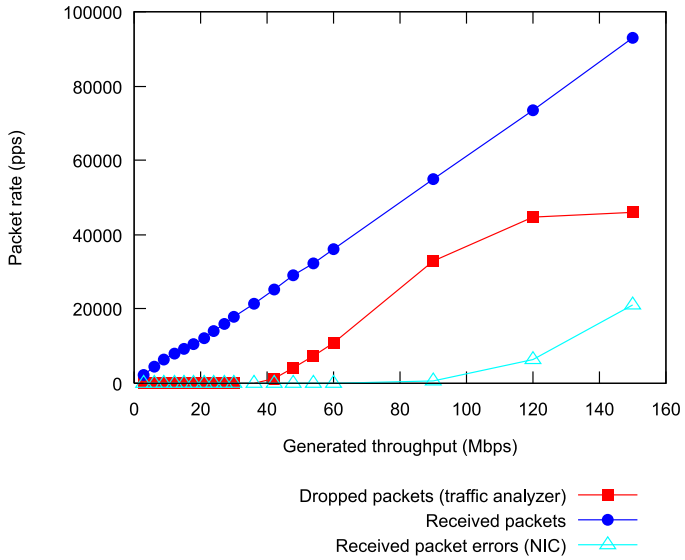


Figure 2.15: Capacity of the traffic analyzer for increasing input traffic.

an excessive increment of the CPU load. The analyzer is not able to examine all received traffic and it starts to lose packets. For an input traffic of 120 Mbps, the CPU load reaches 1 and gets completely saturated. At this point, the analyzer is unable to properly handle all the incoming packets, hence the dropped packet rate saturates, and the received error rate increases.

2.10.2 Scalability

To verify the scalability of the ParNIDS architecture shown in Figure 2.3, we carry out several stress testing experiments for different numbers of traffic analyzers. All the architecture components are installed on machines equipped with the same hardware configuration described previously, while the switch is an HP procurve 2428. In each test we measure the highest network traffic that the system is able to analyze while maintaining packet loss below 1%. We use several traffic generator to reproduce IDEVAL traffic through the scatterer input interface. The scalability results are shown in Figure 2.16, where the x-axis denotes the number of parallel analyzers, and the y-axis the highest system throughput achievable in terms of Mbps. From this figure, it is immediate to observe that ParNIDS capacity of analysis grows almost linearly for increasing numbers of traffic analyzers.

While there is no theoretical limit to the number of traffic analyzers that can be integrated into the parallel architecture from the functional and performance point of view, the experimental results confirm that the system capacity is limited by the scatterer performance. (This was already expected by the architecture design

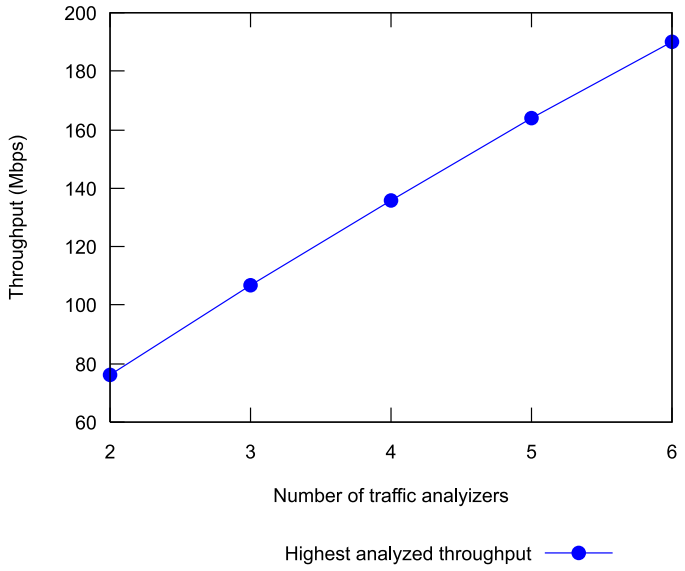


Figure 2.16: Scalability of the parallel NIDS architecture.

shown in Figure 2.3.) For this reason, we carry out several experiments that aim to evaluate the maximum amount of network traffic that the scatterer can manage in terms of bandwidth and packet rate. We install the scattering software on a machine equipped with one Intel Xeon 2.4 GHz CPU, 1 GByte RAM, a 32 bit and 33 MHz PCI bus and four Ethernet NICs at 1 Gbit. One of the Gbit Ethernet NICs is used as the input interface, while the others are connected to three slicers. To generate the required traffic, we use multiple machines, each running an instance of Tcpreplay. The traffic generators and the scatterer are directly connected to a Gbit switch, that forwards all the generated traffic to the scatterer input interface. Traffic generators replay the IDEVAL traffic dump at different rates, and for each run we measure the traffic coming out from the scatterer. Experimental results are presented in Figures 2.17 and 2.18. In both figures, the x-axis represents the throughput of the generated traffic expressed in Mbps. In Figure 2.17, the y-axis shows the traffic received and transmitted by the scatterer measured in Mbps. In Figure 2.18, the y-axis shows the traffic received and transmitted by the scatterer in terms of packet rate, as well as the packet error rate.

Figure 2.17 shows that the scatterer is able to receive all the generated traffic until 800 Mbps, corresponding to a packet rate of about 500.000 packets per second. Above these values, the input NIC saturates, and it is unable to handle all the generated traffic. However, from the functional point of view, the scatterer reaches its limit even before 800 Mbps. We can see that the traffic transmitted by the scatterer remains equal to the received traffic (that is, all received traffic is properly handled and retransmitted) until 200 Mbps that corresponds to a packet rate of

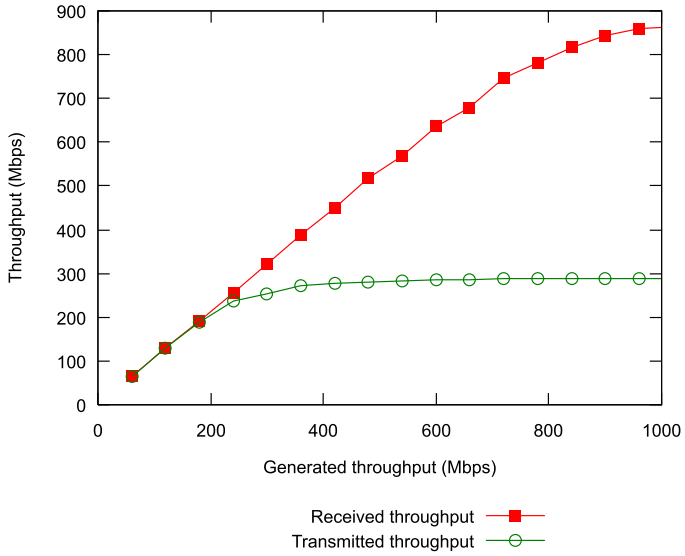


Figure 2.17: Scatterer throughput in terms of Mbps (IDEVAL traffic).

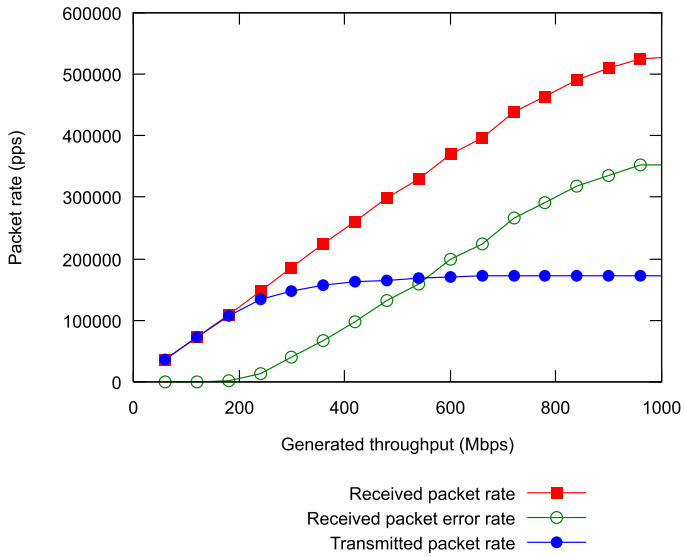


Figure 2.18: Scatterer throughput in terms of packet rate (IDEVAL traffic).

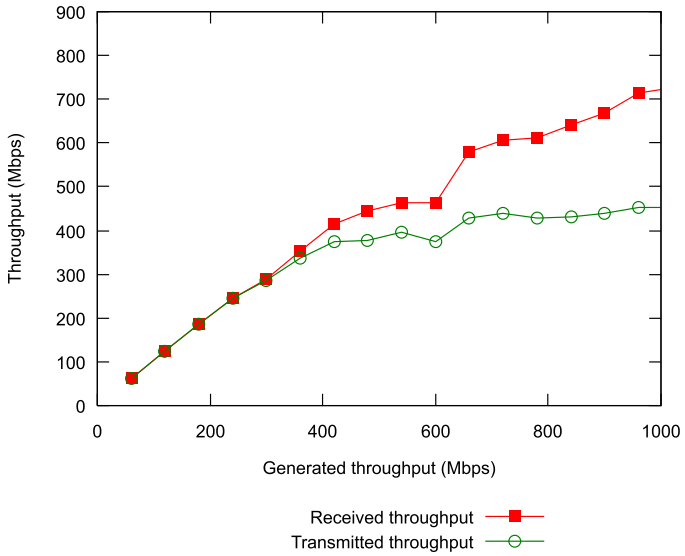


Figure 2.19: Scatterer throughput in terms of Mbps (jumbo frame traffic)

about 120.000 packets per second. Further analyses allows us to conclude that the saturation of the scatterer is due to the excessive number of interrupt requests (IRQs) generated by NICs that are receiving or sending traffic at high packet rates. IRQ saturation is demonstrated also by the CPU utilization of the *ksoftirqd* kernel daemon, that reaches values above 0.5 when the generated traffic is higher than 200 Mbps. We observe that the scatterer runs on a single CPU machine, that has to manage all IRQs generated by three Gbit Ethernet network interfaces. A multi-core machine for the scatterer software could easily increase the analysis capacity of the ParNIDS architecture.

To demonstrate the scatterer ability to manage higher throughput, we enabled jumbo frames in all the Gbit Ethernet NICs, and repeated the same experiments by replaying a jumbo frame traffic dump. Experimental results for the three slicers configuration are shown in Figures 2.19 and 2.20.

Using jumbo frames, the scatterer is able to effectively manage much higher network bandwidths. Indeed, the transmitted traffic remains equal to the received traffic for bandwidth values below 350 Mbps. This limit is caused by the saturation of the 32 bit and 33 MHz PCI bus of the scatterer. Increment in the maximum bandwidth manageable by the scatterer can be achieved even through more effective PCI buses, such as PCI-X and PCI-Express.

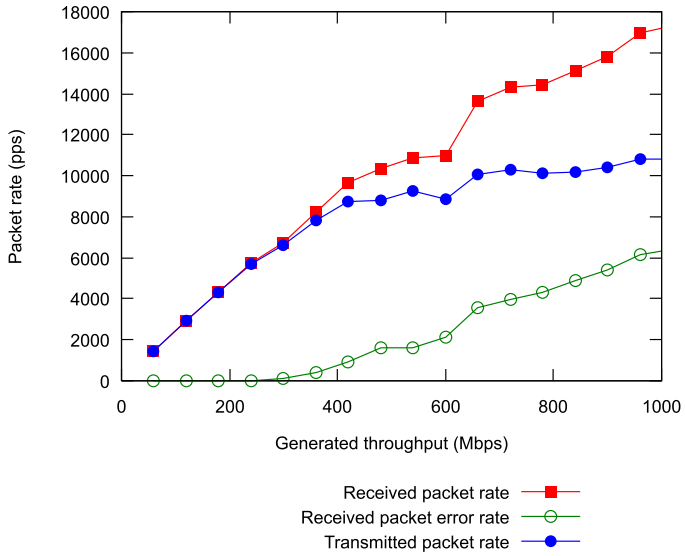


Figure 2.20: Scatterer throughput in terms of packet rate (jumbo frame traffic)

2.10.3 Load balancing

The load balancing framework of the ParNIDS architecture dynamically adjusts the traffic partitioning that is initially performed through static slicing rules. The goal is to avoid possible overloads and consequent packet losses at the traffic analyzers. To evaluate the performance of the proposed system we configured a complete ParNIDS architecture with three slicers and three traffic analyzers.

The IDEVAL traffic is generated by five machines, that replay the traffic dump at a constant rate of 18 Mbps, thus achieving an aggregate bandwidth of 90 Mbps. All the generated traffic reaches the scatterer input NIC. Slicers enforce typical slicing rules that are based on the packet IP addresses. We define a different traffic slice for each machine connected in the internal network. To evaluate the performance of the system in the worst-case scenario, we initially assign all the traffic slices to the same traffic analyzer, and leave the other two analyzers completely idle.

As the main load metric, we consider the traffic received by each analyzer Ethernet NIC in terms of Mbps. Indeed, the experimental results shown in Figure 2.15 show a strong correlation between the incoming throughput and the packet loss. The load collector samples these measures every second. The high threshold value for activating load migration is 40 Mbps, that is the highest throughput sustainable by the traffic analyzers without packet loss (see Figure 2.15). The low threshold value for deactivating load migration is set to 28 Mbps (70% of the high threshold).

As preliminary experiments demonstrated the impact of different load mea-

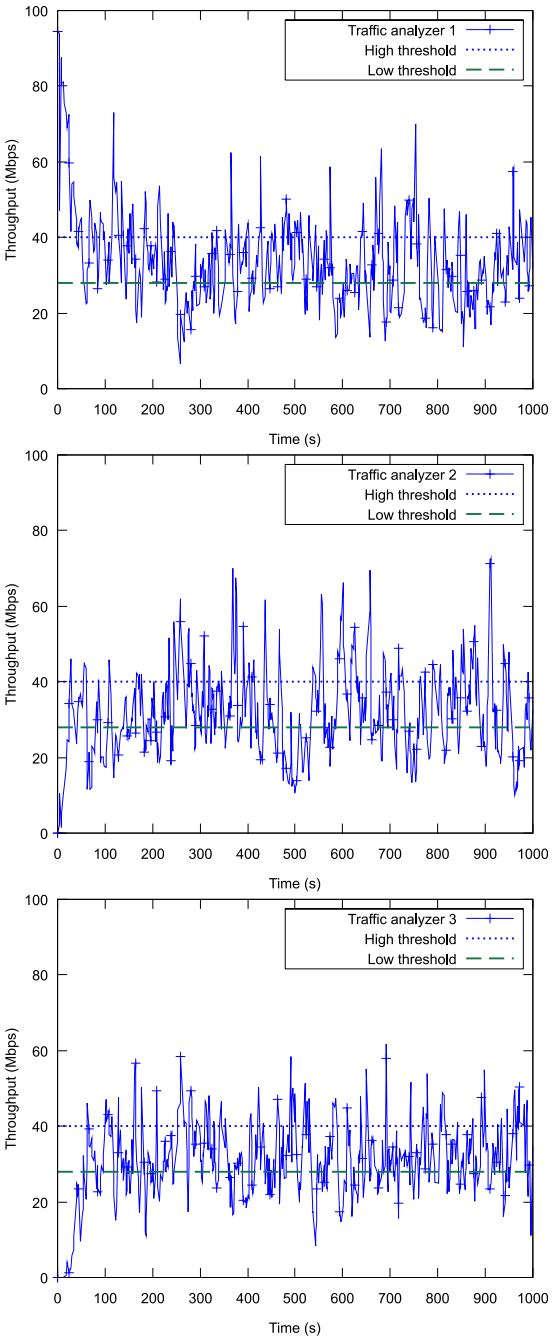
sures on the load balancing efficacy, we repeat the same experiment several times for three load metrics gathered by the load collector: samples, aggregation through the SMA model, and aggregation through the EMA model. The results are shown in Figures 2.21-2.23, where the x-axis denotes the time in seconds of the experiment, and the y-axis denotes the throughput in Mbps. The lines represent the traffic load values in each of the three traffic analyzers (1, 2, 3) as a function of the experiment time.

In Figure 2.21, the load collector uses samples with no aggregation. We can see that at the beginning of the experiment, all the traffic slices are assigned to the traffic analyzer 1 (Figure 2.21(a)), hence its load is well above the high threshold, while the other traffic analyzers are idle. The load balancing algorithm detects this critical situation and starts to redistribute traffic slices among the other two traffic analyzers. As a consequence, the load of the traffic analyzer 1 quickly decreases, while the load on the other analyzers increases. The load balancing stops when the load of traffic analyzer 1 falls below the low threshold. This is an important result because it demonstrates the correct operations of the load balancing and the state migration mechanisms. On the other hand, the three plots in Figure 2.21 clearly show that the system is highly unstable. This unsatisfactory result is caused by the high variability and non-uniform distribution of network traffic, that is typically characterized by bursty behaviors and heavy-tailed distributions [32, 40]. The consequence is that load samples exhibit sudden spikes that overshoot the high threshold for a very short amount of time. Such network activities may not lead to packet loss on the traffic analyzer, because its buffering system is able to handle very short lived traffic peaks. However, the load balancing algorithm detects an overload, and reacts with unnecessary traffic redistributions. These overreactions may bring the system to instability with consequent computational load peaks on the traffic analyzers and packet losses.

Poor results of load balancing based on samples is confirmed by the results reported in Figure 2.24(a) that shows the most and least loaded traffic analyzers during the experiment. There is a significant difference between the most and the least loaded traffic analyzers, hence we can conclude that the network traffic and the computational load is not balanced at all. Moreover, in the 50% of the samples, the highest load is above the high threshold. This means that for almost half of the experiment, at least one of the traffic analyzers is overloaded and requires an activation of the load balancing mechanism with consequent overheads.

These results motivate the need to manage load balancing activation/deactivation through different load measures. As already anticipated, we refer to sample aggregations based on *Simple Moving Average* (SMA) and *Exponential Moving Average* (EMA) models.

Figure 2.22 reports the results of the same experiment carried out using as aggregation function the SMA of the last 20 load samples of each traffic analyzer.



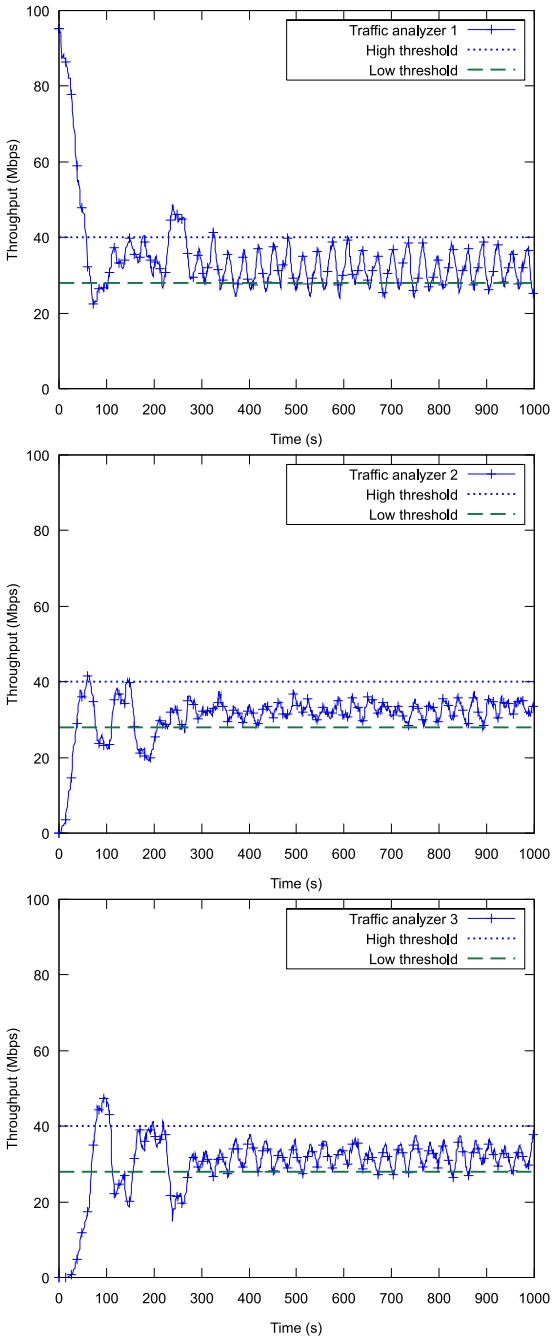
(a) Traffic analyzer 1

(b) Traffic analyzer 2

(c) Traffic analyzer 3

Figure 2.21: Performance of the load balancer mechanism based on load samples.

We can see the same initial effect of network traffic that is entirely routed to traffic analyzer 1, the consequent activation of the load balancing mechanism that brings some traffic to the other two analyzers.



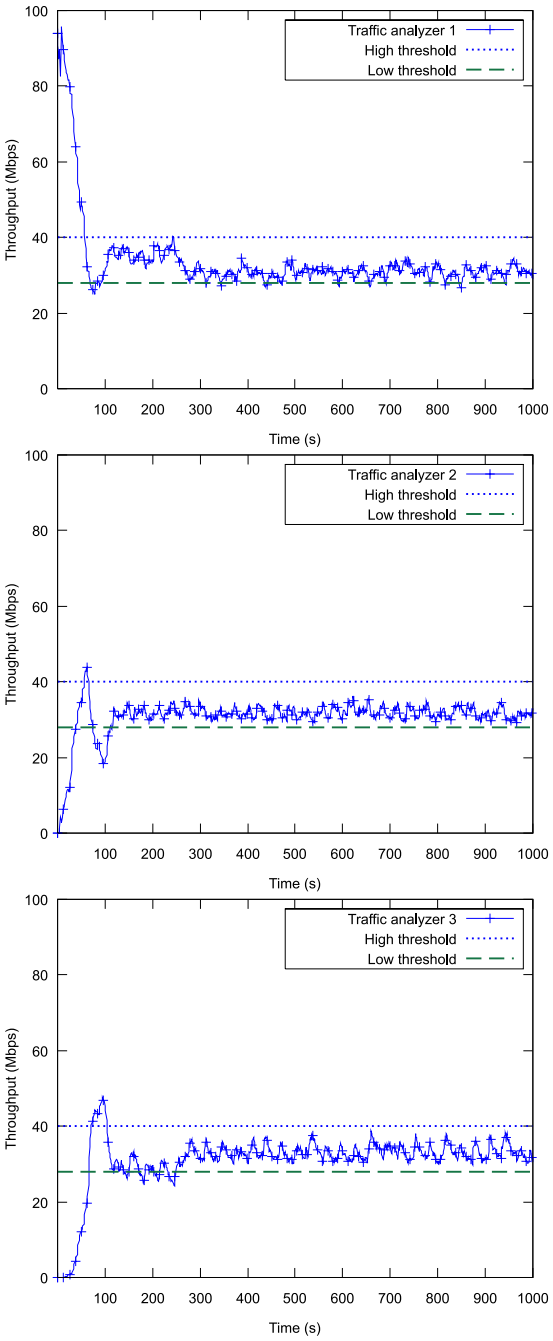
(a) Traffic analyzer 1

(b) Traffic analyzer 2

(c) Traffic analyzer 3

Figure 2.22: Performance of the load balancer mechanism based on SMA load aggregation.

Load aggregation causes a remarkable reduction in the number of load spikes with respect to load balancing based on samples (compare Figure 2.21 and Fig-



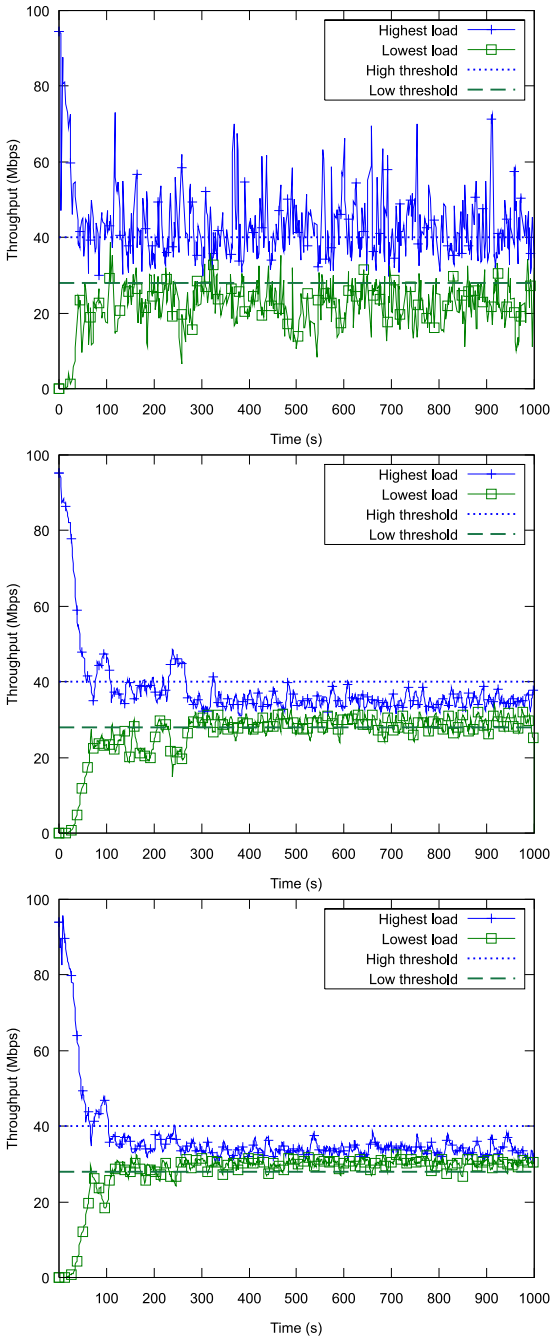
(a) Traffic analyzer 1

(b) Traffic analyzer 2

(c) Traffic analyzer 3

Figure 2.23: Performance of the load balancer mechanism based on EMA load aggregation.

ure 2.22), that is very useful to avoid unnecessary load balancing activations. The better stability of the system load is confirmed by the results in Figure 2.24(b)



(a) Load samples

(b) SMA load aggregation

(c) EMA load aggregation

Figure 2.24: Comparison of the highest and the lowest load on traffic analyzers based on different load metrics

showing the highest and the lowest load on the traffic analyzers during the experiment. We can see that the load balancing algorithm is able to fairly distribute

	No load aggregation	SMA load aggregation	EMA load aggregation
Overloading time (%)	50.86	13.81	8.87
Load balancer activations	188	38	22

Table 2.4: Load balancing performance indexes for different load aggregation functions.

the load among the three traffic analyzers in about 300 seconds. Initial oscillations are not rapidly eliminated because of the latency introduced by load aggregation in the feedback loop, that makes load balancing less responsive to load changes with respect to not aggregated load samples. While the SMA leads to an impressive improvement with respect to the first experiment, these delays open the possibility to a third alternative represented by the EMA. Unlike SMA, the EMA model gives a higher weight to the more recent load samples, hence it can smooth traffic spikes without introducing excessive delays in the feedback loop. Figure 2.23 reports the results when the load balancer is based on an EMA aggregation function of the last 30 load samples of each traffic analyzer. The impact on the load trend of the three analyzers is similar to the SMA case during the first 100 seconds (compare Figure 2.22 and Figure 2.23). Unlike the previous case, now the load remains balanced during the entire experiment with just one small spike at the instant $t = 242$. The load balancer reacts immediately to this spike through a small load migration that brings the overall architecture to a long lasting balanced state. Figure 2.24(c) gives an immediate confirmation that the EMA load aggregation has the smallest difference between the highest and the lowest loaded traffic analyzers. This model eliminates traffic spike effects as the SMA model does, but its responsiveness is more similar to the model that uses samples with no aggregation.

For each of the previously described experiments (load samples, SMA load aggregation and EMA load aggregation), we compute two performance indexes that are representative of the load balancing efficacy. The results are reported in Table 2.4. *Overloading time* represents the percentage of time during the experiment in which at least one traffic analyzer is overloaded. *Load balancer activations* measures the number of load balancing and consequent state migration activities that are triggered during the experiment. This table confirms that the EMA model clearly outperforms both the load samples and SMA load aggregation schemes, because it achieves the most even load distribution together with the lowest number of load balancer (and state migration) activations. We recall that the reported results refer to the worst case scenario, in which the experiment begins with the highest possible unbalance (all the traffic is routed to only one traffic analyzer). We also carried out several other experiments (for different activation thresholds, sample sets, number of traffic analyzers, traffic intensity), all of them confirming

the main conclusions about load balancing efficacy.

Chapter 3

Architectures and protocols for mobile connection security

3.1 Introduction

In this chapter we address some security issues related to node mobility by focusing on two main problems:

- the inability to perform stateful analysis of traffic generated by mobile nodes;
- incompatibility between triangular routing (default routing scheme of Mobile IP) and *egress filtering* best practices.

We are witnessing a steady increase in both the number and the computational power of Internet-ready mobile devices, such as smartphones, PDAs, Internet tablets, subnotebooks and laptops. However, the great potential of mobile devices is limited by the lack of node mobility support in the IP protocol. Whenever a mobile node roams between two different networks, its IP address has to change for it to be valid in the new destination network. Hence, the mobile node is not anymore reachable through its previous IP address, and all the previously opened connections are interrupted. As an example, the lack of mobility support in the IP protocol prevents a mobile node to roam through different networks while downloading long files or streaming multimedia contents.

Mobility in IP networks can be achieved through the *Mobile IP* (MIP) protocol [87, 89, 90], that allows transparent routing of IP datagrams to mobile nodes across the Internet. This goal is reached through the introduction of the *Home Agent* and the *Foreign Agent*, that receive and route datagrams on behalf of the mobile node by leveraging tunneling techniques. Thanks to this additional routing layer, a mobile node is always reachable with its home IP address, even though connected through a foreign network with a completely different address space.

While Mobile IP solves the mobility problem in IPv4 networks, it introduces side effects that may be detrimental for the security of MIP-enabled networks. In particular, we identify and address two mobility-related security issues.

The first contribution of this chapter is represented by the definition of a new attack strategy, called *mobility-based evasion* that attackers in Mobile IP contexts can leverage to perform "stealth" attack, undetectable even by modern stateful NIDS. To the best of our knowledge, this issue has never been studied before. Mobility-based evasion combines traditional evasion techniques with well-timed roaming. As an example, suppose that a mobile node engaged in a TCP connection with a correspondent node roams to a different network. In this context, the network packets sent by the mobile node to the correspondent node follow different paths, depending on the network to which the mobile node is currently connected. In particular, none of the NIDS installed in the two MIP-enabled networks are able to analyze the whole connection and to perform a *stateful* analysis. This simple technique (and several variants, dealt with in Section 3.3) allows a mobile attacker to perform stealth attacks, not detectable by state-of-the-art NIDS sensors.

The second contribution presented in this chapter is a novel detection technique that, for the first time, enables distributed NIDS to perform stateful analysis of mobile connections, thus solving the problem represented by mobility-based evasion. In the proposed approach, distributed NIDS architectures cooperate by exchanging their internal state information. The enabling technology is the state migration framework described in Section 2.4. The basic idea is to perform migration of internal state information while executing the normal Mobile IP roaming operations. This allows the NIDS state information to "follow" a mobile node while it roams to a new network.

The last contribution presented in this chapter is the design of a new dynamic and automatic traffic filtering approach, called *secure triangular routing* able to overcome the incompatibility between Mobile IP triangular routing and egress filtering best practices. A serious problem of the triangular routing scheme lays in its incompatibility with the *egress filtering* policy [127], recommended as best practice [54] because it prevents source IP address spoofing. If the foreign network implements egress filtering, all the IP datagrams generated by the mobile node are blocked at the foreign network boundary, because their source IP address do not belong to the network address space. The commonly adopted solution is to employ an alternative routing scheme: *reverse tunneling*. While being compliant with egress filtering policies, reverse tunneling employs a longer routing path, invariably resulting in higher round trip time and lower throughput. It is then possible to identify a clear trade-off between network security and mobile connections performance.

We propose *secure triangular routing*, a novel routing scheme that guarantees the same performance of triangular routing while fully complying with egress filtering best practices. The basic idea is to dynamically update traffic filtering rules to reflect the presence of a mobile node inside the protected network. When a mobile node roams into a foreign network, the foreign agent (trusted element in any MIP enabled network) interacts with the traffic filtering element (typically a firewall, or a border router) adding the rules needed for IP datagrams generated by the mobile node to pass through the network border without being blocked. The same rules are automatically removed as soon as the mobile node leaves the foreign network.

Viability of both the proposed solutions is demonstrated through prototypes, developed by modifying the source code of *Dynamics*, a popular and open source MIP implementation [38].

The rest of the chapter is organized as follows:

In Section 3.2, we present a brief overview of the MIP protocol, focusing on the elements that are relevant to the considered problems.

In Section 3.3 we present a new attack strategy, defined as *mobility-based evasion*, that allows attackers to evade detection by exploiting node mobility.

Our strategy to defeat mobility-based evasion, based in NIDS cooperation, is presented in Section 3.4.

In Section 3.5 we present *secure triangular routing*, the new routing scheme able to solve the incompatibility between triangular routing and egress filtering.

Finally, in Section 3.6 we discuss related work.

3.2 Mobile IPv4 protocol overview

The aim of the Mobile IP (MIP) [87, 89, 90] protocol is to allow mobile hosts to communicate with other mobile or fixed hosts in the Internet through the IP protocol. To this purpose, MIP provides a way to contact a mobile node using its home address (i.e., the IP address assigned to the mobile node when it is connected to its home network) independently of its current position. This goal is reached through two mobility agents installed in both the home network and the foreign network (defined as the network to which the mobile node is currently connected). These mobility agents are called *home agent* and *foreign agent*, respectively.

It is possible to identify three main operations in the MIP protocol:

- discovery;
- registration;
- tunneling.

The *discovery* phase allows a mobile node to discover the foreign agent and start the registration phase, required to obtain an IP address valid within the foreign network address space. This temporary IP address is called *care of address*. Foreign agent discovery is normally achieved through *agent advertisement* messages, periodically broadcasted by the foreign agent itself. Alternatively, an impatient mobile node may force a foreign agent to immediately send an agent advertisement message by broadcasting an *agent solicitation* message.

Once discovered a foreign agent inside the foreign network, the mobile node starts the *registration* phase by sending a *registration request* message to the prospective foreign agent. This request is forwarded to the home agent, that sends a *registration reply* message to grant or deny the registration. The reply is processed by the foreign agent and then forwarded to the mobile node, concluding the registration phase. To improve the MIP security, *MIP authentication extensions* provide the ability to mutually authenticate mobile node, home agent and foreign agent, as well as to prevent reply attacks.

Once completed the discovery and registration phases, a mobile node connected to the Internet through a foreign network has two different IP addresses: the (permanent) home address and the (temporary) care of address. MIP defines two different routing and tunneling schemes that can be used to make a mobile node reachable through its home address independently from its current position: *triangular routing* and *reverse tunneling*.

3.2.1 Triangular routing

The path followed by packets sent through triangular routing is sketched in Figure 3.1.

IP datagrams sent by the correspondent node to the mobile node are always able to reach the home network by mean of normal IP routing. Once those datagrams are inside the home network, they are intercepted by the home agent and sent to the current mobile node care of address through a tunnel, whose endpoint is the foreign agent. The encapsulated datagrams are then received by the foreign agent that extracts the original datagrams from the tunnel and delivers them to the mobile node. Hence, communication between correspondent node and mobile node is mediated by home agent and foreign agent.

On the other hand, IP datagrams originating from the mobile node and addressed to the correspondent node are delivered through normal IP routing, without the need for any tunneling operation.

Triangular routing is the default routing scheme adopted by the MIP protocol, however its use can be prevented by the enforcement of popular traffic filtering policies at the perimeter of the involved networks. In particular, it is not possible to

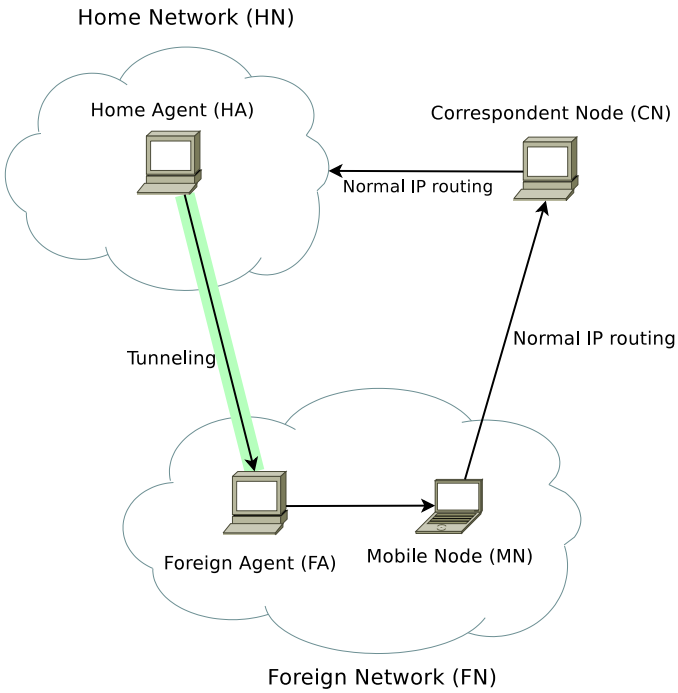


Figure 3.1: Triangular routing

communicate through the triangular routing scheme if the foreign network applies *egress filtering*.

3.2.2 Egress filtering

Egress filtering [127] is a traffic filtering policy applied by firewalls and border routers at the boundary of networks. The main idea behind egress filtering is to prevent the nodes connected inside a network to generate IP datagrams having a spoofed source address. Source address spoofing is a very simple and effective way to hide the real source of an IP datagram, and this technique is widely used in several network-based attacks (such as DoS and DDoS) and stealth network scanning activities. For these reason, adoption of egress filtering policies is a widespread best practice [54]. In most of the cases, egress filtering policies are implemented by checking whether or not the source address of IP datagrams generated from inside the network belongs to the network address space. In the positive instance, the source IP address is topologically valid, and the datagram is allowed to leave the network. If the source address does not belong to the network address space the datagram is dropped.

Unfortunately, egress filtering is not compatible with Mobile IP triangular

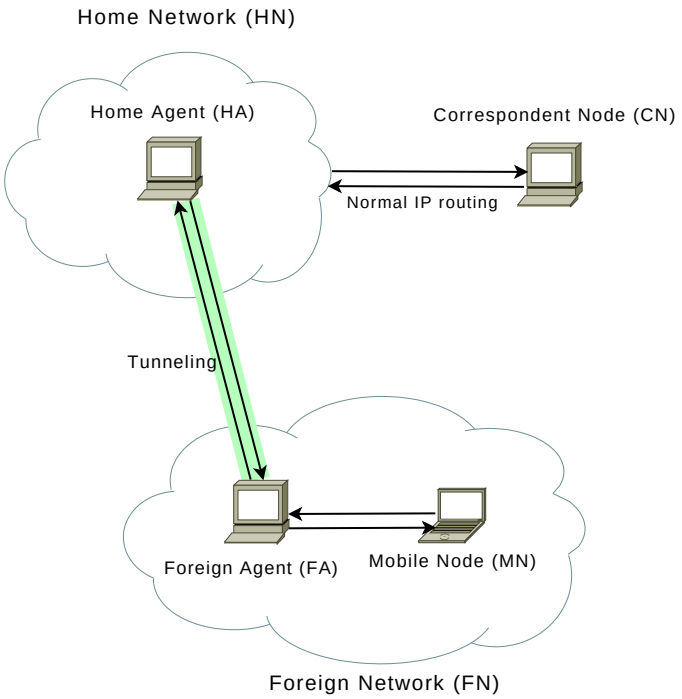


Figure 3.2: Reverse tunneling

routing. All the datagrams generated by a mobile node connected to the Internet through a foreign network have its home address as source address. Such address is a routable IP address belonging to the home network of the mobile node, hence it is not topologically correct within the foreign network address space. As a result, if the security policies of the foreign network include the egress filtering best practice (as they should), all the datagrams generated by the mobile node are dropped at the foreign network border, thus preventing any communication between mobile node and correspondent node. To enable mobile nodes to communicate even if the foreign network applies egress filtering, the MIP protocol allows to fall back to the less efficient reverse tunneling.

3.2.3 Reverse tunneling

As shown in Figure 3.2, reverse tunneling differs from triangular routing in the path followed by IP datagrams generated by the mobile node.

To avoid sending invalid network packets through the network boundary, all the datagrams generated by the mobile node are intercepted by the foreign agent and forwarded to the home agent through a tunnel. The source address of network packets used for tunneled communication is the IP address of the foreign

agent, which is topologically valid inside the foreign network and not dropped by the network device implementing the egress filtering policy. The mobile node then decapsulates the original datagrams from the tunnel, and relays them to the correspondent node through normal IP routing. The source IP address of these network packets is the IP address of the mobile node, which is topologically valid with respect to the address space of the home network. Hence this routing scheme complies with the egress filtering security policy.

The main downside of reverse tunneling is that the use of a tunnel to forward packets from the foreign agent to the home agent necessarily implies a longer routing path, thus resulting in a higher round trip time between mobile node and correspondent node. Moreover a higher round trip time may also cause a severe throughput decrease, as is the case of TCP.

3.3 Mobility-based NIDS evasion

In this section we describe a novel attack strategy, defined as *mobility-based NIDS evasion*, that an attacker can exploit to perform "stealth" attacks, not detectable even by state-of-the-art NIDS. The main idea behind mobility-based evasion is to combine traditional NIDS evasion techniques [97], such as fragmentation, with well timed roaming. Well known evasion techniques do not represent a problem for any stateful NIDS, but node mobility prevent an NIDS to perform stateful analysis on mobile connections, thus creating new opportunities for attackers.

To better describe how mobility-based evasion works, several case studies, based on realistic network scenarios, are presented in the following sections. Section 3.3.1 presents an attempt of traditional NIDS evasion without any mobility, that is easily detected by a stateful NIDS. Sections 3.3.2, 3.3.3 and 3.3.4 present three different variants of successful mobility-based NIDS evasion, that even a stateful NIDS is not able to detect.

3.3.1 Traditional NIDS evasion

Let us consider a simple network scenario, represented in Figure 3.3, in which two nodes are communicating through the Internet. We consider a *mobile node*, installed in a MIP-enabled network, communicating with a *correspondent node*. No assumption is made on the nature of the correspondent node. It can be a fixed Internet node, as well as another mobile node. In this scenario, the mobile node remains within its home network, without performing any handover. We assume that the home network includes a stateful NIDS that monitors all the Internet traffic coming to and from the home network itself (including the traffic generated and received by the mobile node). In this first example, the mobile node aims

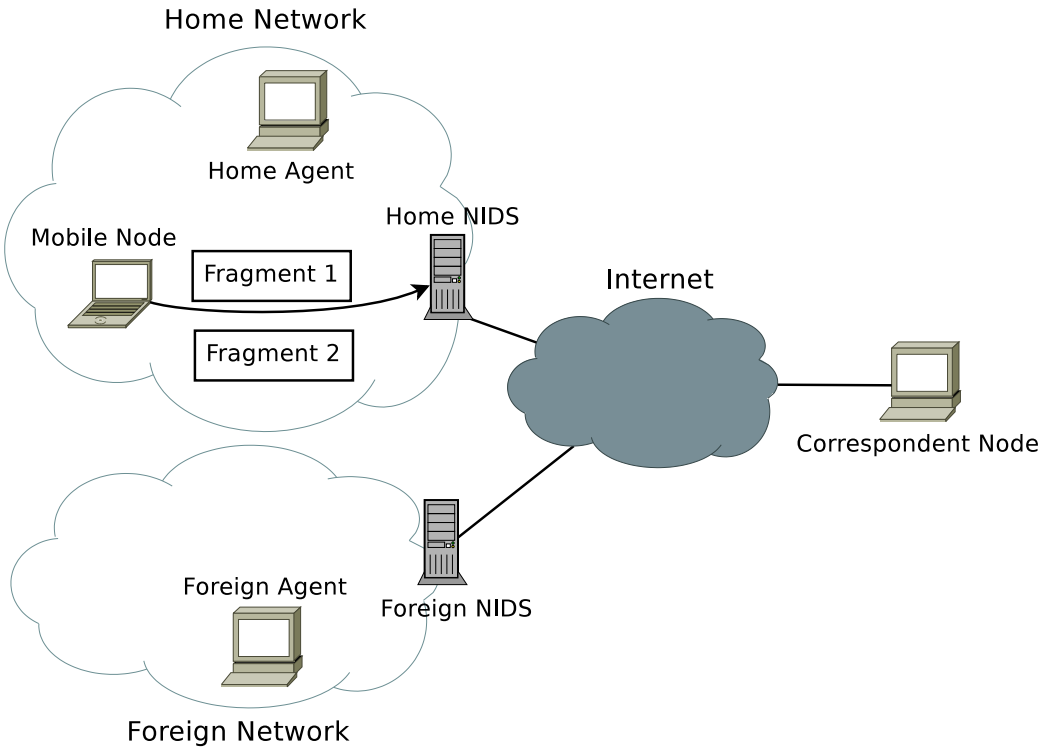


Figure 3.3: Fragmented attack without mobility

to exploit a remote vulnerability of the corresponding node by sending network packets containing a malicious payload. Moreover, the mobile node is trying to evade detection by the NIDS (or at least to make the analysis of its network activities more difficult) by separating the attack in two different fragments (Fragment 1 and Fragment 2, in Figure 3.3), each transmitted in a separate network packet.

Having assumed that the mobile node does not perform any handover while sending the attack, we can ignore the presence of the *home agent* within the home network. A mobile node residing in its home network behaves exactly as any other node in the Internet, relying only on the IP protocol for packet routing. Hence, all the network packets containing the attack fragments are intercepted by the NIDS deployed in the home network, that is able to reassemble the malicious payload and detect the attack.

3.3.2 Mobility-based NIDS evasion: mobile attacker

The scenario represented in Figure 3.4 is similar to the previous one, with the only difference that the mobile node performs an handover to a foreign network while sending the two attack fragments to the correspondent node. As for the

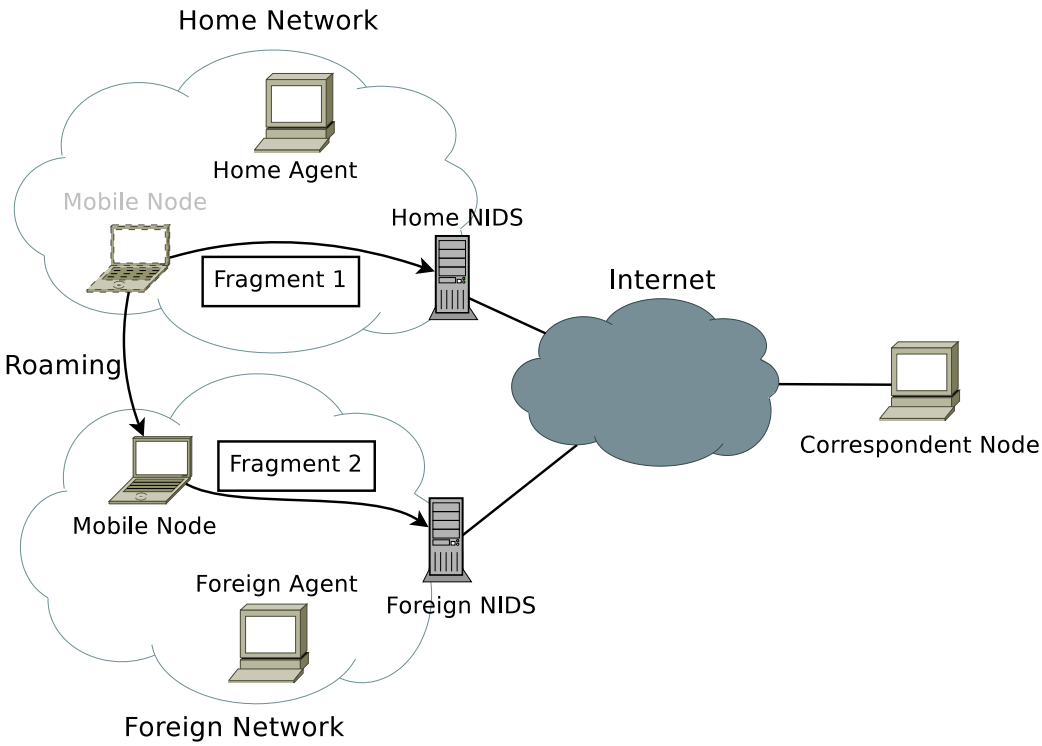


Figure 3.4: First mobile-evasion scenario: mobile attacker

home network, we assume that the foreign network includes a stateful NIDS. In particular, the exact sequence of activities performed by the attacker is as follows:

1. the mobile node sends the first attack fragment to the correspondent node;
2. the mobile node roams from its home network to a foreign network;
3. the mobile node sends the second (last) attack fragment to the correspondent node.

Similarly to the previous case (in which no mobility was involved), the first fragment of the attack is sent through the home network, and is intercepted and analyzed by the home NIDS. Being only a portion of the attack, the NIDS updates its state information without generating any alert. Then, in compliance with the *triangular routing* (default routing scheme used by Mobile-IP, described in Section 3.2.1) the second fragment of the attack is sent from the foreign network, and is routed directly to the correspondent node. This fragment is intercepted by the foreign NIDS that, having not received the previous fragment, does not have the internal state information necessary to reconstruct and recognize the whole attack.

This state information is only possessed by the NIDS deployed in the home network, but it is useless, since the home NIDS never receives the second fragment of the attack.

As a result, none of the two NIDS installed in the home and foreign networks is able to detect the attack. Hence a mobility event allowed the malicious mobile node to perform a stealth attack, that would have been easily detected if mobility had not been supported.

In this scenario, the attack can only be detected by a hypothetical NIDS possibly monitoring all the network traffic directed to the correspondent node. If this is the case, the correspondent NIDS will raise an alert concerning a network attack that appears to originate from the home network of the mobile node. The home network could then be held accountable for the possible consequences of the attack without having any mean to prevent it, or even to know that an illicit activity have been carried out. In any case, both the home and the foreign network infrastructure can be exploited by the attacker to damage third parties, without the network administrators being able to prevent, stop, or even detect the attack.

A new NIDS cooperation scheme, allowing home and foreign NIDS to exchange their internal state and nullify the detrimental effects of mobility on stateful intrusion detection, is presented in Section 3.4

3.3.3 Mobility-based NIDS evasion: mobile target

In this scenario, depicted in Figure 3.5 we consider a fixed attacker trying to compromise a mobile node. With respect to the case study presented in Section 3.3.3, the roles are reversed: the correspondent node is the attacker exploiting mobility to evade detection, while the mobile node is the target.

As in the previous case, we assume that both the foreign and home networks are monitored by a stateful NIDS sensor. We also assume that the correspondent node knows when the mobile node roams across different networks. Depending on the topology of the involved networks, this knowledge can be easily acquired by monitoring the performance of the connection to the mobile node. For example, the round trip time between the correspondent and the mobile node is bound to increase after the mobile node roams from the home network to a roaming network. The additional delay is a direct consequence of how triangular routing works, cannot be eliminated, and is easily measurable by the attacker. Moreover, it may be possible for the attacker to exploit some previous knowledge about the target behavior. If the attacker knows that the mobile node roams frequently among different networks, it may be sufficient just to wait for a given amount of time before sending the last attack fragment. (In case of an attack delivered over a TCP connection, depending on the implementation details of the vulnerable application and of the TCP/IP stack, it is possible to keep the connection alive without

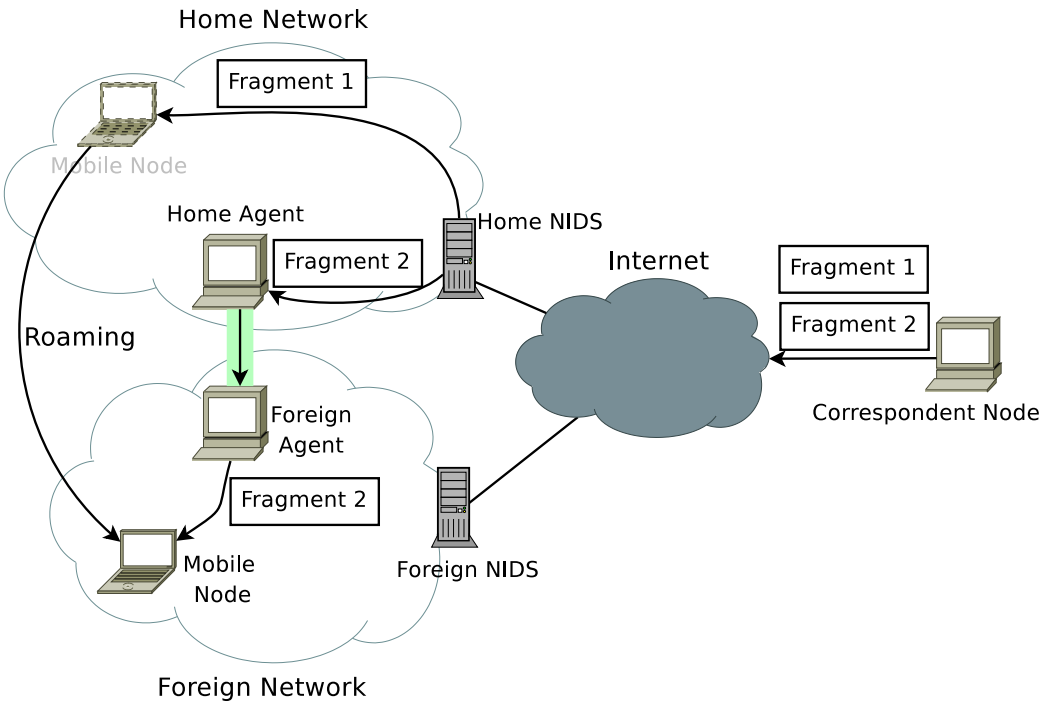


Figure 3.5: Second mobile-evasion scenario: mobile target

sending any data for long time.)

The attack proceeds as follows:

1. the correspondent node sends the first attack fragment to the mobile node;
2. the correspondent node waits for the mobile node to roam from the home network to the foreign network;
3. the correspondent node sends the second attack fragment to the mobile node.

In this case, the NIDS deployed within the home network is able to analyze both the fragments of the attack and to raise the related alert. However, this does not prevent the home agent to forward the second fragment of the attack to the foreign agent, in compliance with triangular routing specification. When the second attack fragment reaches the foreign network, its NIDS is not able to detect the attack, because it has never analyzed the first fragment of the attack. At the end of the attack only the home network knows that the mobile node is possibly compromised. On the other hand the foreign network, to which the mobile node is now directly connected, and that may suffer the consequences of hosting a compromised node, is unaware.

In this scenario, the attack can also be detected by a NIDS possibly installed in the network of the correspondent node, provided that the whole correspondent network is not controlled by the attacker.

3.3.4 Mobility-based NIDS evasion: mobile target and attacker

The last mobility-based evasion scenario is a combination of the two scenarios described in Sections 3.3.2 and 3.3.3. In this case, both the attacker and the victims are mobile nodes. We assume that all the involved networks are monitored by a stateful NIDS, and that the attacker knows when the victim roams to the foreign network.

The attack proceeds as follows:

1. the attacker sends the first attack fragment to the victim;
2. the attacker node waits for the victim to roam from the home network to the foreign network;
3. the attacker itself roams to a different network;
4. the attacker sends the second attack fragment to the victim.

Note that the order of steps 2 and 3 can be inverted without changing the attack outcome.

In this scenario, only the home network of the victim will be able to detect the attack, while the victim foreign network (to which the possibly compromised victim is now connected) and the attacker home and foreign networks (used to launch the attack) are completely unaware.

3.4 Defeating mobility-based evasion through NIDS cooperation

While traditional NIDS evasion techniques based on simple fragmentation are no longer an issue for stateful NIDS, Section 3.3 demonstrates how fragmentation can be effective when coupled with node mobility. The main issue preventing a stateful NIDS to monitor a mobile connection is represented by the inability of a sensor to receive all the network packets exchanged between the correspondent and the mobile node. By receiving only a fraction of the packets, a NIDS is not able to build a complete state of the connection, thus behaving like a stateless NIDS that can be easily evaded through fragmentation and through all the other traditional evasion techniques [97].

3.4.1 NIDS cooperation scheme

To solve mobility-based evasion problems we propose an innovative NIDS cooperation technique, based on the exchange of relevant state information among the intrusion detection systems deployed within the home and the foreign networks. Let us take as example the simple mobility-based evasion scenario with a mobile attacker, represented in Figure 3.4. The NIDS deployed in the foreign network is not able to detect the attack because only its second fragment is received. However, the internal state information that would allow a NIDS to correctly detect the attack after analyzing only the second fragment exists, and it is stored within the internal data structures of the NIDS deployed in the home network, that analyzed the first fragment of the attack. The main idea is to enable cooperation among the two sensors, by migrating the state information generated by the home NIDS to the foreign NIDS. When migration take place, the internal state information necessary to detect the fragmented attack *follows* the mobile node to the new network, thus allowing the new NIDS to carry out a stateful analysis on the mobile connection.

The state migration framework described in Section 2.4, designed to enable the ParNIDS architecture 2 to perform both stateful analysis and dynamic load balancing, can be used to extract, migrate and fuse NIDS internal state information. To take advantage of state migration, both the the networks among which the mobile node is roaming have to be monitored by a state-migration enabled NIDS, such as our modified version of Snort [26]. State migration activities can than be coordinated by the Mobile IP agents already involved in the roaming process.

To better explain the sequence of activities involving mobile node roaming and NIDS state migration, let us consider the mobile attacker evasion scenario, represented in Figure 3.4. The mobile node starts the attack by sending the first fragment to the correspondent node, and then it roams to the foreign network. The activity diagram in Figure 3.6 represents a graphical representations of all the messages exchanged among the mobile node, the mobility agents, and the home and foreign NIDS.

After having discovered the foreign agent in the foreign network, the mobile node sends a *registration request* to the foreign agent. The foreign agent then relays the registration request to the home agent of the mobile node. When the home agent receives the registration request from the foreign agent, it asks the home NIDS to export all the internal state related to the mobile node. The home agent also sends a *registration reply* message to the foreign agent. The foreign agent relays the registration reply to the mobile node thus completing the Mobile IP registration phase. Meanwhile, the home NIDS exports the internal state related to the mobile node, and sends the serialized state representation to the home agent. The home agent relays the serialized state representation to the foreign NIDS, that

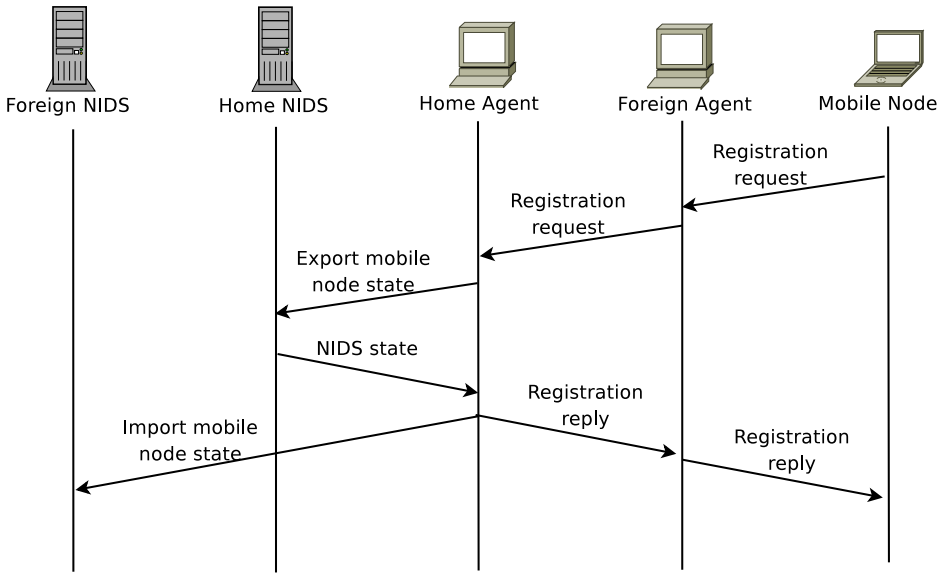


Figure 3.6: Roaming and state migration from the home network to the foreign network

imports it.

With respect to the standard Mobile-IP protocol, we only add three messages: the request of the internal state sent from the home agent to the home NIDS, the reply of the NIDS containing the serialized internal state, and the message from the home agent to the foreign NIDS containing the serialized internal state to be imported. These messages can be generated and transmitted in parallel with the standard Mobile-IP registration phase, without any additional delay. The mobile node will be reachable with its home address as soon as it receives the registration reply from the foreign agent, while the foreign NIDS will be able to detect the attack as soon as the serialized state representation is imported.

It is important to notice that no strict synchronization is required between the registration phase and the state exchange. In the ideal scenario, the foreign NIDS receives the serialized state information before analyzing the second fragment of the attack. Hence, when this attack fragment is received, the NIDS already contains the complete state of the mobile connection and correctly reassembles and detects the attack. However, as demonstrated in Section 2.8.1, the foreign NIDS is able to correctly recognize the whole attack even though the internal state generated by the home NIDS is received after the second fragment of the attack has already be analyzed.

Similarly, the state related to the mobile node is transferred back from the foreign NIDS to the home NIDS as soon as the mobile node registration with the foreign agent terminates.

3.4.2 Prototype implementation

The NIDS cooperation scheme described in Section 3.4.1 has been implemented through a prototype based on open source software.

The first requirement for NIDS cooperation to take place is to rely on intrusion detection systems able to export and import external state representation. To this purpose, we used our improved version of Snort [26], whose implementation is described in Section 2.7.1.

Moreover, in order for the home agent to coordinate the state migration process, we modified the Open Source *Dynamics* [38] Mobile IP implementation. Our modified version of Dynamics include an XMLRPC client used to interact with the XMLRPC server embedded in our custom Snort implementation. To enable the interaction between the foreign agent and the packet filter, we identified the functions used by Dynamics to establish and tear down the tunnel between foreign agent and home agent. This tunnel is used in both the triangular routing and reverse tunneling configurations, is established only after a successful registration and is destroyed immediately after the registration expires or the mobile node becomes unreachable. In Dynamics, those functions are called *create_tunnel_upwards* and *set_expr_timer*, respectively.

After receiving a registration request form a foreign agent, the home agent extracts from the request the home address of the mobile node that is trying to connect to the foreign network. This address is used to generate an XMLRPC request to the home NIDS, asking for all the state information related to the mobile node. After having received the requested state information from the home NIDS, the home agent sends a new XMLRPC request to the foreign NIDS, containing the state information related to the mobile node.

To allow the home agent to send the serialised state information to the foreign NIDS, we add a new field in the registration request message that the foreign agent sends to the home agent. This field is used to transmit the IP address of the foreign NIDS, that can be directly contacted by the home agent.

We remark that it is also possible to perform state migration without modifying any Mobile IP message by making the foreign agent to send its XMLRPC request to the foreign NIDS indirectly, using the foreign agent (whose IP address is already known) as a proxy. This design choice guarantees complete compatibility with the Mobile IP standard, at the expense of a longer time required to complete the state migration among the home and foreign NIDS.

This prototype has been used to perform experimental validation of the proposed state migration scheme.

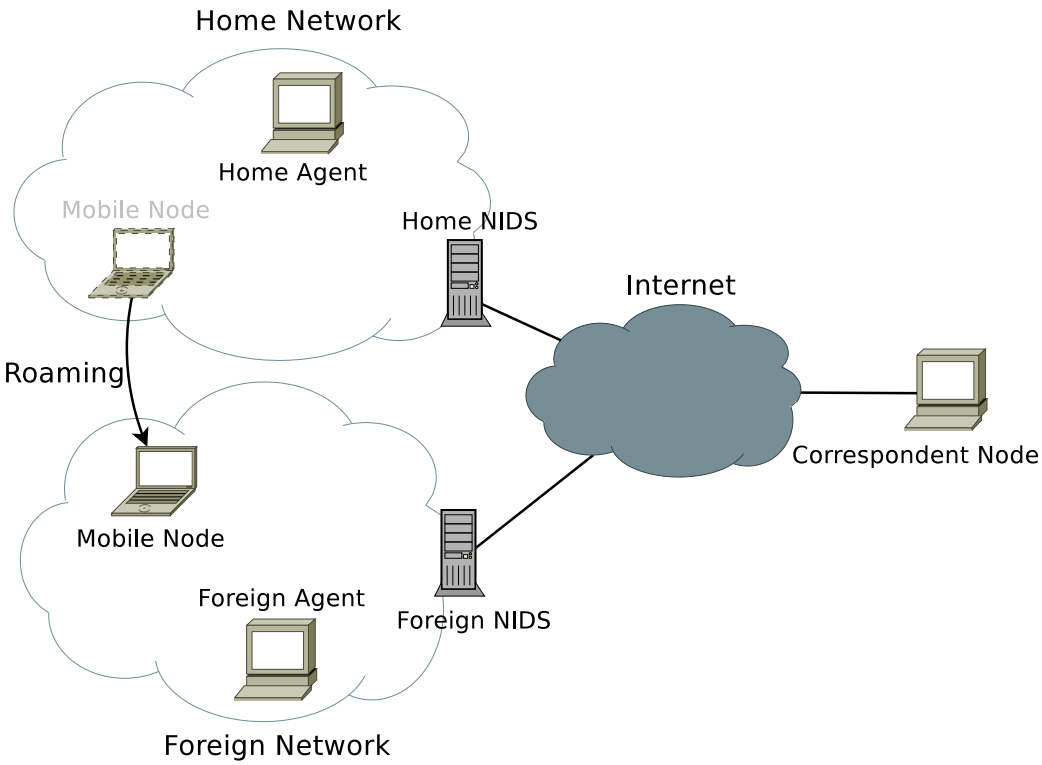


Figure 3.7: Emulated network topology

3.4.3 Experimental validation

We evaluate the ability of the proposed solution to defeat mobility-based evasion through experiments carried out in an emulated network setup, where the topology is shown in Figure 3.7.

We deploy the MIP agents in two different networks connected to the Internet. Home agents and foreign agents have been implemented with our modified version of Dynamics, installed on Linux machines. Home NIDS and foreign NIDS have been implemented with our modified version of Snort, that is able to execute state migration. In order for the intrusion detection systems to correctly analyze the network traffic flowing in tunnels created by the mobility agents, we configured Iptables [50] to forward a copy of the network packets directly toward the NIDS input interfaces, before they are sent inside the tunnel. This configuration allows the two sensors to analyze the traffic flowing among the two Mobile IP agents without requiring any preprocessing, that would have been necessary had the packets been encapsulated. Moreover, this solution is completely independent of the tunneling techniques used by the mobile agents.

Correspondent node and mobile node are Linux machines, and do not require

any MIP-related software, nor modification in the TCP/IP stack. For the sake of simplicity, in the described experimental setup the correspondent node is a fixed host connected to the Internet. However we do not make any assumption on the correspondent node ability to roam among different networks, and the correspondent node may well be a mobile node itself. On the other hand, the mobile node is assumed to roam from the home network to the foreign network.

The described network topology has been realized by emulating all the required hosts through User Mode Linux (UML) [121]. Our experimental setup has been validated by opening TCP connections between the mobile node and the correspondent node (the mobile node downloaded large files from an Apache web server installed in the correspondent node) and verifying the mobile node ability to roam to the foreign network while maintaining the connection active.

In this experiment, the mobile node is the attacker, trying to execute a stealth attack on the correspondent node. The mobile attacker sends to the correspondent node a string of 21 “a”. While being innocuous, a packet containing this string triggers a Snort signature designed to detect a common NOOP-sled widely used by many buffer overflow attacks. This payload is fragmented in two different portions.

In the first run, the state migration is disabled. The mobile node sends the first fragment of the attack, roams to the foreign network, and sends the second fragment of the attack. As expected, none of the two intrusion detection systems deployed in the home and in the foreign network are able to detect the attack, meaning that the attacker successfully evaded detection.

In the second run, we activate state migration. As before, the mobile node sends the first fragment of the attack and roams to the foreign network. However, this time the home agent sends to the foreign NIDS the state information related to the mobile node. Hence, when the mobile node sends the second fragment of the attack, the foreign NIDS is able to reconstruct the complete payload, detect the attack and raise the related alert:

```
[**] [1:1394:5] SHELLCODE x86 NOOP [**]
[Classification: Executable code
was detected] [Priority: 1]
12/07-11:54:06.198166 xxx.xxx.xxx.xxx:xxxxx
-> xxx.xxx.xxx.xxx:xxxxx
TCP TTL:64 TOS:0x0 ID:29085 IpLen:20
DgmLen:74 DF
***AP*** Seq: 0x2F18460F Ack: 0x2F1507E8
Win: 0xB50 TcpLen: 32
TCP Options (3) => NOP NOP TS: 278454 295001
```

Similar experiments have been repeated to emulate all the scenarios presented in Sections 3.3.3 and 3.3.4. In all the cases, the NIDS provided with the second attack fragment and with the state information related to the first attack fragment have been able to correctly detect the attack and raise the expected alert.

3.5 Secure triangular routing

As described in section 3.2.2, the default Mobile IP routing scheme, *triangular routing*, is not compatible with the *egress filtering* best practices [54]. In compliance with triangular routing, a mobile node connected to a foreign network sends IP datagram to the corresponding node using the mobile node home address as source IP address. This IP address is not valid within the address space of the foreign network, hence it is blocked by any modem, router and firewall implementing the egress filtering best practice. To avoid this incompatibility, allowing Mobile IP to be deployed in networks implementing egress filtering policies, it is possible to fall back to the alternative *reverse tunneling* 3.2.3 routing scheme, at the price of longer routing paths, higher round trip times and lower throughput.

Several optimized routing schemes for IP mobility that solve the incompatibility issue and improve the performance of mobile connections have been proposed [91, 128, 136], however they require modifications not only to the mobility agents and the mobile node, but also to the correspondent nodes and/or their networks.

In this chapter we propose *secure triangular routing* (STR) [68], a novel routing scheme for Mobile IP able to solve incompatibility issues between triangular routing and egress filtering. The main idea behind the proposed solution (described in Section 3.5.1) is to enable cooperation between the mobility agents and the packet filter deployed in MIP-enabled networks. This solution does not require any modification neither in correspondent nodes nor in their networks.

3.5.1 Secure Triangular Routing design

When egress filtering policies are enforced on routers and firewalls at the network boundaries, the packet filter checks that the source address of all the outgoing packets is valid within the address space of the protected network. A more effective egress filtering approach is to require the IP address of each outgoing network packet to belong to a host that is currently active inside the network. Such a dynamic filtering policy prevents a node to generate IP datagrams with spoofed IP addresses belonging to the network address space but currently unused. However, this approach is more complex to implement. It requires a tracking mechanism to keep trace of all the active hosts connected inside the network. The list of

active addresses has to be continuously updated to reflect host connections, disconnections and IP address changes. The tracking mechanism has to be secured to prevent an attacker to make the system believe that arbitrary IP addresses are currently active (leading to the introduction of unauthorized traffic filtering rules) or inactive (thus preventing its legitimate owner to communicate). These requirements are not practical, especially for networks with non-trivial size and characterized by high churn rate.

Our proposal is based on the observation that, within the scope of mobile nodes relying on the Mobile IP protocol, a constantly updated and secure tracking mechanism already exists. Indeed, the foreign agent always knows both the care of address and the home address of all the mobile nodes that are currently allowed to connect to the Internet through the foreign network. This information is constantly updated, and Mobile IP registration mechanisms provides strong security guarantees through the use of authentication extensions.

We propose to add a new functionality to the Mobile IP foreign agent, by allowing it to dynamically reconfigure the packet filter. This design choice allows for a dynamic and completely automatic update of the access control rules used to enforce the egress filtering security policy, by taking into account the mobile nodes that are currently connected inside the network. Following that idea we design secure triangular routing by extending the MIP registration phase to add the steps required for dynamic packet filter reconfiguration. To better explain how secure triangular routing works, we consider as case study the network topology shown in Figure 3.8.

In this example a mobile node roams from its home network to a foreign network. We assume that the foreign network is protected by a packet filter implemented through a firewall, enforcing egress filtering best practices. An activity diagram describing the registration scheme of secure triangular routing is shown in Figure 3.9.

The first four messages exchanged in Figure 3.9 represent the standard Mobile IP registration phase. After having received a registration reply message from the home agent, the foreign agent opens a connection with the packet filter (a firewall, in this example) and adds a single rule that allows IP datagram having the mobile node home address as the source IP address to leave the protected network. This rule is also automatically removed by the foreign agent as soon as the registration expires (for example, because the mobile node roams to a different foreign network, or because the registration is not renewed). Hence, after the mobile node registration expires, a possible attacker (the “spoofer node” in Figure 3.9) is not able to send spoofed IP datagrams using the home address of the mobile node as source address.

Secure triangular routing allows the use of the more effective triangular routing scheme, without the need to relax the security policies of the foreign network.

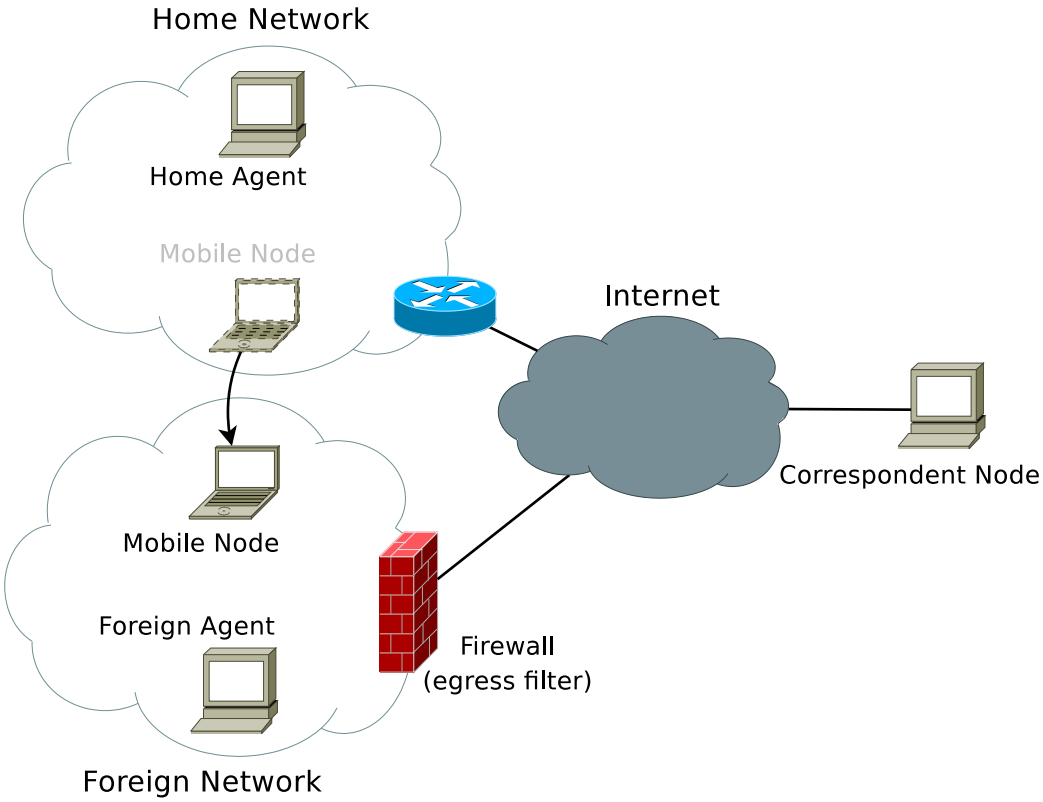


Figure 3.8: MIP-enabled network topology with egress filtering

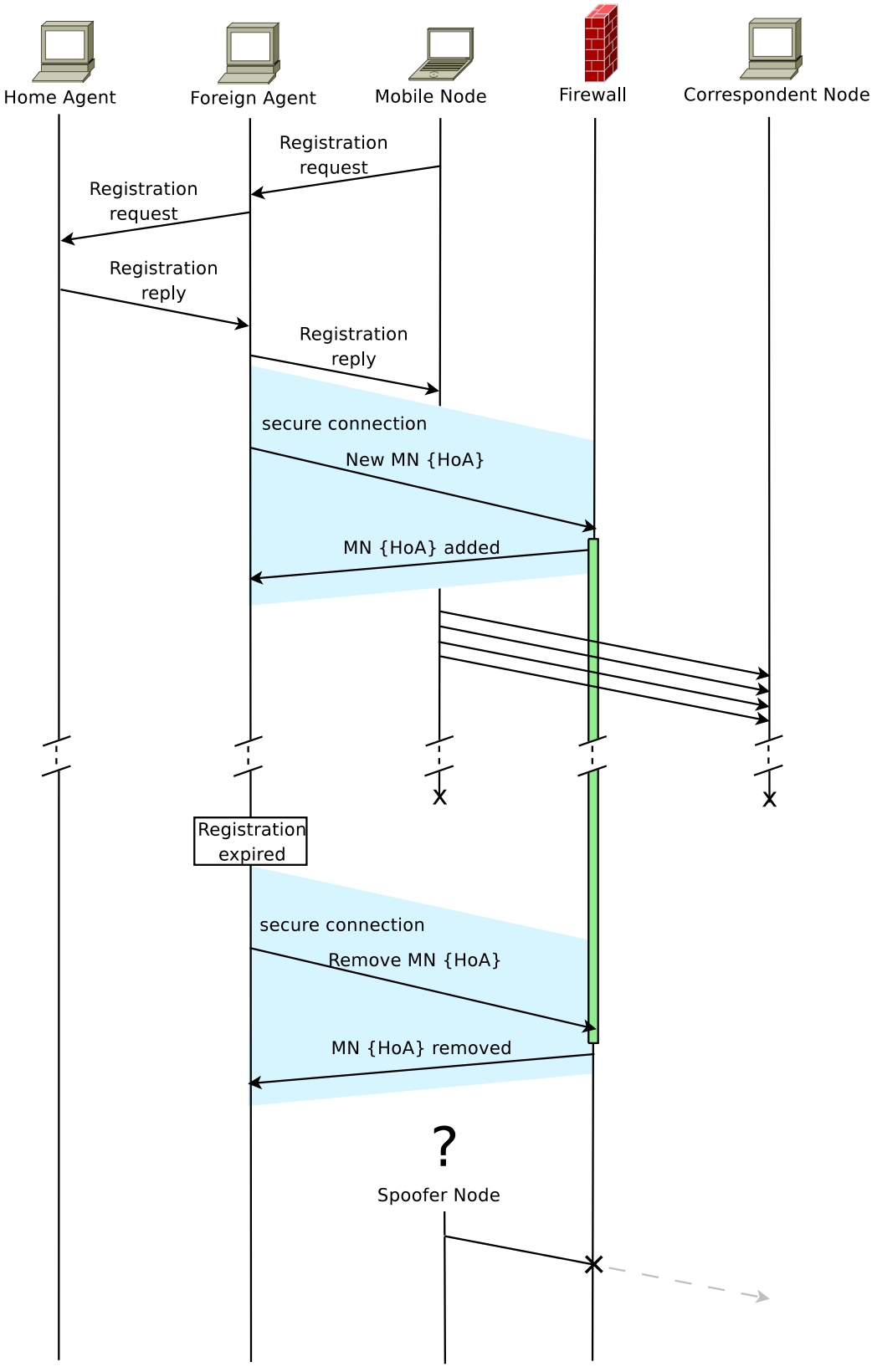


Figure 3.9: Secure Triangular Routing: activity diagram

STR can be deployed just by modifying the implementation of the foreign agent. Moreover, being the modified foreign agent fully compliant with the Mobile IP standard, no modification in other mobility agents, network components or correspondent and mobile nodes is required.

The introduction of the new rule necessary to allow IP datagrams having the home address of the mobile node as the source address to pass through the packet filter is *not* a relaxation of the egress filtering security policy. Each mobile node already registered to the foreign agent, although only temporarily, is connected to the network and is authorized to send its network packet through the network boundary.

The integration between foreign agent and packet filter allows us to leverage all the security properties of the Mobile IP protocol with authentication extension. Mutual authentication among foreign agent, mobile node and home agent is guaranteed, as well as protection from replay attacks. These mechanisms effectively prevent attacker nodes from exploiting the STR implementation by replaying previously eavesdropped traffic to force the foreign agent in a fake registration and to feed stale or deliberately false rules to the packet filter. The connection between the foreign agent and the packet filter has also to be secured, providing mutual authentication among these two network elements, as well as protection from man in the middle and replay attacks. All these requirements are met through the application of widespread cryptographic primitives (see Section 3.5.2).

Increased security levels can also be achieved by limiting the ability of the foreign agent to modify the access control list enforced by the packet filter. In particular, the foreign agent only needs to be able to:

- add a new rule allowing IP datagrams with a single known source address that does not belong to the network address space to *leave* the network (no new traffic is allowed to enter the protected network)

- remove a packet filter rule, but *only if that rule has previously been added by the foreign agent itself* (all the other filtering rules are not modified)

Other limitations can be arbitrarily imposed on the highest number of rules that the foreign agent may add and on the highest rate at which those rules can be added, thus preventing a compromised or malfunctioning foreign agent to disrupt packet filter performance by adding too many rules at a too high rate.

All these security measures effectively prevent the exploitation of dynamic packet filter reconfiguration by an attacker. Hence, we state that STR is as secure as a standard Mobile IP implementation, while being compatible with egress filtering and achieving better performance.

3.5.2 Prototype implementation

We demonstrate the viability of STR through a prototype based on a modified version of the *Dynamics* [38] Mobile IP implementation. To enable the interaction between the foreign agent and the packet filter, we identify the functions used by Dynamics to establish and tear down the tunnel between foreign agent and home agent. This tunnel is used in both the triangle routing and reverse tunneling configurations, is established only after a successful registration and is destroyed immediately after the registration expires or the mobile node becomes unreachable. In Dynamics, those functions are called *create_tunnel_upwards* and *set_expr_timer* respectively.

In our STR implementation, we modified those function to spawn a new process with the purpose of interacting with the packet filter. Filtering rules update is then performed in parallel with tunnel management, thus minimizing the time required to complete the mobile node handoff. This concurrency does not introduce race conditions, because communications between home agent and foreign agent occur through IP datagrams that comply with egress filtering policies, and that are not dropped by the packet filter even though filtering rules have not been updated yet. The current implementation also checks the outcome of the packet filter interaction. If for any reason (for example, a temporary inability to communicate with the packet filter) the filtering rules can not be updated, the foreign agent is able to fall back to the slower reverse tunneling scheme, thus guaranteeing mobile nodes connectivity.

While the proposed STR approach is generally applicable, the design and deployment of the interface between the foreign agent and the packet filter is necessarily influenced by the technology used to implement the egress filtering policy. In our prototype, traffic filtering policies are enforced through a Linux firewall, running Iptables [50].

The foreign agent adds and removes Iptables rules indirectly, by opening an SSH connection to the firewall and executing an interface script. The SSH connection provides public key mutual authentication between foreign agent and firewall, as well as protection from eavesdropping, replay and men in the middle attacks. The script implements the interface that the foreign agent uses to interact with the packet filter, implements all the restrictions discussed in Section 3.5.1, and decouples foreign agent activities from the specific commands used to modify the Iptables ruleset.

3.5.3 Experimental results

The performance gain that can be achieved by adopting STR has been evaluated through several experiments carried out in the emulated network shown in Fig-

ure 3.8.

We deployed the Mobile IP agents in two different networks, connected to the Internet. The foreign network is connected to the Internet through a firewall, implementing the egress filtering best practice: only IP datagrams with a routable source address belonging to the network address space are allowed to go through the network boundary. The home network is connected to the Internet through a gateway, that may also implement traffic filtering best practices (we do not make any assumption on the ability of the home network gateway to filter network packets). Home agents and foreign agents have been implemented with our modified version of Dynamics, installed on Linux machines. The correspondent node is a normal Linux machines, and do not require any MIP-related software, nor modification in its TCP/IP stack. For the sake of simplicity, in the described experimental setup the correspondent node is a fixed host connected to the Internet. However we do not make any assumption on the correspondent node ability to roam among different network, and the correspondent node may well be a mobile node itself. On the other hand, the mobile node is assumed to roam from the home network to the foreign network.

The described network topology has been realized by emulating all the required hosts through User Mode Linux (UML) [121]. The *netem* [80] WAN emulator has also been installed on all the nodes connected to the Internet (the firewall in the foreign network, the gateway in the home network, and the correspondent node) to emulate different bandwidth, network delay and packet loss constraints. Our STR implementation has been installed in the foreign agent, while the home agent runs an unmodified Dynamics installation (though nothing prevents the use of STR in the home agent, also). Our experimental setup has been validated by opening TCP connections between the mobile node and the correspondent node (the mobile node downloaded large files from an Apache web server installed in the correspondent node) and verifying the mobile node ability to roam to the foreign network while maintaining the connection active.

This experimental setup has been used to measure the performance of network communications between correspondent node and mobile node in both reverse tunneling and secure triangular routing scenarios. In particular, we experimentally evaluated two metrics that are relevant to the end-to-end communication performance: the round trip time and the throughout of TCP connections.

The graph shown in Figure 3.10 compares the round trip time (RTT) values of IP datagrams sent between the mobile node and the correspondent node using reverse tunneling and secure triangular routing schemes. RTT measurements have been repeated under different network conditions. The y-axis represents the RTT, measured in milliseconds, wile the x-axis represents the delay introduced by netem on the gateway of the home network, affecting all the network packets flowing between the home network and the Internet. By varying this delay, we are

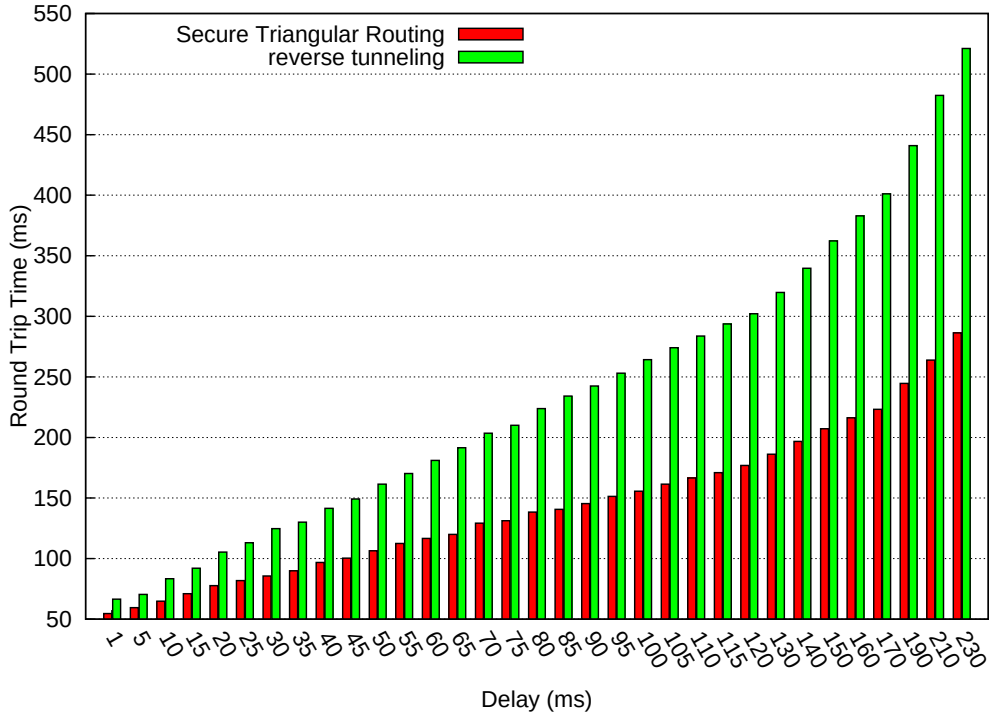


Figure 3.10: Round Trip Time comparison between secure triangular routing and reverse tunneling

able to increase the "network distance" between the home network and both the correspondent node and the foreign network. The delay introduced by netem on the gateway of the foreign network is fixed and equal to 45ms.

As expected, the RTT for both STR and reverse routing grows as the delay between the home network and the Internet increases. However, due to its shorter routing path, the RTTs of the STR are always lower with respect to reverse tunneling. Moreover, triangular routing performance gain steadily increases for higher network delays.

To evaluate the impacts of higher RTT values on end-to-end MIP performance, we measured the throughput of TCP connections between the mobile node and the correspondent node. Throughput measurements have been carried out using the network benchmark tool *netperf*[1].Throughput has been measured for different delays introduced by netem on the gateway of the home network. Each measurement has been repeated until netperf reached a error interval of 2% with 99% confidence. Experimental results are shown in Figure 3.11. The x-axis represents the delay introduced by netem on the gateway of the home network, while the

[1]http://www.netperf.org/

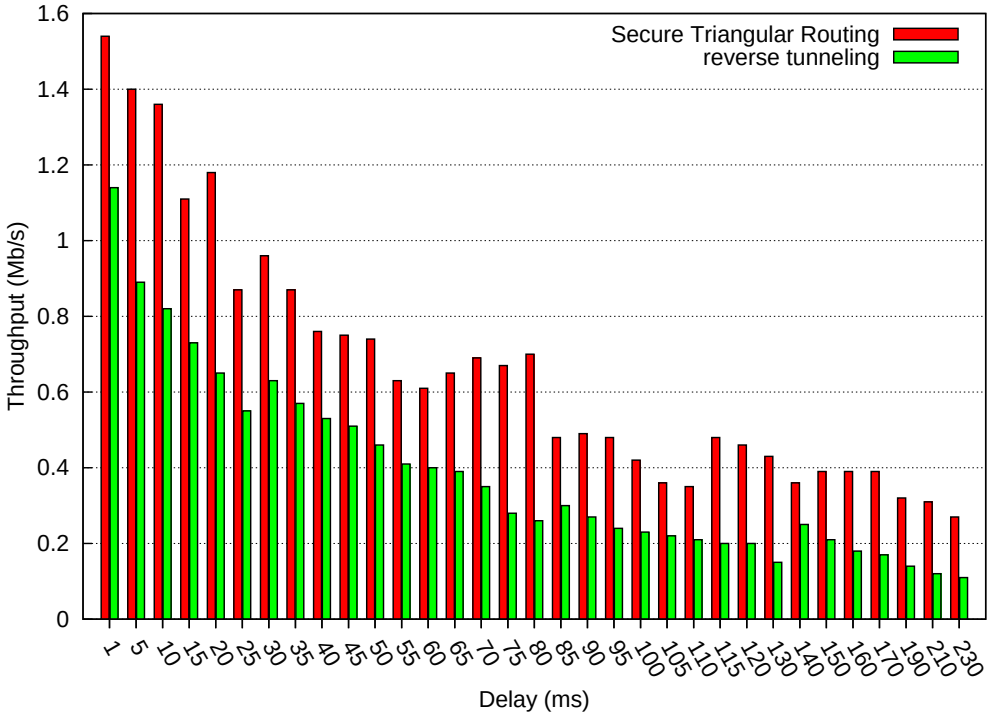


Figure 3.11: Throughput comparison between secure triangular routing and reverse tunneling

measured throughput is shown on the y-axis.

It is possible to observe how detrimental a large value of RTT is for the TCP protocol. Higher delays result in a lower throughput for both the routing schemes. However, the throughput achieved through secure triangular routing is consistently higher than that achieved by the reverse tunneling. As observed for the RTT, the performance gain of triangular routing grows proportionally to the growth of the network delay.

3.6 Related work

Several works addressing performance and security issues of the Mobile IP protocol have been published in literature. To the best of our knowledge, the problem of stateful analysis of mobile connections have never been highlighted or solved before. On the other hand, secure triangular routing directly compares with other proposals in the areas of routing optimization and security concerns.

3.6.1 Routing Optimization

There is a fundamental trade off between improved routing efficiency and backward compatibility with all the standard IP hosts commonly connected to the Internet. To reach high applicability, Mobile IP favours compatibility and transparency over performance. This choice allows mobile nodes to communicate with correspondent nodes without requiring any modification in their software stack or in their networks. The main drawback of this choice is that triangle routing may be far from the optimal IP routing, especially if there is a significant network distance between the home network and the foreign network. This problem is further exacerbated when, due to packet filtering policies applied by the involved networks, reverse tunneling becomes necessary.

The sub-optimal routing issue has been addressed by the same authors of the Mobile IP protocol in [91], where an extension to Mobile IP is proposed that allows the correspondent node to send IP datagrams directly to the mobile node, thus achieving the same efficiency level of normal IP routing. To obtain this result, the correspondent node needs to maintain an updated cache of *binding addresses*, used to store the care of addresses currently associated to all the mobile nodes with whom the correspondent node is communicating. The need to modify the networking stack of all the correspondent nodes accordingly represent the main drawback of this solution, as well as the main reason that prevents the integration of route optimization within the MIP protocol specifications.

Similar in principle to the main idea in [91], are the solutions proposed in [136] and [128]. In particular, in [136] authors propose to move the burden of storing and maintaining the binding cache list from the correspondent node to the border router of the correspondent node network. While this solution allows a mobile node to communicate transparently with unmodified correspondent nodes, it requires modifications of the border router software of (ideally) every network.

On the other hand, in [128] the binding cache is not maintained by the border router of the correspondent node, but by a completely new network element, called *correspondent agent*. From a compatibility perspective, [128] differs from [91] and [136] because no modifications are required in correspondent nodes or in their existing network components. However, the introduction of a new mobility agent is required in (ideally) every network.

Our work clearly differentiates from the existing solutions. We believe that any new scheme requiring modifications in the structure of existing networks and in the software stack of their components and hosts is not practical. Hence our main concern is to improve Mobile IP performance while maintaining the highest possible degree of transparency, compatibility and interoperability with all the currently deployed hosts and network equipment. In this context, we acknowledge the triangular routing as sub-optimal but necessary routing scheme, and focus on

removing the need of reverse tunneling.

3.6.2 Security concerns

The previous efforts in improving Mobile IP security are mainly related to handoff and security associations with mobility agents. Solutions to these problems are described in the Mobile IP protocol specification [89] and further investigated in [88]. In particular, in [89] the problem represented by egress filtering policies is also considered, and solved thanks to the introduction of reverse tunneling.

Several works exist that deal with integration between Mobile IP tunnels and Virtual Private Networks (VPN) [9, 22] and/or firewalls [73]. The proposed solutions provide secure mobile connections through the extensive use of (possibly encrypted) tunneled communications, having VPN concentrators (possibly implemented through modified firewalls or gateways) as endpoints. As an example, In [73] authors assume that networks are protected through a SKIP (Simple Key management for Internet Protocols, [115]) firewall, modified to be aware of Mobile IP packet format and semantic.

All of the proposed solutions rely on the underlying assumptions that both mobile and correspondent nodes are willing and able to establish an encrypted tunnel or a dedicated VPN for each mobile connection. While this assumption can be true for specific scenarios (such as *Intelligent Transportation Systems* devised in [22]) this is not the general case for hosts connected to the Internet.

On the other hand, we only assume that it is possible to modify the filtering rules applied by the packet filter used to protect the foreign network. Moreover, our proposal is fully compliant with the Mobile IP standard, hence it can be easily integrated with VPN or more sophisticated firewalls and gateways whenever that is required by the specific deployment scenario.

3.6.3 Other Mobile IP optimizations

Several other works have been published that aim to improve performance of mobile connections. Interesting solution have been devised to avoid packet losses during handoffs through packet buffering schemes [92, 134]. While those proposals do not shorten the handoff delay, they prevent packet loss and the consequent slow down of TCP connections. Handoff delay can be effectively reduced through regional [41,116] and hierarchical [12,47,64] registration schemes, as well as multiple links [72], and specific configuration of wireless network interface in wireless LAN scenarios [108]. Other ways to improve the end-to-end throughput of Mobile IP connections have been investigated, like the integration of Mobile IP traffic with Multi Protocol Label Switching (MPLS) [99] and the use of Stream Control Transmission Protocol (SCTP) instead of TCP as transport layer protocol [44].

All these optimization are orthogonal to our work. Moreover, being the proposed solution fully compliant with Mobile IP, nothing prevents their integration.

To the best of our knowledge, secure triangular routing is the first proposal able to achieve the same performance of triangular routing in presence of egress filtering, while not relying on modifications to the Mobile IP protocol, or to correspondent nodes and their networks.

Chapter 4

Cooperative architectures for malware detection and analysis

4.1 Introduction

The Internet is growing steadily and rapidly, both in the number of connected hosts and in the speed that network connections can achieve. The large number of users, together with the standardization of the hardware and software configurations of most Internet hosts, enable attackers to compromise a very large number of computers through automated and self-propagating malware targeting widespread software vulnerabilities. Legions of compromised machines are used by attackers to build large networks of federated bots, referred to as *botnets* that are used to carry out malicious activities. In particular, botnet infrastructures are responsible for the vast majority of the unsolicited emails (*spam*) and for *Distributed Denial of Service* attacks (DDoS).

The defense mechanisms currently deployed against the most virulent forms of malware and botnets are inadequate. Some estimates [45] show that the botnet created through the *Storm* worm reached the size of two million machines, thus giving its owner a computational power theoretically higher than the world top supercomputers. Other recent data concerning worm spread can be obtained from the ShadowServer website [107]. Botnets activities can also be inferred by the high number of attacks sustained by any personal computer connected to the Internet through a public (even though dynamic) IP address.

Traditional monitoring techniques are not able to preemptively alert the network administrator about ongoing malicious activities before they reach the protected network. Network intrusion detection systems can only generate alerts for malicious activities that have already been perpetrated, hence their alerts cannot be used to deploy proactive defenses, but only to restore the compromised machines.

The time lag between the infection of a machine and the corrective actions taken by an administrator can be used by the attacker to control the machine, generating spam, attacking third parties hosts, sniffing traffic on the local network, hosting rough files or websites and stealing data. Moreover, each network implements the appropriate countermeasures individually, without any knowledge of what is happening in other networks.

Another weakness of modern NIDS is their inability to assess whether a detected attack represents a real threat for the protected network. Attacks carried out by automated malware target known vulnerabilities in popular software packages, without any previous assessment of the software and hardware configurations of the attacked machines. It is a responsibility of the network administrator to examine each alert and to verify if the targeted machine is vulnerable to the attack.

Our goal is to devise new cooperation schemes that allow network administrators to be notified of possible attacks before being targeted by them, thus enabling the deployment of preventive defenses. In particular, to better react to threats whose nature is inherently distributed, such as self-replicating malware, or intrusions leveraging widespread vulnerabilities, we think that distributed network monitoring approaches are best suited.

In this context, we design and implement two distributed architectures for cooperative intrusion detection and malware analysis [27, 70], based on some sort of collaboration among sensors placed in different networks.

The first cooperative architecture [27] is characterized by a hierarchical topology. Its main innovation is the ability to gather alerts and malware specimens from a wide network space, thus allowing for early detection of emerging threats. Moreover, all the networks involved in the cooperative intrusion detection effort (for example by hosting one or more sensors) can be alerted about new threats as soon as they are detected.

The second architecture [70] relies on a peer-to-peer infrastructure that is based on a distributed DHT overlay. This design choice makes it possible to realize a self-organizing cooperation infrastructure, characterized by high dependability and without single points of failure.

The proposed architectures are designed to fulfill the following goals.

Reduction of the detection delay. Worms usually select the Internet hosts to attack by generating a list of IP addresses of possible victims. A common strategy to generate such list is to guess IP addresses in a range near the address of the infected machine, because those addresses are probably valid and there is a good chance that the corresponding machines have a similar configuration. By deploying a network of cooperating sensors distributed in heterogeneous networks it is possible to spot massive attacks timely.

Behavioral analysis of malware. There have been some proposals in the past for the creation of coordinated network IDS infrastructures [48, 67, 135] (see Section 4.4 for further details), but the proposed architectures have the added value of allowing the collection of real malware samples.

The cooperative architectures proposed in this thesis rely on honeypots, that are able to capture specimens of the detected malware. This approach enables us to isolate the malicious code and submit it to behavioral analysis tools (such as sandboxes) that can determine the activities carried out by the malware. The execution of the malware in a simulated environment makes the analysis immune to the detrimental effects of polymorphism and metamorphism. By knowing in advance the malware behavior it is possible to deploy proactive countermeasures aiming to prevent malware infection and communication.

Improved analysis efficiency. Sandbox-based analysis is computationally expensive. Hence, in a distributed environment the collection of malware samples in a centralized analysis farm is convenient and efficient, because it prevents duplicate analysis of identical malware samples.

Moreover, we propose a runtime alert filtering scheme [28] that is able to automatically rank the security alerts produced by intrusion detection systems. The proposed scheme correlates at runtime the NIDS alerts with other information coming from multiple, heterogeneous and possibly unstructured (or semi-structured) data sources, such as vulnerability databases and information related to the software configuration of the protected hosts.

Effectiveness and scalability of the proposed architectures and of the new alert ranking technique have been demonstrated through experiments based on real prototytes and through at-scale simulations.

The remainder of this chapter is organized as follows.

Section 4.2 describes the first cooperative architecture for malware detection and analysis, based on a hierarchical design, and the proposed alert ranking scheme.

Section 4.3 presents the second cooperative architecture, based on a peer-to-peer infrastructure.

Finally, Section 4.4 compares our proposals with related works in the fields of cooperative security infrastructures and alert ranking.

4.2 Hierarchical architecture for IDS cooperation

In this section we present the structure and the main operations of a hierarchical architecture for intrusion detection and malware analysis. The proposed archi-

tecture has several innovative features. Unlike few existing collaborative systems that are mainly oriented to spam fighting [34, 35, 124], the proposed system is oriented to early malware detection, analysis and dissemination of analysis results. Its flexibility and scalability are intrinsic in the architecture design that is based on a decentralized communication scheme and a multi-tiered hierarchy of geographically distributed components. The proposed solution is general and allows cooperative networks to take advantage of other network security sensors, while requiring a very unobtrusive trust scheme.

Current initiatives which require the cooperation of many users to collect malware [76] represent an appreciable start, but most of the analysis work is manual and there is still a separation between anti-malware research and deployment of countermeasures. On the other hand, the proposed architecture does not need human intervention. Finally, it is worth to observe that we take advantage of honeypots [117] properties to share information between different organizations without raising privacy concerns.

The proposed architecture performs the following high-level functions:

- to receive security alerts and the malware specimens from the cooperative sensors (see Section 4.2.1);
- to aggregate the received security alerts and analyze malware specimens (see Sections 4.2.2 and 4.2.3);
- to provide useful feedback to all the participating networks, defined as all the networks hosting at least one cooperative sensor (see Section 4.2.4).

The scheme in Figure 4.1 shows the main components of the proposed multi-tier system: *sensors*, *managers*, *collector*, That are described in Sections 4.2.1, 4.2.2, and 4.2.3, respectively.

To authenticate the communication endpoints and to digitally sign all the information exchanged among the architecture components, sensors, managers and the collector use public key cryptography. This choice guarantees the traceability of communications, together with the certainty that only registered sensors and managers can communicate with higher level managers and with the collector. Public key cryptography also guarantees confidentiality to the communications.

4.2.1 Cooperative sensors and networks

A *cooperative sensor* is defined as an intrusion detection component that relays the generated alerts to a cooperative intrusion detection architecture. We define a *cooperative network* as a network in which at least a cooperative sensor is deployed. Theoretically, any machine connected to the Internet has the same chance

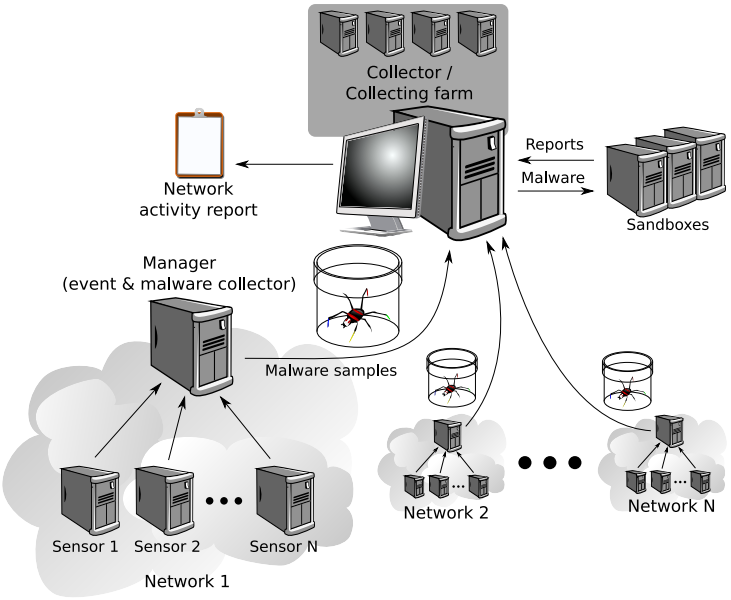


Figure 4.1: Hierarchical cooperative architecture for malware detection and analysis

of being targeted by a worm, however the presence of firewalls has the effect of slowing the infection because some protocols are blocked for inbound connections. By distributing IDS sensors and honeypots throughout a large number of heterogeneous and geographically distributed networks we can thoroughly monitor both malware spread and attack trends.

The current implementation of the proposed hierarchical architecture can leverage heterogeneous sensors.

- **Host IDS.** Intrusion detection systems monitoring activities on the same host in which they are deployed.

- **Network IDS.** Sensors able to capture all network traffic flowing on the monitored link(s) and analyze each packet looking for malicious content. They can be implemented through dedicated hardware appliances or installed on general purpose computer. When an illicit activity is detected, they generate an alert providing information on the related network packets and on the nature of the attack.

- **Operational IDS.** These instruments, also known as *honeypots*, are able to collect malware and to trace malicious activities. They consist in a vulnerable computer (real or emulated) connected to a network and waiting to be attacked. This allow the administrators of the honeypots to study attackers

strategies. Moreover, attackers waste time by attacking a bogus target.

In particular, low interaction honeypots such as *nepenthes* [79] are the best suited sensor type for our purpose, as they are not only able to detect an attack, but also to collect and store a copy of the malware payload through which worms compromise networked hosts.

4.2.2 Managers

Manager nodes represent the core of the proposed hierarchical architecture. Their role is to collect security alerts and malware payloads generated by the cooperative sensors, to perform in-network aggregations and correlations, and to relay relevant security alerts and unknown malware to the *collector*.

A manager installation is composed by two elements:

- a manager process handling NIDS, HIDS and honeypots security alerts;
- a polling agent which retrieves unknown malware payloads from the controlled sensors.

As Figure 4.2 shows, the manager nodes are connected in hierarchical fashion, and each manager relays unknown malware and alerts to the higher layer of the architecture. This topology maximizes the collection capability while keeping low the number of transferred payloads, because they are retrieved by a higher level manager only if they have not already been received. This solution guarantees the transmission of each new malware variant to the collector, while avoiding unnecessary retransmissions of already analyzed malware specimens.

The flexibility of the architecture is guaranteed by the possibility of having any number of intermediate manager levels. In such a way, small organizations can connect its cooperative sensor(s) directly to a remote manager, while complex organizations may have multiple levels of managers that are controlled locally and connected to one or more remote managers.

4.2.3 Collector

The *Collector* is the root element of the hierarchical architecture. Each cooperating network contributes to the collection of alerts and malware hosted by the collector. For each incoming malware sample, the collector runs an automated thorough analysis. Analysis results are stored and utilized to classify the malware and to generate network activity reports, sent to all the administrators of all the cooperative networks.

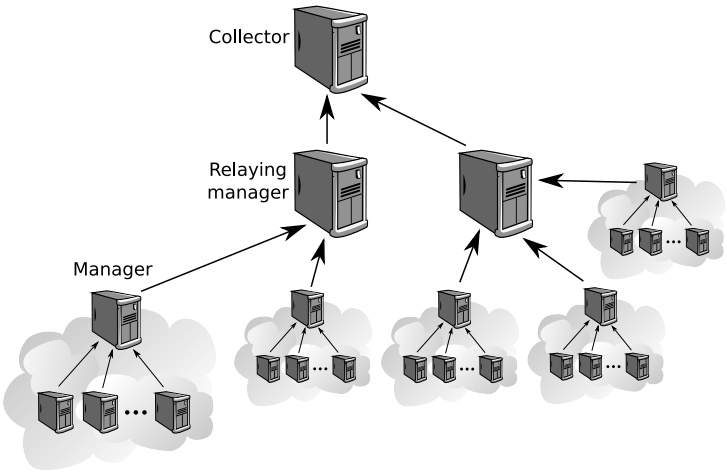


Figure 4.2: Hierarchical organization of managers

Upon collection, both intrusion detection alerts and malware payload samples are labeled using a cryptographic hash function (MD5, in the currently implemented prototypes).

The hash function is computed on relevant fields of intrusion detection alerts. Those fields include the *signature ID* (identifying which attack has been detected by the IDS sensor), IP source and destination addresses, and TCP source and destination port numbers. The hashes computed on different fields are used as indexes to efficiently cluster alerts characterized by similar features. Alert clustering represents the basis on which anomaly detection algorithms rely to identify anomalies in the alerts, such as a surge in a particular type of attack (typically a symptom of a malware trying to spread across several networks), or the presence of multiple and distributed attack sources targeting the same victim (typical of distributed denial of service attacks).

When a malware specimen is captured, the hash function is computed on the whole infective payload. Classification through hash functions makes it easy to determine whether or not a malware sample captured by a cooperative sensor has already been detected by other sensors participating to the cooperative architecture, thus preventing the unnecessary (and detrimental) overhead of repeated analysis of the same executable payload. Unknown malware samples undergo a two-step analysis.

They are first analyzed through several heterogeneous antivirus engines. The currently implemented prototypes relies on external analysis services, such as the one offered by VirusTotal [125]. The use of multiple independent engines ensures higher detection rates, and allows us to maintain a vendor-agnostic approach. While the behavior of most of the already-known malware payloads can be ex-

tracted from the antivirus analysis results, no useful information can be obtained related to malware that have not been already detected (0-day malware, or new variants of a known polymorphic or metamorphic malware).

If the analysis results generated by antivirus engines do not include a complete description of the malware activities, the malware payloads undergo *behavioral analysis* based on *sandboxing*. The analyzed malware is executed within a protected and tightly monitored environment (the *sandbox*) where its activities can be monitored. The currently implemented prototypes relies on two external sandboxing technologies: CWSandbox [33] and Norman SandBox [83]. While the behavioral analysis is not as precise as a signature analysis in identifying malware variants (two different worms may be characterized by similar behaviors), it represents the best way to collect information related to the structure of botnets.

The process of submitting the malware samples is handled by a custom modular software which can be easily extended to support other analysis services.

4.2.4 Network activity reports

The ability to generate and disseminate timely and detailed information on network-based attacks and malware spreading through the Internet represents the main purpose of the proposed architectures, as well as the strong incentive for network administrators to participate in a cooperative effort by installing one or more cooperative sensors.

A side effect of using multiple remote antivirus engines and sandboxes for malware analysis is that the corresponding results will be not homogeneous, often unstructured and not directly comparable. In order to allow the cooperating networks to take advantage of the analysis results, all the heterogeneous information has to be further processed and adapted, by compiling *Network activity reports*. These reports represent a structured summary of the relevant IDS alerts and of the activities performed by captured malware specimens, with main focus on the network-related activities, such as the endpoint of the connections related to the attacks and the amount and nature of transferred data. Actual examples of network activity reports generated by the proposed architecture are presented in Section 4.2.7.

Network activity reports are sent to all the cooperating networks, even to those not yet reached by the malware. In this way, all the participating parties receive the information needed for deploying defensive countermeasures, such as blocking certain network connections, closing ports or patching some software applications. The exchange of information between the cooperating networks is the most effective measure against the spread of malware. Furthermore, the knowledge about the infection vector may be shared very early with the vendor of the targeted software, which can start correcting the problem when only few machines

have already been attacked.

One of the most effective countermeasures that can be adopted against malware is the interruption of connections towards the malware distribution servers and the *Command and Control* servers. For example, if the command and control servers are identified by behavioral analysis, it is possible to put into place new firewalling rules that prevent any infected host behind the firewall to connect and download further instructions from the malware controller. It is also possible to write blacklists of known command and control servers (although some recent worms such as Storm use decentralized communication systems). We remark that, as a part of the adaptation phase, the information delivered through network activity reports are semantically tagged. Hence, they can be easily parsed by a machine and used to generate and deploy automatic countermeasures without human intervention, thus greatly reducing the time required to react to a network attack.

4.2.5 Alert ranking

The proposed scheme for prioritizing alerts [28] integrates at runtime the knowledge on vulnerabilities and local software configurations with intrusion alerts and network activity reports produced by our cooperative architecture. Among all the security alerts received by cooperating networks, only a few of them represent a real threat. By knowing the vulnerabilities that affect the machines of the protected network we can highlight the attacks exploiting these vulnerabilities by raising the priority of the related alerts. On the other hand, the priority of alerts related to attacks trying to exploit vulnerabilities that do not affect the monitored hosts can be safely lowered, and their analysis postponed.

The design of the proposed architecture is shown in Figure 4.3.

The complete list of applications that are vulnerable to a specific attack can be extracted from several vulnerability databases [1] that are freely available online. These sources are analyzed offline by a set of custom parsers, and the extracted information are used to build the *structured vulnerability information database*, ready to be used for alert filtering and prioritizing.

The *installed software database* is used to store and retrieve all the knowledge related to the software installed on the machines contained in the protected network. While (to the best of our knowledge) a single automated tool able to perform this task without human intervention does not exist, useful information can be gathered through package management software, network vulnerability

[1] Open Source Vulnerability Database (`http://osvdb.org/`), Common vulnerabilities and exposures (`http://cve.mitre.org/`), National Vulnerability Database (`http://nvd.nist.gov/nvd.cfm`), Packet Storm security advisories (`http://www.packetstormsecurity.org/alladvisories/`), SecurityFocus™vulnerabilities (`http://www.securityfocus.com/vulnerabilities/`)

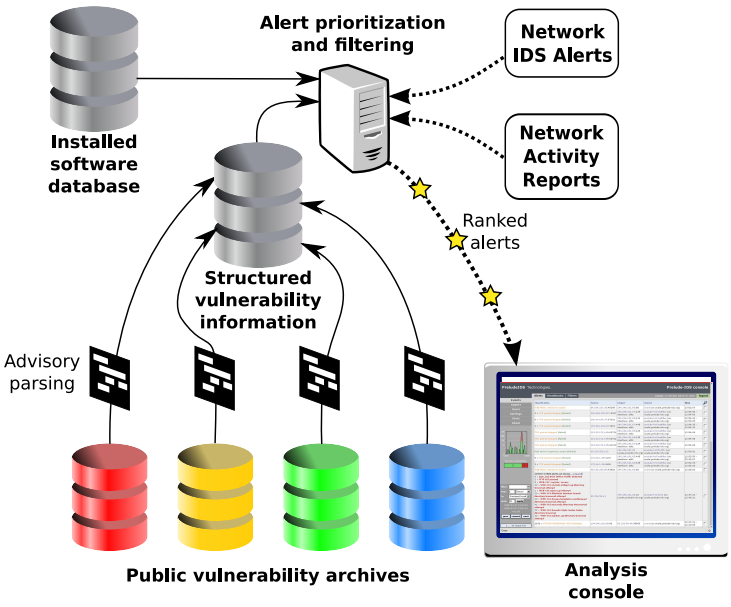


Figure 4.3: Architecture design

scanners, and network reconnaissance software able to perform application and operating system fingerprinting (such as *nmap* [82]).

The *alert filter* is the central element of the proposed architecture. Its role is to receive all the alerts and the network activity reports, to evaluate their level of threat for the protected machines and to rank each alert based on that treat level. Ranked alerts are then presented to the administrator for further analysis.

For each alert, the alert filter extracts:

- information related to the vulnerability that the attacker tried to exploit;
- the identity of the targeted host (typically, its IP address).

After having identified the vulnerability, the alert filter looks up all the vulnerable applications in the structured vulnerability information database. Concurrently, the identifier of the host targeted by the attack is used to retrieve a complete list of the installed applications. These information are compared in order to assess whether the attacked system runs one or more vulnerable applications. In the positive instance there is a high probability that the attacker succeeded in compromising the targeted machine, and the alert priority is increased. Otherwise, the attack failed, and the alert priority can be safely lowered. We remark that the computational cost related to the comparison among the two application lists (which usually contain only a few elements each) is very low, and comparable, if not

lower, to the computational complexity of the packet analysis performed by any modern intrusion detection system. Hence, all this operation can be performed at runtime with respect to the NIDS activities.

This alert ranking scheme does not depend on cooperation among heterogeneous networks, and it is generally applicable. However it is extremely useful to help a network administrator to manage the alerts and reports generated by a cooperative architecture gathering alerts from a high number of cooperative sensors.

4.2.6 Prototype implementation

The feasibility of the proposed architecture is demonstrated through a prototype based on open source software and custom integration scripts. The prototype has been validated experimentally in controlled conditions, by exposing it to known malware. After these preliminary tests, the prototype has been deployed in real networks by connecting its cooperative sensors to the Internet. The described prototype allowed us to collect and automatically analyze unknown malware samples (not detected by all the tested antivirus engines).

Cooperative sensors have been implemented through Snort [111] and nepenthes [79]. In particular, nepenthes is best suited for malware analysis, because it is able to capture real malware specimens. Nepenthes is a modular daemon that provides the following functions:

- socket binding and listening for incoming connections;
- identification of targeted vulnerability;
- analysis of the exploit code for the extraction of information necessary for downloading the worm payload;
- payload retrieval;
- logging and issuing alerts, potentially to a remote server through the IDMEF protocol.

Each of these functions is implemented in a separate module, that is identified by a descriptive prefix, such as *dnsresolve-*, *download-*, *module-*, *log-*, *shellcode-*, *shellemu-*, *sqlhandler-*, *submit-* and *vuln-*. For example, modules whose name starts with “vuln” contain the vulnerability simulation logic that is needed to reply correctly to attacks and to retrieve a specimen of the malicious payload.

Malware distribution can occur in many ways, and the honeypot software has to support as many methods as possible. Sometimes the shellcode opens a remote connection with a TCP or UDP stream from which it transfers subsequent commands; otherwise a TFTP, FTP or HTTP download is used to transfer the worm

payload. Nepenthes computes a SHA512 or MD5 hash of the malware and it stores a copy of the harmful binary. Such binaries can be moved to remote servers with different methods or be submitted[2] to organizations that search for new malware specimens, like the mwcollect Alliance [76], or Norman [83]. Nepenthes also provides a *libprelude* output module which can be used for issuing alerts to a *prelude manager* [96].

A critical element of the proposed distributed architecture is the communication infrastructure. We designed this part by taking into account the following requirements:

- forwarding of alerts and malware binaries from sensors to managers and between managers;
- transfer of unknown malware samples to the collector;
- authentication of all exchanged messages;
- confidentiality.

In addition to these functional requirements, it is essential for the format of the exchanged messages to be a recognized standard that allows for high interoperability between heterogeneous threat detection systems. To fulfill these requirement we choose to implement the alert management framework through the open source software *prelude* [96].

Prelude is an Open Source software that allows the deployment of a *hybrid* intrusion detection system — an aggregate of heterogeneous sensors employing different technologies and approaches to detect attacks. A typical use case is the integration of Host and Network IDS in large networks. The format of the exchanged messages is IDMEF [46]. Prelude offers a library (*libprelude*) that security-related software can use to issue alerts and communicate with Prelude managers. Communications are encrypted using public key cryptography and relaying of messages is supported by managers, so it becomes possible to build a hierarchical network of malware-collecting nodes.

A key requirement of the proposed architecture that is not supported by the hybrid IDS Prelude is the submission of the binaries downloaded by nepenthes to a remote collection node.

Our implementation is based on a script which is executed periodically on the collector and manager nodes. The script spawns a remote shell to each machine that is managed on the immediately lower level of the architecture, lists the

[2] submission is done though the GOTEK protocol or with custom solutions that are different for each collection service

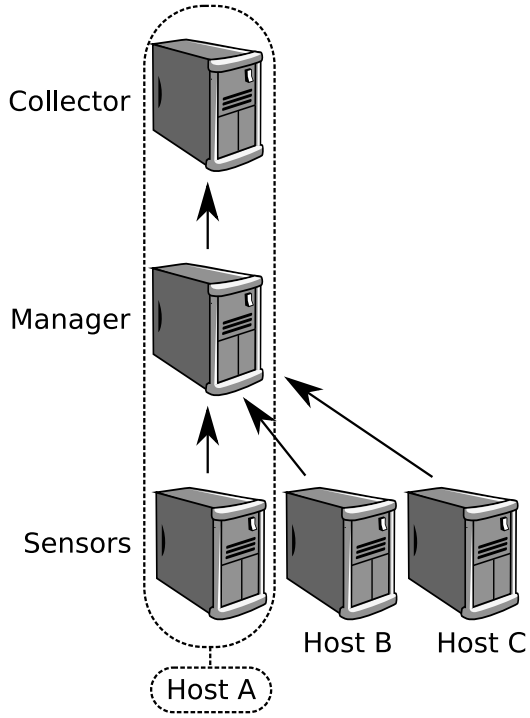


Figure 4.4: Experimental setup for the hierarchical architecture for malware detection and analysis

collected binaries and retrieves the unknown ones. For example, the collector examines the caches of the highest level managers, which in turn collect the malware binaries from their respective subordinate managers. The lowest level managers collect binaries from their pool of cooperative sensors.

4.2.7 Experimental results

All the functions of the proposed prototype have been experimentally. The test setup used in our experiments is represented in Figure 4.4.

A single host is being used as a sensor, manager and collector, and two sensors have been installed in other machines. The manager collects alerts and malware samples from three sensors, and the collector does the same on a single manager. Through this setup we can validate:

1. the collection of binaries performed by the manager;
2. the forwarding of alerts from the sensors up to the collector;
3. the analysis of binaries performed by the collector;

4. the generation of network activity reports and its dissemination to the cooperating networks.

A first test has been performed by exposing the sensors to known malware specimens in a controlled laboratory environment. Nepenthes manages to collect the binaries and correctly issues alerts that are forwarded by the manager to the collector. Accordingly, the binary payload of the malware is transferred from the sensors to the manager and then to the collector, which proceeds with the analysis, since the hash of the binary is unknown. The payload is sent to the Norman sandbox analysis tool, a network activity report is generated on the basis of the analysis result and sent to the administrators of the three simulated networks.

To verify the behavior of the system in a realistic and hostile setup, some nepenthes sensors have been exposed to the Internet by deploying them on machines connected through ADSL lines and having public IP addresses. This experiment led to the issuing of 3866 alerts and to the collection of 52 unique malicious binaries over a span of about eight hours. Seven of the collected binaries were previously unknown to our collector, and they were submitted to the available online scanning services. In many cases the behavioral analysis performed in CWSandbox has lead to the classification of malware as an Internet worm, and to the identification of several command and control hosts.

The following is an example network activity report produced by parsing the analysis result generated by the sandbox analysis service:

```
0995104827bee951abc4fcc93cdf85ee :
INFECTED with W32/Malware
(Signature: W32/Malware.LNH)
    * Connects to "j4m4lz.B3D3RPIERO.INFO"
      on port 6137 (TCP).
    * Connects to IRC Server.
    * Possible backdoor functionality
      [Authenticate] port 113.
Network Activity:
  Opened listening TCP connection on port: 113
    * C&C Server: 69.64.36.188:6137
    * Server Password:
```

The worm tries to connect to a command and control server and opens a backdoor on port 113.

```
13ff667bebcc58253faba2313dce7b89 :
INFECTED with W32/Kut.gen1
(Signature: W32/Poebot.ADT)
```

```
  * C&C Server: 140.116.199.57:8998
Network activity
  * Server Password: PING
```

In this case it has been possible to intercept the password used by the malware for *authenticating* to its command and control servers.

```
03fb1ecf2cbcfb74ab5c29dcd247e132 :
INFECTED with W32/Endom.A (Signature: Allaple.gen1)
    * Sends data stream (76 bytes) to remote
        address "124.86.6.4",
      port 139.
    * Connects to "124.86.6.4" on port 445 (TCP).
    * Sends data stream (76 bytes) to remote
        address "124.86.8.6",
      port 139.
    * Connects to "124.86.8.6" on port 445 (TCP).
      * Sends data stream (76 bytes) to remote
        address "124.86.10.8", port 139.
    * Connects to "124.86.10.8" on port 445 (TCP).
    * Connects to "124.86.6.4" on port 9988 (TCP).
       * Sends data stream (255 bytes) to remote
         address "124.86.6.4", port 9988.
```

This result demonstrates that the proposed solution allows us to detect and block malware communications even though they rely on a complex, multi-tier control network, as this worm does. Similar strategies make it difficult to block the malware communications only by inspecting network traffic anomalies, because of the multiple servers and the different TCP ports used by the malware. However, by examining the malware behavior we know at least its connection points to the command and control network, and we can prevent newly infected machines from joining the botnet.

4.3 Peer-to-peer architecture for IDS cooperation

The hierarchical design of the cooperative architecture presented in Section 4.2 is characterized by three main drawbacks:

- it cannot dynamically adapt to changes in network connectivity;
- the centralized collector represents a single point of failure, and the failure of one or more managers reduce the architecture ability to collect and analyze malware specimens;

- the hierarchical management architecture has to be configured manually.

In this section, we present a new collaborative architecture for intrusion detection, malware gathering and analysis. The main novelty of our proposal is in the peer-to-peer scheme used to organize all the sensors deployed among heterogeneous networks in a single distributed architecture. This design choice has several advantages.

First of all, the proposed peer-to-peer overlay network of sensors, based on Distributed Hash Tables (DHT), is characterized by the lack of single point of failures, that are inherent in centralized and hierarchical architectures. This characteristic represents a significant dependability improvement and allows us to achieve a more robust architecture, that is able to endure sudden failures of some of its nodes and/or network links without suffering a significant service disruption.

The proposed collaborative architecture provides better scalability than centralized and hierarchical designs. The scalability of all the architectures having a centralized analysis and management scheme is obviously limited by the capacity of the central manager to receive and correlate alerts generated by multiple sources. On the other hand, the proposed architecture inherits both the high scalability and self-organizing properties of DHT overlays, thus enabling collaborations among large and dynamic networks of sensors.

To achieve high scalability, fault tolerance, dependability and self-organization capabilities, the proposed architecture relies on a completely distributed overlay of collaborative alert aggregator. Each element is responsible for collecting relevant alerts and malware samples, and to make them available to the other nodes of the collaborative overlay.

A representation of the logical components is provided in Figure 4.5. Each collaborative element is composed by three different layers: the *sensor* layer, the *local aggregation* layer and the *collaboration* layer, that are described in Sections 4.3.2, 4.3.3 and 4.3.4, respectively.

4.3.1 Sensors layer

The proposed peer-to-peer architecture can be used to enable cooperation among heterogeneous and distributed sensors. The current implementation of the proposed architecture can rely on the sensors already described in Section 4.2.1. Cooperative sensors can be distributed within the monitored network, and configured to relay their alerts and the captured malware specimens to the local alert collector.

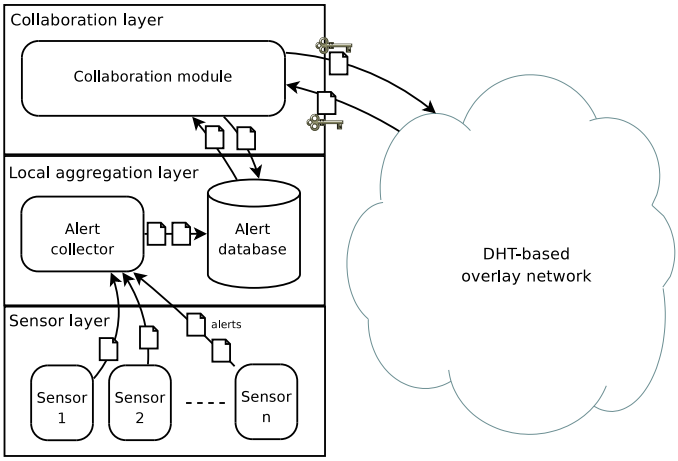


Figure 4.5: Design of a collaborative alert aggregator

4.3.2 Local aggregation layer

The local aggregation layer is responsible for collecting, filtering and aggregating all the heterogeneous alerts received from the cooperative sensors (as defined in Section 4.2.1) of the lower layer. Within this layer, the *alert collector* receives all the heterogeneous alerts and performs preliminary analysis and correlation operations. While it is possible for the alert collector to execute arbitrarily complex aggregation and correlation algorithms, its fundamental task is to pre-process all the received alerts, that are possibly syntactically and semantically heterogeneous. All these alerts are classified and stored in the local *alert database*. The database is used by the upper layer as the only mean to store and retrieve alerts, and provides a well defined and sensor-independent interface.

4.3.3 Collaboration layer

The *collaboration layer* represents the only element which is connected to the DHT-based overlay network. Being part of the overlay, each *collaboration module* is provided with a unique node identifier $nodeID$. The collaboration module has three main purposes.

It is responsible for retrieving the new alerts that have been stored in the local alert database. These alerts are submitted to the DHT-based overlay network using a key computed over significant fields of the alert. As an example, the key used to submit a malware specimen can be computed by applying a hash function to the malware itself, while an NIDS alert can be submitted to the overlay network twice, using as keys the signature ID and the IP address from which the illicit network packet originated.

Moreover, the collaboration module receives messages submitted by other modules connected to the same overlay network. In particular, each of them is responsible for a portion of the hash space (determined by the $nodeID$ and by the implementation details of the DHT algorithm) and receives all the messages whose keys fall within that hash space. This design choice allows each collaboration module to receive all the messages that are relevant to a specific scenario.

As an example, to detect a surge in self-replicating malware, it is possible to configure collaboration modules to submit malware samples to the overlay network using the hash of the malicious code as a key. As a result, the collaboration module that is responsible for the hash space comprising the hash of the malware will receive all the related messages, and will be able to perform meaningful statistical aggregations on the malware replication rate. Moreover, given that all the identical malware samples are sent to the same collaborative alert aggregator, each malware specimen will be analyzed only once, independently of the number of sensors that will detect it.

Similarly, an attacker targeting hosts belonging to different domains can be easily pinpointed by submitting IDS alerts using the hash of the illicit packet source address as key. This allows a single collaboration module to receive all the alerts caused by the same source, thus realizing an effective network-based and distributed alert aggregation and correlation scheme.

Finally, each collaboration module disseminates the analysis results to the other nodes connected to the overlay network. Timely dissemination provides useful information and alerts to all the collaborative nodes and represent a strong incentive to participate to the collaborative architecture.

4.3.4 Main benefits of the peer-to-peer architecture

The distributed nature of the proposed peer-to-peer architecture for IDS cooperation is characterized by important advantages over existing architectures. In this section we analyze four main strength points:

- fault tolerance;
- load balancing;
- scalability;
- storage efficiency.

Fault tolerance

The completely distributed nature of the proposed architecture is inherently fault tolerant, because it lacks single points of failure that are typical of hierarchical and

centralized architectures. Multi-tiered systems concentrate alert aggregation, correlation and analysis functions in the root node of the architecture [27]. According to this paradigm, the root node represents a single point of failure, meaning that the complete hierarchical architecture will not be able to perform any useful task in case the root node becomes unreachable.

Moreover, the effectiveness of hierarchical architectures can be severely impaired even by failures of the nodes belonging to intermediate layers of the tree. As an example, a failure of one of the tier-1 nodes will result in the isolation of the complete sub-tree having the faulty node as root.

On the other hand, the proposed peer-to-peer architecture leverages a completely distributed, network-driven aggregation and correlation technique. Each node is responsible for aggregating and correlating only a small subset of alerts and malware samples. If a node becomes suddenly unreachable, only the messages that would have been handled by the faulty node are lost, while all the other element of the distributed architecture are not influenced by the failure. Moreover, depending on the implementation details of the DHT routing scheme used to realize the overlay network, the collaborative nodes will eventually detect the failure of a peer, and autonomously modify their local overlay routing tables accordingly. Hence, the proposed architecture is able to autonomously reorganize itself, restoring full efficiency without requiring human intervention.

Message replication schemes can also be used to reduce the (minimal and transitory) message loss due to the failure of a collaborative node. In the current implementation of the proposed architecture it is possible to set a configurable *replication constant* k. For each alert, the collaborative alert aggregators will issue k messages. One of these messages is sent to the collaborative alert aggregator whose unique identifier $nodeID$ is responsible for the message key. The other $k-1$ messages are sent to the $k-1$ nearest neighbours, thus guaranteeing full reliability for up to $k-1$ failures at the same time. An experimental evaluation of message loss probability for higher number of contemporary faults is presented in Section 4.3.6. By tuning the value of k it is possible to achieve the desired trade-off between load and fault tolerance.

Load balancing

The second major advantage of the proposed DHT-based distributed architecture is represented by its inherent load balancing properties. Let us consider a realistic scenario in which a network participating to a collaborative hierarchical IDS architecture is targeted by an attacker, while the other participating networks are not. This scenario is represented in Figure 4.6. In a similar situation, the load related to alert management and correlation is unevenly distributed. Indeed, only the managers connecting the leaf of the tree representing sensors in the attacked

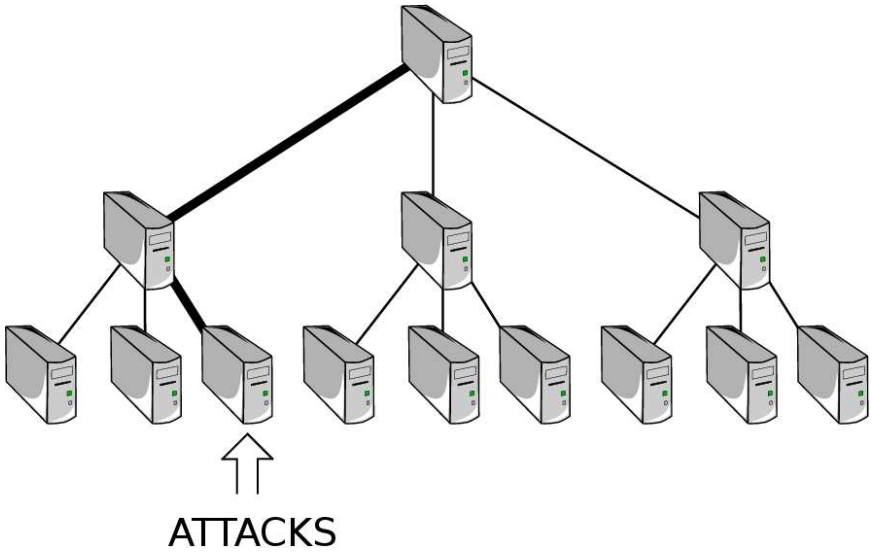


Figure 4.6: Load unbalancing in hierarchical cooperative architectures

network to the root manager are involved in alert management. By attacking only networks that relay alerts to the same higher-level manager, it is possible for an attacker to overload the path connecting the attacked networks to the hierarchy root.

Moreover, hierarchical architectures concentrate malware analysis and alert correlation tasks on the root manager to avoid replicated and useless analysis. Hence the computational load on this component is significantly higher than the load of the intermediate managers.

Uneven load distribution is effectively mitigated by the proposed distributed alert aggregation and correlation approach. Lacking a single, centralized aggregator to which all the alerts have to be transmitted, there is no single path through which all the alerts generated by a sensor (or a set of sensors in the same network) are transmitted. As represented in Figure 4.7, alerts gathered by the collaborative alert collector in an hypothetical attacked networks are routed to multiple different collaborative nodes, based on the key characterizing each alert.

Scalability

Hierarchical IDS architectures are based on a multi-tier management infrastructure connecting the lowest layer alert managers (the leaves of the management tree) to the root alert collector. Each alert manager is characterized by limited computational power and bandwidth, hence it will able to collect, aggregate and relay alerts generated by a finite number n of sensors and managers. It is then

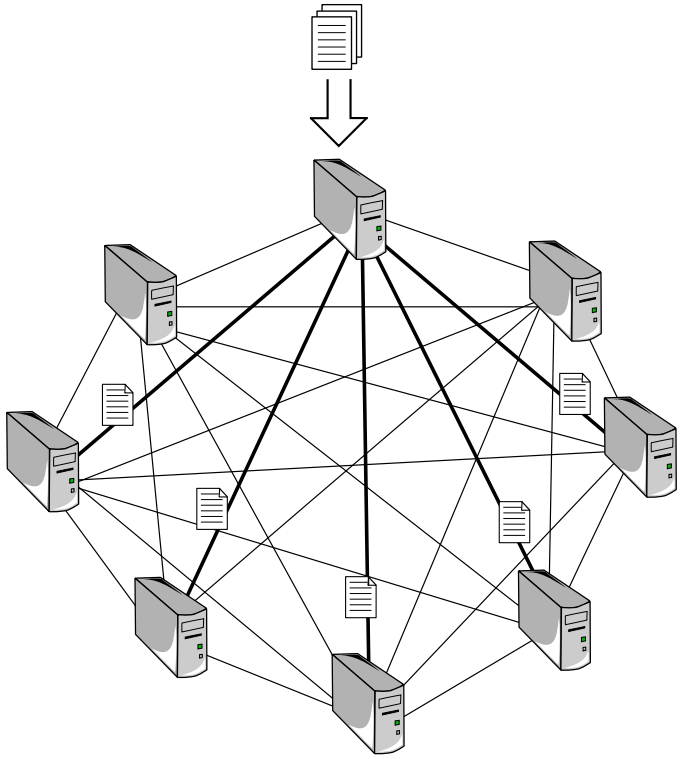


Figure 4.7: Load balancing in peer-to-peer cooperative architectures

possible to describe a generic hierarchical architecture for intrusion detection as an n-ary tree, whose number of intermediate elements grows logarithmically with the number of the leaves.

On the other hand, in the proposed design all the alert management operations are distributed among the leaves, and there is no need for a separate management infrastructure. This is a big advantage in terms of scalability and architecture management. Indeed, it is not necessary to manually reconfigure the architecture hierarchy and to add new layers to the management tree whenever the number of leaves increases.

Storage efficiency

Another advantage of the peer-to-peer architecture is the smaller number of stored alerts and malware specimens. In a hierarchical architecture, a copy of each different alert and malware specimen is maintained in each node of the management tree by which the alert has been received. Let us consider a tree of managers having order n, l leaves and height $h = \log_n(l)$. If c represents the number of copies of each different alert stored by all the nodes belonging to the hierarchical architecture, then

$$h \leq c \leq \sum_{i=0}^{h-1} n^i$$

This means that the number of copies c is comprised between the number of managers in the path between the single leaf generating the alert and the root of the tree (h) and the total number of managers, when the same alert has been issued by all the leaves in the tree ($\sum_{i=0}^{h-1} n^i$). As h grows with the logarithm of the number of leaves, then the number of copies of each different alert (and malware specimen) that a hierarchical architecture needs to maintain also grows logarithmically with the number of leaves.

On the other hand, in the peer-to-peer architecture the number of copies of each different alert and malware specimen is determined by the replication factor k, which is a configurable parameter independent of the number of cooperative nodes connected to the overlay.

A comparison between the number of copies stored by the proposed architecture and examples of hierarchical architectures is presented in Table 4.1.

The first row of the table represents the number of copies stored in the proposed architecture with a replication factor $k = 5$. The number of copies does not depend on any other parameter. The following rows contain the number of copies stored in hierarchical architectures characterised by different number of nodes and order. As an example, the second row shows that the number of copies of the same alert in an hierarchical architecture with $l = 1000$ nodes and order

Type of Network	Minimum	Maximum
DHT overlay, $k = 5$	5	5
Hierarchical, $l = 1000$, $n = 10$	3	111
Hierarchical, $l = 10000$, $n = 10$	4	1111
Hierarchical, $l = 100000$, $n = 10$	5	11111
Hierarchical, $l = 8000$, $n = 20$	3	421
Hierarchical, $l = 160000$, $n = 20$	4	8421

Table 4.1: Comparison of the number of malware copies stored in hierarchical and peer-to-peer cooperative architectures

$n = 10$ is comprised between 3 and 111, depending on how many leaves issued the same alert.

4.3.5 Prototype realization

Viability of the proposed architecture has been demonstrated through a prototype. In compliance with the architecture description provided in Section 4.3 each cooperative node comprises at least four different software modules:

- one or more sensors;
- alert collector;
- alert database;
- collaboration module.

The current implementation of the collaborative alert aggregator can rely upon heterogeneous sensors, thus being able to detect a wide range of threats. In particular, we use Snort [111] (standard de-facto for signature based network intrusion detection) as NIDS sensor, and nepenthes [79] as low-interaction honeypot. All the cooperative sensors described in Section 4.2.1 can be used as alert and malware sources.

The local alert collector has been implemented through Prelude [96], that is an Open Source hybrid IDS. We made extensive use of the features provided by the Prelude manager software module. All the communications between the prelude manager and the sensors are based on the Intrusion Detection Message Exchange Format (IDMEF) [46], which is a widely used standard for the generation and management of messages related to security incidents, made by IETF. IDMEF output is supported (either natively or through plug-ins) by a wide variety of network, host and operational IDS, hence our prototype can be easily modified to interact with different sensors.

The local alert collector is configured to store all the collected alerts and malware samples in the local alert database, implemented with MySQL [77].

The collaboration module has been implemented in Java, thus obtaining a platform independent application. The management of the DHT-based overlay network relies on the FreePastry libraries [42], a Java implementation of Pastry network [15–17, 65, 104]. These libraries, in addition to managing the overlay network, provide an useful emulation environment.

We modified PAST [37, 102], that is a software for persistent data storage in peer-to-peer networks, to implement the alerts and malware specimens storage system. PAST is based on Pastry for message routing and for network management. Its main features are the balancing of storage space between all the nodes and good fault tolerance.

Multicast communication among cooperative nodes, required for the dissemination of the analysis results, is realized through a modified version of Scribe [18, 103].

The collaboration module can be configured by editing an XML [131] file. The interface with the local alert database is implemented through JDBC drivers [51], thus guaranteeing high levels of interoperability with all the common DBMS.

Alerts are read from the database, classified according to their nature, and managed by different Java classes. In the current version of the prototype we have implemented three different classes, for the management of three heterogeneous types of alerts: malware samples, information related to attempted hostile connections to a honeypot, and alerts generated by a NIDS sensor. The modular implementation of the collaboration module allows us to easily extend its functions by implementing new classes for the management of other alert types.

Once a message is received, the collaboration module is responsible for its storage, its analysis, and the dissemination of the related network activity reports to the other cooperative nodes (as described in Section 4.2.4). It is also possible for the cooperative nodes to subscribe to specific areas of interest, thus receiving only the relevant analysis results and information.

4.3.6 Validation and performance evaluation

Both viability and performance of the proposed solution have been demonstrated through extensive experiments and simulations, carried out through the prototype implementation described in Section 4.3.5. Small scale experiments, including only a few tens of nodes, have been realized by executing several instances of the collaboration module in the same host and by binding them on different port numbers. Those experiments allowed us to validate our implementation.

To carry out tests comprising a higher number of nodes we modified our prototype to leverage the network emulation environment provided by the FreePastry

libraries. That solution allowed us to launch up to a thousand nodes on the same Java Virtual Machine.

Finally, en experiment involving up to ten thousand nodes have been carried out through ad-hoc simulators, focusing on the high-level behavior of hierarchical and DHT-based architectures. In this experiment we omit all the low level details related to the actual transmission of messages over wide-area networks and of their storage within the local alert database. Although simplified, the simulators used to perform these experiments use the same routing schemes of the real prototype. Moreover, their correctness has been assessed by executing the same experiments with the complete prototype (using only the FreePastry network simulation) and the simplified simulators, and verifying the complete agreement of the results.

Dependability and Fault tolerance

The completely distributed nature of the proposed architecture is inherently fault tolerant, and lacks single points of failure that characterize hierarchical architectures. While the failure of only a few nodes does not impair the architecture dependability, it is still possible to lose some alerts sent to a faulty node.

To minimize the risks of losing alerts, the proposed architecture relies on the message replication scheme described in Section 4.3.4. It is possible to configure a replication factor k, so that each message is sent to the k collaborative alert aggregators whose $nodeID$ is nearest to the message key. Moreover, the PAST layer that handles message storage runs periodic checks (at configurable time intervals) to verify the existence of all the k copies of each message, and replicates the missing copies in case some of the k nodes responsible for a message are no longer reachable. In this case a message can be lost only if all the k nodes that received the alert fail, before PAST restores the correct number of copies.

The ability of our architecture to sustain faults and node churn has been demonstrated through several experiments. In each run we simulate an overlay network comprising a variable number of collaborative nodes (from 1000 to 10,000), and we randomly generate a set of messages. Once granted that each collaborative node is responsible for at least one message, we simulate the concurrent failure of a given percentage of collaborative alert aggregators, ranging from 0.5% to 10% of the nodes. Finally, we waite for PAST to run a scheduled update, thus restoring k copies of each message that has not been lost due to the concurrent node failures, and we check if all of the messages injected at the beginning of the simulation are still available. Simulation results are presented in Tables 4.2 and 4.3.

In Table 4.2 we compare networks of different size (between 1000 and 5000 nodes) using a replica factor $k = 5$ (5 copies of each message). The number of nodes killed is expressed as a percentage of the total node number. For each

Concurrent fault rate (%)	1000 nodes	2000 nodes	5000 nodes
1	0	0	0
2	0	0	0
3	0	0	0,01
4	0,01	0,02	0,04
5	0,02	0,04	0,12
6	0,06	0,12	0,24
7	0,11	0,22	0,6
8	0,19	0,42	1,04
9	0,35	0,76	1,8
10	0,58	1,16	2,99

Table 4.2: Message loss probability for $k = 5$ and a variable number of nodes

failure rate and for each network size we run the simulation 100,000 times. The message loss rate is expressed as percentage of the 100,000 runs in which *at least* one message has been lost. As an example, the cell at the fourth row and second column in table 4.2 denotes that a network of 1000 collaborative nodes with 40 concurrent failures (4% of the number of nodes) lost one or more messages in only 0.01% of the 100,000 runs.

In Table 4.3 we report the results of simulations aiming to assess how the replica factor k influences the probability of losing a message. In this series of simulation we used a constant network size of 10,000 collaborative nodes, and we varied both the concurrent failure rate (expressed as percentage of the number of nodes) and the replica factor (with values of k ranging from 4 to 6). As in the previous case, for each combination of fault rate and replica factor we run 100,000 simulations, and we measure the packet loss probability as the percentage of simulations in which at least one message have been lost.

As an example, the cell at the fourth row and third column denotes that using a network with 10,000 cooperative nodes, replication factor $k = 5$ and the concurrent failure of 200 nodes, one or more messages are lost in only 0,003% of the 100,000 tests. These experiments demonstrate that by using appropriate values of the replica factor k it is possible to achieve negligible message loss probability even for large networks and concurrent failures of hundreds of nodes.

Load balancing and Scalability

The set of experiments presented in this section aims to demonstrate the load balancing properties of the proposed architecture. In particular, we compare the load of the collaborative alert aggregators connected through a DHT-based over-

Concurrent fault rate (%)	k=4	k=5	k=6
0,5	0,002	0	0
1	0,009	0	0
1,5	0,029	0	0
2	0,16	0,003	0
2,5	0,359	0,006	0
3	0,735	0,019	0,001
3,5	1,272	0,036	0
4	2,117	0,075	0,002
4,5	3,392	0,161	0,003
5	5,022	0,219	0,015
5,5	7,027	0,363	0,017
6	9,732	0,542	0,037
6,5	12,888	0,872	0,064
7	16,64	1,186	0,081
7,5	20,886	1,591	0,108
8	25,859	2,172	0,159
8,5	31,107	2,844	0,224
9	36,682	3,774	0,315
9,5	43,234	4,571	0,406
10	48,685	5,904	0,529

Table 4.3: Message loss probability for a network of 10,000 nodes and different values of k

lay network with respect to the load of the lowest layer of alert manager in the hierarchical IDS architecture presented in [27].

Experiments are carried out by simulating a network of 5,000 collaborative nodes, and a hierarchical architecture with the same number of leaves. Each intermediate manager belonging to the management infrastructure of the hierarchical network is connected to a random number of elements in the lower level, randomly chosen between 5 and 20. The resulting tree has 4 layers and 420 low-level managers, directly connected to leaf sensors. Figures 4.8, 4.9 and 4.10 compare the load distribution among the 5,000 collaborative alert aggregators in the DHT-based overlay and the load distribution among the 420 low-level managers of the hierarchical architecture.

Figure 4.8 represents the best-case scenario for the hierarchical architecture, in which all the 500,000 messages (100 message for each leaf, on average) are uniformly distributed among the leaves. The uniform distribution among all the leaves emulates a scenario in which network attacks are uniformly distributed among all the sensors, that generates alerts at exactly the same rate. The two solid lines of Figure 4.8 represent the cumulative distribution function (CDF) of the number of messages that each node in the collaborative architecture (line labelled *P2P*) and each low-level manager in the hierarchical architecture (line labelled *Hierarchical*) have to manage. The x-axis represents the ratio between the number of messages actually handled by a node and the expected average (i.e. $500,000/5000 = 100$ messages per collaborative node in the collaborative architecture, and $500,000/420 = 1190.48$ messages per manager in the hierarchical architecture). The dotted line represents the ideal load distribution, in which 100% of the nodes handle a number of messages that is equal to the expected average (hence the ratio between the handled messages and the expected ratio is 1).

As shown in Figure 4.8, for a uniform alert distribution the load of the hierarchical architecture is better balanced than the load of the collaborative overlay. Indeed, all the nodes in the hierarchical architecture handle a number of messages comprised between 40% and 150% of the expected average. However, even in the best-case scenario for the hierarchical architecture it is possible to see that the load distribution of the collaborative architecture is comparable. 20% of the nodes in the collaborative architecture are more loaded than the most loaded nodes in the hierarchical set-up, but the highest load is limited to 3,2 times the expected load.

A more realistic scenario is shown in Figure 4.9, in which the 500,000 alerts are not generated by all the sensors uniformly, but they follow a heavy-tailed Zipf distribution[3]. The two CDFs in this graph clearly shows the benefits of the hash-

[3]The used formula is $f(P_i) = \frac{c}{i}$, where i is the rank, P_i indicates the event which occupies the i-th rank (the i-th most frequent), $f(P_i)$ is the number of times (frequency) that the P_i event occurs, c is a constant equivalent to P_i

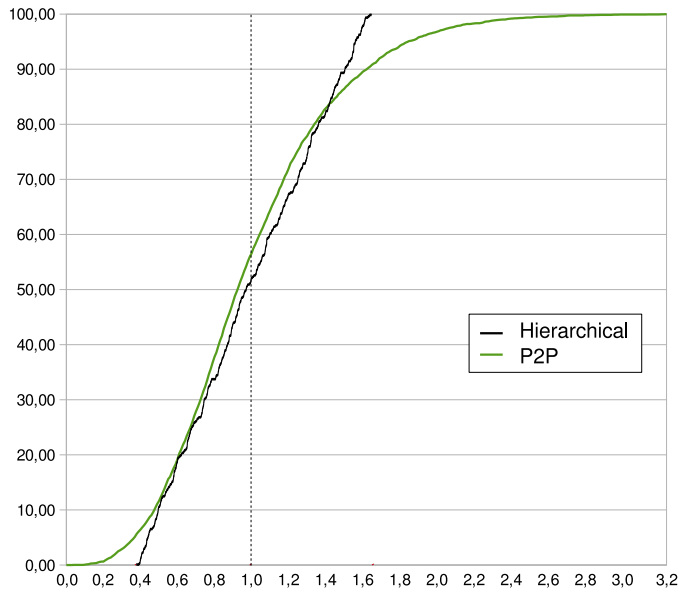


Figure 4.8: CDF of the number of messages managed by nodes of the hierarchical and peer-to-peer cooperative architecture, attacks are received by sensors with uniform distribution

based alert distribution algorithm implemented by the DHT overlay network. The node responsible for managing an alert is determined by the alert content, and not by the sensors that generates it. Hence the load distribution of the proposed architecture in this scenario is identical to the one presented in Figure 4.8. On the other hand, the load distribution in the hierarchical architectures is highly unbalanced: the most loaded manager handles a number of messages equal to 103 times the expected average.

Finally, Figure 4.10 represents the attack scenario in which all the alerts are generated by a single sensor (or by several sensors, connected to the same low-level manager). This is the worst case for a hierarchical architecture, because all the alerts have to be handled by a single manager, while the other managers are completely idle. The resulting DFS is heavily skewed, because a single manager sustains a load 420 times higher than the expected average. On the other hand, the proposed collaborative architecture has the same load distribution already shown in the two previous scenarios.

4.4 Related work

The two cooperative architectures for intrusion detection and malware analysis, consisting of geographically distributed sensors represents in itself two original

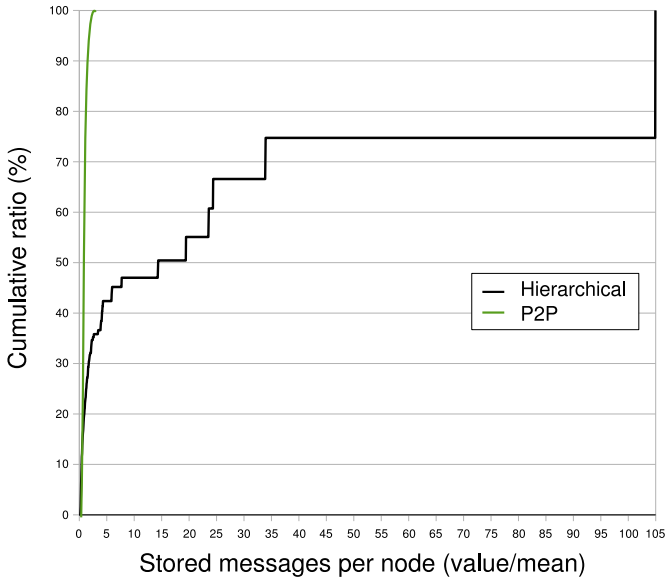


Figure 4.9: CDF of the number of messages managed by nodes of the hierarchical and peer-to-peer cooperative architecture, heavy tailed attack distribution

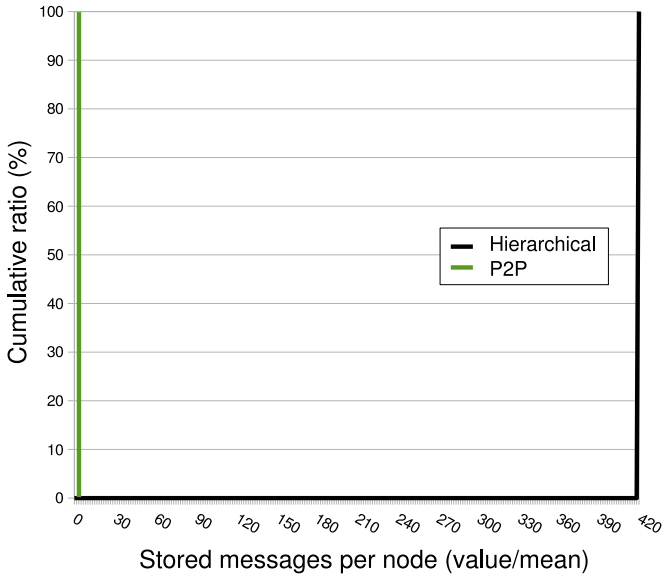


Figure 4.10: CDF of the number of messages managed by nodes of the hierarchical and peer-to-peer cooperative architecture, attacks are received by a single sensor

proposals. However, other interesting cross-organization security initiatives exist. We can cite the DNS black lists (DNSBL) [52], Botnet investigation [43,107], IDS alert correlation frameworks [48, 122], partial information disclosure [133]. DNS black lists (DNSBL) are one of the existing automated facilities for limiting the activities of suspicious hosts, but they are still highly focused on a single service (email). Our approach is general and avoids blacklists, since many home Internet connections have dynamic IP addresses and the inclusion of such addresses would be detrimental to the efficacy of the blacklist and to the user experience.

Authors of [135] and [67] already proposed peer-to-peer architectures in the field of intrusion detection. In particular, [135] allows peers to exchange information related to IP addresses that each peer considers suspicious. By integrating several lists of suspicious addresses their system is able to pinpoint sources that are malicious with high probability. These addresses are used to build a collaborative blacklist. On the other hand, [67] uses a DHT overlay to make peers exchange snapshots of their internal behavior, consisting in traces of the executed system calls. By correlating several traces it is possible to pinpoint suspicious sequences that are executed by many peers in a short time frame, and that can be caused by the propagation of a malware. The peer-to-peer architecture proposed in this thesis clearly differentiates from the existing works, since we aim to capture and analyze real malware specimens, instead of just recognize that a malware is spreading. Moreover, by compiling network activity reports on the basis of behavioral analysis of the captured payload, we are able to provide cooperating network with a complete description of the malware behavior, that provides much more useful information than a blacklist of IP addresses.

Investigation of botnets is still highly manual. The existing efforts are more oriented to help law enforcement agencies, but not so much to limit malware spread. Instead, we aim mainly to counter worm infections and botnet activities by alloeing network administrator to receive timely and preventive alerts.

Malware collection organizations, such as the *mwcollect Alliance* [76], focus on grasping the dynamics of infection spreading and detecting new malware types and variants. This operation is carried out mainly through manual analysis of binary code and extraction of call graphs. We seek to obtain a substantial reduction of the human interaction required for identifying new malware.

The worst problem that administrators have to face while dealing with the alerts generated by a NIDS is represented by the *false positive* rate (the impact of false positives on Intrusion Detection Systems reliability has been analyzed in [7]). By false positive we mean all network activities that are licit or harmless to the protected systems, but that have been erroneously signaled by the NIDS as a security threat. We should also consider that several techniques can be utilized by skilled attackers to cover up the attack traces by forcing a NIDS to generate storms of irrelevant alerts [75,86]. If the NIDS is working properly, the real attack

is likely identified and signaled, but important alerts are hidden among several thousands of other irrelevant and misleading alerts. This kind of diversion is very effective because it prevents a timely deployment of proper countermeasures since the administrator is overwhelmed by the large amount of alerts.

The issue of intrusion detection alert management has been extensively addressed in literature. NIDS *alert fusion* techniques [61] help network administrators by aggregating several homogeneous alerts. However they suffer several drawbacks that are solved by our proposal. First of all, aggregation itself is not useful to assess the real alert dangerousness. Moreover, alert fusion algorithms rely only on intrusion detection alerts, and do not leverage any external information source. The authors of [74] propose a formal aggregation scheme able to correlate alerts exploiting the same vulnerability. To this purpose, they use vulnerability information, but the proposal is just a more sophisticated fusion algorithm that does not assess the threat level of an alert.

The process of automatically ranking vulnerabilities on the basis of a computed threat level has been defined as *alert verification* in [122] and [56]. Following the taxonomy proposed in previous works, our proposal can be classified as a passive alert verification scheme. This means that our alert verification is carried out without an active interaction with the system(s) targeted by an attack. This characteristic is indeed desirable, and allows our system to verify alerts in real time, without imposing load on the monitored systems and without relying on results generated by a possibly compromised machine. In [122] and [56], authors propose an alert verification scheme based on a single, structured vulnerability database. The dependency on a single information source is clearly identified as a drawback in [98], due to possible incompleteness, lack of timely updates, and lack of trust in the single information provider.

The main strength of our approach, that differentiates the proposed alert verification scheme from all the previous works, is the ability to leverage a vast amount of vulnerability data, built upon multiple, heterogeneous and unstructured information sources. This critical task is carried out by parsing and semantically tagging unstructured information, with techniques borrowed by the data mining field. Data mining techniques have already been applied in the network intrusion detection context [58, 95]. However, they have only been used for attack detection, while intrusion alert management is not considered. Other attempts at identifying complex attack scenarios to help analysis have been discussed in [81]. However, that approach is suitable for offline analysis, while the solution proposed in this thesis is able to perform runtime threat evaluation, thus allowing for a timely deployment of proper countermeasures.

Chapter 5

Conclusions

In this thesis we present several distributed architectures and algorithms that aim to address some security issues of modern network-based systems. We consider three main problems that may affect present and future systems:

1. large capacity networks will make it difficult to perform stateful traffic analysis;
2. node mobility will cause novel security breaches;
3. self propagating malware will generate massive attacks, targeting networks and systems distributed across the Internet.

For each problem we present novel architectural and algorithmic solutions based on the cooperation among distributed components.

To address the first problem, we propose ParNIDS [69], a new parallel architecture for traffic analysis that represents a significant improvement with respect to previous works, because it guarantees dynamic load balancing [5] while preserving a stateful and reliable traffic analysis. To this purpose, we design and validate a novel state migration [26, 29] mechanism that allows traffic analyzers to exchange their internal state information. To the best of our knowledge, this is the first framework that permits the dynamic migration of even complex state information among cooperative NIDS, thus achieving higher accuracy and maximum flexibility in the packet analysis. Thanks to this framework, it is possible to detect fragmented attacks even if none of the deployed NIDS sensors receives the complete attack signature. The proposed scheme can be applied in parallel NIDS architectures where multiple sensors cooperate on the same network segment for reliability and/or performance purposes, in wireless mobile networks, and in all other distributed systems in which network topology and packet routing may dynamically evolve. The advanced features of the proposed scheme are integrated into Snort, which is the most popular signature based NIDS. A large set of

experiments validates the framework functionality and demonstrates that the complexity of state migration is fully compatible with the performance requirements of runtime network traffic analysis.

Traffic distribution and state migration are driven by an innovative load balancing mechanism, that guarantees stateful analysis even on migrated traffic. We also address the issues related to load balancing activations in a context of highly variable traffic characterized by heavy-tailed distributions. A large set of experimental results validates the main functions of the proposed system and demonstrates the scalability of the architecture, as well as the performance of the load balancing and state migration framework. These properties come together with low costs and high flexibility that is guaranteed by a complete software implementation.

We then consider two security issues that are related to node mobility: the inability of traditional NIDS to perform stateful analysis on mobile connections and the incompatibility between *triangular routing* (default routing scheme of Mobile IP) and *egress filtering* best practices.

We initially highlight a new attack strategy, called *mobility-based evasion*, that can be used by an attacker to perform "stealth" attacks that evade detection from state of the art NIDS. The effectiveness of mobility-based evasion is demonstrated through several scenarios based on realistic network topologies.

We then propose a new NIDS cooperation scheme that for the first time enables distributed NIDS to perform stateful analysis on mobile connections, and to defeat mobility-based evasion techniques.

A third contribution in the area of node mobility is represented by a novel dynamic packet filtering approach, called *secure triangular routing* (STR) [68], that is able to solve the trade-off between triangular routing and egress filtering. Secure triangular routing is designed to provide same performance as that of *triangular routing* without the need of relaxing security policies in Mobile IP-enabled networks. The proposed solution does not require any modification in correspondent nodes or in their networks, and it fully complies with the Mobile IP protocol specifications.

Massive attacks generated by self-spreading malware represent a critical and growing threat to the security of network-based systems. To mitigate their effects, we propose to enable cooperation among distributed networks, aiming for early detection of new threats and allowing cooperating networks to deploy *preventive* countermeasures. While a similar principle is already used in cooperative tools for fighting spam, this is the first proposal of architectures able to automatically and efficiently collect and analyze malware specimens and to disseminate behavioral analysis results to all the participating networks. We present two cooperative architectures for malware detection and analysis.

The first architecture [27] is based on a hierarchical design, and aims to improve analysis efficiency by preventing duplicate analysis of the same threat, even

though detected in different networks. It also leverages behavioral analysis to generate comprehensive *Network Activity Reports*, containing detailed information about network activities of analyzed malware specimens. These reports are immediately disseminated to all the cooperating networks, that can leverage this knowledge by implementing pro-active countermeasures, hopefully before being attacked by the malware.

The second cooperative architecture proposed in this thesis [70] improves the hierarchical scheme by referring to a peer-to-peer architecture. Alert aggregation and analysis is performed by a self-organizing overlay of cooperative alert aggregators, rather than by a static hierarchical architecture. This innovative design choice guarantees increased scalability, self adaptation, and dependability due to the lack of a single point of failure.

The effectiveness of all the architectures and algorithms proposed in this thesis are demonstrated by experiments carried out on prototypes based on Open Source software. Each prototype is tested to verify its functions and to guarantee that its performance allows (soft) real time operations in realistic network scenarios.

Several interesting results have been achieved and presented in this thesis, however future activities are needed to further improve the security of network-based information systems that are and will be even more characterized by high capacity network links, supports for node mobility, and attacks carried out by self-propagating malware and botnets.

We believe the ParNIDS architecture to be an effective solution for the problems related to the stateful analysis of large volumes of traffic. Some future activities may aim to improve the scalability of the scatterer component, that represents the bottleneck of the proposed architecture.

Security problems related to node mobility represent a novel field in which there is a lot of space for future works. In this thesis we have identified and solved two specific problems, but many other issues will emerge with the diffusion of node mobility.

Cooperative malware detection and analysis is another topic that may lead to further research activities. We are working on the integration of heterogeneous sensors applied to real networks, although we can envision that the integration of heterogeneous data coming from different sources will represent an interesting challenge by itself.

Bibliography

[1] M. Abd-El-Malek, G. Ganger, G. Goodson, M. Reiter, and J. Wylie. Fault-scalable byzantine fault-tolerant services. In *Proc. of the twentieth ACM symposium on Operating systems principles (SOSP 05)*, Oct. 2005.

[2] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth. BAR fault tolerance for cooperative services. In *Proc. of the twentieth ACM symposium on Operating systems principles (SOSP 05)*, Oct. 2005.

[3] M. Alam, Q. Javed, M. Akbar, M.R.U.Rehman, and M. Anwer. Adaptive load balancing architecture for snort. In *Proc. of the Internationale Conference on Networking and Communication (INCC 2004)*, Lahore, Pakistan, Jun. 2004.

[4] L. Alvisi, A. Clement, M. Dahlin, M. Marchetti, and E. Wong. Making byzantine fault tolerant systems tolerate byzantine faults. In *Proc. of the 6th Usenix Symposium on Networked Systems Design and Implementation (NSDI 2009)*, Boston, MA, USA, April 2009.

[5] M. Andreolini, S. Casolari, M. Colajanni, and M. Marchetti. Dynamic load balancing for network intrusion detection systems based on distributed architectures. In *Proc. of the sixth IEEE International Symposium on Network Computing and Applications (NCA 2007)*, Cambridge, MA, USA, July 2007.

[6] M. Andreolini, M. Colajanni, and M. Nuccio. Scalability of content-aware server switches for cluster-based web information systems. In *Proc. of the 12th International World Widw Web Conference (WWW2003)*, Budapest, Hungary, May 2003.

[7] S. Axelsson. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *ACM Conference on Computer and Communications Security*, pages 1–7, 1999.

[8] Base home page, available online at http://ebtables.sourceforge.net/.

[9] T. Braun and M. Danzeisen. Secure mobile ip communication. In *Proc. of the 26th Annual IEEE Conference on Local Computer Networks (LCN'01)*, Tampa, FL, USA, November 2001.

[10] L. Bu and J. A. Chandy. FPGA based network intrusion detection using content addressable memories. In *Proc. of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, USA, Apr. 2004.

[11] D. Burroughs, L. Wilson, and G. Cybenko. Analysis of distributed intrusion detection systems using bayesian methods. In *Proc. of the 21st IEEE International Per-*

formance Computing and Communication Conference, Phoenix, AZ, USA, April 2002.

[12] R. Cáceres and V. N. Padmanabhan. Fast and scalable handoffs for wireless internetworks. In *Proc. of the 2nd ACM Annual International Conference on Mobile Computing and Networking (ACM MOBICOM'96)*, New York, USA, November 1996.

[13] C. Canali, M. Rabinovich, and Z. Xiao. *Utility computing for Internet applications*. Spribger Verlang, 2004.

[14] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu. The state of the art in locally distributed web-server systems. *ACM Computing Surveys*, 34(2):263–311, 2002.

[15] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Security for structured peer-to-peer overlay networks. In *Proc. of the 5th Symposium on Operating Systems Design and Implementaion (OSDI 02)*, Boston, MA, USA, Dec 2002.

[16] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron. Exploiting network proximity in distributed hash tables. In *Proc. of the International Workshop on Future Directions in Distributed Computing (FuDiCo)*.

[17] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. One ring to rule them all: service discovery and binding in structured peer-to-peer overlay networks. In *Proc. of the 10th workshop on ACM SIGOPS European workshop*, pages 140–145, New York, NY, USA, 2002. ACM.

[18] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. J. Wang, and A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *Proc. of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2003)*, San Franciso, CA, USA, April 2003.

[19] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 2002.

[20] B. Caswell and J. Hewlett. Snort users manual, 2006.

[21] I. Charitakis, S. Anagnostakis, and E. P. Markatos. An active traffic splitter architecture for intrusion detection. In *Proc. of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS 2003)*, Orlando, FL, USA, Oct. 2003.

[22] A.-T. Cheng, C.-H. Wu, J.-M. Ho, and D. Lee. Secure mobile ip communication. In *Proc. of the 2004 IEEE International Conference on Networking, Sensing and Control*, Taipei, Taiwan, March 2004.

[23] C. R. Clark, W. Lee, D. E. Schimmel, D. Contis, M. Koné, and A. Thomas. A hardware platform for network intrusion detection and prevention. In *Proc. of the Workshop on Network Processors and Applications at HPCA (NP-3)*, Madrid, Spain, Feb. 2004.

[24] A. Clement, M. Marchetti, E. Wong, L. Alvisi, and M. Dahlin. Bft: The time is now. In *Proc. of the Workshop on Large-Scale Distributed Systems and Middleware (LADIS 2008)*, Yorktown, NY, USA, September 2008.

[25] C. J. Coit, S. Staniford, and J. McAlerney. Towards faster string matching for intrusion detection or exceeding the speed of snort. In *Proc. of the DARPA Information Survivability Conference and Exposition*, 2001.

[26] M. Colajanni, D. Gozzi, and M. Marchetti. Enhancing interoperability and stateful analysis of cooperative network intrusion detection systems. In *Proc. of the ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ACM/IEEE ANCS 2007)*, Orlando, FL, USA, Dec. 2007.

[27] M. Colajanni, D. Gozzi, and M. Marchetti. Collaborative architecture for malware detection and analysis. In *Proceedings of The IFIP 23rd International Information Security Conference (SEC 2008)*, September 2008.

[28] M. Colajanni, D. Gozzi, and M. Marchetti. Selective alerts for the run-time protection of distributed systems. In *Proc. of the Ninth International Conference on Data Mining, Protection, Detection and other Security Technologies (DATAMINING 2008)*, Cadiz, Spain, May 2008.

[29] M. Colajanni and M. Marchetti. A parallel architecture for stateful intrusion detection in high traffic networks. In *Proc. of the IEEE/IST Workshop on "Monitoring, attack detection and mitigation" (MonAM 2006)*, Tuebingen, Germany, September 2006.

[30] A. Constantine and R. Stadler. Adaptable sevrer cluster with QoS constraints. In *Proc. of the 9th IFIP/IEEE International Symposium on Integrated Network Management*, Nice, France, May 2005.

[31] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. HQ replication: A hybrid quorum protocol for Byzantine fault tolerance. In *Proc. of the 5th Symposium on Operating Systems Design and Implementaion (OSDI 06)*, Nov. 2006.

[32] M. Crovella. *Performance Evaluation with Heavy-Tailed Distributions*. Springer, 2001.

[33] Cwsandbox, behavior-based malware analysis remote sandbox service, available online at http://www.cwsandbox.org.

[34] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. P2p-based collaborative spam detection and filtering. In *Proceedings of the Fourth International Conference on Peer-to-Peer Computing (P2P 04)*, Washington, DC, USA, August 2004.

[35] Distributed checksum clearinghouses homepage, available online at http://www.rhyolite.com/dcc/.

[36] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer. Operational experiences with high-volume network intrusion detection. In *Proc. of the 11th ACM conference on Computer and communications security*, 2004.

[37] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001.

[38] Dynamics mobile ip home page, available online at http://dynamics.sourceforge.net.

[39] Ebtables home page, available online at http://ebtables.sourceforge.net/.

[40] G. Fishman and I. Adan. How heavy-tailed distributions affect simulation-generated time averages. *ACM Transactions on Modeling and Computer Simulation*, 16(2):152–173, 2006.

[41] E. Fogelstroem, A. Jonsson, and C. E. Perkins. Mobile ipv4 regional registration. *Request For Comments 4857 (Experimental), Internet Engineering Task Force*, June 2007.

[42] Freepastry, an open source implementation of pastry, available online at http://www.freepastry.org.

[43] F. C. Freiling, T. Holz, and G. Wicherski. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. In *In Proceedings of 10 th European Symposium on Research in Computer Security, ESORICS*, pages 319–335, 2005.

[44] S. Fu and M. Atiquzzaman. Improving end-to-end throughput of mobile ip using sctp. In *Proc. of the 2003 Workshop on High Performance Switching and Routing (HPSR 2003)*, Turin, Italy, June 2003.

[45] S. Gaudin. Storm worm botnet more powerful than top supercomputers. *Information Week*.

[46] I. I. D. W. Group. The intrusion detection message exchange format (idmef), http://tools.ietf.org/html/rfc4765, 2007.

[47] R. Hsieh, Z. G. Zhou, and A. Sereviratne. S-mip: a seamless handoff architecture for mobile ip. In *Proc. of the 22nd Annual Joint Conference of the IEEE Computer and Communication Societies (INFOCOM 2003)*, San Francisco, USA, March 2003.

[48] W.-Y. Hsin, S.-C. Lin, and S.-S. Tseng. A study of alert-based collaborative defense. In *ISPAN '05: Proceedings of the 8th International Symposium on Parallel Architectures,Algorithms and Networks*, pages 148–153, Washington, DC, USA, 2005. IEEE Computer Society.

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

[50] Iptables home page, available online at http://www.netfilter.org/projects/iptables/index.html.

[51] The java database connectivity (jdbc), available online at http://java.sun.com/javase/technologies/database/index.jsp.

[52] J. Jung and E. Sit. An Empirical Study of Spam Traffic and the Use of DNS Black Lists. In *Internet Measurement Conference*, Taormina, Italy, October 2004.

[53] Juniper networks home page, available online at http://www.juniper.net.

[54] T. Kilallea. Recommended internet service provider security services and procedures. *Request For Comments 3013, Internet Engineering Task Force*, November 2000.

[55] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: speculative byzantine fault tolerance. In *Proc. of the twentyfirst ACM Symposium on Operating Systems Principles (SOSP 07)*, 2007.

[56] C. Kruegel and W. Robertson. Alert verification: Determining the success of intrusion attempts. In *Proc. of the First Workshop the Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 2004)*, 2004.

[57] C. Kruegel, F. Valeur, G. Vigna, and R. Kemmerer. Stateful intrusion detection for high-speed networks. In *Proc. of the IEEE Symposium on Research on Security and Privacy*, Oakland, CA, USA, May 2002.

[58] W. Lee and S. J. Stolfo. Data mining approaches for intrusion detection. In *Proc. of the Seventh USENIX Security Symposium (SECURITY '98)*, 1998.

[59] H. C. Li, A. Clement, M. Marchetti, M. Kapritsos, L. Robison, L. Alvisi, and M. Dahlin. Obedience vs choice in cooperative services. In *Proc. of the 8th Usenix Symposium on Operating Systems Design and Implementation (OSDI 2008)*, San Diego, CA, USA, December 2008.

[60] H. C. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. Bar Gossip. In *Proc. of the 7th Symposium on Operating Systems Design and Implementaion (OSDI 06)*, 2006.

[61] Z. Li, Y. Chen, and A. Beach. Towards scalable and robust distributed intrusion alert fusion with good load balancing. In *Proc. of the SIGCOMM Workshop on Large Scale Attack and Defense (LSAD06)*, 2006.

[62] D. J. Lilja. *Measuring computer performance. A practitioner's guide*. Cambridge University Press, 2000.

[63] R. Lippmann, J. W. Haines, D. J. Fried, J. Korba, and K. Das. Analysis and results of the 1999 darpa off-line intrusion detection evaluation. In *Proc. of the Third International Workshop on Recent Advances in Intrusion Detection*, Toulouse, France, October 2000.

[64] W. Ma and Y. Fang. Dynamic hierarchical mobility management strategy for mobile ip networks. *IEEE Journal on Selected Areas in Communications*, 22(4):664–676, May 2004.

[65] R. Mahajan, M. Castro, and A. Rowstron. Controlling the cost of reliability in peer-to-peer overlays. In *Proc. of the Second International Workshop on Peer-to-Peer Systems (IPTPS 2003)*, 2003.

[66] M. V. Mahoney and P. K. Chan. An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In *Proc. of the Sixth International Workshop on Recent Advances in Intrusion Detection (RAID 2003)*, Pittsburgh, PA, USA, Sep. 2003.

[67] D. J. Malan and M. D. Smith. Host-based detection of worms through peer-to-peer cooperation. In *Proc. of the 2005 ACM workshop on Rapid Malcode (WORM 2005)*, 2005.

[68] M. Marchetti and M. Colajanni. Adaptive traffic filtering for efficient and secure ip mobility. In *Proc. of the 4th ACM International Symposium on QoS and Security for Wireless and Mobile Networks (Q2SWinet 2008)*, Vancouver, British Columbia, Canada, October 2008.

[69] M. Marchetti and M. Colajanni. Parnids: Parallel network intrusion detection system for stateful and reliable analysis. Technical report, Submitted for publication. University of Modena and Reggio Emilia, Modena, Italy, 2008.

[70] M. Marchetti and M. Messori. Dependable distributed architecture for collaborative intrusion detection and malware analysis. Technical report, Submitted for publication. University of Modena and Reggio Emilia, Modena, Italy, 2009.

[71] D. S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou. Process migration. *ACM Comput. Surv.*, 32(3):241–299, 2000.

[72] T. Min, T. Lin, and K. Jianchu. A seamless handoff approach of mobile ip based on dual link. In *Proc. of the First IEEE International Conference on Wireless Internet (WICON'05)*, Budapest, Hungary, July 2005.

[73] G. Montenegro and V. Gupta. Sun's skip firewall traversal for mobile ip. *Request For Comments 2356, Internet Engineering Task Force*, June 1998.

[74] B. Morin, L. Mé, H. Debar, and M. Ducassé. M2d2: A formal data model for ids alert correlation. In *Proc. of the 5th symposium on Recent Advances in Intrusion Detection (RAID 2002)*, 2002.

[75] D. Mutz, G. Vigna, and R. Kemmerer. An experience developing an IDS stimulator for the black-box testing of network intrusion detection systems. In *Proceedings of the 2003 Annual Computer Security Applications Conference (ACSAC '03)*, pages 374–383, Las Vegas, Nevada, December 2003.

[76] mwcollect alliance homepage, available online at http://alliance.mwcollect.org/.

[77] Mysql home page, available online at http://www.mysql.com/.

[78] J. Nash. Non-cooperative games. *The Annals of Mathematics*, 54:286–295, Sept 1951.

[79] Nepenthes home page, available online at http://nepenthes.mwcollect.org/.

[80] Netem network emulator, available online at http://www.linuxfoundation.org/en/net:netem.

[81] P. Ning, Y. Cui, and D. Reeves. Constructing attack scenarios through correlation of intrusion alerts, 2002.

[82] Nmap, free security scanner for network exploration and security audits, available online at http://nmap.org/.

[83] Norman home page, available online at http://www.norman.com.

[84] A. Orebaugh, S. Biles, and J. Babbin. *Snort cookbook*. O'reilly, 2005.

[85] M. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, 1994.

[86] S. Patton, W. Yurcik, and D. Doss. An achilles' heel in signature-based ids: Squealing false positives in snort. In *Proc. of the fourth International Symposium on Recent Advances in Intrusion Detection (RAID 2001).*, 2001.

[87] C. E. Perkins. Mobile networking through mobile ip. *IEEE Internet Computing*, 2(1):58–69, January 1998.

[88] C. E. Perkins. Mobile ip and security issue: an overview. In *Proc. of the First IEEE/Popov Workshop on Internet Technologies and Services*, Moscow, Russia, November 1999.

[89] C. E. Perkins. Mobile ip. *IEEE Communications Magazine*, 40(5):66–82, May 2002.

[90] C. E. Perkins. Mobility support for ipv4. *Request For Comments 3344, Internet Engineering Task Force*, August 2002.

[91] C. E. Perkins and D. B. Johnson. Route optimization in mobile ip. *IETF Internet Draft*, February 2000.

[92] C. E. Perkins and K.-Y. Wang. Optimized smooth handoffs in mobile ip. In *Proc. of the 1999 IEEE Symposium on Computers and Communications*, Red Sea, Egypt, July 1999.

[93] V. A. Pham and A. Karmouch. Mobile software agents: An overview. *IEEE Communication Magazine*, 36(7):16–37, 1998.

[94] P. Porras, D. Schnackenberg, S. Staniford-Chen, Davis, M. Stillman, and F. Wu. The common intrusion detection framework architecture, available online at http://gost.isi.edu/cidf/drafts/architecture.txt, 1999.

[95] L. Portnoy, E. Eskin, and S. J. Stolfo. Intrusion detection with unlabeled data using clustering. In *Proc. of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*, 2001.

[96] Prelude hybrid intrusion detection system, available online at http://www.prelude-ids.org/.

[97] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, 1998.

[98] T. R. Gula. Correlating IDS alerts with vulnerability information, available online at available at http://www.nessus.org/whitepapers/va-ids.pdf, 2002.

[99] Z. Ren, C.-K. Tham, C.-C. Foo, and C.-C. Ko. Integration of mobile ip and multi-protocol label switching. In *Proc. of the 2001 IEEE International Conference on Communications (ICC'2001)*, Helsinki, Finland, June 2001.

[100] L. G. Roberts. Beyond moore's law: Internet growth trends. *Computer*, 33(1):117–119, 2000.

[101] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proc. of the 13th USENIX Conference on System Administration (LISA '99)*, Seattle, WA, USA, Nov. 1999.

[102] A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proc. of the eighteenth ACM symposium on Operating systems principles (SOSP 01)*, pages 188–201, New York, NY, USA, 2001. ACM.

[103] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. In *Proc. of the Third International Workshop on Networked Group Communication, (NGC 2001)*, 2001.

[104] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[105] L. Schaelicke, T. Slabach, B. Moore, and C. Freeland. Characterizing the performance of network intrusion detection sensors. In *Proc. of the Sixth International Symposium on Recent Advances in Intrusion Detection*, Pittsburgh, PA, USA, September 2003.

[106] L. Schaelicke, K. Wheeler, and C. Freeland. Spanids: a scalable network intrusion detection loadbalancer. In *Proc. of the 2nd conference on Computing frontiers*, Ischia, Italy, May 2005.

[107] Shadowserver foundation homepage, available online at http://www.shadowserver.org.

[108] S. Sharma, N. Zhu, and T. cker Chiueh. Low-latency mobile ip handoff for infrastructure-mode wireless lans. *IEEE Journal on Selected Areas in Communications*, 22(4):643–652, May 2004.

[109] S. Snapp, J. Brentano, G. Dias, T. Goan, L. Heberlein, C. Ho, K. Levitt, B. Mukherjee, T. G. D. Mansur, K. Pon, and S. Smaha. A system for distributed intrusion detection. In *Compcon Spring '91. Digest of Papers*, San Francisco, CA, USA, 1991.

[110] S. R. Snapp, J. Brentano, G. V. Dias, T. L. Goan, L. T. Heberlein, C.-L. Ho, K. N. Levitt, B. Mukherjee, S. E. Smaha, T. Grance, D. M. Teal, and D. Mansur. Dids (distributed intrusion detection system) motivation, architecture, and an early prototype. *Internet besieged: countering cyberspace scofflaws*, pages 211–227, 1998.

[111] Snort home page, available online at http://www.snort.org.

[112] R. Sommer and V. Paxson. Exploiting independent state for network intrusion detection. In *Proc. of the 21st Annual Computer Security Applications Conference*, Tucson, AZ, USA, December 2005.

[113] H. Song and J. W. Lockwood. Efficient packet classification for network intrusion detection using fpga. In *Proc. of the 2005 ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays (FPGA 05)*, Monterey, CA, USA, Feb. 2005.

[114] H. Song, T. Sproull, M. Attig, and J. Lockwood. Snort offloader: A reconfigurable hardware NIDS filter. In *Proc. of the 15th International Conference on Field Programmable Logic and Applications (FPL)*, Tampere, Finland, Aug. 2005.

[115] M. Song, J. Huang, R. Feng, and J. Song. Simple key managementfot internet protocols (skip). In *Proc. of the Internet Society's 1995 International Networking Conference (INET'95)*, Honolulu, HI, USA, June 1995.

[116] M. Song, J. Huang, R. Feng, and J. Song. A distributed dynamic mobility management strategy for mobile ip networks. In *Proc. of the 6th International Conference on ITS Telecommunications (ITST 2006)*, Chegdu, China, June 2006.

[117] L. Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[118] Tcpreplay home page, available online at http://tcpreplay.sourceforge.net.

[119] Top layer networks home page, available online at http://www.toplayer.com.

[120] N. Tuck, T. Sherwood, B. Calder, and G. Varghese. Deterministic memory-efficient string matching algorithms fo intrusion detection. In *Proc. of the IEEE Conference on Computer Communication*, Hong Kong, China, March 2004.

[121] The user-mode linux kernel home page, available online at http://user-mode-linux.sourceforge.net/.

[122] F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer. A comprehensive approach to intrusion detection alert correlation. In *IEEE Transactions on Dependable and Secure Computing*, 2004.

[123] P. K. Varshney. *Distributed Detection and Data Fusion*. Springer-Verlag New York, Inc., 1996.

[124] Vilpul's razor: a collaborative spam detection and filtering network. available online at http://razor.sourceforge.net/.

[125] Virustotal, a malware analysis service offered by hispasec sistemas, available online at http://www.virustotal.com.

[126] D. Winer. XMLRPC home page, available online at http://www.xmlrpc.com/.

[127] A. Wool. Direction-based filtering in firewalls. *Elsevier Computers and security*, 23(6):459–468, September 2004.

[128] C.-H. Wu, A.-T. Chen, S.-T. Lee, J.-M. Ho, and D. T. Lee. Bi-directional route optimization in mobile ip over wireless lan. In *Proc. of the 56 IEEE Vehicular Technology Conference (VTC 2002)*, Vancouver, Canada, September 2002.

[129] K. Xinidis, I. Charitakis, S. Antonatos, K. G. Anagnostakis, and E. P. Markatos. An active splitter architecture for intrusion detection and prevention. *IEEE Transactions on Dependable and Secure Computing*, 03(1):31–44, 2006.

[130] K. G. A. Xinidis K and E. P. Markatos. Design and implementation of a high-performance network intrusion prevention system. In *Proc. of the 20th International Information Security Conference (SEC 2005)*, Chiba, Japan, May 2005.

[131] Extensible markup language (xml), available online at http://www.w3.org/xml/.

[132] Xml-rpc for c and c++, available online at http://xmlrpc-c.sourceforge.net/.

[133] D. Xu and P. Ning. Privacy-preserving alert correlation: a concept hierarchy based approach. *Computer Security Applications Conference, 21st Annual*, Dec. 2005.

[134] L. Zhang, J. Cao, and S. K. Das. A mailbox-based scheme for improving mobile ip performance. In *Proc. of the 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03)*, Providence, RI, USA, May 2003.

[135] C. V. Zhou, S. Karunasekera, and C. Leckie. A peer-to-peer collaborative intrusion detection system. In *Proc. of the 13th IEEE International Conference on Networks*, 2005.

[136] P. Zhou and O. W. W. Yang. Reverse routing: An alternative to mip and romip protocols. In *Proc. of the 1999 IEEE Canadian Conference on Electrical and Computing Engineering*, Alberta, Canada, May 1999.