TKN Telecommunication Networks Group

# Protecting Communication Infrastructures Against Attacks with Programmable Networking Technology

vorgelegt von
Diplom-Ingenieur
Andreas Hess

Von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
Dr.-Ing.

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. Heiß
Gutachter: Prof. Dr. Wolisz
Gutachter: Prof. Dr. Rathgeb

Tag der wissenschaftlichen Aussprache: 19.06.2008

Berlin 2008
D 83

# Zusammenfassung

Die Angriffsstatistiken spiegeln ein klares Wachstum registrierter Schutzzielverletzungen wider. Die Ursachen hierfür sind zahlreich. Grundlage sind zum einen die sich stetig vergrößernden Mengen der Internetnutzer und der Internetrechner. Zum anderen lässt sich ein fehlendes bzw. mangelhaftes Sicherheitsbewusstsein vieler Nutzer und Administratoren beobachten, welches in mangelhafter Systempflege resultiert. Betriebssystem- und Softwarehersteller stellen Sicherheitsupdates für das Beseitigen bekannt gewordener Sicherheitslöcher—so genannte Patches—bereit, die aber nicht oder nur stark zeitverzögert eingespielt werden. Als Beispiel hierfür kann der als Blaster bekannt gewordene Internetwurm herangezogen werden, welcher eine Schwachstelle des RPC-Dienstes von Windows-Systemen ausnutzte. Am 16. Juli 2003 stellte Microsoft auf seiner Seite ein entsprechendes Sicherheitsupdate zur Verfügung und dennoch erfolgte am 11. August ein Ausbruch des Internetwurms. Symantec stufte den Wurm in die Kategorie 4 (ernsthafte Bedrohung / weltweite Verteilung) ein.

Mithilfe einer dynamischen, automatisierten Verteilung und Integration von proaktiven Schutzmechanismen innerhalb eines Netzwerkes, kann mit "geringem" Aufwand eine große Zahl an Endsystemen in kurzer Zeit geschützt werden. Es ist dabei jedoch zu beachten, dass Angriffsunterdrückungssysteme das Verhalten (Durchsatz, Latenzzeit, Jitter, etc.) von Netzwerken beeinflussen, da der Netzwerkverkehr vor der Weitervermittlung auf verdächtigen Inhalt geprüft wird. Software-spezifische Schwachstellen und folglich auch die Menge der darauf basierenden Angriffe, sind zumeist spezifisch für eine bestimmte Menge von Anwendungen oder Betriebssystemen. Hierdurch bietet sich eine Modularisierung der zu den Schwachstellen korrespondierenden Schutzmechanismen an.

Das Ziel dieser Arbeit ist die Realisierung eines auf aktiver Netzwerktechnologie basierenden und sich selbstorganisierenden Angriffsunterdrückungs-Netzwerks. Aktive Netzwerke bieten die Möglichkeit, anwendungsspezifische Dienste, z.B. spezielle Angriffsunterdrückungsfunktionalitäten, dynamisch auf aktiven Knoten zu starten und zu beenden. Es wird also untersucht wie die Ermittlung und Erfüllung des spezifischen Schutzbedarfs einzelner Netzbereiche automatisiert werden kann. Dabei werden die folgenden Punkte diskutiert und entwickelt:

- die Architektur des auf aktiver Netzwerktechnologie basierenden Angriffsunterdrückungsystems,
- die Analyse des zu schützenden Netzwerks hinsichtlich Topologie und verbundener Systeme, und
- die Entscheidung auf welchen Router welche Schutzmechanismen platziert werden.

Im Rahmen der Arbeit wird ein Konzept eines solchen Angriffsunterdrückungs-Netzwerks entwickelt und prototypisch implementiert. Es wird exemplarisch gezeigt wie ein Netzwerk hinsichtlich Topologie und Systemkonfigurationen untersucht werden kann. Zur Berechnung der "optimalen" Verteilung der Schutzmechanismen wird ein mathematisches Model aufgestellt. Es werden dabei die folgenden Platzierungsstrategien entwickelt:

- Erfüllung aller Sicherheitsanforderungen bei gleichzeitiger Minimierung der Anzahl der verwendeten aktiven Router.
- Erfüllung aller Sicherheitsanforderungen bei gleichzeitiger Minimierung der maximalen Belastung eines Routers.

Abschliessend wird das entwickelte Angriffsunterdrückungs-Netzwerk auf die Experimentierumgebung Deter-Testbett portiert und es werden zwei konkrete Netzwerkkonfiguration emuliert und bewertet. Die erzielten Resultate zeigen deutlich den Gewinn des entwickelten Ansatzes.

# Abstract

The continued explosion of new virus/worm and other security attacks in the Internet, the tremendous propagation speed of self-propagating attacks, and the still increasing number of hosts connected to the Internet has led to network security being considered as a design criterion rather than an afterthought. Beyond this, also the diversity of software is increasing and still the quality of many software solutions is insufficient, especially in terms of security vulnerabilities resulting from programming errors.

These problems are aggravated by an inappropriate security awareness of many network and system administrators as well as users which has (again) been clearly shown by the W32/Blaster worm [37]. The worm which started on August 11th 2003 exploited a vulnerability that has already been known four weeks earlier. Actually, since July 16th 2003 Microsoft had provided a patch in order to fix this flaw. But still, the worm could diffuse itself in a manner such that Symantec rated it as category 4 (severe threat, global distribution). As a consequence thereof, we can clearly see that the idea of quickly patching all vulnerable systems upon detection of a new security hole is not an appropriate measure to cope with the evolution of execution speed of computer attacks, and that, therefore, attackers will continue to be able to break into systems and deploy them for their purposes in the future.

Attack prevention, detection, and mitigation mechanisms can be broadly classified as network based or host based. Network based security mechanisms have been shown to be much more effective than host based mechanisms, primarily because of the former's ability in identifying attack traffic that is further upstream from the victim and closer to the attack source. In the context of network based mechanisms, we propose a flexible overlay network of security systems running on top of programmable (active) routers. In such an architecture, security services can be dynamically distributed across the network, which provides flexibility for load-balancing of services across nodes and addition of new services over time. The thesis discusses:

- the architecture of the intrusion prevention system on top of active networking technology,
- the process of analyzing the network to be protected in terms of topology and connected systems,
- the third functional part decides which intrusion prevention services are deployed on which programmable routers.

A contribution of the thesis is the conceptual design of the autonomous intrusion prevention overlay network on top of programmable networking technology, also a corresponding pro-

totype is implemented. We exemplarily show how to gather network information in terms of topology and connected systems. Further on, we develop an optimization framework that specifies the optimal placement of security services. The following objective functions are introduced:

- minimize the number of programmable routers used while fulfilling all security requests;
- minimize the maximal workload of a programmable router while fulfilling all security requests.

Finally, the intrusion prevention architecture is setup on the Deter testbed. We show the benefit of the presented approach by emulating two concrete networking scenarios.

# Contents

# Chapter 1

# Introduction

Daily life is more and more influenced by the Internet which, for example, allows to comfortably accomplish tasks like bank transferals, shopping or travel bookings. But the benefit of the Internet is accompanied by the threatening danger of fraud and misuse. For instance, *phishing* which is a form of Internet fraud—it aims to steal valuable information such as credit card numbers, social security numbers, user IDs and passwords—caused solely in the USA in 2007 a damage of over 3 billion US dollars [90].

An initial prerequisite for a successful attack is the existence of a *vulnerability* which is a weakness in an information system or service that can be used by an attacker to alter its intended operation. Further on, recent developments show that communication networks like the Internet are vulnerable and that they cannot be secured by sporadic and uncoordinated security devices like firewalls at users and cooperates sites. The reasons behind this trend originate from multiple developments.

The number of DNS-registered hosts in the Internet keeps growing [42], as does the number of computers connected to the Internet via DSL or cable modem—high bandwidth and permanent access currently represents one of the fastest growing markets in the telecommunications business [91]. In 2007 over 70 % of all Germans used a computer at home and additionally, about 50 % of all households had a broadband Internet access [28]. This implies an accordingly increasing number of vulnerable hosts and offers an ever growing number of potential targets for malicious activities.

Further, the diversity of software is increasing but the quality of many software solutions in terms of security vulnerabilities resulting from programming errors is insufficient—the Computer Emergency Response Team (CERT) counted 4,129 vulnerabilities in 2004 and 5,990 in 2005 [3]. Additionally, users and administrators are very slow in applying fixes to vulnerable systems [112], either because they are overstrained patching the systems or unaware of the threats they represent. The Internet worm Netsky-P was responsible for 15.7 % of all virus incidents that were registered in 2005 although it was spotted the first time in March 2004 and an effective protection against it has been made available at the end of that month [120].

On the other side, the attacks are becoming more aggressive. The propagation speed of self-distributing attacks is increasing [32]—the authors of [123] argue that under certain conditions a small worm "can infect almost all vulnerable servers on the Internet in less than thirty seconds." As a consequence, the time window to invoke countermeasures against

potential attacks is shrinking. For instance, according to [120] there is a chance of 50 %, of being infected within 12 minutes online when running an unpatched Windows system without a firewall.

Countermeasures in terms of attack prevention, detection, and mitigation mechanisms—namely software updates, firewalls, antivirus software and intrusion detection/prevention systems—can be applied to improve the protection of the systems of a communication network. These mechanisms can be broadly classified as network based or host based. Now, referring to the above trends, it can hardly be expected that all users and administrators will be able to keep their system(s) secure. In addition, fixing security holes as soon as patches become available can hardly be done in time on all end systems, and the installation, configuration and operation of an intrusion detection/prevention systems requires a knowledge that most users do not have. In this context, we believe network based security mechanisms to be much more effective than host based mechanisms. Thus, in order to relieve end-users and administrators from continuously having to deal with today's massive amount of security challenges, we propose to use the routers of a network to protect the connected end systems against intrusions. A fundamental requirement for the intrusion prevention architecture is flexibility to cope with both the evolution of attacking techniques and the varying protection demands of the end-systems. A means of realizing flexibility is the usage of *programmable networks* [36] consisting of programmable nodes on which services, for example intrusion prevention services, can be dynamically deployed. Summarizing, the philosophy followed in this thesis is not to modify end-systems and to do intrusion prevention as a software process on routers in the network. The effort to place a limited set of programmable routers in a network that are capable to do intrusion prevention is smaller than to modify a significant higher number of end-systems.

However, the operation of an intrusion prevention system (IPS) inevitably decreases network performance as packets are analyzed for malicious content before being forwarded. To address efficiency two concepts are introduced:

- *demand-driven* intrusion prevention and
- operation of an intrusion prevention *overlay network* (ON).

Demand driven intrusion prevention makes use of the fact that an attack requires the existence of a concrete vulnerability to succeed. In the proposed approach an intrusion prevention service provides protection against attacks exploiting a concrete vulnerability. Hence, the amount of security services required to adequately protect the end-systems of a network can be restricted by a previous identification of the present vulnerabilities and according to our philosophy it makes little sense to scan traffic for attacks which cannot harm a host. Next, the operation of an intrusion prevention overlay network provides the capability to intelligently distribute the security services among the programmable routers. In this context a central question of the thesis is:

> **What intrusion prevention functionality is required at which places of a network for the purpose of adequately protecting the end-systems of that network while simultaneously fulfilling a chosen objective function?**

To answer the question the following aspects must be considered. Further, how to decide on the set of security services that must be installed? And finally, what security services deployment strategies are reasonable?

The presented concept of an intrusion prevention overlay network is designed for limited networks like an autonomous system (AS) as well as for high-speed networks (backbone). The functionality provided by the intrusion prevention overlay network (ON) depends on the chosen scenario. When deployed in a limited networking environment—in terms of a limited amount of connected end-systems and traffic volumes—the IPS overlay network is able to autonomously identify network topology, end-systems (operating system, running applications, etc.) and the set of required security services. Subsequently, the requested security services are intelligently distributed in the network such that, for example, the impact on the network performance—caused by the intrusion prevention overlay network—is minimized. When deployed in a backbone network the IPS framework does not analyze network topology, end-systems and required security services as the amount of systems to be analyzed is too big. In this scenario, the related information must be provided to the autonomous intrusion prevention overlay network. Summarizing, the intrusion prevention overlay network comprises three functional parts:

1. the first functional part provides the intrusion prevention framework that actually allows to dynamically deploy intrusion prevention services on programmable nodes in the network.

2. the second functional part (not for high-speed deployment) includes the intelligence to analyze the network to be protected:

   - identification of the network topology,
   - discovery of the hosts that are running and
   - hosts profiling, identification of operating system and running applications.

3. the third functional part decides which intrusion prevention services are deployed on which programmable routers.

A contribution of the thesis is the conceptual design of an autonomous intrusion prevention overlay network on top of programmable networking technology. A fundamental requirement of the intrusion prevention architecture is the capability to dynamically deploy a security service at runtime on a chosen router. Further requirements and a detailed description of the intrusion prevention architecture is given in Chapter 4. To assess the performance of the proposed intrusion prevention overlay network a Linux-based prototype including a set of seven intrusion prevention services—using existing attack detection techniques—was implemented and analyzed.

Further, the concept of an intrusion prevention overlay network was designed for both limited networking environments and high-speed networking environments. In the former case, the architecture is able to autonomously recognize and deploy the security services, necessary to adequately protect the end-systems of a network, while simultaneously minimizing the impact on the network performance. A prerequisite is knowledge of network topology, end-system configurations and flow volumes. An exemplary case study that shows how to gather necessary network information is presented in Chapter 5.

Chapter 6 discusses the question: "What intrusion prevention functionality is required at which places of a network in order to efficiently protect the end-systems against attacks spreading over the network?". To answer this, an optimization framework was developed that considers the scenarios:

- Security services distribution along predefined routes:
  - single-path
  - multipath
- Joint traffic routing and security service distribution

Predefined routing implies that at least one path from any source to any destination is specified—as a result the routing tables are set. In a single-path routing environment, a single path exists between any two systems. In contrast, in a multipath routing environment multiple paths exist between systems, allowing for load-balancing. In the last scenario the optimization framework specifies traffic routing—a single path from each source to each destination—and the distribution of the requested security services. The optimization framework allows to choose the routing scenario as well as one of the following objective functions:

- minimize the number of programmable routers used while fulfilling all security requests and keeping all router queues bounded;
- minimize the maximal workload of a programmable router while fulfilling all security requests and keeping all router queues bounded.

Chapter 7 presents the performance evaluation of the proposed intrusion prevention framework and the optimal deployment strategies described above. To assess the intrusion prevention overlay network emulations were conducted on the *Cyber Defense Technology Experimental Research* testbed (DETER). Two different network topologies—a limited tree network and a high-speed network—were emulated for varying sets of traffic traces. The achieved results show that intrusion prevention can be done as a software process in case that the requested security services are intelligently distributed among the router nodes. The benefit of the approach proposed is evident for both networks.

# Chapter 2

# A Review of Network-Based Intrusion Detection and Prevention

In this chapter an overview of network-based intrusion detection and prevention systems is given. Initially, communication networks including active/programmable routers are introduced in Section 2.1. Afterwards a general definition of a network-based attack is given in Section 2.3. In addition, we discuss security threats as well as the corresponding security services in the same section. Subsequently, a general intrusion detection/prevention system architecture and a method to classify them is given. Further the section elaborates on the differences between intrusion detection and prevention systems, and it also discusses the hardware that is used for the realization. Finally, the section describes how to evaluate intrusion detection and prevention systems. In Section 2.4 network information gathering techniques are discussed.

## 2.1 Communication Networks

The concept introduced in this thesis is designed for networks using the *Internet Protocol version 4* (IPv4) for data transmission. In this context, we remark that our view of a communication network is restricted to network elements that operate on the network layer and higher layers of the *Open Systems Interconnection Basic Reference Model* (OSI). Accordingly, the topology of a communication network is specified by a varying number of end-systems that are connected with each other over a set of routers and links. Network components that operate in the data link layer like hubs or switches remain invisible to us.

A link is a passive component of a specific capacity and delay whereas a router forwards and routes IP packets. To do so it inspects the packet headers of arriving IP packets and checks the local routing table to identify the outbound interface. Finally, an end-system's configuration is defined by its hard- and software. The former includes among others the *Central Processing Unit* (CPU) and the chipset used. The software configuration is determined by the *Operating System* (OS) and the applications that are running on the system. Finally, end-systems are both sender and receiver of network traffic. Measurements of local-area [88], [136] and wide-area [107] network traffic have shown that packet-switched data traffic is self-similar. A self-similar object is exactly or approximately similar to a part of itself, a popular

example are fractals as they can be divided into parts and each part is a reduced copy of the whole. In the context of network traffic self-similarity means that packet bursts, also known as packet trains, appear on a wide range of time scales.

In practice, the topology of a network steadily changes. A reason for this are end-systems that are connected to the network via dial-in lines and consequently, they are not permanently part of the network. Another reason would be a newly connected web-server. Furthermore, the configuration of an end-system might change over the time. New applications/services can be installed and launched on it. We assume the network topology as well as the end-systems configurations to be static.

Next, the routing defines the path over which data is sent from a sender to a receiver. Routing algorithms can be classified static or dynamic. A static routing algorithm calculates routes which are valid forever and in contrast a dynamic routing algorithm considers the network state and is able to adapt the routing topology. For instance, the dynamic routing protocol *Open Shortest Path First* (OSPF) is able to recalculate routes due to link-/router-failures or overload situations. In this thesis we do not consider dynamic routing protocols.

Apart from that also source- and destination-address of an IP packet can be modified by network elements doing *Network Address Translation* (NAT) [17]. Throughout the thesis NAT is not considered any further. In the networking scenarios considered by us a set of routers is replaced by active routers that are explained in the following section.

## 2.2 Active/Programmable Networking Technology

*Active/programmable networks* provide a framework for flexible and rapid service creation on top of existing networks and they were considered as a new networking paradigm in the early 90's [36, 128]. In an active network, the routers or switches of the network perform customized computations on the messages flowing through them. Or in other words, an active node is expected to offer multiple active networking services.

The concept of active networking was intended to satisfy emerging user demands as well as service provider demands, to facilitate the integration of new technologies into existing networks and to improve the overall network performance. Service examples are media transcoding services, network security enhancing services, protocol boosters, proxies and overlay networks [66, 137, 21].

Active networks are classified *discrete* or *integrated* [129]. The discrete active networking approach separates the transport of active networking service and input data. This means, active service installation and message processing are independent of each other [57, 82]. In detail, active services are installed on chosen active nodes in advance. Each active node examines the arriving packets and forwards them to the appropriate active service(s).

In contrast to this, the integrated approach uses smart packets so-called *capsules* that have their own executable code [134, 13, 117]. A capsule is a combination of a program (or a small fragment) and input data which is automatically executed at each active router it traverses. The programs that are included in a capsule are assumed to consist of simple instructions.

Summarizing, the integrated approach prefers simplicity to security as further management or evaluation functionality is not required. In contrast, the discrete approach is adequate

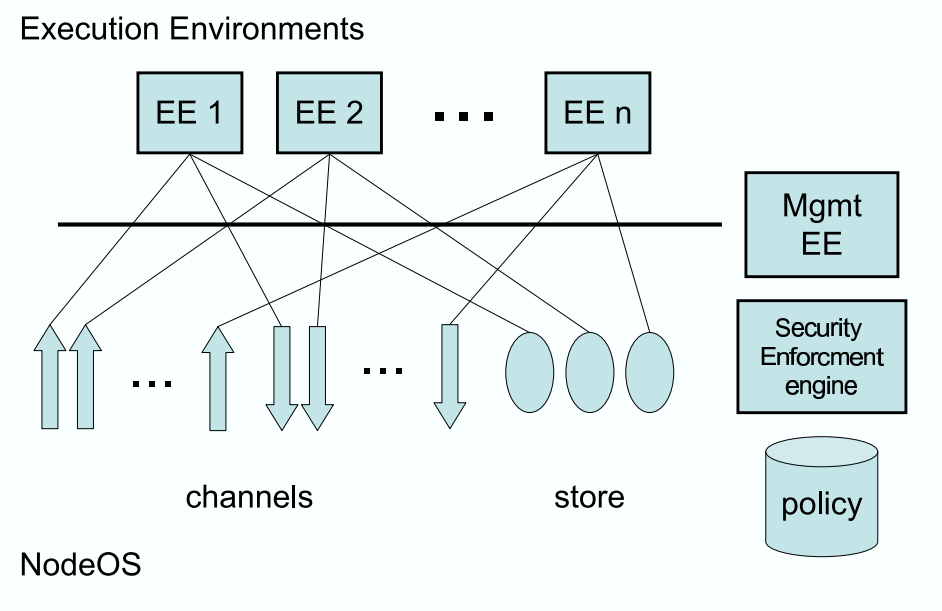



Figure 2.1: The AN architectural framework

for complex active services which can hardly be realized as capsules. In addition, the discrete approach seems to be more applicable for active services that are expected to run permanently like security enhancing services. For example, an intrusion detection service is intended to inspect all packets that are routed through an active node and thus, it is required to run permanently. Moreover, we assume a programmable networking environment to provide the following capabilities:

- the programmable environment must allow to add/remove services at runtime,
- the programmable infrastructure must support the deployment of chosen services on specified programmable routers at a given time,
- the programmable infrastructure must be able to dynamically relocate a service from a programmable router to another one, and
- the programmable infrastructure must provide means to securely exchange information between programmable routers.

In the remainder of this section we introduce the fundamental discrete active node architecture which is depicted in Figure 2.1 and that was developed by the DARPA active network community [35, 63].

The primary functional components are the *Node Operating System* (NodeOS) and the *Execution Environments* (EEs). Each entity of an active networking service runs inside an

EE which provides the interface through which the networking services are provided to users. In addition, an active node is expected to offer multiple active networking services, each one running in its own EE, and accordingly, it is required that the EEs are completely independent of each other. The NodeOS manages the requests for local resources of the EEs. An active networking service triggers its EE to send a resource request to the NodeOS in case that the service requires access to the local resources (e.g. memory). The NodeOS in turn forwards the request to the security enforcement engine which consults the policy database whether to fullfil the request. The connections between network and active networking services are realized via *channels*. Arriving packets are analyzed by an active node which in turn inserts them in the channel connected to the EE that runs the specified active networking service. The active networking service processes the incoming packets and sends them subsequently via the connected outgoing channel. Finally, the *management execution environment* component is responsible for node configuration aspects as for example the modification of a security policy.

A detailed description of the architecture is given in[62]. It was picked up by many later active networking projects, as for example the *AMnet 2.0* architecture [57].

However, challenging security issues are related to the operation of active networks [97]. Network operators do not feel comfortable about executing "arbitrary code" on routers in the network and in addition, users may have the authorization to install active services. Accordingly, an active node itself could be the victim of an attack, or the active node itself could be exposed to attack other end systems. This topic as well as a corresponding concept to address the security isssues of programmable networks are discussed in references [67, 68, 69].

## 2.3 Intrusion Detection and Prevention Systems

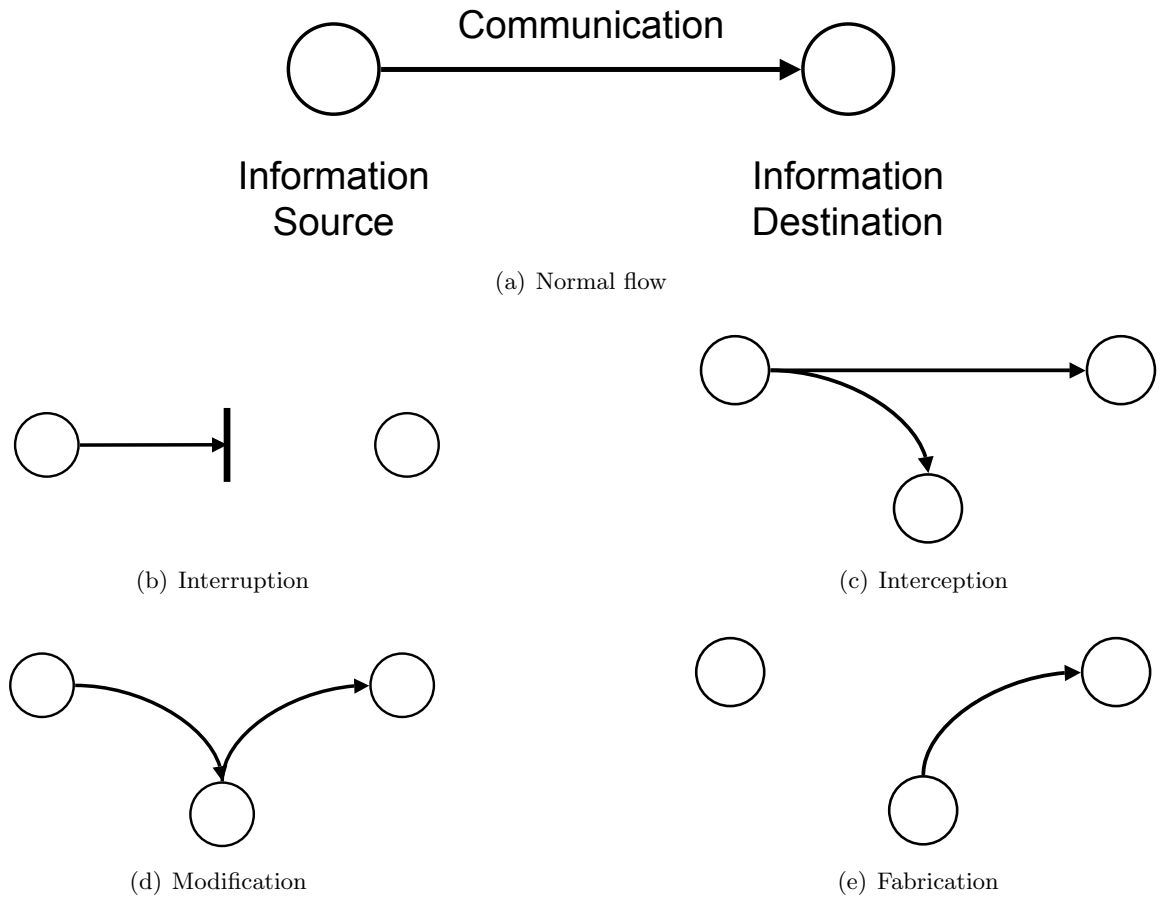
"*Intrusion Detection* is the process of identifying and responding to malicious activity targeted at computing and networking resources" [14]. The concept of an *intrusion detection system* (IDS) was introduced in 1980 by Anderson [16]. A few years later Denning formalized a first intrusion detection model that tried to detect anomalies based on profiles that characterize the behavior of subjects/users with respect to objects/system's resources [51].

### 2.3.1 What is an Attack?

Many slightly varying definitions of a network attack exist. For example, Schäfer defines an attack as the actual realization of a threat, whereas a threat is any possible event or sequence of actions that might lead to a violation of one or more security goals [115]. Lindqvist and Jonsson define an *intrusion* as a successful attack that consists of: 1.) an attack which exploits a *security flaw* (aka *vulnerability*), and 2.) a *breach* (aka *compromise*) which is the resulting violation of the security policy [89]. The definition of an attack used throughout this thesis is more general:

> **An attack is any sequence of actions that changes the behavior of a system in an unwanted way.**

Since the emergence of network attacks, researchers made an effort to categorize them (e.g. [38, 89, 99, 122]). In the following two attack taxonomies are presented: the process-




(a) Normal flow

(b) Interruption

(c) Interception

(d) Modification

(e) Fabrication

Figure 2.2: Security attacks

based taxonomy from Stallings [122] and the threat-based taxonomy of Schäfer [115].

Stallings introduced a process-based taxonomy [122] differentiating between four attack techniques (see Figure 2.2): **interruption**, **interception**, **modification** and **fabrication**. Figure 2.2(a) depicts a normal communication flow; an information source produces data and sends this data to the chosen information destination.

An interruption attack (see Figure 2.2(b)) prevents that all data sent by the information source is received by the information destination. Attacks in this category do a denial-of-service (DoS) attack against communication acts. For example, an attacker could try to overload the victim's link such that packets will be dropped. Another possibility, would be to physically destroy the communication channel between source and destination.

The consequence of an interception attack (see Figure 2.2(c)) is that an unauthorized third party gets access to the information. For example, an attacker could eavesdrop the communication channel between source and destination. Generally, it is difficult to notice such an attack as all information is correctly received by the destination. A modification attack (see Figure 2.2(d)) manipulates the information that is sent from the source to the destination. A requirement for such an attack is that the attacker has access to a router/

network device that is on the data-path between source and destination. Finally, in case of a fabrication attack (see Figure 2.2(e)), an attacker creates counterfeit messages—using the information source's identity—and sends them to the information destination.

In contrast, Schäfer distinguishes between the following technical threats:

- *Masquerade:* An entity pretends to have the identity of another entity.
- *Eavesdropping:* An entity reads information that is meant for someone else.
- *Authorization violation:* An entity uses services or resources although it does not have appropriate permission.
- *Loss or modification of information:* Certain information is destroyed or modified.
- *Forgery:* An entity creates new information using the identity of another entity.
- *Repudiation:* An entity falsely denies having participated in a particular action.
- *Sabotage:* Any action that is aimed at reducing the availability or correct functioning of services or systems.

In reality, a concrete attack often involves a combination of the threats mentioned above. An intrusion into a system often involves sniffing the access identification and related password. The identity of the sniffed identification is then provided for the access check with the latter representing a masquerade.

Nevertheless, for the purpose of identifying intrusions (definition: the unwanted changing of the behavior of systems) it is a mandatory prerequisite to precisely define a corresponding security policy. A security policy is the set of rules, principles, and practices that determine which *security goals* should be realized in what fashion. Basically, the following security goals are distinguished:

- **Confidentiality:** Transmitted or stored data should only be disclosed to authorized entities.
- **Data integrity:** It should be possible to detect unintentional or deliberate changes to data. This requires that the identification of the originator of the data is unique and cannot be manipulated.
- **Accountability:** It must be possible to identify the entity responsible for a particular event (e.g., use of a service).
- **Availability:** The services implemented in a system should be available and function properly.
- **Controlled access:** Only authorized entities should be able to access certain services and data.

Next, *security services* are applied to enforce the security goals defined in a given security policy. In detail, the following security services exist:

- **Authentication**: the process of determining whether someone or something is, in fact, who or what it is declared to be.
- **Integrity**: is the assurance that data created by a specific entity cannot be modified by third without being detected; accordingly the security service of integrity requires authentication.
- **Confidentiality**: is the process of ensuring the secrecy of protected data.
- **Access control**: is the process of permitting or denying the use of an object (e.g. a system or file) by a subject (e.g. a user or a process).
- **Non repudiation**: is the process of of ensuring that a communication peer cannot later deny its participation.

Further, each security service belongs to one of the following three general categories [85]:

- **Attack Prevention:** the set of security mechanisms that protect systems against attacks by restricting access to critical resources. An important member of this category is the technique of access control which is used to define or restrict the rights of authorized users, application programs, systems, or processes to information system resources (e.g. multi-user computer system, firewall).
- **Attack Avoidance:** this set of security mechanisms is applied in case that an attacker has access to the critical resource like a message that is transmitted over the public Internet. Then the information is transformed in manner that makes it useless to an attacker. A technique to secure information is cryptography.
- **Attack Detection:** the security mechanisms in this set actually do not protect system resources, they rather identify attacks/attack attempts that take or have taken place. Intrusion detection systems belong to this category. Attack detection is an important prerequisite for the recovery process of compromised systems.

The focus of this thesis is on network-based techniques to prevent attacks from becoming successful. Hence, the subsequent section discusses the concept of a firewall which is the most widely-used approach to protect end-systems against attacks.

### 2.3.2 Access Control

*Access control* is the process of permitting or denying the use of an object by a subject. In the context of network security, the most common applied access control mechanism is a *firewall*. A firewall controls the network traffic that might enter/leave the trusted network in accordance with a set of rules specified in the security policy. A *personal firewall* is placed on an end-system whereas a *network firewall* runs on a dedicated network device like a gateway router for the purpose of:

- restricting traffic to enter at one carefully controlled point,
- restricting the traffic that may enter corresponding to rules (source-address, destination-address, protocol, etc) and

- restricting traffic to leave at one carefully controlled point.

Moreover, firewalls are further categorized into *network layer* and *application layer* firewalls. Network layer firewalls–despite their name—operate on the network (IP) and transport layer (UDP/TCP). Accordingly, a network administrator is able to specify rules using attributes like: IP source/destination address, source/destination port or specific protocol flags (e.g. *TCP SYN* or *ACK* flag). In contrast, an application layer firewall, which most often consists of a set of *proxy servers* allows for a more fine-grained access control on the application layer. A proxy server is a server that acts as an intermediary between end-systems and the Internet. Initially, all end-systems' requests for Internet services are sent to that proxy. Next, those requests that are authorized by the security policy are executed by the proxy server. An example would be an application layer firewall that can be configured to allow *WWW*-requests only for a set of authorized *Uniform Resource Locators* (URL).

A further distinctive feature is whether a firewall keeps state or not. A *stateful firewall* keeps track of network packets over a specified period of time in order to block packets that do not belong to an existing communication. For example, a stateful firewall observes the 3-way handshake of TCP peers for the purpose of evaluating whether packets belong to an established TCP session. A *stateless firewall* does not keep track of past events.

Finally, it must be stated that firewalls cannot protect against new threats, viruses or worms which is the scope of intrusion detection and prevention.

### 2.3.3 Intrusion Detection and Prevention System Architecture

The *Common Intrusion Detection Framework* (CIDF) working group made an attempt to enable and facilitate the cooperation of different intrusion detection components/systems [125] by developing a general IDS-architecture that consists of components that can be found in most common IDSs:

- *event generators/E-boxes*: an E-box is the interface between the IDS and the computational/networking environment. It collects data and generates events based on that data. An exemplary E-box is a filter that passively monitors a network and which generates events based on the traffic gathered.

- *event analyzers/A-boxes*: an A-box receives the events from the E-boxes, analyzes them and generates new ones. An example is an A-box which tries to detect specific attack signatures in the events.

- *event database/D-boxes*: D-boxes are used to store events and accordingly, D-boxes allow for an exhaustive postmortem analysis.

- *response units/R-boxes*: response units receive and process the events produced by the event analyzers. This might include actions like the killing of processes or the resetting of specific connections. R-boxes consume events.

Figure 2.3 depicts a CIDF representation of an exemplary network-based IDS. The E-box copies the network traffic and forwards it to the three A-boxes constituting the first stage of the analysis process. Each A-box of the first stage analyzes the network traffic in terms

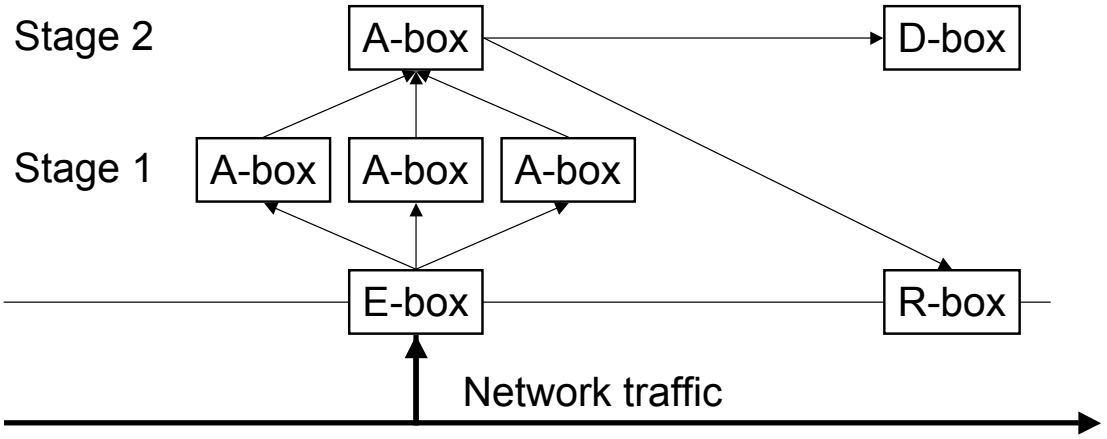



Figure 2.3: CIDF representation of an exemplary IDS-architecture

of malicious contents and forwards its result to the A-box of stage 2. The A-box of stage 2 correlates the results of stage 1 and initiates the adequate response by instructing the R-box. In addition, it forwards the information to be logged to the D-box.

### 2.3.4 Taxonomy of Intrusion Detection and Prevention Systems

Much research was done in the field of intrusion detection leading to a multitude of approaches. According to the taxonomy given in [50] IDSs can be categorized on the basis of:

- audit source location: host-/network-based
- the detection method: anomaly/misuse
- behavior on detection: passive/active
- usage frequency: continuous/periodic
- state awareness: stateful/statelss

First, IDSs are categorized into the two types *network intrusion detection systems* (NIDS) and *host-based intrusion detection systems* (HIDS). A NIDS monitors packets on the network wire and attempts to discover if a hacker is attempting to break into a system. A typical example is a system that watches for large number of TCP connection requests (SYN) to many different ports on a target machine thus, discovering if someone is attempting a TCP port scan. In contrast, a HIDS is locally installed on an end-system and it focuses its monitoring and analysis on the internals of the system rather than on its external interfaces. For example a HIDS monitors which processes try to access/modify the local password file.

Second, the evaluation method whether or not an attack is taking place differ between IDSs. The *anomaly technique* assumes that all intrusive activities are necessarily anomalous. Thus, if it would be possible to create a "normal" behavior pattern of a system/

communication network, every occurrence that does not accord to this pattern would be identified as an intrusion. Thereby, the main problem is the creation of the behavior patterns. Different approaches have been discussed including e.g. neural networks but still many problems exist. Another drawback of this method is that it is not possible to identify an attack by name. The system detects an anomaly but it is not inevitably able to link the anomaly to a specific attack and, hence it is difficult to invoke adequate countermeasures. In contrast thereto, the *misuse detection* technology (aka *signature detection*) scans the gathered information (e.g. network traffic) for known attack signatures. An alert is generated in case that the gathered information matches an attack signature. A simple signature example is the *Land*-attack, the source and destination IP addresses in a packet are the same" [102]. The strength of this approach is that each detected attack is named correctly which helps invoking the correct response. But one drawback hereby is that a signature-based IDS is only able to detect known attacks and that the signature database must be continuously maintained. Furthermore, through the crafting of numerous packets which match attack signatures, alarms on an IDS can be conditioned or disabled and then exploited [105], [141]. An attacker who uses existing evasion techniques could trick signature-based IDSs. For example one instruction belonging to the Nimda-attack [92] trying to exploit the so-called "Directory Traversal" vulnerability could look like:

- GET /SCRIPTS/../../winnnt/system32/cmd.exe?/c+dir,
- GET /SCRIPTS/../../winnnt/system32/cmd.%65xe?/c+dir,
- GET /SCRIPTS/..%c0%af../winnnt/system32/cmd.exe?/c+dir.

The given instructions look different but they are identical. Either an attack signature is generated for each thinkable permutation or appropriate functions are integrated into the IDS. In the first case, an enormous amount of attack signatures must be managed by the IDS. Nevertheless, a modern IDS should be able to upgrade the set of attack signatures and to integrate new functionalities in order to react on newly detected evasion techniques.

Third, IDSs differ in their response to attacks. An active IDS provides a certain set of response mechanisms to an attack like:

- the re-configuration of a firewall or
- the sending of a *TCP RESET* packet in order to stop the attack session (only applicable to TCP based attacks and obviously, this results in a race condition between attack and response mechanism)
- etc.

Active IDSs are also known as *intrusion detection and response systems* (IDRS). In contrast, a passive IDS solely triggers an alert that an attack was detected.

Fourth, IDSs are categorized in accordance to their usage frequency. A real-time IDS must run continuously whereas some distributed systems do an periodical offline audit as they collect the log files from different sensors in the network.

Fifth, the distinctive feature of *stateful* and *stateless* IDSs is added. A stateful IDS keeps track of a certain amount of packets allowing to detect more sophisticated attacks—for

example attacks that span multiple packets. A stateless IDS solely inspects one packet a time and accordingly, stateless IDSs miss many attacks. Today, most IDSs are stateful.

### 2.3.5 Difference between Intrusion Detection and Intrusion Prevention

Most network-based IDSs do a *passive* packet analysis (see Figure 2.4). This means a promiscuous network device or a sniffer creates copies of the network traffic and hands those to the analysis engine. In CIDF terminology, the E-box copies the network traffic and forwards it to the A-box(es). The strengths of the passive packet analysis are that it does not degrade the network performance and that the presence of a promiscuous network device is hard to detect by other machines.

However, Ptacek revealed in his seminal report [72] fundamental flaws of the passive packet analysis technique. He demonstrates that a skilled attacker might use evasion techniques—which exploit ambiguities in the network traffic as seen by the NIDS—for the purpose of avoiding detection by a NIDS. Traffic ambiguities occur due to the fact that "there isn't enough information on the wire on which to base conclusions about what is actually happening on networked machines".

For example a NIDS must be capable to reassemble overlapping/out-of-order IP fragments in the same manner as the receiving end-system; different OSs reassemble overlapping/out-of-order fragments in different manners. This means that in case of overlapping fragments addressed to a Windows NT 4.0 host, the NIDS must favor old data while reassembling. If the fragments are addressed to a Linux host, the NIDS must favor new data while reassembling. A related problem is the question of which packets a NIDS should accept in order to process identical packets as the end-hosts. For example some end-hosts will accept source routed packets, but many will not. What about a packet seen by the NIDS that has a TTL of 2, does it reach its destination or not?

Further, as a matter of principle, passive packet analysis IDSs are *fail-open*, which means that the IDS does not provide any further protection in case that it is crashed/overloaded. Accordingly, in case of a denial of service attack that tries to exhaust the victim's resources by flooding it, a fail-open IDS ceases attack detection. In contrast, a *fail-close* IDS would block all traffic in case it crashes. A thorough discussion of the general flaws of network-based intrusion detection systems can be found in the technical report of Ptacek [72].

For the purpose of addressing the above mentioned issues a NIDS should be provided with:

- knowledge about the network,
- mechanisms that allow to identically process and interpret the packets as the receiving end-system and
- a fail-close mechanism.

With regards to the second aspect, it must be taken into account that standards do not specify the complete behavior of network protocols and hence, it is not sufficient to supply an IDS with a protocol stack that is compliant to the standard. Actually, an *ideal* IDS should be supplied with mechanisms that allow to accurately reproduce the protocol stack behavior of all existing network protocol implementations.

Besides this, the following approaches are considered to eliminate the network traffic ambiguities of an IDS:

- The deployment of a HIDS on each end-system which guarantees an accurate view of the network traffic. But the operation of a HIDS on every system implicates a huge management effort as each system must be deployed, configured and maintained.
- Gathering and providing the required network knowledge (network topology, end-systems OSs). The challenging issue is to identify which host is running what operating system (version, patch-level). Tools that assist the knowledge gathering process—like network scanners as *nmap* [58]—exist and and are emerging.
- The *bifurcating analysis* technique creates multiple analysis threads in order to examine each of the possible traffic interpretations for malicious content [106]. The approach scales well as long as the number of possible interpretations is rather small.
- The operation of a *traffic normalizer* [65] which is deployed directly in the path of traffic for the purpose of eliminating traffic ambiguities. For example, a traffic normalizer effaces fragmentation specific ambiguities by reassembling the IP fragments itself before forwarding them. But it must be taken into account that traffic normalization might affect end-to-end semantics and in addition, traffic normalization has an impact on the network performance (e.g. a packet cannot be reassembled until all fragments are received by the traffic normalizer).

Based on the technology of traffic normalizers IDSs started to evolve into *inline intrusion detection systems*. An inline IDS is directly in the data path and packets are either routed through or bridged across it, both allowing to normalize the traffic to a certain degree. Next, inline IDSs that allow to block attacking packets are called *intrusion prevention systems* (IPS). Table 2.1 lists for the concepts of IDS and IPS the architectural differences, the disadvantages and the strengths.

First, the approaches differ in the realization of the E-box. IDSs do a passive packet analysis whereas IPSs do an active packet analysis routing all traffic across the IPS itself and, consequently, an IPS must at least be supplied with two network interface cards (NICs). The system receives the packets on one NIC, analyzes them and sends them—in case of benign communication activities—out via the other NIC. In addition, the realization of the E-box determines the behavior of the IPS in overload situations and in the unlikely event of an IPS crash. In this context an E-box can either be designed to drop packets (fail-close) or to forward them un-inspected (fail-open).

Second, IPS and IDS differ in the functionality of the R-box(es). Generally, an IDS is not designed to prevent attacks rather than to inform about an ongoing one. IDS response mechanisms—if available at all—are limited to actions like the sending of a *TCP RESET* packet or the reconfiguration of a firewall. Obviously, this results in a race condition between attacker and response mechanism (often in favor of the attack). In contrast, the purpose of an IPS is to efficiently stop attacks by doing access control up to the application layer meaning that packets which belong to an attack can actually be filtered by the IPS. In other words, an IDS analyzes a copy of the network traffic whereas an IPS analyzes the original traffic. However, the evaluation method whether an attack is taking place—the realization

Table 2.1: A comparison of IDS and IPS

| | **IDS** | **IPS** |
|---|---|---|
| **E-box** | Copies network traffic and forwards it to the A-box(es) | Redirects the original network traffic to the A-box(es) |
| **A-box** | For the purpose of identifying an attack IDS and IPS both use the same set of detection techniques/algorithms | |
| **D-box** | IDS and IPS both provide the same set of logging capabilities | |
| **R-box** | Commonly integrated response mechanisms—if present at all—are limited to the re-configuration of a firewall or the sending of a *TCP RESET* packet in order to stop the attack session (only applicable to TCP based attacks). But obviously, this results in a race condition between attack and response mechanism (often in favor of the attack). | Theoretically, an IPS provides the same set of response mechanisms like an IDS but these are not required as an IPS additionally provides the capability to drop/filter the attacking packets. The IPS approach allows for a fail-close realization. |
| **Weaknesses** | | |
| | • vulnerable to traffic ambiguities<br>• fail-open | • impact on the network performance<br>• traffic normalization might affect end-to-end semantics |
| **Strengths** | | |
| | • does not affect normal network operations | • allows for efficient protection<br>• reduces traffic ambiguities<br>• fail-close (optional) |

of the A-box(es)—do not differ between IDS and IPS because both can use the same set of anomaly- and misuse-detection techniques respectively. Analogous is the realization of the D-box(es) as there is no conceptually difference between IDS and IPS. An important rule of thumb is that a good IPS can always be operated as an IDS.

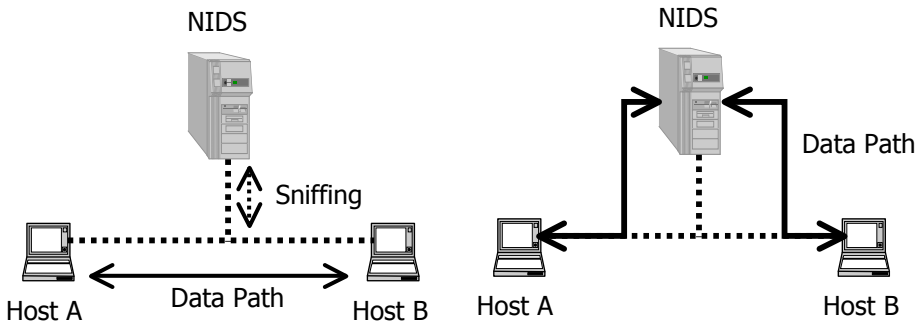
Summarizing, an IPS allows to efficiently protect end-systems against attacks and in

Table 2.2: A Comparison of existing Security Technologies

| Technology | Function | Strength | Drawback |
|---|---|---|---|
| **Firewall** | blocks or permits traffic according to specified rules | real-time filtering | no attack detection functionality |
| **Misuse-based IDS** | screens gathered data for known attack traffic patterns | clear identification of attacks | fails to detect unknown attacks, in addition, multiple evasion techniques exist |
| **Anomaly-based IDS** | observes gathered data for anomalies | detection of unknown attacks | high false alarm rate |
| **IPS** | screens original network traffic for attacks making use of anomaly and misuse detection methods | provides an efficient protection against attacks as malicious packets are immediately filtered | the operation of an IPS has an impact on the network performance as each packet is analyzed in terms of malicious content before being forwarded |

addition, an IPS can be realized in a fail-close manner. The disadvantages in the operation of an IPS are that an IPS has an impact on the network performance as each packet is analyzed in terms of malicious content before being routed. Further, an IPS normalizes the network traffic to a certain degree which might affect end-to-end semantics.

The concepts of IDS and IPS are not contradicting. An IDS does not become obsolete due to the existence of an IPS. The intention of an IDS is to monitor a networking/computing environment in order to *identify* malicious activities like an ongoing network attack. An IDS does not provide an efficient protection against network attacks whereas an IPS *prevents* attacks from becoming successful by filtering the corresponding packets. An IPS combines the blocking capabilities of a firewall with the more thorough packet inspection of an IDS. IDS and IPS can be operated in a complementary manner. The IPS is assigned to filter out the attacks and the IDS monitors if the IPS succeeded. Table 2.2 summarizes the distinctive features, the advantages and the disadvantages of the discussed techniques.




(a) Passive (b) Active

Figure 2.4: Packet Analysis

### 2.3.6 Approaches to Realize Intrusion Detection and Prevention Systems

In realizing an IDS or IPS two general approaches exist: hardware-based and software-based. Hardware-based intrusion prevention systems consist of special purpose hardware like Application-Specific Integrated Circuits (ASIC) and Field Programmable Gate Arrays (FPGA). In contrast software-based IPSs consist of General Purpose Processors (GPP) and potentially of Network Processors (NP). Hybrid approaches make use of special purpose hardware as well as of GPPs and NPs. A GPP is an integrated circuit (IC) which is not specifically developed for an application. The strength of a GPP is its flexibility, its generic instruction set allows to implement a great variety of applications. In contrast a network processor (NP) is an IC that contains an instruction set that is optimized for the processing of information contained in packets headers (OSI/ISO layers 2. 3 and 4). Consequently, header fields (e.g. IP-/TCP-header) are more efficiently evaluated by a NP compared to a GPP. But a GPP, for example, is faster in searching for attack patterns in a packet's payload. A Field Programmable Gate Arrays (FPGA) is an IC that can be programmed to do a set of operations on the data passed to it. Moreover, a FPGA can be reprogrammed in order to change the set of operations performed by it. But the task of reprogramming FPGAs is still complex [29]. Finally, an Application-Specific Integrated Circuit (ASIC) is an IC that is specifically designed for a purpose. It contains hard coded instructions which it performs on the data passed to it. An ASIC cannot be reprogrammed but in comparison to a FPGA it is faster and it is cheaper in case of high production volumes. The initial costs to launch the production of a specific ASIC are not negligible. An overview of the strengths and drawbacks of the mentioned components is given in Table 2.3.

### 2.3.7 Evaluation of Intrusion Detection and Prevention Systems

The evaluation of an IDS/IPS is the systematic determination of its value for a networking environment including answers to questions like how well does a given system detect intrusions and how much effort is necessary to tune this system for a given networking environment?

Initially, an alarm generated by an IDS/IPS does not inevitably indicate the happening

Table 2.3: A Comparison of Hardware used for IPSs

| Hardware | Description | Strength | Drawback |
|---|---|---|---|
| **GPP** | An IC that contains a generic instructions set; a GPP is not designed for a specific application but rather for a great spectrum of applications | The provided instructions allow for the implementation of a great spectrum of operations | Tradeoff between provided capabilities and performance |
| **NP** | An IC that is specifically designed for the processing of network traffic; contains a specific instruction set for processing layers 2,3,4 headers | Very efficient in analyzing packet's headers | Inefficient in analyzing payload |
| **ASIC** | An IC that performs a set of hard-coded instructions on the data passed to it | Very fast and cheap in case of high production volumes otherwise the costs per ASIC are high | Lack of flexibility; an ASIC is customized for a specific operation(s) and it is not possible to reprogram it |
| **FPGA** | An IC that can be programmed to perform certain operations on the data passed to it | Gain of flexibility compared to ASIC due to programmability | Lower performance compared to an ASIC; the process of programming a FPGA is still complex |

of an attack, it rather represents one of the following scenarios:

1. **alarm/true positive**: an IDS/IPS correctly identifies an attack which would harm the targeted end-system.
2. **false positive**:
   - an IDS/IPS thinks that the inspected network traffic belongs to an ongoing attack although current activity is benign.

- an IDS/IPS correctly identifies an attack but the attack is non-threatening or not applicable to the site.

3. **false negative**: a non-event and thus not correlated to an alarm, indicating that the IDS/IPS failed to identify an applicable attack.

Based on the given alarm categories it is possible to evaluate IDSs/IPSs by using one or more of the metrics that were introduced in [50, 85, 109]:

- **accuracy**: this metric represents that capability of an IDS/IPS to differentiate between legitimate and illegitimate actions (aka *false positive* rate). Namely, an IDS/IPS should not identify legitimate actions as illegitimate.
- **completeness**: an *incomplete* IDS/IPS fails to detect the entirety of all attacks. In practice, it is hard to evaluate this metric because it would require knowledge of unidentified attacks, which is not available.
- **performance**: the performance of an IDS/IPS is the rate at which audit events are processed. This is the decisive factor whether an IDS is qualified for doing real-time intrusion detection. Regarding an IPS the performance is the limiting factor in terms of how much traffic can be analyzed.

Besides the given metrics, an IDS/IPS should also respect the properties of:

- **fault tolerance**: an IDS/IPS itself must be resistant against attacks.
- **scalability**: an IDS/IPS must always be able to process all events without ignoring any. With respect to steadily growing traffic volumes and increasing number of attacks this is an important property especially for network-based IDSs/IPSs.

Despite the given and measurable metrics like performance it remains difficult to compare multiple IDSs/IPSs. First, one IPS can be tuned for a test such that it just tests for those attack patterns that will be included in the network traffic whereas other IPSs check for all known attack pattern. Or, some system may keep state whereas others do not. Second, the traffic mix used for the evaluation of the IPSs has a strong impact on the individual results. An example are single packet attacks versus multiple packet attacks or the fraction of IP fragments.

The issue of evaluating IDSs is still an open topic in the research community. Up to now no standard has been specified. An overview of is given in [94]. The need for an standardized evaluation process for IDSs was also recognized in 1998 by the *Defense Advanced Research Projects Agency* (DARPA) which in turn initiated with the help of the *MIT Lincoln Laboratory* (MIT/LL) the *DARPA Intrusion Detection Evaluation* project. The project created—based on network traffic that was collected at an Air Force base—the first formal and repeatable evaluation process for IDSs [44, 64]. But the traffic traces collected in 1998 do not any longer represent a realistic traffic mix which still can be observed in today's networks.

Another issue is addressed by Axelson who explains the *base-rate fallacy phenomena* for intrusion detection systems [20], saying that the effectiveness of an IDS is determined by the ability of the system to suppress false alarms. He further deduces that an effective IDS has to achieve a false positive rate in the challenging order of $1 \cdot 10^{-5}$ per event.

## 2.4 Gathering Network-Related Information

An attack requires the existence of a concrete vulnerability to succeed. For example, the *Code Red* attack exploits a buffer overflow of certain versions of Microsoft's IIS (with enabled indexing service). To qualify IDS alerts into false positives and true positives it is essential to know whether the attack that triggered the corresponding alarm can actually harm the victim. The process of identifying and quantifying vulnerabilities in a system is called *vulnerability assessment* (VA). In addition, VA is also used to appropriately configure IDSs.

Generally, three techniques exist to gather vulnerability information: *active scanning*, *passive fingerprinting* and *cooperation*. An active vulnerability scanner sends specifically crafted packets to well defined addresses and evaluates the replies. The scanner either operates in an *intrusive* or *non-intrusive* manner. The former technique identifies the vulnerabilities of an end-system by actually exploiting them. On the one hand the results gained are accurate but on the other hand this might result in a service-/system-crash. In contrast, a non-intrusive network scanner identifies a vulnerability on the basis of the type and version of a running service. Consequently, the results produced by a non-intrusive network scanner are not as accurate as the results of an intrusive one, but normal network operations are not affected by it. Either way, the gathered vulnerability information is only as current as the latest scan and accordingly, the questions arise of when and how often to scan? In addition, an active scanner creates traffic for the purpose of identifying end-systems. What might result in a huge amount of scanning tasks for large networks.

The passive fingerprinting technique uses a network interface card (NIC) in promiscuous mode to sniff the network. On the one hand the passive fingerprinting technique does not collide with normal network operations but on the other hand it is impossible to predict the point in time when all end-systems of an network will be identified. End-systems that do not communicate cannot be detected by the passive fingerprinting technique, but still, those systems are potential victims.

The cooperation technique requires an agreement between end-systems and network analysis tool. One approach is that each end-system runs a small application that registers at the analysis tool and subsequently, it provides the analysis tool with the required knowledge in terms of operating system and running applications. In a second approach, the network analysis tool logs into the end-systems and autonomously gathers the relevant data, but the end-systems must provide a login to the network analysis tool. The advantage of the cooperation technique is accuracy. Operating system and running applications—including version number and patch level—are precisely identified. But not all users feel comfortable about either running a small network analysis client application on their machines or providing a login to the network analysis tool.

### Host Discovery

This section describes existing methods—starting with the active ones—to detect end-systems in a given network. The first method is to query the *Domain Name System* (DNS) which stores information associated with domain names in a distributed database. For example, it translates Internet domain names into IP addresses and vice versa, a DNS reverse lookup resolves an IP address into a domain name. But not every end-system of a network is

necessarily registered in the Domain Name System and in addition, the existence of such an entry does not guarantee that the corresponding system really exists.

Moreover, the *Internet Control Message Protocol* (ICMP) provides mechanisms to discover the hosts in a network. A straightforward approach—also known as *ping sweep*—is to send an *ICMP Echo Request* message to each potential target IP address. In accordance to RFC 1122 [31], a running end-system should answer by sending an *ICMP Echo Reply* message. The same result can be achieved by sending *ICMP Echo Requests* to the specific addresses $x.x.x.0$ (network address) and $x.x.x.255$ (broadcast address). But two problems may arise. Systems can be configured to remain silent upon the reception of an *ICMP Echo Request* message and *ICMP Echo Requests* addressed to specific network addresses can be filtered.

A more enhanced technique to discover the end-systems of a network is to provoke *ICMP* error messages:

1. an *ICMP Destination Unreachable Protocol Unreachable* message (Code 3 Type 2) is sent by a host in case that it received an IP packet with an unused protocol number. Protocol numbers $134 - -254$ are currently not used.

2. an *ICPM Destination Unreachable Port Unreachable* message (Code 3 Type 3) is sent by a host in case that it received an UDP/TCP packet which is addressed to an unused port.

3. an *ICMP Time exceeded Fragment Reassembly Time Exceeded* message (Code 1 Type 11) is sent by a host in case that it only received the first part of a fragmented IP packet. After a certain period of time the host discards the first part and generates the mentioned error message.

It must still be taken into account that ICMP is not reliable and, accordingly, it might happen that an ICMP error message get lost on the way. Hence, in cases that no response is received from a scanned address the scan should be repeated for the purpose of accuracy.

In contrast to the active methods presented above, the hosts of a network can also be passively identified by observing source and destination addresses of the network traffic. But passive host identification is accurate only if all hosts produce network traffic that can be observed.

#### Remote Service Discovery

Now, once the end-systems of a network are known, it is of interest what services are offered by each host. The most common active technique to identify running services is to do a portscan. The seminal paper of Fyodor [59] introduces:

- TCP connect scanning,
- TCP SYN scanning,
- TCP FIN scanning,
- TCP ftp proxy scanning,

- SYN/FIN scanning using IP fragments,
- UDP recvfrom() scanning,
- UDP raw ICMP port unreachable scanning.

The two first scanning methods are shortly explained in the following, a detailed description of all listed methods is given in [59]. A *TCP connect* scan executes the *connect* command of the local operating system which tries to establish a TCP connection with the specified destination (IP-address and TCP port). The system-call succeeds in case that the counterpart listens on the addressed port. Moreover, the usage of non-blocking system calls allow to scan in parallel many ports on the target machine within a short period of time. A further method to identify open ports of a remote host is to do a *TCP SYN* scan. Here, the scanner sends connection requests, *TCP Syn* packets, to the specified addresses. The reception of *TCP Syn Ack* message indicates that the system scanned runs a service that listens on the corresponding port whereas a *TCP Rst* packet indicates the contrary. Subsequently, the scanner closes the half-opened connection by sending a *TCP Rst* packet.

The question is what service is bound to which listening port. To identify the listening ports of an end-system, theoretically the enormous amount of $65,536$ TCP- and $65,536$ UDP-ports must be scanned. But usually not all ports are of interest as many services were initially assigned by the *Internet Assigned Numbers Authority* (IANA) to a distinct port number [114]. For instance, on *Linux*-based systems the file "/etc/services" contains information about the pairing of port number and service. Nevertheless, sometimes well known services are configured to run on non-standard ports in which case it is more difficult to identify the running service. An approach is to try to connect with a set of upper-layer protocol clients (e.g. *TELNET*, *SSH* or *FTP*). Consequently, a scanning tool requires a client for each protocol to be tested slowing down the scanning process. Summarizing, the disadvantages of an active scanner are:

- Difficulty to discover end-systems that are rarely online,
- Difficulty to identify services that are not bound to standard ports and
- An active scanner may affect normal network operation:
  - scanning traffic and
  - unpredictable behavior of scanned machines.

Another possibility that does not suffer under the above mentioned weaknesses is to passively identify services. For the purpose of identifying TCP-based services, a network sniffer—a network interface card running in promiscuous mode—screens the network traffic for *TCP Syn Ack* packets. Subsequent, the network sniffer searches those TCP messages for a set of keywords/protocol patterns allowing to determine the service that is bound to the related source port. But the detection of UDP-based services—besides services bound to the standard ports—is more difficult. The fact that the UDP transport protocol is connectionless complicates the task of identifying whether a UDP packet contains a request or a response. Hence, all UDP packets must be analyzed for specific keywords/indications in order to identify UDP-based services that are bound to non-standard ports.

Recapitulating, a network sniffer enables the identification of end-systems that are rarely online and it allows to identify TCP-based services that are bound to unconventional port numbers. The downturn related to the operation of a passive network sniffer are:

- Services cannot be identified until they start to communicate and
- Identification of UDP-based services that are bound to non-standard ports.

Further, a passive network sniffer identifies services as soon as these start to communicate and hence, it is impossible to predict how long it takes to get a complete picture of all services that are offered in a network.

**Remote Operating System Identification**

Another aspect in constructing an accurate view of a network is the active or passive determination of the operating system that is running on an end-system. Active and passive scanners exploit the fact that every operating system's IP stack has its idiosyncrasies. An active scanner sends malformed packet to the target system and subsequently it analyzes the responses which vary from OS to OS.

For example, the provocation of *ICMP* error messages can be exploited to collect hints for OS-specification. Normally, an error message quotes the header and at least 8 bytes of the offending packet. But again, some operating systems quote more than 8 bytes and others quote inaccurately (e.g. Linux adds 20 bytes). Some OSs even quote the IP header incorrectly, e.g. the checksum and the *TTL* field are two candidates. Taken these effects together, with up to five tests it is possible to specify groups of OSs. For instance the operating systems *Windows 95*, *Windows 98* and *Windows NT 4.0* can be identified as follows. In a first step an *ICMP ECHO* request is sent to the target host and subsequently, the corresponding *ICMP ECHO* reply is analyzed. A *TTL* value of 32 clearly identifies a *Windows 95* system. Other *Windows* operating systems set by default a *TTL* value of 128. In a next step an *Address Mask request* message is sent to the target host. A host which is running *Windows 98* or *Windows NT 4.0* under *Service Pack 4* would reply to the request. To distinguish between these two Windows operating systems a *Timestamp request* message is sent. If the host is running *Windows NT*, it would not respond to the request.

In [121] Spitzner describes how the OS of a host can be passively evaluated based on the fields: *TTL*, *Window Size*, *DF* and *TOS*. For example, a *TTL* value of 45, a *Window Size* of $0x7D78$, the *DF*-bit set and a *TOS* value of $0x0$ points towards a Linux box, potentially *Red Hat 6.0* kernel 2.2.5.

# Chapter 3

# State of the Art of Network Based Intrusion Detection and Prevention

This chapter presents a related selection of intrusion detection and prevention systems. Section 3.1 introduces host-based intrusion detection and prevention systems and Section 3.2 continues with a detailed presentation of network-based systems. Afterwards, Section 3.3 discusses limitations of the presented approaches and gives an overview of approaches taken to address these. Finally, Section 3.4 recapitulates the state of the art and identifies open issues in intrusion detection and prevention.

## 3.1 Host-Based Systems

**Tripwire** [9] emerged in 1992 and it is today one of the best-known and most widely-spread HIDS. It exists in three versions: **Open Source Tripwire**, **Tripwire for Servers** and **Tripwire for Enterprises**. The first is suitable to monitor a small number of servers where centralized control and reporting is not needed. Both remaining versions provide a centralized control station as well as professional support. The main principle used by all three versions is the same: each Tripwire entity observes the integrity of chosen files by creating a secure database of file and directory attributes. The database is used to detect changes of files which indicate an attack (e.g. the replacement of the login application). Thus, Tripwire is only capable of detecting an attack if it changes a supervised file and after it happened. Open-source HIDS that make use of the file integrity testing method are also known as *System Integrity Verifier* (SIV). Further SIVs are: Aide [1], md5deep [5], Samhain [138] or AFICK (Another File Integrity CHecker) [60].

**Snare** (**S**ystem i**N**trusion **A**nalysis & **R**eporting **E**nvironment) [8] is an open-source HIDS for Linux-based systems which provides a C2-style logging system. C2 is a security standard specified by the US Government National Computer Security Council (aka the *Orange Book*) [10]. The standard differentiates between the security levels:

- **D**: a non-secure system
- **C**:
  - C1 requires user log-on, but allows group ID

  - C2 requires individual log-on with password and an audit mechanism.

- **B**:
  - B1 access is based on standard Department of Defense (DoD) clearances
  - B2 guarantees the path between the user and the security system and provides assurances that the system can be tested and clearances cannot be downgraded
  - B3 requires that the system is characterized by a mathematical model that must be viable.

- **A1**:
  - requires DOD clearance levels
  - requires that a system can be characterized by a mathematical model that can be proven.

Most Unix-based systems fulfill the C1 standard by default but further on, they can be upgraded such that they satisfy the C2 requirements (e.g. Snare does so). In practice, this means that certain forms of system calls performed by every process are logged with an unique audit ID. This includes calls to open and close files, change directory, alter user process parameters, and so on. The audit ID allows to identify the user who initiated the supervised operation. In accordance to this, Snare consists of three parts: a kernel-module, a user-space daemon and a graphical user interface. The kernel-module monitors which process executes which system call and further who possesses the corresponding process. That information is passed to the user-space damon which is responsible for the logging. Finally, the graphical user interface is used for configuration and reporting issues.

The **Linux Intrusion Detection System** [4] is better known as LIDS and despite its name it is not a real host-based IDS. First of all, it allows to limit the capabilities of root on a given system. On common Linux-based systems root is allowed to do anything he wants to and, users/processes (especially malicious ones) that have root privileges pose a huge security risk. LIDS allows to limit the damage that could be caused, for example, by a malicious user with root privileges, by enhancing the security of Linux based systems. In practice, LIDS introduces access control lists (ACL) that allow for a fine-grain specification of the authorizations of users and root. LIDS triggers an alarm in case that a user/root violates a specified rule.

**OSSEC HIDS** [6] is another open-source host-based intrusion detection system for Unix-based systems that does integrity checking, services detection and alerting. In addition, it provides the capability to execute a specified binary as a response to a detected attack, to monitor multiple systems (agents) on a central system (server) and to detect rootkits. A rootkit is a collection of tools that an intruder brings along to a victim computer after gaining initial access. It allows him to conceal his presence, processes and files on a system to the legitimate user/administrator. Rootkits are categorized into kernel level, library and application level rootkits:

- Kernel level rootkits either add code to the kernel or replace existing kernel routines; on a Linux-based/Windows-based system this can be achieved via the means of loadable kernel modules/device drivers.

- Library rootkits modify or replace system libraries in order to manipulate a chosen set of system calls.
- Application level root-kits modify or replace existing application binaries on a system.

For the purpose of easily gaining access at a later point an intruder most-often installs a back door—a means of remote access to a system that bypasses security mechanisms —on a compromised system and consequently, this results in a "unusual" open port. Accordingly, an approach to detect an installed rootkit is to do a portscan from a remote system. OSSEC HIDS does this in a slightly different manner, it tries to bind to every port on the local machine in order to detect a suspicious open port.

**PortSentry** by Psionic Technologies (now Cisco) [111] is a HIDS that identifies suspicious port connection requests and provides means to invoke countermeasures. The assumption made is that as an attacker requires knowledge about operating system and/or open ports/ running services on a system and, accordingly, a system scan most often indicates an approaching attack. PortSentry is able to detect a variety of scans, for example a so-called Null-scan. Thereby a TCP probe packet with no flags set is sent to a specific port. If the port is closed then a RST/ACK packet is received, whereas if the port is open no response is received. PortSentry provides the following set of responses to a detected scan:

1. the printing of an alert message to *syslog*,
2. the dynamic setting of a route in order to forward all traffic originating from the scanning IP address to a non-existent destination,
3. the blocking of the scanning IP-address by adding an appropriate rule to a local packet filter/firewall
4. the insertion of the scanning host into */etc/hosts.deny*. Hosts that are listed in /etc/hosts.deny cannot connect to any service as these are monitored by a local *TCP Wrapper*.

It must be considered that responses 2, 3 and 4 can be exploited by an attacker who continuously forges his IP source address. In an extreme case this results in the blocking of all network traffic.

The presented host-based intrusion detection systems make use of the techniques system integrity verification, log analyses and system-call monitoring. Some HIDS have in addition further capabilities: for example LIDS allows to restrict the power of root or OSSEC HIDS provides capabilities to detect rootkits. In the remainder of this section the focus is on network-based intrusion detection and prevention systems.

## 3.2 Network-Based Systems

This section discusses network-based intrusion detection and prevention systems. Initially, stand-alone systems that are designed to run on a single machine are introduced. Afterwards, the attention is drawn towards distributed approaches, including systems that make use of mobile agents as well as systems requiring programmable routers.

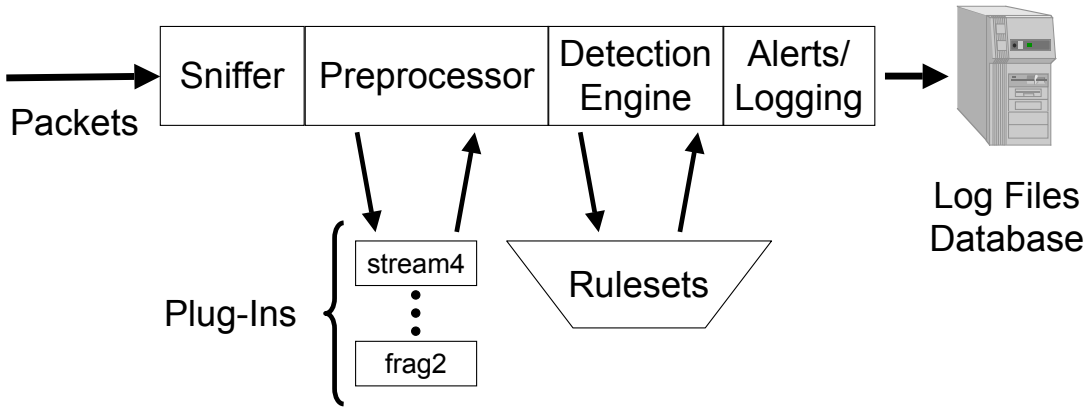



Figure 3.1: Snort: Packet processing

### 3.2.1 Stand-alone Systems

**Snort** [24] is the most popular open-source network-based intrusion detection system. Initially, Snort started in 1998 when Marty Roesch, the author, realized a packet sniffer for Linux-based systems. One year later, in 1999, he added a first small signature-based analysis engine to Snort. In the course of time, Snort developed into a full-grown (mainly) signature-based NIDS.

Snort can be operated as sniffer, packet logger or network-based intrusion detection system. When Snort is operated as sniffer it displays the current network packets on a local console. The packet logging mode additionally allows to write the network packets into a file. Further, Snort can be operated as a signature-based NIDS. Figure 3.1 depicts the Snort architecture[1] and the corresponding packet processing.

Basically, Snort consists of a sniffer, a preprocessor, a configurable amount of plug-ins, a detection engine, a corresponding ruleset and a logging mechanism. The sniffer is used to tap the network traffic. As already mentioned, a sniffer creates copies of the network packets it sees and does not have any impact on the network performance. Next, the sniffer forwards the network packets to the preprocessor which prepares the packets for the detection engine. The preparation itself is done by a configurable amount of plug-ins. Two exemplary plug-ins are the *stream4* and the *frag2* plug-ins. The former integrates the concepts of TCP statefulness and session-reassembly into Snort. TCP statefulness means that only packets belonging to an established TCP connection are forwarded to the detection engine; all others are neglected. This plug-in was realized as an answer to tools like *Stick* [61] or *PCP* [105] which were designed to trigger huge amounts of alarms on Snort. Session reassembly refers to the capability to keep a memory of past TCP packets of existing TCP connections in order to detect attacks that span multiple packets.

Further, the Snort-ruleset is organized by category, examples are: Trojan horses, buffer overflows and access to various applications. Since new attacks emerge it is necessary to

[1]The discussed architecture refers to Snort 2.0

regularly update the ruleset. A Snort-rule consists of (see Figure 3.2):

- a rule header specifying rule action, protocol, source and destination,
- a rule body determining the pattern to look for in the network packets.

The rule header defines—based on the criteria of protocol, source and destination—if a network packet must be checked against that specific rule and in addition, it assigns the action to take (rule action) in case of a match. The next line contains a fictitious Snort rule:

```
alert udp any any -> $INTERNAL 21974 (msg: "Bad Worm Backdoor";)
```

This rule triggers an alert (as the rule action is *alert*) with the message "Bad Worm Backdoor" in case that a UDP packet originating from any source (specified by *any any*) is addressed to an internal IP address (defined by the variable *$INTERNAL*) with the destination port 21974. Finally, in case that the data matched a rule (with a rule action: log, alert or activate) then Snort logs the corresponding information and/or triggers the appropriate alert. Snort provides the capability to send alerts via UNIX sockets or to trigger Windows Popups. In addition, it is possible to integrate further output plug-ins between detection engine and the alert/logging facility of Snort. At last, **Snort-Inline**—a modified version of Snort—processes network packets inline and in addition, it can be configured as an intrusion prevention system.

**Bro** [106] developed at the Lawrence Berkeley Labs is another well-known Unix-based open-source NIDS which does a passive protocol analysis. The principal structure of the Bro system is depicted in Figure 3.3. The system is divided into two components:

- the event engine filters irrelevant packets of the incoming packet stream (from the libpcab) and subsequently transforms the relevant ones in a stream of higher-level network events.
- the policy script interpreter that is used to express a site's security policy.

Moreover, Bro uses the functionality that is provided by the libpcap—an application programming interface for packet capturing—to restrict the amount of packets that are captured from the network. At the beginning of the Bro project, libpcap was configured to capture any packet addressed to applications for which Bro was capable to perform specialized analysis (Finger, FTP, Ident, Telnet, Rlogin and Portmapper) as well as any TCP packet with the SYN, FIN, or RST control bits set (for the purpose of identifying start- and end-points of

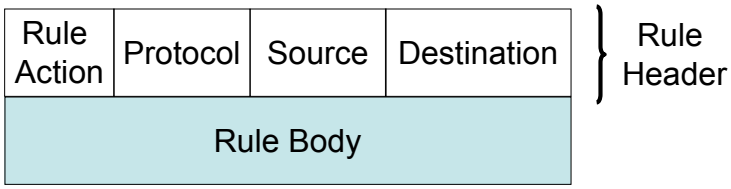



Figure 3.2: The Snort rule structure

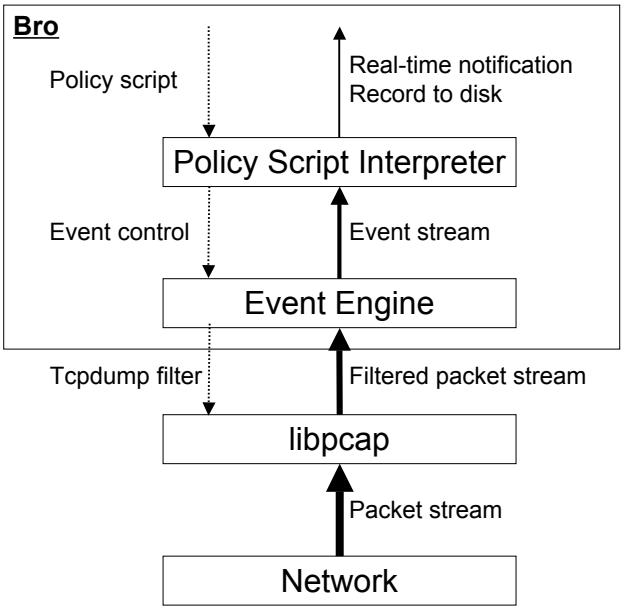



Figure 3.3: The structure of the Bro system

TCP connections). Packets that correspond to the described criteria are passed to the event engine.

The event engine can be compared with the preprocessor of Snort as it is responsible for packet reassembly and TCP connection state tracking. Besides this, it checks if a packet header is in conformity with the protocol specifications, for example, it verifies the header checksum. Packets which do not pass this initial test are discarded by the event engine. The remaining packets are transformed into events and forwarded via a first-in-first-out (FIFO) queue to the policy script interpreter.

Actually, the policy script interpreter detects attacks based on rules. Therefore, it executes policy scripts—written in the specific Bro language and which contain the rules describing what sorts of activities are deemed malicious—against the stream of events. Consequently, this is a more generic definition of an attack signature allowing for a more abstract analysis. A policy script for example is able to create new events, to log data, to send real-time notifications or to record data to disk.

**ARPWatch** [87] also developed by the Network Research Group at Lawrence Berkeley National Laboratory addresses the issue of *ARP spoofing*. The *address resolution protocol* (ARP) provides a mechanism to map an IP address to a host's hardware address. Hence, in case that an attacker has gained control over a system of an Ethernet LAN, then he can use *ARP spoofing/ARP (request) poisoning* to eavesdrop/manipulate communications between two systems of the switched network. In detail, an attacker pretends to be the legitimate receiver for IP packets, addressed to a specific IP address, by sending appropriately spoofed ARP messages. ARPWatch monitors and memorizes Ethernet/IP address pairs and in addition, the system provides the capability to inform via email about changing pairs.

The discussed systems mainly focus on signature detection and additionally contain a limited set of anomaly detection functionalities. They are intended to run on a single machine and, accordingly, an automated and intelligent distribution of security operations over a set of machines is not possible.

### 3.2.2 Distributed and Coordinated Systems

An initial project towards distributed intrusion prevention is described in [74]. The authors explain the implementation of a distributed firewall system. Usually, a network firewall is installed at the point where the protected subnetwork is connected to the Internet. This concept is fine as long as one point of contact exists between the trusted subnetwork and the Internet. In this context, the locations where to place the firewall entities can easily be determined as the points of contact between the trusted subnetwork and the untrustworthy Internet. Furthermore, the authors focus on the distribution and synchronization of policies.

Further, much work is done towards the distributed collection of intrusion relevant data. For example, **Prelude** [7] is a *hybrid* IDS which means that it is both a network-based and a host-based IDS. Basically, the system is made up of the following components:

- sensors which gather information. By default, Prelude comes with the following set of exemplary sensors:
  - *Prelude LML* a host-based sensor which monitors logfiles like syslog; it can also receive syslog messages from remote hosts
  - *Prelude NIDS* a network based IDS
  - *Snort*—a patch which enables Snort to become a Prelude sensor is provided on the project's web-side.
  - etc.
- manager which process the data gathered by the sensors. Managers can forward alerts to other managers or to Counter Measure Agents. In a distributed environment managers can act as a relay to the dedicated central manager.
- Counter Measure Agents which receive and process the alerts sent by the managers.

A particularity of the Prelude IDS is that it allows to integrate other security tools like for example Snort as sensors. For this purpose Prelude provides the *libprelude* library which provides an Application Programming Interface (API) to create *Intrusion Detection Message Exchange Format* (IDMEF) based events. "The purpose of the Intrusion Detection Message Exchange Format is to define data formats and exchange procedures for sharing information of interest to intrusion detection and response systems, and to the management systems which may need to interact with them" [48].

The architecture of the Prelude NIDS is depicted in Figure 3.4. To capture packets Prelude uses also the capabilities offered by libpcab. In addition, similar to Snort and Bro, Prelude does initial checks on the network packets in order to filter irrelevant ones and IP de-fragmentation as well as TCP stream reassembly is immediately done after the capturing of network packets. The second stage of the Prelude NIDS consists of protocol plug-ins that

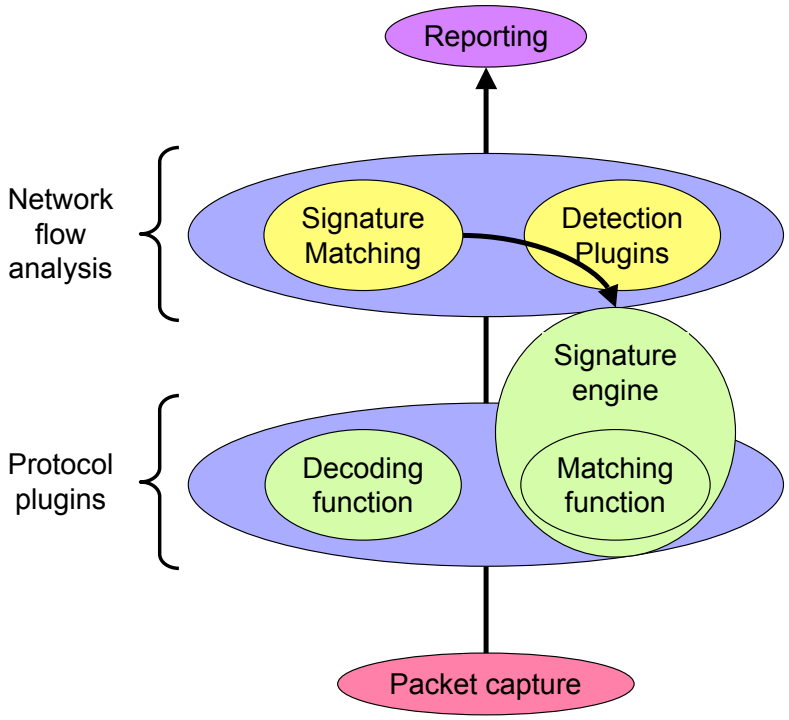



Figure 3.4: The architecture of the Prelude NIDS

are capable of decoding several higher-level protocols. For example, the integrated *HTTP Decoding Plug-in* searches for HTTP requests in TCP packets and transcodes/normalizes all included escaped characters (HTTP-, UTF-8- and Unicode-escaping). Subsequently, the normalized traffic is forwarded to the next level of the Prelude NIDS which actually screens the packets for attacks, for instance, the *Scan Detection Plug-in* triggers an alert in case that too many connections on different ports are detected within a given amount of time. Another example is the *Snort Rule Plug-in* that allows to use the original rule set of Snort for signature matching. Alert reporting is done via the mentioned libprelude.

Summarizing, Prelude is a hybrid IDS meaning that data is captured on end-systems and in the network. Further, effort was made to integrate further security tools like for example Snort as sensors into the Prelude framework. It turned out that Snort is the preferred NIDS sensor (compared to the described Prelude NIDS) as it provides more functionality and, in addition, its ruleset is regularly updated. The project shows how a cooperation of multiple security tools can be realized in order to achieve a better overall protection of a network.

**Shadow** is the open-source and Unix-based **Secondary Heuristic Analysis for Defensive Online Warfare** system [54] developed by the US Navy. It consists of sensors capturing network traffic and an analysis station actually doing intrusion detection based on the data gathered by the sensors. In contrast to the NIDS discussed so far, Shadow uses a technique called *traffic analysis* doing network layer anomaly detection rather than content analysis for the detection of attacks. In addition, since Shadow was designed as a supplement

to a misuse-based network intrusion detection system it cannot detect simple string based attacks like a *phf*-attack against a WWW-server. Consequently, the analysis station is not interested in the packets' payload and accordingly, the sensors only capture IP-headers using *tcpdump*. In addition, each sensor stores the network traffic in one hour chunks which are regularly downloaded by the analysis station (no realtime detection). Furthermore, to detect traffic anomalies the analysis station applies a set of filters on the downloaded data. An exemplary filter collects information about how many different internal IP addresses and/or ports each external source address contacts in order to identify scan attempts. Another filter script is executed every 24 hours to generate a daily page of IP statistics.

The **Domino** (**D**istributed **O**verlay for **M**onitoring **I**nter**N**et **O**utbreaks) project [140] proposes an architecture for a globally distributed intrusion detection system on the basis of an overlay network consisting of heterogeneous systems. The project focuses on the detection of Internet outbreaks like, for example, the identifying of a yet unknown and self-distributing worm; misuse-based NIDSs would fail to detect such a worm as so far no attack signature exists. The system architecture includes an axis overlay, satellite communities and terrestrial contributors. Each axis node runs a honeypot which is used to monitor unused IP addresses. Satellite nodes and terrestrial contributors gather data from locally running NIDSs and firewalls. In contrast to terrestrial contributors satellite nodes are capable to do alarm clustering. All data gathered by terrestrial contributors and satellites is forwarded to the axis overlay which subsequently analyzes the data.

The **Indra** project [77] follows a distributed approach to network intrusion detection and prevention. A prerequisite is that all hosts on a network run an Indra security daemon. According to the PhD thesis of Howland [71], it takes a large number of attack attempts before an attack finds a vulnerable machine. Most likely, the unsuccessful attempts could be observed by the Indra clients which subsequently inform the other Indra clients of the network in order to invoke countermeasures (e.g. blocking of the corresponding IP address). This assumes that self-spreading attacks try to compromise further end-systems in a brute-force manner, which is not always true; some worms use the technique of scanning to detect vulnerable systems which will subsequently be attacked. In contrast to this approach, our philosophy is not to modify the end-systems but to do intrusion prevention on routers in the network.

### Agent-Based Systems

The **Intrusion Detection Agent System** (IDA) [19], developed by the *Information-technology Promotion Agency* (IPA) in Japan, is an IDS that uses mobile agents to trace intruders, gather intrusion-related information along the intrusion-route, and decide whether an intrusion has occurred. To detect intrusions IDA watches for events related to attacks which the authors name *Marks Left by Suspected Intruder* (MLSI). An exemplary MLSI is a modification of a security sensitive file like, for example, the system's password file. But, the observation of one MLSI cannot mandatorily be equated with an attack (perhaps a legitimate user changed his password). IDA consists of a manager, sensors, bulletin boards, message boards, tracing agents, and information-gathering agents (see Figure 3.5).

The manager detects intrusions by analyzing the data that is gathered by the information-gathering agents (IA). Moreover, it coordinates the mobile agents and the bulletin board, and

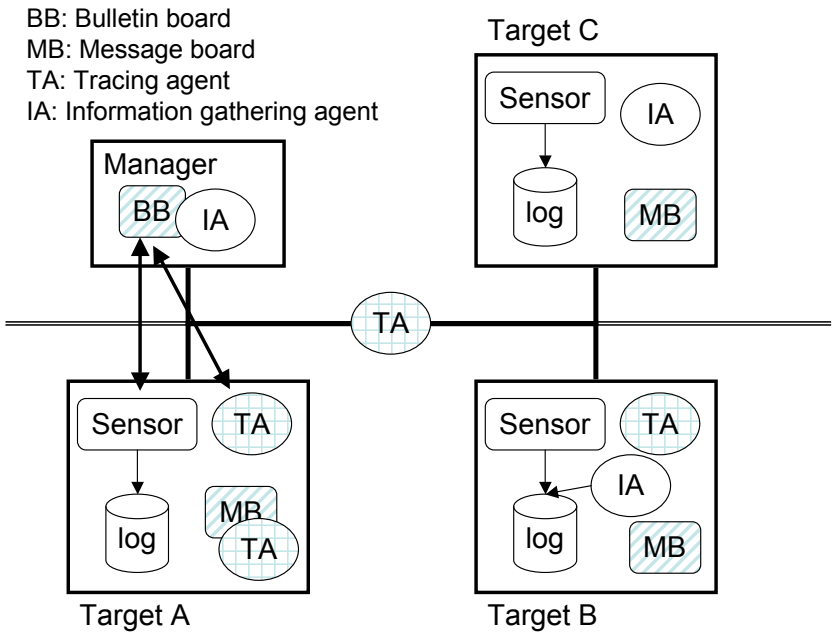



Figure 3.5: The Intrusion Detection Agent System (IDA)

finally, it is the interface between the IDA system and the administrator. A sensor resides on each target system and at that place it monitors the system's log file to spot MLSIs which it reports to the manager. On receiving a MLSI the manager delegates a tracing agent (TA) to the corresponding target system. The TA activates an information-gathering agent (IA) on the target system and tries to identify the users which caused the MLSI remote site. For this purpose the IDA system accumulates information about network connections and running processes in advance. Further, the TA tries to move to the user's remote site. There it activates another IA. In case that the TA cannot move any further it returns to the manager. The purpose of the IA is to gather data correlated to the MLSI. Subsequently, the IA returns to the manager and writes its report to the bulletin board (BB).

Recapitulating, the IDA system tries to identify intrusions based on log file monitoring. The authors do not precisely describe how multiple MLSIs are correlated in order to decide whether an attack occurred on a system. Moreover, the system requires that sensors and mobile agents have access to all target systems. In fact, a sensor must reside in advance on each target system and it must have access to the system's log file.

The **Intrusion Blocker based on Active Networks** (IBAN) [55] consists of a management station, mobile vulnerabilities scanners, and mobile intrusion blockers. A mobile scanner is an application designed to detect one particular vulnerability by looking at system fingerprints. If the scanner has found a vulnerable service an intrusion blocker is placed close to the corresponding system which inspects the traffic for the vulnerable service and blocks the traffic if it detects an attack attempt. IBAN focuses on the detection of automated known attacks. A scanner and a blocker are designed for one particular vulnerability. Consequently,

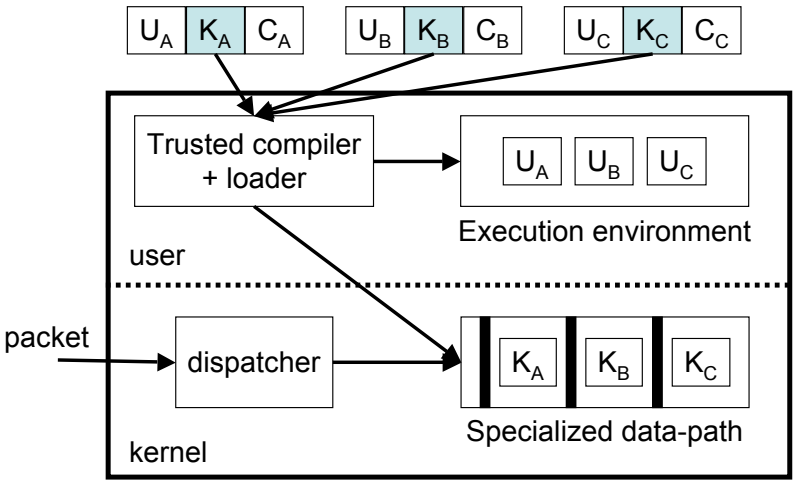



Figure 3.6: The FLAME architecture

numerous mobile applications could exist in an average network. Further, each application observes the traffic for a specific traffic pattern, thus each mobile application performs a big set of identical operations. As the authors of the paper write themselves, often it is more difficult to write a detection tool for a specific vulnerability than to provide an adequate defense mechanism. Consequently, IBAN would not deploy a defense mechanism close to a vulnerable host as long as it is not able to detect it.

**Systems on Top of Programmable Routers**

The approaches described in the following make use of enhanced/programmable routers.

The **FLAME** project [15] developed an architecture (Figure 3.6) that allows users to create their own individual network monitoring application that will be deployed on enhanced routers in the network—the proposals include active networks and an enhanced network interface card. In detail, a user application is a kernel module that passively monitors packets in real-time. A module consists of three components:

- kernel-level code $K_x$—responsible for time critical packet processing,
- user-level code $U_x$—offers additional functionality at lower time-scales—and
- a set of credentials $C_x$.

The user-code component can for example be used to open a standard socket in case that an application requires to communicate with another peer. Code is written in the C-like language *Cyclone* [78] and is processed by a trusted compiler. The set of credentials $C_x$ is used at compile to verify that the module is authorized to perform the requested actions. The kernel-code is supervised by additional code that performs the policy checks (represented by the black units in front of each $K_x$). An additional overhead is caused by doing run time

checks in order to ensure security. The project addresses the issue of executing user-code in kernel-space for the purpose of performance. A downturn is that the presented approach requires the disclosure of algorithms and programming code; source code must be transferred to the system.

The approach presented in reference [130] discusses the issue of intelligently placing packet filters on network routers in IP-carrier networks. A packet filter consists of a classifier and an action. The latter is either of type *allowing*—packets that match the classifier are forwarded—or *disallowing*—packets that match the classifier are dropped. A *false negative* occurs when malicious packets pass an allowing packet filter, and analogously, a *false positive* occurs when legitimate packets are dropped by a disallowing packet filter. The authors present an approach to deploy packet filters on network routers such to minimize the risk that is associated with a distributed packet filter configuration. To calculate the associated risk the specification of a set of parameters is required. The authors differentiate between source nodes $s_O$, adversary nodes $s_A$ and destination nodes $d$. Legitimate traffic is transported through the network via paths $p_O \in P_O$ whereas attack traffic propagates through the network via paths $p_A \in P_A$. Furthermore, parameter $\omega_{p_O}/\varphi_{p_A}$ denotes the probability of a false positive/false negative occurring on path $p_O/p_A$. Finally, $D_O/D_A$ denotes the damage related to a false positive/ false negative. The authors demonstrate that it is possible, on the basis of these parameters, to optimally deploy the packet filters on network routers also considering further aspects like rerouting issues.

The authors of [81] make use of enhanced routers that are capable to do content filtering. They developed a heuristic for the placement of content-filtering nodes in a network, and, given the placement of such nodes, designed a fully polynomial time approximation scheme (FPTAS) that maximizes the traffic carried by the network subject to the constraint that all traffic passes through a content filtering node at least once. The authors did not take into account the computational costs of doing content filtering; they rather considered costs in terms of money.

### 3.2.3 Commercial Network-Based Intrusion Detection Systems

For the sake of completeness, also commercial solutions like *RealSecure*, *Netprowler*, *Network Flight Recorder* and *NetRanger* must be mentioned. A detailed discussion of them is impossible due to a lack of publicly accessible documentation.

**RealSecure** from ISS [75] is one of the most widely deployed commercial IDS. It consist of managers and sensors. A manager is responsible for administrative and operational issues whereas a sensor—which can be a network-based one as well as a host-based one—-creates events.

**Netprowler** which originally was realized by Axent (now integrated into Symantec) [126] is known for its proprietary *Stateful Dynamic Signature Inspection* (SDSI) technology which separates intrusion analysis from the signature database. Accordingly, NetProwler allows to dynamically load new updates without taking down the sensor.

**NetRanger** from Cisco [40] consists of network-based sensors and an analysis station which is called "director". A sensor inspects the routers' system logs as well as the packets transmitted over the wire. In addition, NetRanger provides the capability to reassemble packets in order to detect attacks which are distributed over multiple IP fragments.

Finally, the **Network Flight Recorder** (NFR) [100] is a general purpose security device that allows to collect huge amount of data. For this purpose it uses a modified version of the *libpcap* which is an application programming interface for packet capturing developed by the *Lawrence Livermore Laboratory of the Department of Energy*. In addition, the filter language—used to configure the modified libpcab—allows among other things to create filters that monitor the network traffic for specific patterns.

## 3.3 Limitations of Intrusion Detection and Prevention Systems

In theory, the concept of IDSs and IPSs is clear and, numerous systems were implemented. But in practice, those systems have shown to have their limitations in terms of accuracy, performance and flexibility. This section analyzes the limitations and presents approaches that were developed to address them.

### 3.3.1 Accuracy of Intrusion Detection/Prevention Systems

Intrusion detection systems are known to produce large amount of alarms per day [80, 119] at which most of them are irrelevant ones in respect of the overall security of a network. For example, the notification about an ongoing *IIS*-specific attack targeted against an *Apache* web server is not relevant, since the targeted end-system is not vulnerable to this exploit. Further, in [20], Anderson introduces the *base-rate fallacy* for intrusion detection systems. He shows that the effectiveness of an IDS is determined by the ability of the system to suppress false alarms. Multiple approaches have been taken to improve the accuracy of alerts.

An approach to reduce the false-positive rate of intrusion detection/prevention systems is *alert verification*. In [84] the authors use this technique to determine the success of intrusion attempts. In detail, all alerts executed by an intrusion detection/prevention system are forwarded to an alert verification entity. The task of the alert verification entity is to evaluate whether the reported attack harms the addressed system. The host-specific vulnerability information is either gathered in an active or passive way which means:

- passive - the vulnerability information is gathered in advance,
- active - the vulnerability information is gathered after the occurrence of an alert.

In accordance to above given categorization, the *Active Mapping* approach [118] passively analyzes the network, including the creation of a host profile for each end-system in the network. The profiles are constructed by sending specially crafted packets to each host and interpreting the responses. On the basis of that knowledge a NIDS is capable of deciding whether an attack actually reaches the target host and if the attack is applicable.

**Shoki** [27] is an open-source signature-based NIDS which is able to identify the operating system of an end-system via passive fingerprinting in order to evaluate the priority of an alarm. Therefore, Shoki uses the techniques provided by *p0f* [142] which is a versatile passive OS fingerprinting tool.

Article [119] compares three commercial IDSs—*Cisco's Threat Response*, *ISS's Fusion* and *Tenable's Lightning Console*—that consist of: a network scanner to collect and manage

vulnerability information, an intrusion detection system and a console that verifies the alerts from the IDS. Cisco's follows an active alert verification process, meaning target information is gathered after the occurrence of an alert. The Cisco system requires a login to the target machines in order to collect the requested target information. In contrast ISS uses a scanner to regularly gather host-specific vulnerability information. The approach of Tenable is able to collect the target information either by using an active (regularly) or passive scanner (continuously). All three systems demonstrate that the approach of alert verification helps to improve accuracy.

A further set of approaches uses the *alarm-/alert-correlation* technique which combines alerts that are created by multiple IDSs [45, 46, 47, 49, 76, 93, 101, 124, 131]. The assumption is that a real attack would be remarked by more than one IDS. Accordingly, alerts can be collected, pre-processed (e.g. transformation into a readable format as the alerts stem from varying IDSs) and aggregated into significant high-level alarms.

The techniques alert-verification, alert-correlation and active mapping require the correlation of information stemming from different sources. For example, it must be possible to evaluate—based on vulnerability information gathered by a network scanner and an alert produced by an IDS—whether an harming attack is taking place. Reference [96] proposes M2D2 which is a formal data model for IDS alert correlation that allows to interrelate four types of information:

- network topology and end-systems (products),
- vulnerability information,
- the security tools used,
- generated events.

The network topology is modeled as a hypergraph and a product, a logical entity that is executed by a host, is represented by quadruplet **(vendor_id, product_id, version_id, type)**. The field *type* differentiates between operating system, server-like applications, local applications and other entities. The configuration of a host is defined by the running products of type: operating system, server-applications and local applications. The *Common Vulnerabilities and Exposures* list [2] which provides the most comprehensive index of standardized names for vulnerabilities is used to identify vulnerabilities within M2D2. The third information specifies what kind of security tools (e.g. vulnerability scanner, NIDS, etc) and their methods (misuse/anomaly) were used to create events. Finally, the fourth information type stores the events that are produced by the security tools.

Julisch follows another approach of alert correlation. He argues that each alarm that is triggered by an IDS can be backtracked to a limited set of *root causes* [79]. The root cause of an alarm is the reason why it occurs and further, according to Julisch, in most environments a relatively small number of highly predominant root causes exist which are the reason for over 90% of all alarms. Therefore he proposes a two step process: The alarms are clustered by an offline data mining process which subsequently allows to identify the root causes.

Also automated learning has been considered as a means to improve IDS accuracy. In [108] the authors describe the *Adaptive Learner for Alert Classification* (ALAC) which is an approach to reduce the amount of false positives. The system presented in the paper uses

supervised machine learning to tune an alert classification system based on observations of a human expert. The ALAC systems learns to differentiate between false positives and true positives by observing a security analyst who actually classifies alerts into true positives and false positives. Alert classification is defined as attaching a label to an alert indicating its importance.

### 3.3.2 The Requirements of Scalability and Flexibility

Scalability is the requirement that an IPS is able to analyze all network packets and to decide in real-time whether an attack is taking place at the same time. Another important requirement for an IPS is flexibility. To address the evolution of attacks it is of great benefit to be able to dynamically deploy new detection and prevention techniques on the security systems. Considering the hardware, two approaches to construct intrusion prevention systems can be differentiated:

- Realization of a centralized high performance IPS using multiple GPPs, NPs as well as special-purpose hardware like ASICs and FPGAs;
- Realization of a distributed IPS consisting of multiple software-based IPS entities on top of "conventional" hardware like GPPs and NPs.

As discussed in Section 2.3.6 special-purpose hardware (ASICs and FPGAs) is able to analyze network packets at high-speed, but on the contrary it is very expensive and inflexible. Further, the authors of [23] identified a set of weaknesses of centralized IDSs—systems that analyze the gathered data, including systems that collect data from multiple sensors, for evidence of attacks at one central place. The two most important arguments are:

- single point of failure and
- limited scalability as the performance of the central analyzing unit limits the amount of data that can be evaluated in a given time.

For this purpose the authors propose the *Autonomous Agents For Intrusion Detection* architecture (AAFID) that consists of autonomous agents, transceivers and monitors. An autonomous agent is an entity that resides in a host and performs a set of specific security operations. An end-system can host any number of autonomous agents and if at least one autonomous agent resides in a system, a transceiver must also be installed. All agents of a host report their findings to the local transceiver which might perform data reduction and which subsequently forwards its results to one or more monitors. Finally, a monitor analyzes the received data—originating from multiple systems and enables the detection of attacks that involve several hosts. The approach requires that a user agrees on running multiple autonomous agents and a transceiver on its system. Moreover, the authors do not indicate how to correlate the gathered data.

Also, the authors of [29] recognized the difficulty in realizing a central intrusion detection/prevention system that is capable to scan all network traffic. They propose to deploy an intrusion detection/prevention system on each end-system's network interface card. For this purpose, they developed *CardGuard* which is a software-based signature detection system

that uses a NP—located on the network interface card. The advantages of CardGuard are that no resources of the end-system are requested and the limited traffic volume that must be analyzed. The downturns are that the intrusion detection capabilities of CardGuard are limited to signature detection and, in addition, all users must agree to run CardGuard. Furthermore, the authors do not address the issue of configuring CardGuard.

For example, the configuration of Snort requires the setting of:

- environment variables,
- configuration parameters,
- preprocessor configuration—this must be done for each activated preprocessor,
- output module configuration,
- definition of new action types (optional) and
- specification of the set of rules that the packets are checked for.

Taking the given possibilities into account, the task of configuring Snort—especially the last item—is too difficult for most users. Moreover, anomaly based systems must repeatedly be trained with current data and misuse-based systems must continuously be updated.

Summarizing, an extreme approach is the construction of a high-performance centralized intrusion detection/prevention system using special-purpose hardware. The advantage of the approach is that only one system must be configured and the downturn is that it lacks flexibility. Further, considering growing traffic volumes—Internet traffic approximately doubles each year [41]—and an increasing number of attacks this demand is difficult to fulfill. The other extreme possibility is the deployment of a software-based intrusion detection/prevention system on every end-system. But in the context of the CardGuard approach the effort to install and configure the system on every network interface card is huge. Hence, a compromise between these two possibilities, the construction of a flexible intrusion detection/prevention system running on selected nodes in the network, seems to be promising. In this context, agent-based approaches have a fundamental downturn in common. An agent is a complete mobile application implemented for a specific purpose, like for example to search for traces of intrusions in a network. An agent performs all required operations on its own, and consequently, an efficient distribution of operations between multiple agents is not possible which affects the overall performance of the system.

## 3.4 What is Missing - A Discussion of the State of the Art

On the basis of the discussion of state of the art the requirements of autonomy from users and end-systems, accuracy, efficiency, scalability and flexibility were identified as important for an IPS.

First of all, an IPS should not require input of users as it cannot be expected that users are able to appropriately protect their end-systems themselves. Further, users must not be obliged to install an intrusion prevention system on their end-systems. The distributed approaches discussed [19, 77, 29, 23] require a modification of the end-systems that are to be

protected. Hence, the relevant users must agree to that and, the corresponding hardware/ software must be installed and configured. The effort for this is too huge. A good IPS must be autonomous from users and end-systems.

Moreover, to be effective an intrusion detection/prevention system must work as accurately as possible. In intrusion prevention systems the false positive rate is even more critical than in intrusion detection systems, since the harmless packets that trigger the alarm are filtered by the system and, hence, valid communications are interrupted or even stopped. The traditional approach to reduce the false positive rate of an intrusion detection/prevention system is to manually maintain and configure it. Systems like Snort—this also counts for Snort-Inline—or Bro belong to this group of security systems.

A promising approach to reduce the false positive rate without manual configuration effort is to combine vulnerability assessment and intrusion detection/prevention. The approaches [84, 118, 119] have in common that they do not use the gathered network knowledge to correctly configure the corresponding systems, but instead, incorporate the knowledge on the output side of the intrusion detection/prevention systems, after the generation of an alarm. As a consequence, each intrusion detection/prevention system performs all security checks. The vulnerability information can be used to restrict the number of security checks that must be performed per end-system which in turn would improve the efficiency of the system.

Section 3.3.2 discussed the requirements of scalability and flexibility. With steadily growing traffic volumes and increasing number of attacks, the realization of a centralized intrusion prevention system becomes impossible. In addition, the strict QoS requirements (end-to-end delay, loss rate, jitter, etc.) of existing and emerging applications are challenging and special-purpose hardware—often used in centralized high-performance IPSs—lacks flexibility in terms of re-programmability. In contrast, software-based intrusion prevention systems provide more flexibility but have a lower performance. Hence, a distributed and software-based IPS is a good compromise to fulfill the requirements for performance and flexibility.

The philosophy followed in this thesis is not to modify end-systems and to do limited intrusion prevention as a software process on selected routers in the network. The effort to place a limited set of programmable routers in a network that are capable to do intrusion prevention is smaller than to modify a significant higher number of end-systems. Furthermore, this statement is also valid for maintenance tasks (update of attack rules, algorithms, etc.). None of the distributed approaches considered intelligent deployment strategies, although the placement of intrusion prevention functionality has an impact on the performance of the network that is to be protected. Further, an intelligent, network-based and distributed intrusion prevention system must be able to adapt itself to network dynamics. For example, the reconfiguration of existing routes—triggered by a dynamic routing protocol—might demand to relocate certain intrusion prevention functionalities to another router in the network.

# Chapter 4

# Fidran: An Autonomous Intrusion Prevention Overlay Network

According to the arguments presented in Chapter 1 it cannot be expected that all users and administrators are able to keep their system(s) secure. Moreover, the fixing of security holes as soon as patches become available can hardly be done in time on all end systems. Thus, in order to relieve end-users and administrators from continuously having to deal with today's massive amount of security challenges the protection of end-systems should be done in the network. This thesis develops a concept for an autonomous intrusion prevention overlay network on top of programmable networking technology which intelligently protects the end-systems of a networking environment against network-based attacks. The central question of the thesis is:

> **What intrusion prevention functionality is required at which places of a network for the purpose of adequately protecting the end-systems of that network while simultaneously minimizing the impact on the network performance?**

It must be considered that the task of protecting end-systems by installing security services on routers in the network poses demands to the architecture of those routers. Further, it is necessary to decide on the set of security services that must be installed. And finally, reasonable security services deployment strategies must be considered.

The presented concept of an intrusion prevention overlay network which is called *Flexible Intrusion Detection and Response Framework for Active Networks* (Fidran) is designed for limited networks like an autonomous system (AS) as well as for high-speed networks (backbone). The functionality provided by Fidran depends on the chosen scenario. When deployed in a limited networking environment—in terms of a limited amount of connected end-systems and traffic volumes—the IPS overlay network is able to autonomously identify network topology, end-systems (operating system, running applications, etc.) and the set of required security services. Subsequently, the requested security services are intelligently distributed in the network such that, for example, the impact on the network performance is minimized. When deployed in a backbone network the IPS framework does not analyze network topology, end-systems and required security services since the amount of systems

to be analyzed is too big. In this scenario, that information must be provided to the autonomous intrusion prevention overlay network. The Fidran overlay network comprises of three functional parts:

1. the first functional part (not for high-speed deployment) includes the intelligence to analyze the network to be protected:
   - identification of the network topology,
   - discovery of the hosts that are running and
   - hosts profiling, identification of operating system and running applications.
2. the second functional part provides the intrusion prevention framework that actually allows to dynamically deploy intrusion prevention services on programmable nodes in the network.
3. the third functional part decides which intrusion prevention services are deployed on which programmable routers.

The next section covers the requirements for an autonomous intrusion prevention overlay network.

## 4.1 Requirements for an Autonomous Intrusion Prevention Overlay Network

The integration of a self-configuring IPS promises to be highly beneficial for private users and administrators. Especially, most consumers are not interested in security internals but in having a well running and protected system. However a couple of pitfalls that lead to a series of requirements for the design of such systems have to be avoided when planning an autonomous intrusion prevention overlay network:

- *Extensibility and flexibility:* As new attacking patterns are appearing almost every day, it is of prime importance that the IPS functionality can be easily augmented with appropriate detection and prevention capabilities. Depending on the attacking pattern, this might be realized by simply adding new attacking signatures up to the introduction of new modules with specific detection logic, including stateful inspection techniques and anomaly detection algorithms. The extension/modification of IPS functionality must occur at runtime without having to restart the IPS.
- *Efficiency and scalability:* as an IPS is integrated into the communication path, efficiency of the packet processing chain is of prime importance. With steadily growing traffic volumes and increasing number of attacks, the realization of a centralized intrusion prevention system becomes virtually impossible because all packets would have to be processed there. The strict QoS requirements (end-to-end delay, loss rate, jitter, etc.) of existing and emerging applications are challenging. The possibility of distributing the security services across the network offers the potential to optimize network performance. Furthermore, the overall system design needs to be scalable in terms of packet flows to be processed as well as attacking techniques to be supervised.

- *User-friendliness and transparency:* the protection of end-systems must occur in a user-friendly manner. Many users are not sensible to security vulnerabilities affecting their own machines or they are overstrained patching these (see Chapter 1). Farther, many believe that they will never become the target of an attack, due to the perception that their system or data is not of value for hackers. In this context it seems sensible to integrate the required intrusion prevention functionalities into the network itself. Moreover, the average user is not expected to be a skilled security expert who spends much time on keeping his systems up-to-date. Furthermore, transparency means that the end-systems and services must not be aware of the running IPS overlay network.
- *No special hardware*: the task of doing intrusion prevention should be realized as a software process running on *General Purpose Processors* (GPP).

The proposed intrusion prevention overlay network makes use of the capabilities provided by an underlying programmable network infrastructure. The decision in favor of the programmable networking technology is motivated in the following section.

## 4.2 Why Programmable Routers

Active/programmable networks were introduced in the early 90's as a new networking paradigma and they were announced to provide a solution to many difficulties in the field of communication networks. The programmable networking technology provides a framework for flexible and rapid service creation on top of existing networks. It uses enhanced nodes—so called active/programmable nodes—within the network for the provision of specific services. A programmable node is able to execute services which are loadable on-demand from a remote *service repository* (SR), and thus can enhance its functionality in a flexible manner. Application examples for this technology are media transcoding services, network security enhancing services or overlay networks, for example [21].

The decision to use active/programmable network technology as underlying infrastructure for the realization of the intrusion prevention overlay network was made due to the following arguing. Referring to Section 4.1, an autonomous intrusion prevention overlay network must fulfil certain requirements. The demand for *extensibility and flexibility* is satisfied by providing the capability to dynamically start/stop arbitrary programmable networking services on any programmable node. For the sake of security this capability must be restricted to a limited group of authorized persons [69, 68]. The requirement for *efficiency and scalability* can be achieved by an appropriate realization of the programmable networking infrastructure. To cope with increasing traffic volumes and to keep the additionally added intrusion prevention delay as small as possible it is of outstanding importance that on the one hand a programmable router captures network traffic as fast as possible and on the other hand the intrusion prevention services are realized in an efficient manner. The scalability part of the requirement is addressed by the ability of a programmable network infrastructure to deploy a chosen service on a specified programmable router. To provide protection an intrusion prevention service must be placed on a programmable router which lies on the data path between attacker and potential victim. Consequently, the ability to deploy an intrusion prevention service on a specified programmable router enhances the scalability of the intrusion

prevention overlay network. In the context of scalability it might be necessary to relocate a running service from one programmable router to another one, for example in case of a changing network topology. For example, a new end-system, requesting the installation of further intrusion prevention services, is attached to a router of the network. To optimally distribute the intrusion prevention services among the programmable routers it might be necessary to relocate running services—including state information like established TCP-sessions—from one programmable router to another one.

Summarizing, a programmable networking environment must provide the following capabilities to qualify itself adequate for the operation of an autonomous intrusion prevention overlay network:

- the programmable environment must allow to add/remove services at runtime,
- the programmable infrastructure must support the deployment of chosen services on specified programmable routers at a given time,
- the programmable infrastructure must be able to dynamically relocate a service from a programmable router to another one, and
- the programmable infrastructure must provide means to securely exchange information between programmable routers.

In case a programmable networking infrastructure fulfils the above listed requirements, then it is up to the developers of the security services to take care of an efficient realization of the required intrusion prevention services.

## 4.3 The Principle of Demand-Driven Intrusion Prevention

This section elaborates on the proposed concept of an autonomous and distributed intrusion prevention framework on top of programmable routers. The presented framework is designed to be operated in an arbitrary network consisting of a number of routers $r_i$ that connect several subnetworks $n_j$ to each other and to the Internet (see Figure 4.1). A subnetwork consists of a varying amount of end-systems which must not be equal. Each router in the network can either be of passive or programmable nature. An active router is automatically a member of the IPS overlay network. For protecting the end-systems, there has to be at least one programmable router on the path between potential attackers (Internet and all subnets) and each subnet. Internal attacks—attacks that completely take place inside one subnet—cannot be stopped, as only traffic between/to subnets is analyzed.

Demand-driven intrusion prevention makes use of the fact that attacks require the existence of one or multiple concrete vulnerabilities to succeed. For example, the *Code Red* attack exploits a buffer overflow that exists in certain versions of Microsoft's *Internet Information Server* (IIS). In the approach followed, an intrusion prevention service provides protection against attacks exploiting a concrete vulnerability. Flows towards an end-system are only analyzed by intrusion prevention services that protect against attacks that could actually harm it. In this way, the proposed approach reduces the amount of signatures for which each flow is checked and as a result this also reduces the rate of false positives, as explained next.

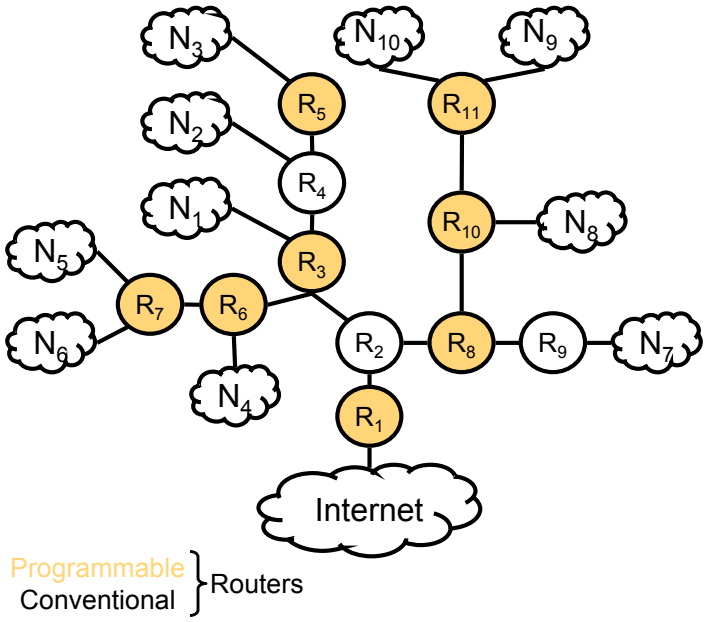



Figure 4.1: An Example Network

Let $p_i$ be the probability of falsely classifying a packet as malicious for signature i (false positive rate of signature i). $(1-p_i)$ is then the probability of correctly classifying the packet, and $(1-p_1)\cdot(1-p_2)\cdot\cdots\cdot(1-p_N)$ the probability that a benign packet is correctly classified by N signatures. The false positive rate for N signatures is then

$$1-\prod_{i=1}^{N}(1-p_i),\ 0\leq p_i\leq 1. \tag{4.1}$$

Independently of the values of $p_i$, the product decreases for increasing N and the false positive rate increases for increasing number of signatures.

This requires knowledge of the relevant end-systems and their vulnerabilities and, since networks vary over time, that knowledge must be continuously gathered automatically. In detail, it is essential to identify:

- the network topology,
- the reachable end-systems:
  - distance between Internet and end-system,
  - running OS and applications or vulnerabilities,
  - amount of traffic that is destined to an end-system.

This information enables to specify both the intrusion prevention services requested by each end-system, and all placement possibilities for each service. The distance between Internet

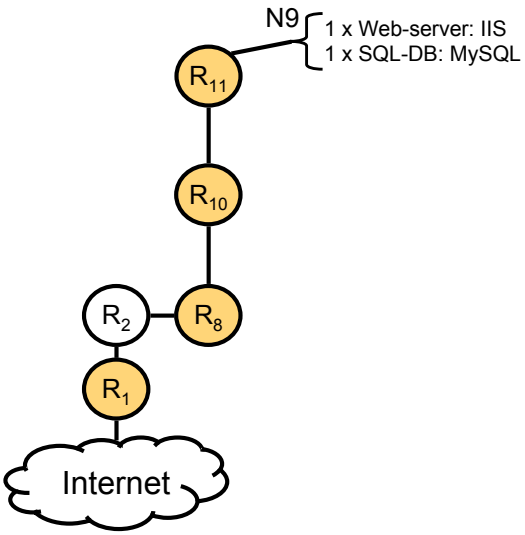



Figure 4.2: A detailed view of a path of the network

and an end-system is used to filter packets that would not reach its destination based on the TTL. Further, the amount of traffic that is destined to an end-system is required to determine an optimal deploy strategy of the intrusion prevention services.

An intrusion prevention service analyzes one or multiple flows (IP-source, IP-destination address, port numbers) for specific attacks. Hence, the requested intrusion prevention services must be distributed in the network such that any protection service required by a subnet is deployed somewhere between potential attackers (Internet and subnets) and that subnet. For each router the following degrees of freedom exist:

- Decision whether or not a router is programmable.
- Choice of protection services to be integrated.
- Specification of which traffic must be analyzed by which protection services.

The set of these decisions per router make up a deployment strategy for the Fidran architecture.

For example, assuming that subnetwork $N_9$ of the exemplary network depicted in Figure 4.2 consists of two end-systems: a Windows NT system running *Microsoft's Internet Information Server* (IIS) with support for *php*-, *cgi*- and *coldfusion*-scripts and a Linux-based system running the SQL database management system *MySQL*. An analysis of both systems results in an initial request to install five application-specific intrusion prevention services (*IIS*, *php*, *cgi*, *coldfusion* and *MySQL*), each service preventing attacks targeting the eponymous application.

Assuming that all subnetworks can be trusted (the Internet is the only potential attacker), this results in an overall number of five intrusion prevention services that must be deployed on the four programmable routers $R_1$, $R_8$, $R_{10}$ and $R_{11}$ that are on the path between Internet

and subnetwork $N_9$. Theoretically, the outcome of this are $4^5 = 1024$ different deployment strategies. In case that all five routers between Internet and subnetwork $N_9$ can be made programmable, then $5^5 = 3125$ different possibilities exist.

The requested intrusion prevention services must be placed on all traffic paths to subnet $N_9$ in case that all subnets are potential attackers, and the amount of placement possibilities can be calculated using Equation (4.2). The number of programmable routers on the path from the origin o to destination d (Internet: o=d=0) is denoted $routers(o, d)$ and $s_d$ denotes the amount of services requested by destination d.

$$\sum_{o=0}^{l} \sum_{d=1}^{l} routers(o, d)^{s_d} \tag{4.2}$$

Assuming that all routers in the network can be turned into programmable ones and all subnets request the installation of five intrusion prevention services, then this results in an overall amount of 468,548 placement possibilities. Consequently, the question is how to configure each router of a network in order to adequately protect the end-systems.

To do so, the self-configuring intrusion prevention overlay network must be capable of:

1. analyzing the network which is to be protected (only for limited networking environments),
2. determining an intelligent deployment strategies of the requested intrusion prevention services,
3. distributing the requested security services in accordance to the previously defined strategy, and
4. adapting itself to network dynamics.

The next section identifies the mechanisms—besides the above mentioned capabilities—that are required to create an autonomous intrusion prevention overlay network.

## 4.4 The Fidran Intrusion Prevention System Architecture

This section discusses the *Flexible Intrusion Detection and Response Framework for Active Networks* architecture depicted in Figure 4.3. Fidran is build on top of an underlying programmable networking environment which allows to dynamically deploy new security services on Fidran routers. Its modular design provides the infrastructure for a constructive cooperation between modules of different security technologies. In addition, the programmable networking infrastructure facilitates maintenance work and enables the distribution of security tasks among different Fidran routers.

For developing Fidran the following design-requirements were identified and followed:

- modular concept:
  - capability to fine-grain configure Fidran nodes thus enabling the principle of demand-driven intrusion prevention

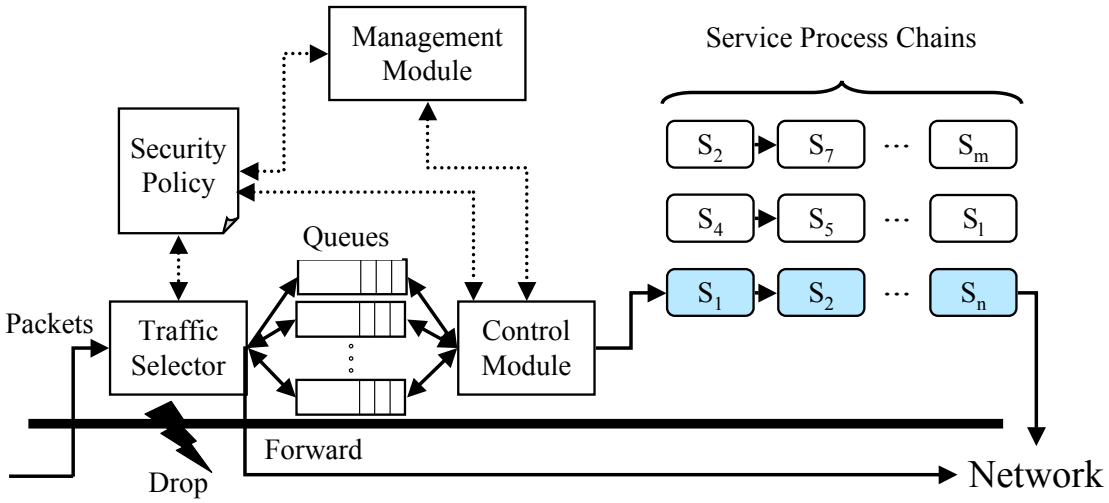



Figure 4.3: The Fidran Architecture

  - capability to dynamically extend the functionality through the integration of new security services
  - facilitation and acceleration of maintenance work and configuration tasks
- efficiency
- (re-) configuration of the Fidran system at runtime

The framework consists of core components which run permanently and of add-on components—the security services—which are dynamically integrated into the system as needed (cf. Fig. 4.3). The core functionality comprises the traffic selector, the security policy, the control/management module, the network scanner and the default queuing discipline. Security services are implemented as loadable modules featuring IPS specific networking services. The system is designed such that a dynamic reconfiguration at runtime (insertion and deletion of security services) is possible. The capabilities provided by the underlying programmable networking infrastructure allow to distribute the Fidran system on programmable routers. The dynamic creation of an IPS overlay network is thereby enabled. Secure communication between programmable nodes is also provided. In the following sections all components of the framework are explained in detail.

### 4.4.1 The Security Policy

Each Fidran node possesses a security policy which is depicted in Figure 4.4 and which specifies the behavior of the system. As the tasks differ between last hop programmable routers and the other programmable routers, two versions of the security policy exist. In detail, they store:

- **all Fidran systems:**

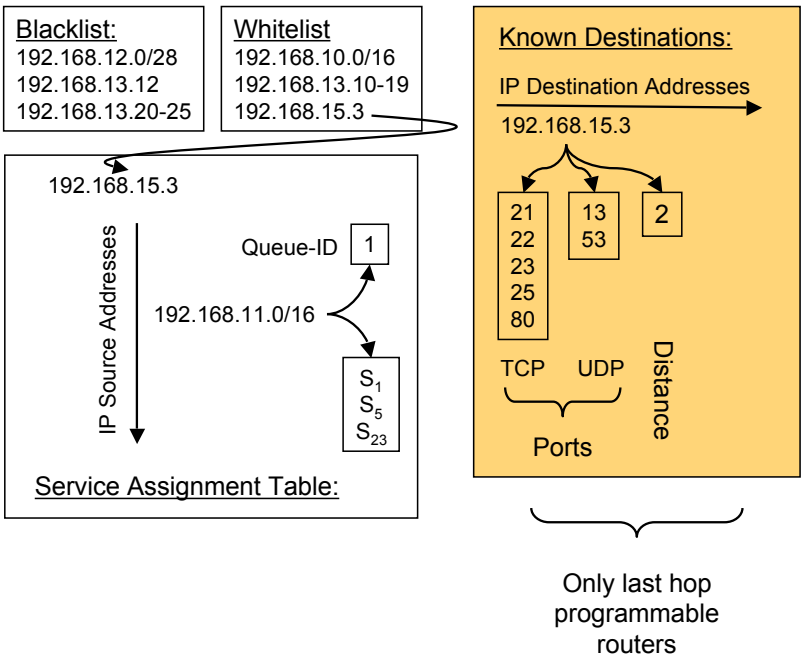



Figure 4.4: The security policy

- the traffic categories:
  - process:
    - mapping between flows and service process chain and
    - mapping between flows and waiting queue
  - drop: specification of packets that are dropped by the traffic selector
  - forward: specification of the flows that are not analyzed by the local FIDRAN system

- **only last hop programmable routers:**
  - the distances between FIDRAN router and end-systems,
  - a list of relevant and reachable destination-addresses

The structure of the security policy is depicted in Figure 4.4. All security policies contain a *whitelist*, a *blacklist* and a *service assignment table*. The whitelist specifies the network traffic belonging to traffic category process and the blacklist defines the network packets which must be dropped by the traffic selector. Network traffic that neither is specified in the whitelist nor in the blacklist is automatically assigned to traffic category forward. To reduce the amount of entries the security policy supports the *Classless Inter-Domain Routing* (CIDR) notation as well as the wildcard "-" which allows to specify small IP address ranges. The *service assignment table* memorizes the mapping between traffic flows and intrusion prevention services and in addition, it assigns the traffic flows to the appropriate waiting

queue (queue-id). To implement flow specific assignment of intrusion prevention services—based on the tuple IP destination and source address—the service assignment table associates the list of assigned intrusion prevention services with the corresponding source and destination address as depicted in Figure 4.4. For this purpose the service assignment table uses the IP destination addresses of the whitelist as primary index and the IP source address as second index. Entry $(IP_B/IP_A)$ of the service assignment table points to the list of intrusion prevention services that are assigned to the flow originating from IP address $IP_A$ and destined to IP address $IP_B$. Identification of intrusion prevention services happens via unique names. The control module uses the service assignment table to generate the service process chains (see Section 4.4.5). Finally, the waiting queue for a flow is specified in the corresponding data field $(IP_B/IP_A)$ →Queue-ID.

Security policies of last hop programmable routers, like router $R_5$ in Figure 4.1, additionally keep track of reachable and relevant destination addresses. The list memorizes IP-addresses and listening port numbers (TCP and UDP) of end-systems that are reachable and that belong to the corresponding subnetwork, for example, router $R_5$ keeps track of all reachable destination addresses of subnetwork $N_3$. The list is continuously observed and regularly updated by the network analysis process (Section 5). Last hop traffic selectors use it to identify new destinations in terms of new end-systems or new running applications. Supplementary, the list memorizes the distances between the local Fidran router and the end-systems of the attached subnetworks. Last hop traffic selectors check the distance fields to filter those packets that would not reach their destination.

### 4.4.2 The Traffic Selector

Initially, all network traffic captured by the network interfaces is forwarded to the traffic. It is responsible for the tasks:

- **all Fidran systems:**
  - of categorizing the network traffic into
    - process - packets are analyzed by a set of security services,
    - forward - packets are directly forwarded without being analyzed by a security service, and
    - drop - packets are dropped by the Fidran system:
      - ingress-/egress-filtering
      - packets that would not reach their destinations
- **only last hop programmable routers:**
  - of measuring traffic volumes
  - of screening network traffic for new destination addresses

All network traffic—downlink (traffic destined to a subnet) as well as uplink (traffic originating from a subnet) traffic—is redirected to the traffic selector. To appropriately divide the network traffic into the categories process, forward and drop the traffic selector accesses the corresponding entries of the security policy (black- and whitelist). First it checks the blacklist

to drop the specified network packets. Next it inserts the network packets, as specified by whitelist and the corresponding entry of the service assignment table, into the appropriate waiting queue. Network packets that neither are specified in the blacklist nor in the whitelist are directly forwarded and not analyzed by any installed security service (see Figure 4.3). It is either not necessary to check this traffic (e. g. encrypted traffic, IPSec), or another programmable node on the route to the end-system is in charge of doing so.

Traffic selectors running either on the BGR or on a so-called last hop programmable router—for example, router $R_7$ in Figure 4.1 is the last programmable hop for subnetworks $N_5$ and $N_6$—check the *time to live* (TTL) values of network packets addressed to a destination within the network and drop those that would not reach their destination. The corresponding distances between end-systems and programmable routers are specified during the network analysis process described in Section 5 and are stored in the security policy (see Section 4.4.1).

The principle of demand-driven intrusion prevention is to analyze flows—identified via the tuple (source-address, destination-address, source-port, and destination-port)—only with the set of intrusion prevention services that protect against the attacks that could actually harm the receiving end-system. To address the issue of IP address spoofing, BGR and last programmable hops are assigned to do ingress/egress filtering. The former is done by the traffic selector running on the BGR: packets that come in from the Internet and that have an IP source address which belongs to the IP-address range of the network are dropped. Traffic selectors running on last programmable hops are assigned to do egress filtering on the uplink network traffic: network packets originating from a subnetwork of the network must have a corresponding IP source address otherwise they are dropped. The relevant information is extracted from the corresponding routing tables.

To optimally deploy the requested intrusion prevention services it is mandatory to know how much traffic is destined to an end-system (see Section 6); this is done by last hop traffic selectors. Finally, to identify new destinations within the network—networks vary over time—last hop traffic selectors scan the network traffic for new destination addresses (IP-address, destination port) that lie within the IP-address range of the network.

### 4.4.3 An Intrusion Prevention Service

It is emphasized that neither the development nor the improvement of new/existing attack detection techniques is part of this thesis. The focus is on implementing an intrusion prevention overlay network that efficiently and autonomously protects vulnerable end-systems of a network. For this purpose the intrusion prevention overlay network uses existing techniques to protect vulnerable systems.

Demand-driven intrusion prevention makes use of the fact that most attacks require the existence of one or multiple concrete vulnerabilities to succeed and hence, flows to an end-system are analyzed only by intrusion prevention services relevant to the systems' vulnerabilities. A vulnerability is a weakness in an information system that can be used by an attacker to alter the intended operation of the system. To take advantage of a vulnerability the attacker uses an *exploit*, for example, the *Code Red* attack exploits a buffer overflow that exists in certain versions of Microsoft's *Internet Information Server* (IIS). Several approaches to name vulnerabilities exist. The *Common Vulnerabilities and Exposures* list [2] provides the most comprehensive index of standardized names for vulnerabilities. Moreover, it must be taken

into account that multiple exploits can address one concrete vulnerability: metamorphic and polymorphic variants of a primary exploit can be generated by altering instructions (or the instruction sequence) or by applying different encryption/decryption techniques, respectively. Polymorphism and metamorphism hamper the implementation of adequate intrusion prevention services, but promising approaches are being developed: reference [33] discusses the automated generation of vulnerability-based signatures, signatures that match all exploits of a given vulnerability. Further approaches are presented in [86, 133, 39, 33].

All intrusion prevention services are stored on a central server, the *Service Repository* (SR), where the programmable routers download the requested intrusion prevention services from. Further, each security service is realized as programmable networking service and accordingly, it can dynamically be integrated/removed on/from a FIDRAN router at runtime—a programmable node must not be rebooted in order to extend its functionality.

An intrusion prevention service provides protection against attacks exploiting a concrete vulnerability: it performs its individual set of operations—signature-detection, anomaly detection algorithms or any kind of security intelligence—on the traffic passed to it. An exemplary intrusion prevention service provides protection against the *Land-attack*. To crash vulnerable end-systems, an attacker sends a single a TCP packet with the SYN flag set and the same destination and source address and port.

To enable the dynamic generation of service process chains—which is done by the control module (see Section 4.4.5)—each security service contains a header part specifying:

- the unique intrusion prevention service *name*,
- the *author* of the intrusion prevention service,
- a short *description* of what the intrusion prevention service is doing,
- the *operating systems* that are protected by the intrusion prevention service
- the *applications* that are protected by the intrusion prevention service.
- the *protocol*-field specifies which network packets must be forwarded to the intrusion prevention service and
- the *priority*-field specifies the position in the process chain,
- finally, the field *call* contains the events to which an intrusion prevention service registers itself.

The fields *name*, *author*, *operating systems*, *applications* and *protocol* are self-explaining. In addition, the approach provides the possibility to label an intrusion prevention service with a priority. The control module uses linked lists to handle the individual execution sequences of packets and the priority determines the position of an intrusion prevention service inside these linked lists. Finally, an intrusion prevention service registers itself—specified in the *call*-field—to one of two events: the arrival of a packet or the reception of an alarm which has been triggered by an intrusion prevention service.

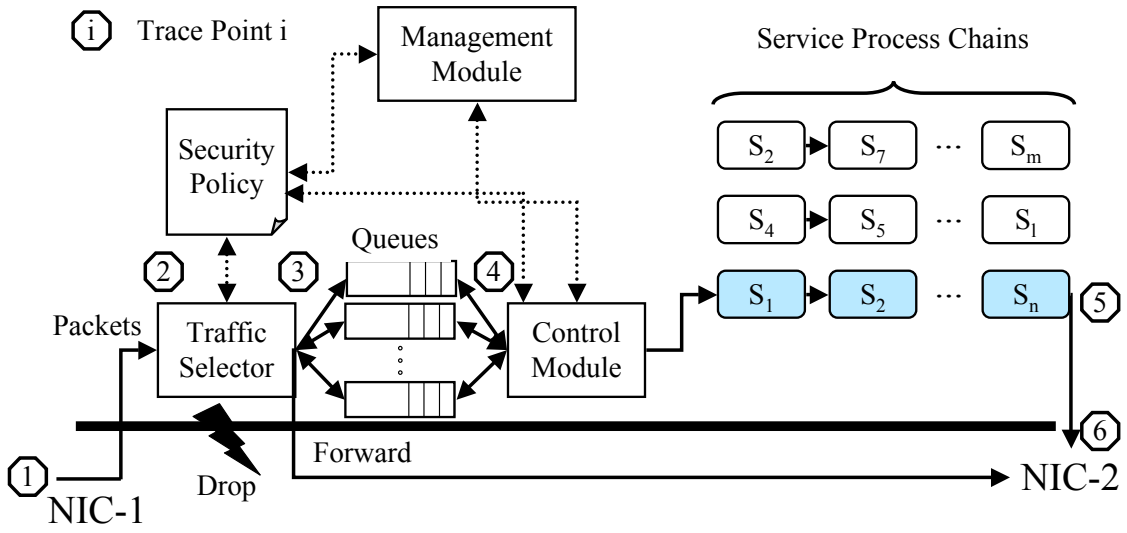



Figure 4.5: The trace points used for the performance evaluation

### 4.4.4 The Waiting Queues

Each Fidran system is provided with a configurable amount of first-in-first-out (FIFO) waiting queues of limited length. A waiting queue is the buffer between traffic selector and remaining Fidran system. A waiting queue is linked to a concrete service process chain. The control module takes the packets from the waiting queues and forwards them to the appropriate service process chain. The matching service process chain is identified by the waiting queue that stored the corresponding packet.

### 4.4.5 The Control Module

The control module is the central unit of the Fidran system. Its main responsibility is to take care of the service processing chains. This includes the task of handling the integration of new and the removing of old security services. Further, the set of integrated security services are dynamically linked by the control module into a set of service processing chains. The particularity is that security services can be integrated into /removed from a Fidran system at runtime. A system stop or a system's reboot is not required. A security service can be member of multiple security service processing chains. To create the security service processing chains the control module evaluates the header parts of the security services and the security policy. A security service is identified via an unique name and moreover, the security policy specifies which security services must analyze which packets.

As stated in the previous section the control module also dequeues the waiting queues whereas all waiting queues are assigned the same portion of processing time meaning that the waiting queues are served without priority. This technique is known as Round-Robin scheduling. Other scheduling algorithms could be integrated.

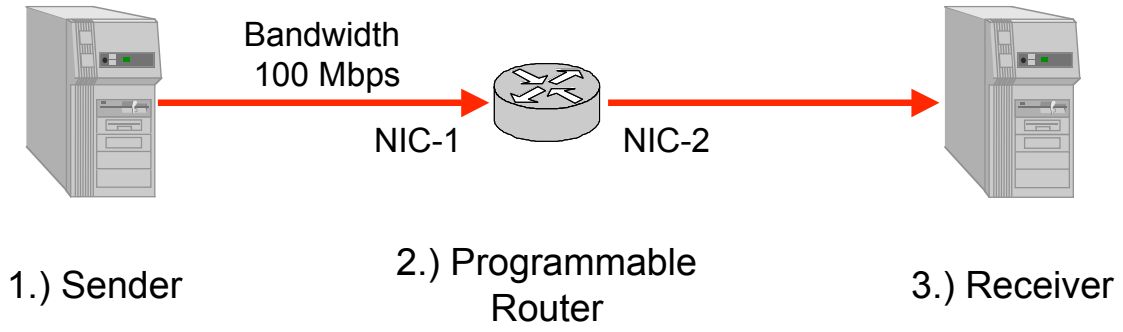



Figure 4.6: The initial testbed

## 4.5 The Impact of a Fidran Router on the Processing of a Packet

In the course of the thesis a Linux-based prototype of the Fidran system was implemented to assess its performance. The prototype includes a small collection of intrusion prevention services using the state-of-the-art pattern matching algorithms Boyer-Moore-Horspool [70], which is a modified version of the original Boyer-Moore algorithm [30], and Aho-Corasick [12]. These intrusion prevention services search for specific patterns in the packets payload. The security services, the control module, the traffic selector and the queuing disciplines were implemented as loadable kernel modules, whereas the management module runs as a user-space process. Moreover, the Fidran architecture is equipped with a configurable number of waiting queues which were realized as first-in-first-out (FIFO) waiting queues.

To determine how a running Fidran system affects the network performance in terms of delay, an initial testbed consisting of three systems was setup. The systems were connected in line via Ethernet with a connection speed of 100 MBit/s (see Figure 4.6) and the middle host—equipped with two network interface cards—was running the Fidran system. The Fidran system was supplied with one waiting queue which was configured to maximally buffer 100 packets. The remaining two hosts acted as sender—generating constant-bit-rate traffic (CBR)—and receiver. No further tasks were running on the systems.

Fidran was analyzed for two types of systems: a Pentium III system, 733 MHz, 1024 MByte main memory (PC733) and and a Dual Xeon node, 3000 MHz, 2048 MByte main memory (PC3000). Sender and receiver were of type PC3000. Furthermore, all systems were running a customized Fedora operating system with kernel 2.6.12. An experiment run included the sending of 30,000 packets for each parameter combination. For each packet, the following timestamps were traced (see trace points in Figure 4.5):

T1) reception time at NIC-1,

T2) arrival time at the Fidran system,

T3) insertion time into the waiting queue,

T4) start time of the processing period,

T5) end time of the processing period and

T6) sending time at NIC-2.

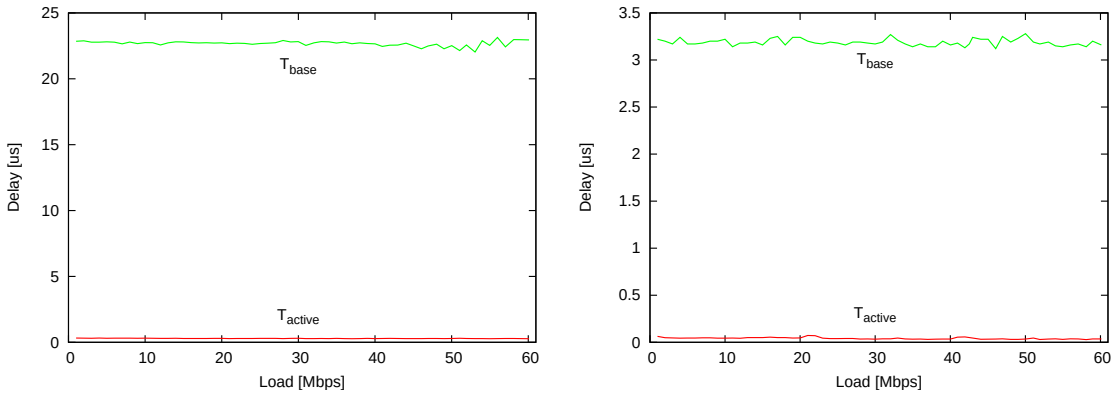
The *Time Stamp Counter* (TSC), a feature of the Intel IA-32 instruction, was used to capture the timestamps. The *Read Time Stamp Counter* instruction (RDTSC) returns the count of ticks from processor reset and Equation 4.3 shows how to convert a number of counted ticks into the corresponding delay. To correctly measure the delays the following precautions were taken. On multi-processor systems—this also counts for multi-core systems—it is not guaranteed that all processors have identical values in their TSC. This issue was addressed by disabling the multi-processor support of the operating system. Consequently, on Dual Xeon nodes packets were only processed by one CPU. In addition, to reduce the number of wasted cycles modern processors support *out of order execution*. The *CPUID* instruction can be executed to serialize instruction execution. Serializing instruction execution guarantees that any modifications to flags, registers, and memory for previous instructions are completed before the next instruction is fetched and executed.

$$delay[s] = \frac{\text{number of ticks}}{\text{CPU frequency[Hz]}} \tag{4.3}$$

In a first set of experiments the basic delay $T_{\text{base}}$ representing the routing delay in a standard network router—here the two systems mentioned above—was specified. In this scenario no Fidran system was running on the programmable router. This delay is denoted by $T_{\text{base}}$ and it also applies in case of a running Fidran system. Throughout the experiments the packet reception times at network interface card 1 (NIC 1), tracepoint $T1$, were traced as well as the sending times at NIC 2, tracepoint $T6$. The routing delay for a packet corresponds to the difference between reception time and sending time.

Apart of the system that is used as router, also packet size and mean packet arrival rate influence the average routing delay of a packet. For this reason $T_{\text{base}}$ was once measured for varying traffic loads with packets of a constant size and once for varying packet sizes at constant traffic load. In detail, $T_{\text{base}}$ was measured for varying constant-bit-rate traffic loads starting at 1 Mbps and ending at 60 Mbps. The traffic rate was increased by 1 Mbps per experiment run and the IP packet size was set to 1500 bytes. $T_{\text{base}}$ was measured for both systems—PC733 and PC3000—and the results are depicted in Figure 4.7. The green curve in Figure 4.7(a) shows the measured routing delay $T_{\text{base}}$ over the offered load for a system of type PC733 and the green curve in Figure 4.7(b) depicts the measured values for a system of type PC3000. It can be seen that the basic delay $T_{\text{base}}$ is a constant for a given system and does not depend on the offered load or in other words the offered load has no influence on $T_{\text{base}}$.

Next, a Fidran system was started on the programmable router but no security services were installed. In case of a running Fidran system, all traversing packets will additionally be delayed because Fidran performs a security policy lookup in order to check whether the current packet must be analyzed by security services. This delay can be measured between

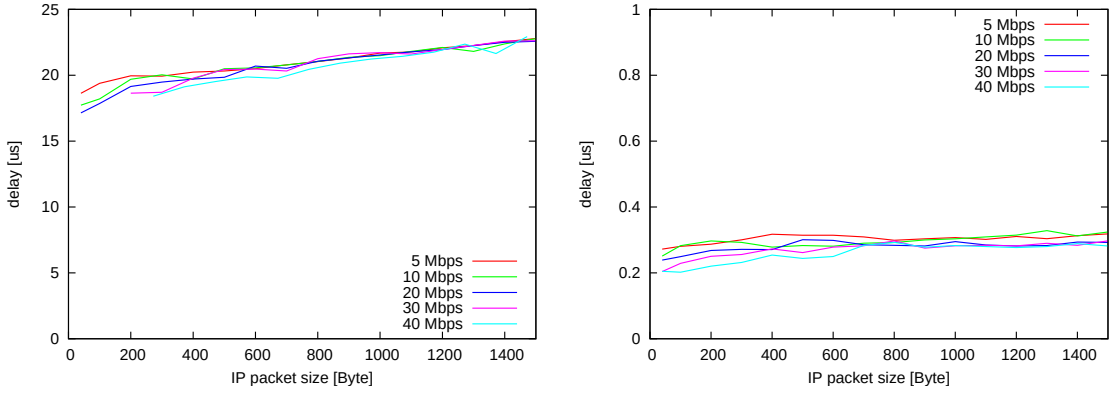



(a) PC733 (b) PC3000

Figure 4.7: $T_{\text{base}}$ and $T_{\text{active}}$: varying traffic rates and IP packet size of 1500 Bytes

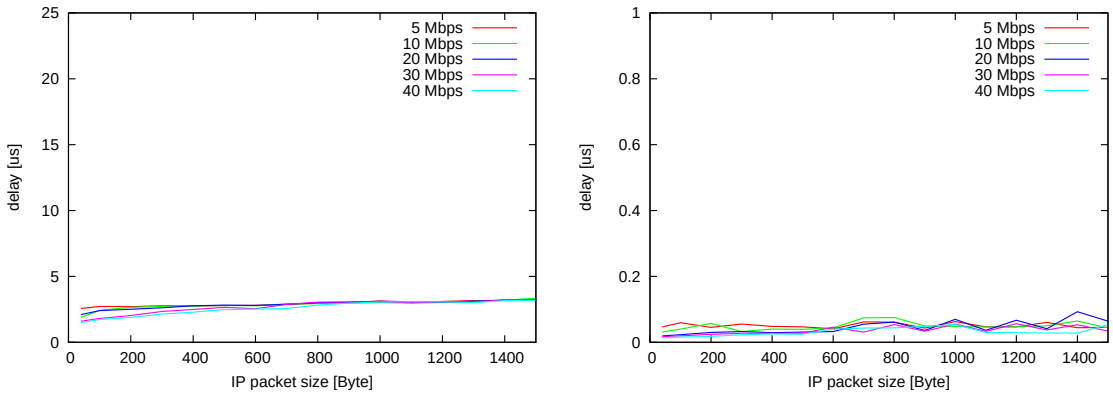
Tracepoint 2 and Tracepoint 3, and is denoted by $T_{\text{active}}$. During the experiment, each packet was checked against ten destinations which are registered in the security policy. The red curve in Figure 4.7(a) represents the security policy lookup delay $T_{\text{active}}$ over the offered load for a system of type PC733 and the scenario of ten possible destination addresses. The analogous values for a system of type PC3000 are depicted by the red curve in Figure 4.7(b). Again, both curves show that for a given scenario—in terms of possible destination addresses—the offered load does not influence the parameter $T_{\text{active}}$.

In the following, the influence of the packet size on the delays $T_{\text{base}}$ and $T_{\text{active}}$—again each packet was checked against ten destinations—was analyzed for both system types using CBR-traffic of 5, 10, 20, 30 and 40 Mbps. At the beginning, IP packets of size 40 Byte were sent and in the course of the measurement the IP packet size was increased in steps of




(a) $T_{\text{base}}$ (b) $T_{\text{active}}$

Figure 4.8: PC733: $T_{\text{base}}$ and $T_{\text{active}}$ for constant traffic rates and varying IP packet sizes



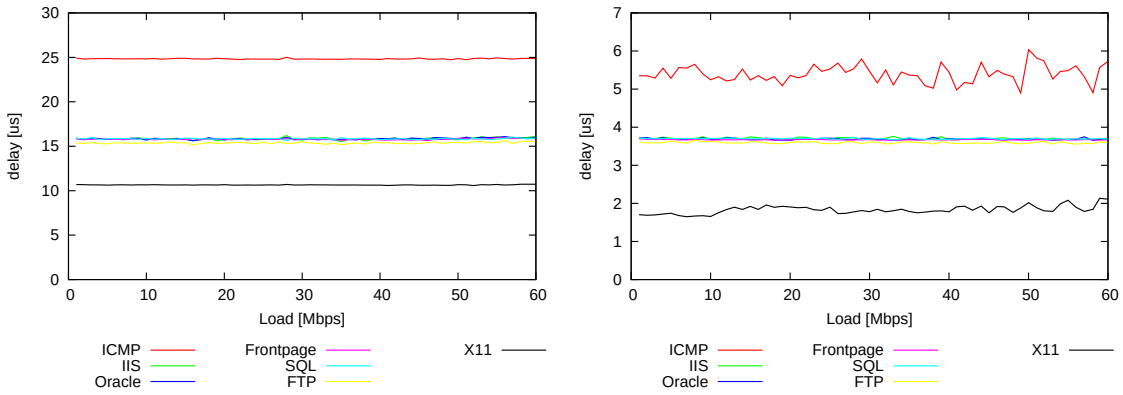

(a) $T_{\text{base}}$ (b) $T_{\text{active}}$

Figure 4.9: PC3000: $T_{\text{base}}$ and $T_{\text{active}}$ for constant traffic rates and varying IP packet sizes

100 Byte up to a maximal size of 1500 Byte. Figure 4.8(a) depicts the routing delay $T_{\text{base}}$ and Figure 4.8(b) shows the delay $T_{\text{active}}$ as functions of the IP packet size for systems of type PC733. The corresponding results for systems of type PC3000 are depicted in Figures 4.9(a) and 4.9(b). Considering the measured $T_{\text{base}}$ values for both systems, it can again be seen that the load has no influence on the delays. But on the opposite the IP packet size has an impact on the routing delay $T_{\text{base}}$ as depicted in Figures 4.8(a). For both systems the routing delay $T_{\text{base}}$ increases with growing packets. The reason for this are the operations that are required once to transfer a packet from the receiving network interface card to the router's network stack and once to forward the packet to the sending network interface card. The greater a packet the longer do these operations take. Finally, the influence of the packet size is stronger for PC733 systems as the slopes of the curves in Figure 4.8(a) are greater than those in Figure 4.9(a). Figures 4.8(b) and 4.9(b) depict the measured delays $T_{\text{active}}$ for both systems. Considering the range of the depicted values $T_{\text{active}}$ can be assumed to be constant for a given system. The reason for this is that the traffic selector (see Section 4.4.2) exclusively evaluates IP-header fields to decide whether a packet must be processed by the local router. Table 4.1 summarizes the measured delays $T_{\text{base}}$ and $T_{\text{active}}$ for both types of systems.

Table 4.1: $T_{\text{base}}$ and $T_{\text{active}}$

| | | Delay [$\mu s$] | |
|---|---|---|---|
| | IP packet size [Byte] | PC733 | PC3000 |
| $T_{\text{active}}$ | 40-1500 | 0.293 | 0.042 |
| $T_{\text{base}}$ | 40 | 17.83 | 1.93 |
| | 500 | 20.12 | 2.71 |
| | 1500 | 22.74 | 3.23 |




(a) PC733 (b) PC3000

Figure 4.10: Processing times of example security services: varying load and constant IP packet size 1500 Byte

In the next set of experiments security services were started on the programmable router, additionally delaying the packets on their way from sender to receiver. The delay caused by running security services can be split into two components:

- waiting time $T_{\text{waiting}}$ and
- processing time $T_{\text{process}}$.

The waiting time $T_{\text{waiting}}$ consists of the time that a packet spends inside the waiting queue (T3–T4). The time $T_{\text{process}}$ is measured between Tracepoints 4 and 5. For implementing security services, the attack signatures of the current Snort rule database were used. The security services used are listed in Table 4.2. The second column indicates the algorithm used (AC: Aho-Corasick and BM: Boyer-Moore-Horspool). It is emphasized here that the security services were only realized for the purpose of the performance evaluation. They have their limitations; for example, they neither keep state information nor do they address fragmentation. Again the processing time of each security service was measured for both systems and for the scenarios: (i) varying load and constant IP packet size (1500 bytes) and (ii) constant load and variable IP packet size (40-1500 Byte). At each experiment run exactly one security service was deployed on the programmable router and each IP packet was checked against ten destinations. Furthermore, the packets' payload was generated by random.

Figure 4.10(a) depicts the processing times over the offered load for the seven security services on a PC733 system and Figure 4.10(b) shows the corresponding results for a PC3000 system. Again for both systems it can be stated that the offered load has no influence on the security services' processing times. The processing times of the security services using the Aho-Corasick algorithm are about the same value. The reason for this is that the algorithm matches all patterns at once against the input stream. To do so it constructs a finite state pattern matching machine based on the provided set of signatures. In contrast the Boyer-Moore algorithm is only able to match an input string against a single signature at a time.

Table 4.2: The exemplary security services

| Name | Algorithm |
|---|---|
| S1 ICMP | BM |
| S2 IIS | AC |
| S3 Oracle | AC |
| S4 Frontpage | AC |
| S5 SQL | AC |
| S6 FTP | AC |
| S7 X11 | BM |

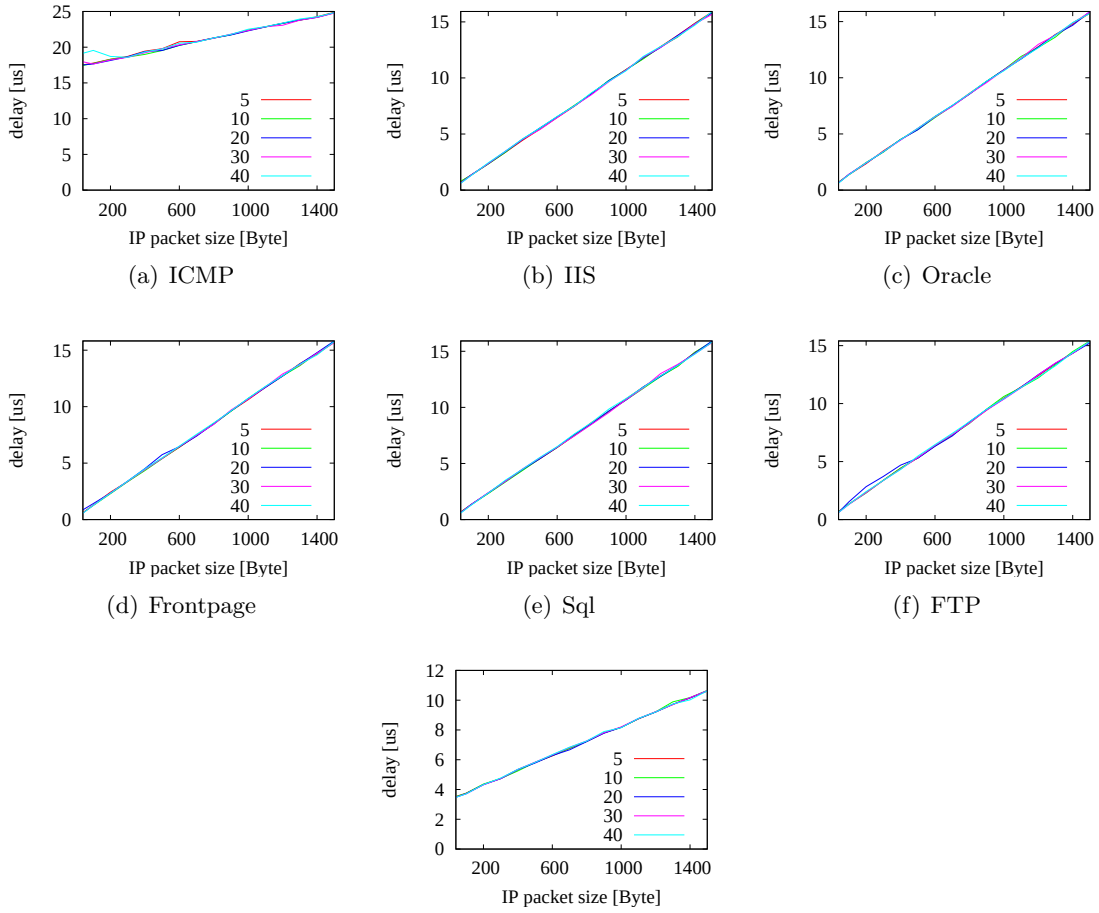
Consequently, the processing times of the ICMP security service (12 signatures) is greater than the processing time of the X11 security service (2 signatures). But according to the figures, the processing time of a security service is independent of the load and moreover, it is a constant for a given security service.

In a subsequent set of experiment runs the impact of the IP packet size on the security services' processing times was studied. For constant traffic rates of 5, 10, 20, 30 and 40 Mbps and IP packet sizes from 40 Byte up to 1500 Byte the security services' processing times were measured for both systems. The results for the PC733 system are depicted in Figure 4.11 and the analogously measured delays for the PC3000 system are shown in Figure 4.12. Each figure depicts a security service's processing time over the IP packet size for the mentioned CBR-traffic loads. All depicted curves show the effect of the IP packet size on the security service's processing time; the huger a packet the longer its processing time. The reason for this is that the complexity of the algorithms Boyer-Moore and Aho-Corasick are linear in the length of the input string. The linear relationship between processing time and input string (IP packet) can clearly be seen in the figures. Finally, the curves of each figure—representing different CBR-load scenarios—again prove the fact that the load does not influence the processing time of a security service.

In a next experiment, the delay in dependence on the number of integrated security ser-

Table 4.3: The processing times $t_s$ of the security services [$\mu s$] for various IP packet sizes [Byte]

| | PC733 | | | PC3000 | | |
|---|---|---|---|---|---|---|
| | 40 | 500 | 1500 | 40 | 500 | 1500 |
| ICMP | 17.92 | 19.68 | 24.83 | 4.08 | 4.78 | 5.46 |
| IIS | 0.69 | 5.48 | 15.82 | 0.17 | 1.29 | 3.70 |
| Oracle | 0.67 | 5.45 | 15.80 | 0.17 | 1.28 | 3.71 |
| Frontpage | 0.65 | 5.49 | 15.75 | 0.12 | 1.27 | 3.67 |
| SQL | 0.63 | 5.47 | 15.83 | 0.16 | 1.33 | 3.72 |
| FTP | 0.64 | 5.34 | 15.33 | 0.14 | 1.25 | 3.61 |
| X11 | 3.50 | 5.82 | 10.63 | 1.00 | 1.32 | 1.75 |

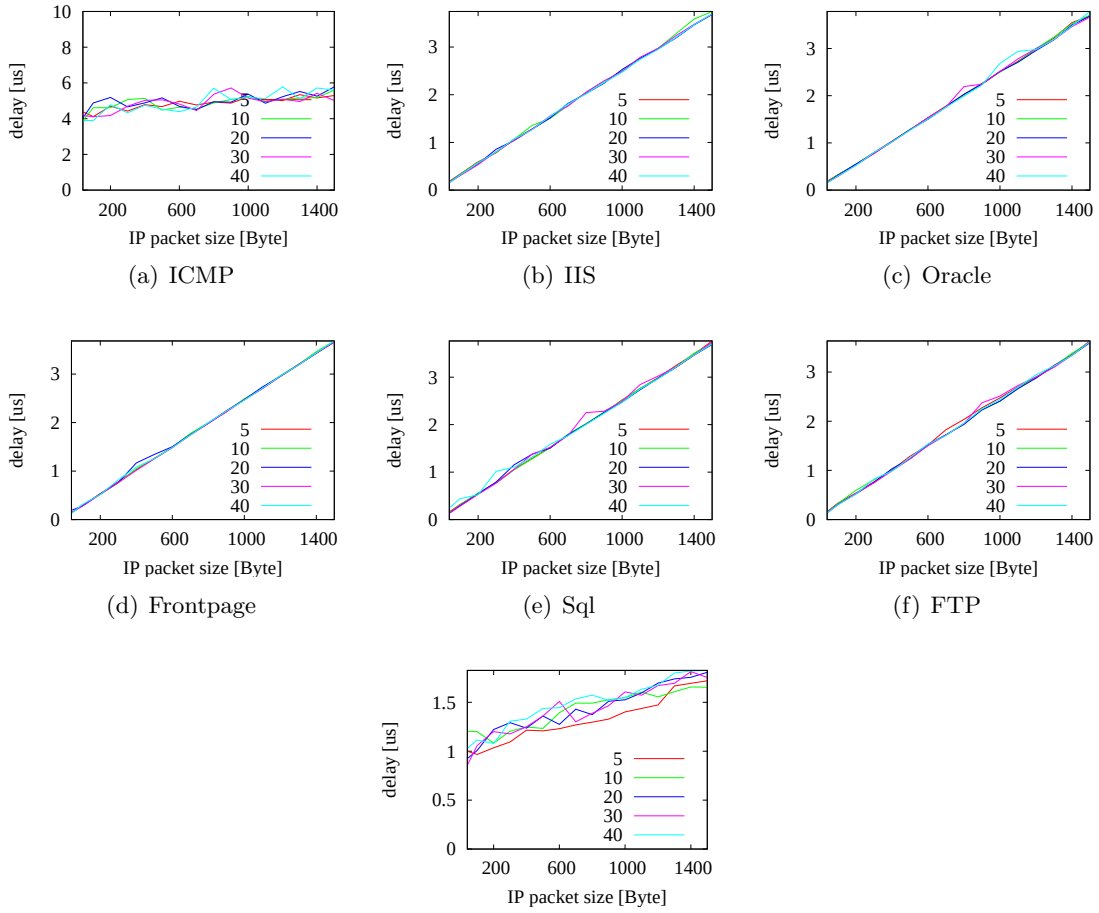



(a) ICMP (b) IIS (c) Oracle

(d) Frontpage (e) Sql (f) FTP

(g) X11

Figure 4.11: PC733 processing times of the example security services: constant loads and varying IP packet size




(a) ICMP (b) IIS (c) Oracle

(d) Frontpage (e) Sql (f) FTP

(g) X11

Figure 4.12: PC3000 processing times of the example security services: constant loads and varying IP packet size

vices was measured. This experiment considers the fact that the current Snort rule database contains over 6,000 attack signatures which are stored in 48 separate files; for example, each of the seven implemented security services contains the attack signatures of one of those files. Consequently, it is expected that multiple security service will be deployed on a programmable router. For the purpose of traceability only one type of security service was used throughout that experiment. Initially, one entity of the *FTP* security service was running on the programmable router and continuously further *FTP* security services were integrated into the FIDRAN system, up to a total number of 20. Figure 4.13 depicts the measured packet waiting time $T_{\text{waiting}}$ and Figure 4.14 depicts the corresponding processing time $T_{\text{process}}$. This time only the results for a PC733 system are depicted as the results for the PC3000 system show the same characteristics at another scale. The $x$-axis of the figure represents the amount of installed *FTP* services, the $y$-axis represents the offered load in MBit/s.

Figure 4.13 depicting the measured delays $T_{\text{waiting}}$ is divided into two regions: a lossy one and a loss-free one. A router can be modeled as a finite queuing system. The packet arrival rate $\lambda$ is an input parameter which is influenced by a number of factors e.g. amount of connected systems or the users' behavior. The service rate $\mu$ is determined by the traffic mix and the set of services that must be applied to each packet. When the arrival rate exceeds the service rate ($\mu < \lambda$) the queue fills up and the system starts to drop packets—it moves from the loss-free to the lossy region. The average waiting time in the lossy and the loss-free region actually depends on several factors: the traffic mix, the packet arrival rate, the security services assignment, the queuing discipline (here: one FIFO queue) and the dropping policy. However, the chart shows that the waiting time is either negligible (loss-free) or dominating the total delay caused by the FIDRAN system (lossy).

The delay $T_{\text{process}}$ represents the time that a packet needs to traverse all security services that are assigned to it. In the experiments conducted, all packets were analyzed by the set of integrated *FTP* security services. Accordingly, $T_{\text{process}}$ is expected to be proportional to the average processing time $t_s$ of the service and the amount $n$ of installed security services ($T_{\text{process}} = n \cdot t_s$). Figure 4.14 depicts $T_{\text{process}}$ over the offered load and the amount of installed services. The graphic shows that the processing time is constant for a given number of installed services. This demonstrates once more that the actual load does not influence the processing time. Furthermore, the delay $T_{\text{process}}$ increases linearly with a growing number of installed services, thus confirming the assumption that $T_{\text{process}} = n \cdot t_s$.

Summarizing, the results show that the additional overhead $T_{\text{active}}$ caused by starting a FIDRAN system and redirecting all packets to it is rather small compared to the routing delay $T_{\text{base}}$. Furthermore, the processing time $t_s$ can reasonably be assumed constant for a given security service $s$ and a given IP packet size. The processing time $t_s$ of a security services increases linearly with growing IP packets. Further, the total processing time $T_{\text{process}}$ for a packet can be approximated by the sum over the processing times $t_s$ of the security services assigned. In addition, Figure 4.13 demonstrates that an IPS might become the bottleneck of a network (lossy region), even in case of limited security services.

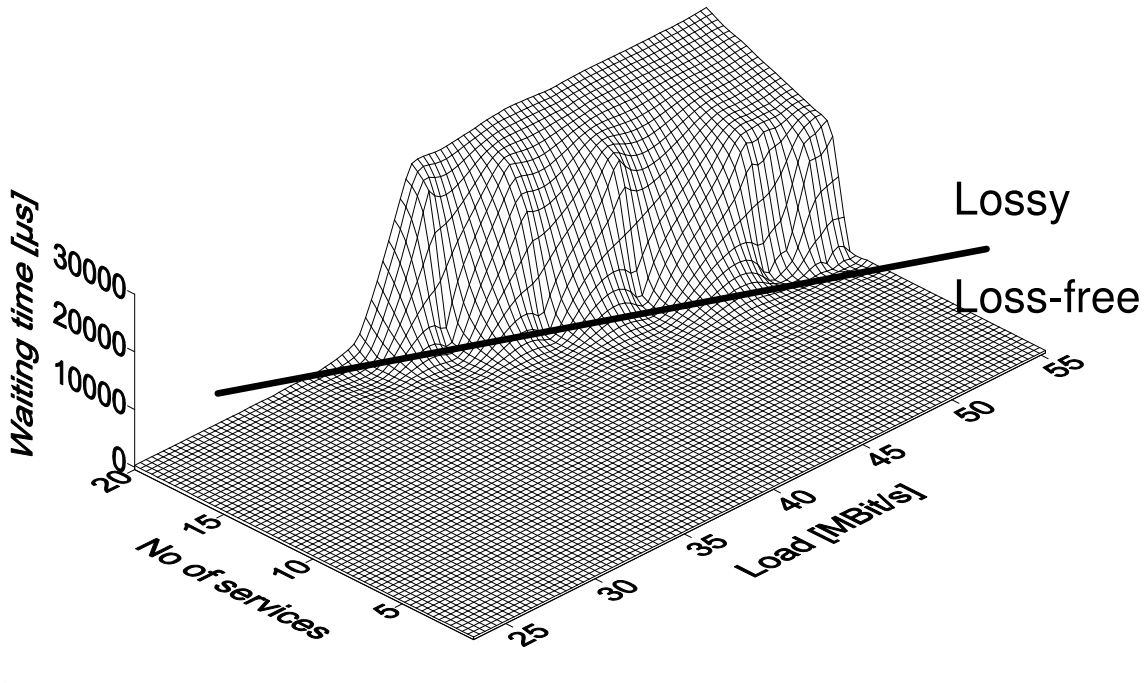



Figure 4.13: PC733: $T_{\text{waiting}}$ (T3-T4) over the number of installed FTP-security services

## 4.6 Summary

This chapter introduced the concept of Fidran, which is a *Flexible Intrusion Detection and Response Framework for Active Networks*. To build an intrusion prevention system either special purpose hardware or general purpose hardware is used. Section 4.1 identified the requirements of extensibility, flexibility and scalability to be important for an autonomous intrusion prevention overlay network but special purpose hardware contradicts them as explained in Section 2.3.6. In addition, when considering increasing traffic volumes and an increasing number of attacks the construction of a centralized intrusion prevention system is hardly to realize.

The modular concept of Fidran addresses the demands of extensibility, flexibility and scalability. Security services can be integrated into /removed from running Fidran systems. Furthermore, scalability is provided by the capability to distribute security services over multiple programmable routers. To do so, knowledge of the network is required to intelligently deploy the security services on the programmable routers. Chapter 5 discusses how that information can be gathered in an automated way and Chapter 6 introduces the deployment strategies developed for Fidran.

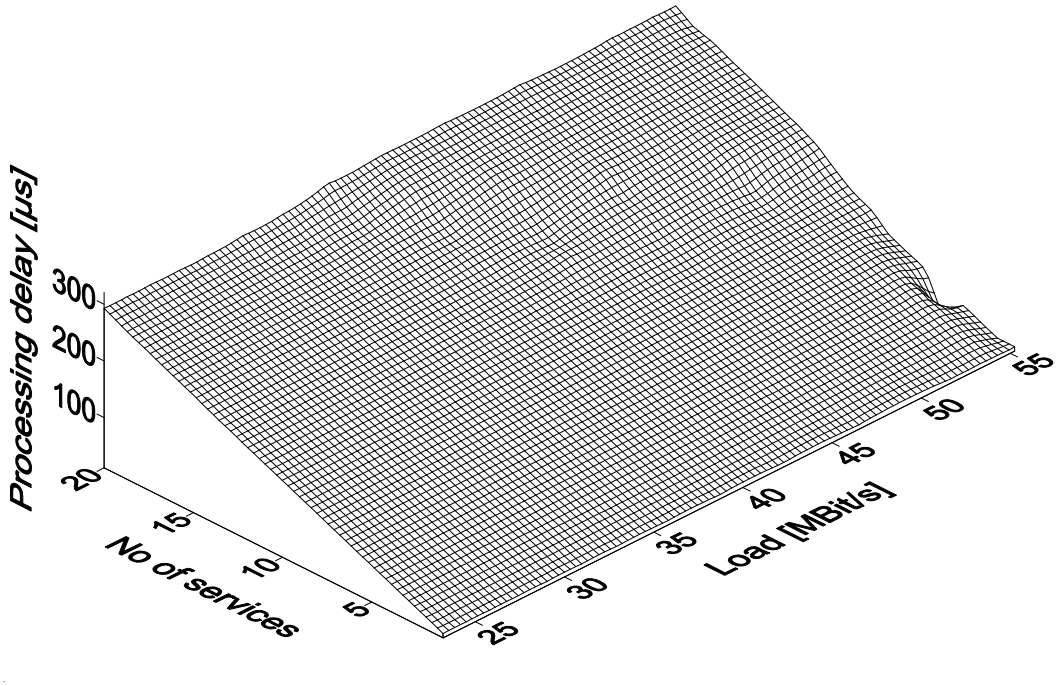



Figure 4.14: PC733: $T_{\text{process}}$ (T4-T5) over the number of installed FTP-security services

# Chapter 5

# Gathering Network Information

The capabilities of the intrusion prevention overlay network depend, as described in Chapter 4, on the deployment scenario. When operated in a backbone network information like, for example, the requested security services must be provided to the overlay network to follow the principle of demand-driven intrusion prevention. When deployed in a limited networking environment, an example is depicted in Figure 5.1, the intrusion prevention overlay network can be configured such to autonomously gather network vulnerability information. Since most attacks exploit one or multiple concrete vulnerabilities that are specific to an operating system (OS) or application, it is essential to identify:

- the topology of the network,
- the reachable end-systems:
  - distance between Internet and end-system,
  - running OS and applications,
  - amount of traffic that is destined to an end-system.

This information enables to specify both the intrusion prevention services requested by each end-system, and all placement possibilities for each service. The distance between Internet and an end-system is used to filter packets that would not reach its destination (based on the TTL). Further, the amount of traffic that is destined to an end-system is required to optimally deploy the intrusion prevention services.

Generally, three techniques exist to gather vulnerability information: *active scanning*, *passive fingerprinting* and *cooperation*. An active vulnerability scanner sends specifically crafted packets to well defined addresses and evaluates the replies. The scanner either operates in an *intrusive* or *non-intrusive* manner. The former technique identifies the vulnerabilities of an end-system by actually exploiting them. On the one hand the results gained are accurate but on the other hand this might result in an application-/system-crash. In contrast, a non-intrusive network scanner identifies a vulnerability on the basis of the type and version of a running application. Consequently, the results produced by a non-intrusive network scanner are not as accurate as the results of an intrusive one, but normal network operations are not affected by it. Moreover, the gathered vulnerability information is only as current as the latest scan and accordingly, the questions arise when and how often to scan? In addition, an active scanner creates traffic for the purpose of identifying end-systems. Considering large networks this might result in a huge amount of scanning tasks.

The passive fingerprinting technique uses a network interface card (NIC) in promiscuous mode to sniff the network. It does not collide with normal network operations but it is impossible to predict the point of time when all end-systems of a network will be identified. End-systems that do not communicate cannot be detected by the passive fingerprinting technique even though those systems are potential victims.

The cooperation technique requires a consent between end-systems and network analysis tool. One approach is that each end-system runs a small application that registers at the analysis tool and subsequently, it provides the analysis tool with the required knowledge in terms of operating system and running applications. In a second approach, the network analysis tool logs into the end-systems and gathers autonomously the relevant data. Therefore, the end-systems provide a login to the network analysis tool. The advantage of the cooperation technique is accuracy. Operating system and running applications—including version number and patch level—are precisely identified. But not all users feel comfortable about either running a small network analysis client application on their machines or providing a login to the network analysis tool.

## 5.1 Requirements and Practical Considerations

The network knowledge gathering process should occur in an efficient manner such that the scanning time is minimized and normal network operation is not disturbed. Further, the data should be collected in a non-intrusive manner, such that services and operating systems do not crash. Additionally, following practical issues must be considered:

- changing routing topologies due to link-/router-failures or overload situations which trigger rerouting mechanisms of the applied *Interior Gateway Protocols* (IGP) like *Open Shortest Path First* (OSPF) or the *Routing Information Protocol* (RIP).
- varying amount of reachable end-systems due to variable online times; for example a special purpose server is expected to be permanently reachable whereas a private end-system might only be connected to the Internet for a short period of time. A related difficulty is the *Dynamic Host Configuration Protocol* (DHCP) [53] which dynamically assigns IP-addresses to clients.
- varying end-system configuration: an end-system might become vulnerable due to the start of a new service for which an exploit exist or a patch is applied to an end-system eliminating a set of security holes.
- the publication of a new security hole/vulnerability which results in the need to install adequate protection mechanisms in front of the relevant systems.
- the application of *Network Address Translation* (NAT) [17] veils the amount of end-systems that are represented by a single visible IP-address. In some cases—depending on the operating systems of the hosts—it is possible to identify the number of systems that are located behind a NAT box [25].

Following the argument that intrusion prevention should not modify end-systems, the cooperation technique is inappropriate for gathering the required information about the network.

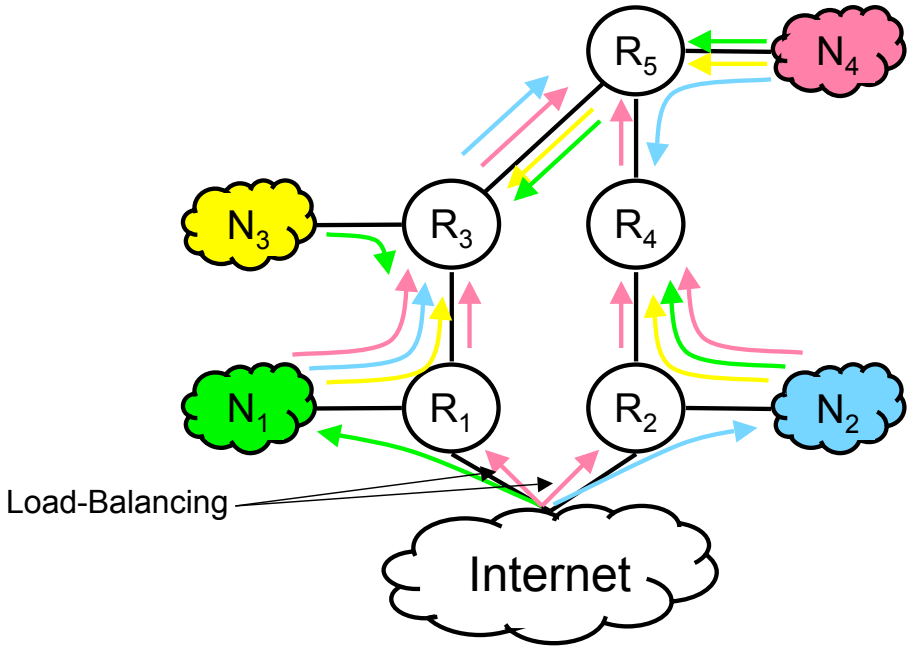



Figure 5.1: A limited networking environment

Hence, the proposed network analysis process makes use of the techniques active scanning and passive fingerprinting.

In detail, the analysis of a network consists of three phases:

1. the *boarder gateway router* (BGR) calculates the network-backbone using the routing tables of all the routers of the network.

2. active scanners are used to discover and analyze the end-systems of the network in terms of distance, operating system and running services.

3. always running passive fingerprinting components measure the end-systems' specific traffic volumes and observes the network traffic for new, not yet registered IP-addresses (indicating new end-systems connected to the network).

The approach described requires the following preconditions: first, the BGRs—here $R_1$ and $R_2$—are programmable routers; second, the BGRs know the addresses of all routers—programmable as well as conventional ones—of the limited networking environment; third, the BGRs are capable to poll the routing tables, for example via SNMP, of the routers. Finally, the routes in the limited networking environment are symmetric such that the active scanner(s) will see the responses sent by the probed end-systems.

## 5.2 Gathering Network Knowledge

Figure 5.1 depicts an example of a limited networking environment consisting of routers ($R_1$–$R_5$) with $R_1$ and $R_2$ acting as BGRs and subnetworks ($N_1$–$N_4$); each one representing a variable amount of varying end-systems. According to its name a boarder gateway router connects a limited networking environment to the Internet and as shown in the scenario multiple BGRs between a network and the Internet can exist. The arrows in Figure 5.1 represent the paths between the subnetworks and to the Internet. In order to provide protection an intrusion prevention service must be placed on the path between attacker and victim.

Initially, the BGRs figure out the structure of the network, meaning they analyze how the routers of the network are connected with each other. It must be remarked that the network topology discovery process is restricted to network elements of the network-layer of the *ISO/OSI* reference model. This means, network components that operate in the data-link layer like hubs or switches remain invisible.

At system startup the BGRs request the routing table of the other routers in the network. On the basis of these routing tables and the assumption of symmetric routes, they calculate the network backbone using Algorithm 1. The algorithm starts by identifying the children of a BGR which are the next-hop routers towards the network's end-systems. For this purpose, the routing tables of all routers of the network are searched for *default* routing entries (aka default routes) which, if present, are the last entries in a routing table. Normally, the default route points towards the router that has a connection to the Internet. Take the following entry as an example:

```
Destination     Gateway          Genmask     Flags     ...     Interface
0.0.0.0         192.168.50.1     0.0.0.0     UG        ...     eth1
```

The default route tells the routing daemon to send all IP-packet that have not yet been routed via interface *eth1* to the next hop, which is specified via the gateway entry (192.168.50.1). The flags *U* and *G* indicate that the route is up and that the specified gateway must be used. Now, in case that the default route lists the BGR as gateway, the router is directly connected to the BGR and is its child. This children identification process is successively repeated for all routers of the network. The outcome of the algorithm is graphically depicted in Figure 5.2. In a next step, the BGR triggers the programmable routers next to subnetworks to scan the corresponding IP-address range.

Each programmable router is supplied with a network scanning component which is used to discover and analyze end-systems. To do so, the scanner sends specifically crafted packets to the addresses specified by the BGR and evaluates the replies (see [58, 59]). The local security policy stores all reachable destination addresses—tuple IP-address, port number—of the corresponding subnetwork in a hash-table. Afterwards, each programmable router sends its results to the BGR which combines them to an overall map of the AS, showing running end-systems and their configurations. The AS-map is used by the BGR to calculate an optimal distribution of the intrusion prevention services (see Chapter 6). Then, the BGR triggers the programmable routers to install the required intrusion prevention services and to remove those that are no longer requested according to the newly calculated optimal distribution.

To address network dynamics—the vulnerability information represents the view of the last scan—the network analysis process is bipartite. On the one hand the active scanning

```
program buildnetwork

Parameters:
  bgrlist;        // List of BGRs of the network
  routerlist;   // List of routers of the network without BGRs
1.) Init
    router r = bgrlist.first;
2.) Identify the children of r
    for(allRouters of routerlist)
      for(allInterfaces of router)
        for(allRoutingEntries of interface)
          if((routingentry == defaultroute) && (gateway == r))
            r.addchild(router)
3.) Goto next router
    if(r $\in$ bgrlist)
      r = r.childrenlist.first()
    else if(r.childrenlist.next() != NULL)
      r = r.childrenlist.next()
    else
      r = bgrlist.next()
    r = r->childlist()->next
4.) Identify the children of router r
    Goto step 2

end buildNetworkHelper
```

**Algorithm 1:** Specifying the Backbone of a given Network

process is repeated in regular intervals to update network map and security policies. On the other hand, to immediately catch changes, last hop traffic selectors scan the downlink network traffic for new destination addresses. Whenever one is detected, the traffic selector triggers the BGR to actively scan it. Afterwards, the BGR updates the security policies.

## 5.3 A Case Study

To assess the feasibility of the approach a network scanner was integrated into the Linux-based Fidran prototype was used. As network scanner *Nessus-3.0.2* [52] was chosen as it performed well in a comparison of existing vulnerability scanners [56, 103]. Moreover, Nessus provides an overall amount of $10,000$ security checks which are updated each day. Further, the testbed which is depicted in Figure 5.3) was setup:

1. host-1: the victim, running Windows XP, SP1, IIS,
2. host-2: the programmable router,
3. host-3: the attacker.

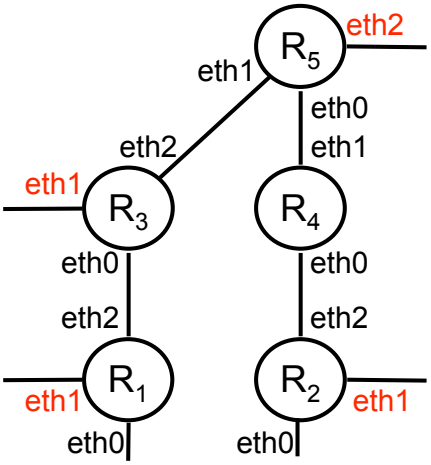



Figure 5.2: Outcome of Algorithm 1: Network backbone

All packets originating from the attacker and destined to the victim are routed via the programmable router and, initially, no Fidran system was running on the programmable router. To attack the victim the *Metasploit* framework [95] which is an open-source platform for developing, testing, and using exploit code. With Metasploit it can be demonstrated that an attacker is able to execute arbitrary code on the victim. The successful attack exploits the vulnerability described in Microsoft's security bulletin *ms03-026* (superseded by *ms03-039*). In detail, the victim is running a vulnerable implementation of the RPC interface which can be compromised by an attacker to execute arbitrary code. Worms like *Blaster*, *Nachia* or *Welchia* used, among others, that security hole to gain access to end-systems.

To specify the amount of intrusion prevention services that are required to protect a typical user's end-system, a host running Microsoft's Internet Information Server on a Windows XP Professional operating system—patched with Service Pack 1—was scanned. Nessus provides several output formats, but all of them are inappropriate for an automated evaluation. The output for two exemplary security holes is shown in Vulnerability 1 and 2.

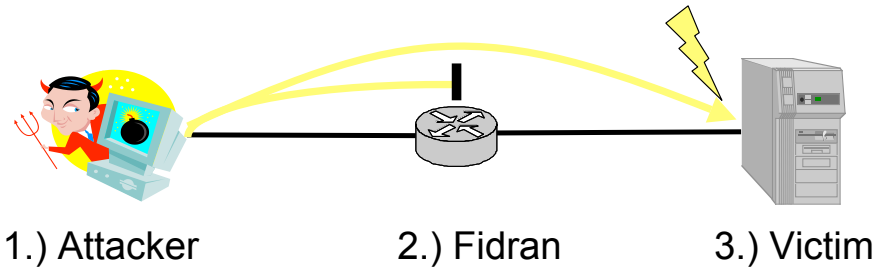



Figure 5.3: Network Knowledge Gathering

```
results|192.168.100|192.168.100.68|epmap (135/udp)|11890|
Security Hole|
A security vulnerability exists in the Messenger Service
that could allow arbitrary code execution on an affected
system. An attacker who successfully exploited this
vulnerability could be able to run code with Local System
privileges on an affected system, or could cause the
Messenger Service to fail. Disabling the Messenger Service
will prevent the possibility of attack.
This plug-in actually checked for the presence of this flaw.
Solution : see
http://www.microsoft.com/technet/.../ms03-043.mspx
Risk factor : High
CVE : CVE-2003-0717
BID : 8826
Other references : IAVA:2003-A-0028, IAVA:2003-a-0017,
                   IAVA:2003-b-0007
```

Vulnerability 1: CVE-2003-0717

Several approaches to name vulnerabilities exist and most of them are supported by Nessus. The *Common Vulnerabilities and Exposures* list [2] provides the most comprehensive index of standardized names for vulnerabilities. Microsoft uses its *Security Bulletin ID* to name Windows-specific vulnerabilities. Further examples, are Bugtraq- and IAVA-ID. But unfortunately, none of them is exhaustive: For example, Vulnerability 1 is stored in the CVE list (*CVE-2003-0717*) but Vulnerability 2 is not. Hence, to automatically configure the Fidran overlay network a script parses and extracts the Nessus report for security holes.

To follow the principle of demand-driven intrusion prevention two types of intrusion prevention services were implemented:

- intrusion prevention services containing all Snort attack signatures that compromise a

```
results|192.168.100|192.168.100.68|microsoft-ds (445/tcp)
|12209|
Security Hole|
Synopsis :
Arbitrary code can be executed on the remote host due
to a flaw in the LSASS service.
Description :
The remote version of Windows contains a flaw in the
function DsRolerUpgradeDownlevelServer of the Local
Security Authority Server Service (LSASS) which may allow
an attacker to execute arbitrary code on the remote host
with the SYSTEM privileges.
A series of worms (Sasser) are known to exploit this
vulnerability in the wild.
Solution :
Microsoft has released a set of patches for Windows NT,
2000, XP and 2003 :
http://www.microsoft.com/technet/.../ms04-011.mspx
Risk factor : Critical
/ CVSS Base Score : 10
(AV:R/AC:L/Au:NR/C:C/A:C/I:C/B:N)
Other references : IAVA:2004-A-0006
```

Vulnerability 2: LSASS-specific

concrete vulnerability—the vulnerability is clearly identified via the *Common Vulnerabilities and Exposures* (CVE) number, and

- intrusion prevention services containing all Snort attack signatures to protect a concrete application—the application is identified via its name.

A Nessus plug-in checks for the presence of one or multiple concrete vulnerabilities and in most cases they are reported via the corresponding CVE numbers. Accordingly, if *all known* vulnerabilities of an application would have a CVE-number assigned then the vulnerable application can be protected by the intrusion prevention service that contains all CVE-specific attack signatures. Otherwise, if Nessus reports *at least one* vulnerability that does not possess a CVE number then the vulnerable application is protected by the intrusion prevention service containing all application-specific attack patterns, even if that application also has CVE-named vulnerabilities. In this way, at least for some applications, the number of attack signatures can be significantly reduced and consequently the false positive rate (see Equation (4.1)).

Next, Fidran was started on the programmable router and Nessus identified the following vulnerabilities:

1. **CVE-2003-0717**: A security vulnerability exists in the Messenger Service that could allow arbitrary code execution on an affected system
2. **CVE-2003-0818**: ASN.1 parsing vulnerabilities, arbitrary code can be executed on the remote host
3. **CVE-2003-0715, CVE-2003-0528, CVE-2003-0605**: RPC interface buffer overrun, arbitrary code can be executed
4. **CVE-2005-0048, CVE-2004-0790, CVE-2004-1060, CVE-2004-0230, CVE-2005-0688**: Arbitrary code can be executed on the remote host due to a flaw in the TCP/IP stack
5. **CVE-2005-1984**: Arbitrary code can be executed on the remote host due to a flaw in the Spooler service
6. **CVE-2005-1206**: Arbitrary code can be executed on the remote host due to a flaw in the SMB implementation
7. **CVE-2004-0212**: A remote code execution vulnerability exists in the Task Scheduler, arbitrary code can be executed on the remote host (task scheduler).
8. **CVE-2006-0034, CVE-2006-1184**: A vulnerability in MSDTC could allow remote code execution.
9. **CVE-2005-2119, CVE-2005-1978, CVE-2005-1979, CVE-2005-1980**: A vulnerability in MSDTC could allow remote code execution.
10. **No CVE number**: Arbitrary code can be executed on the remote host due to a flaw in the LSASS service.

This list would lead to the installation of 19 CVE-specific and 1 application-specific intrusion prevention services. But multiple CVE-specific intrusion prevention services contain the same set of attack signatures, so that the total number of services to install is actually 11.

For example, vulnerabilities *CVE-2003-0715, CVE-2003-0528* and *CVE-2003-0605* (item 3 in the listing), the ones exploited in the attack, require one and the same intrusion prevention service, containing two attack signatures. Fidran automatically loaded the demanded intrusion prevention service; repeated attack attempts failed as Fidran recognized and filtered the attack. A default configured Snort system would check at least 76/810 signatures for each TCP packet addressed to the victim's destination ports 135/445—leading to a much higher false positive rate. Of course Snort could also be manually configured like Fidran, but that would require checking the applicability of each of the 886 rules by hand, and then keeping it up-to-date—clearly a much more expensive and time-consuming task for a single security hole.

Altogether, nine out of ten detected vulnerabilities can automatically be protected by Fidran. But, the approach would fail to protect the task scheduler (Vulnerability 7 of the above listing) because Snort does not provide adequate attack patterns (neither for CVE-2004-212 nor for the Task Scheduler application).

## 5.4 Summary

This chapter discussed the possibilities for an autonomous intrusion prevention overlay network, involving the capability to gather network knowledge and to self-configure itself. Regular active scans are used to discover running hosts and identify their vulnerabilities; in addition, new hosts/applications are immediately identified by traffic selectors on the look out for new destination addresses. The information gathered is then used by Fidran to place the required intrusion prevention services—each service provides protection for a concrete vulnerability or application—in front of the end-system that need them. The principle of demand-driven intrusion prevention reduces the amount of security checks that are performed per flow and, as a consequence, reduces also the overall false-positive rate.

Based on a concrete vulnerability, the case study demonstrated that Fidran is able to protect against known attacks which exploit a vulnerability that can remotely be identified. The protection provided by the Fidran overlay network is limited by the availability of adequate protection mechanisms and the capability of the integrated network scanner to accurately identify vulnerabilities: end-systems remain vulnerable in case that the network scanner fails to identify security holes (false negatives), and superfluous intrusion prevention services are installed in case that Nessus reports vulnerabilities that actually do not exist (false positives). Unfortunately, identifying all false negatives is not possible per definition. Nevertheless, a more thorough study of the accuracy of Nessus would identify false positives.

Finally, an upcoming creation of *vulnerability signatures*—signatures that matches all exploits of a given vulnerability [33]—facilitates the matching between vulnerability and defense mechanism.

# Chapter 6

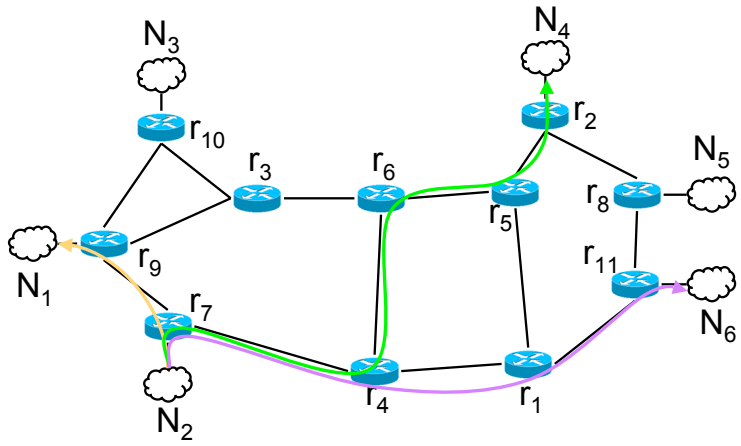
# Optimal Deployment Strategies

Strict quality of service (QoS) requirements of existing and emerging applications are challenging and contradicting the operation of an intrusion prevention overlay network which inevitably decreases network performance as all packets are analyzed for malicious content before being forwarded. *Traffic engineering* is the process of arranging how traffic flows through a network [139]. Further, *constraint based routing* [43] is to find routes that are subject to some constraints such as delay requirements. While determining a route, constraint based routing considers not only topology of the network, but also the specified requirement of the flow and the resource availability of the links/nodes. The solution provided by constraint based routing may consist of longer but lighter loaded paths compared to the heavier loaded shortest paths solution, and consequently, network traffic is thus distributed more evenly.

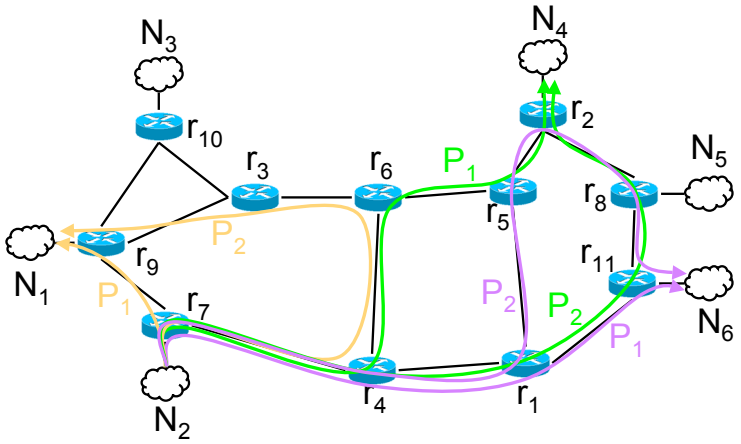
This chapter introduces the optimization framework developed to calculate optimal security service deployment strategies including the determination of the paths to minimize the impact of Fidran on the network performance. The flow requirements considered while specifying the routes are described in Section 6.2. *Mixed Integer Linear Programs* (MILP) are formulated to assign routes and placement of security services. Further, the optimization framework covers the scenarios:

- Predefined routing:
  - single-path
  - multipath
- Joint traffic routing and security service distribution

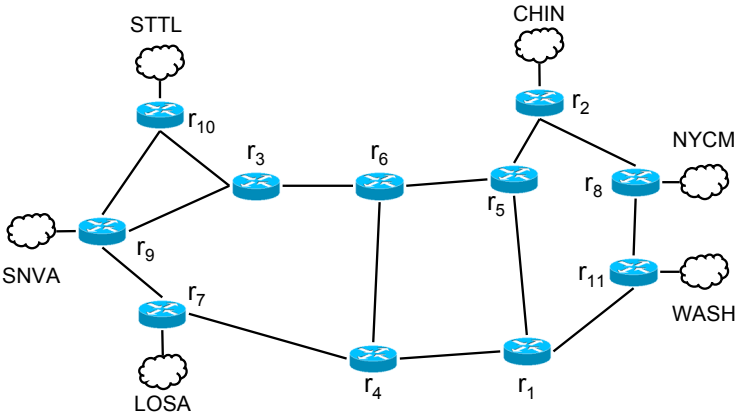
Predefined routing implies that at least one path from any source to any destination is specified—as a result the routing tables are set. In a single-path routing environment, a single path exists between any two subnetworks: Figure 6.1(a) depicts a fixed path from $N_2$ to $N_1$, $N_2$ to $N_4$ and $N_2$ to $N_6$. All traffic originating from $N_2$ and destined to $N_1$, $N_4$ or $N_6$ is routed via the respective path. In a multipath routing environment multiple paths exist between subnetworks, allowing for load-balancing. Figure 6.1(b) depicts for each connection from source $N_2$ to destinations $N_1$, $N_4$ and $N_6$ two possible paths $P_1$ and $P_2$ to route the corresponding traffic. For example, to balance the load half of the traffic can be




(a) Environment with predefined singlepath routes




(b) Environment with predefined multipath routes




(c) Joint routing and service deployment environment

Figure 6.1: Optimization scenarios

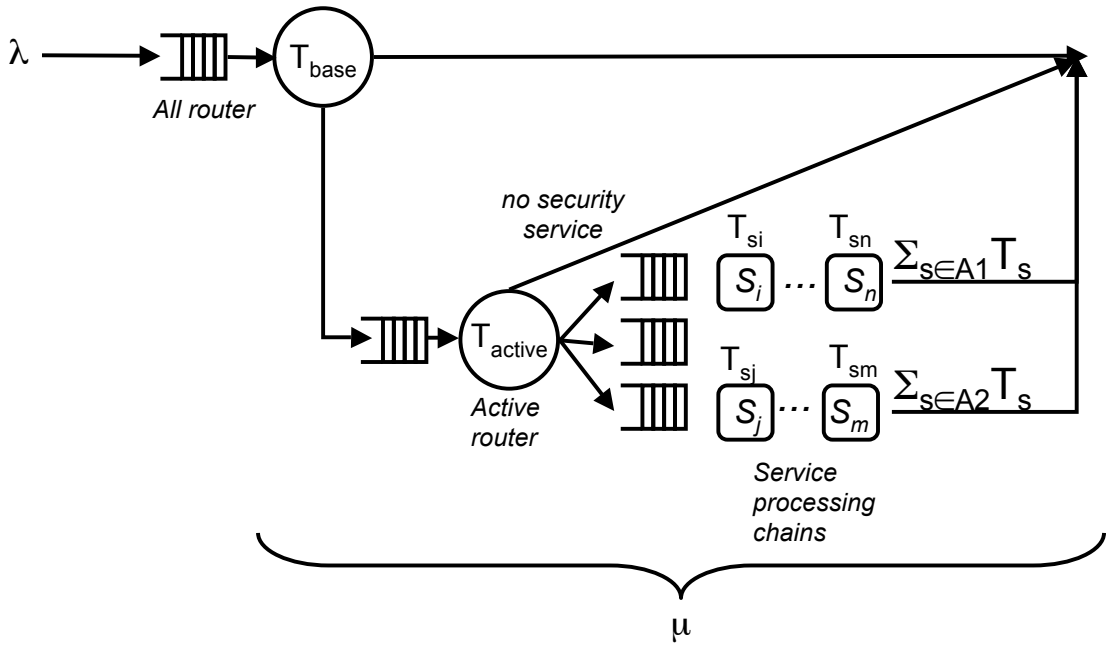



Figure 6.2: Decomposition of packet's delay inside a router

routed via path $P_1$ and the other half via path $P_2$. Finally, in the last scenario the network topology is given but no paths between the subnetworks are defined (see Figure 7.17). Hence, the optimization framework specifies traffic routing—a single path from each source to each destination—and the distribution of the requested security services.

Summarizing, a deployment strategy specifies the placement of programmable router nodes, and the distribution of security services across such nodes so as to optimize certain objectives. In case of joint traffic routing and security services deployment also the routes (single paths) are specified by the optimization framework. The optimization framework provides two objective functions that are discussed in detail in Section 6.2:

1. minimize the total number of programmable routers used and
2. minimize the maximum workload of any router node in the network.

The chapter is structured as follows. The subsequent section explains the system model of a router that was used throughout the optimization framework. Section 6.2 covers both objective functions. The remaining sections (Sections 6.3-6.5) describe the scenario-specific MILPs.

## 6.1 Router System Model

The behavior of a router can be modeled as a finite queuing system consisting of an arrival process, a queueing discipline and a service mechanisms (see Figure 6.2). The packet arrival

rate $\lambda$ is an input parameter which is influenced by a number of factors e.g. amount of connected systems or the users' behavior. The service rate $\mu$ is determined by the traffic mix, the set of services that must be applied to each packet and the number/performance of the integrated processors. For example, in case of a dual-/multi-processor system, one processor can be assigned to handle the arriving packets whereas the remaining processors perform the security analysis. Consequently, the waiting time of a packet is bounded from below by the service requests of the packets which are in front of it. In the following a programmable router is modeled such that it resembles a single-processor system. Multiple processor machines are respected by a performance parameter scaling down the overall processing time.

When the arrival rate exceeds the service rate ($\mu < \lambda$) the queues fill up and the system starts to drop packets. As has been shown empirically in Section 4.5, the processing time of a packet at a router can be modeled as a sum of three components:

- the basic delay $T_{base}$ representing the routing delay in a standard network router;
- the delay $T_{active}$ representing the overhead necessary to decide whether a packet must be processed on the programmable router (applies only if the router is programmable);
- the sum of the processing times $T_s$ for each service $s$ that is applied to a packet.

Accordingly, the total processing time for a packet that receives services in the set $A \subseteq S$ at a node is given by the sum:

$$T_{base} + T_{active} + \sum_{s \in A} T_s \tag{6.1}$$

## 6.2 Objective Functions

The presented optimization framework provides the following objective functions:

(i) the minimization of the number of programmable routers used in a network while fulfilling all security requests and keeping all router queues bounded;

(ii) the minimization of the maximal workload of a programmable router while fulfilling all security requests and keeping all router queues bounded.

The first objective is to minimize the number of programmable routers which can be expressed as following:

$$\min \sum_i x_i \tag{OPT1}$$

The expenses of buying programmable routers as well as their complexity are much higher than for standard routers. As a result of this, network operators are interested in protecting communication infrastructures with the objective to minimize the costs/complexity. This corresponds to objective function (OPT1) which minimizes the overall number of programmable routers in a network under the above mentioned constraints.

The idea for the second optimization model is to evenly distribute the load among routers. The difference between processing time and interarrival time corresponds to the free capacity

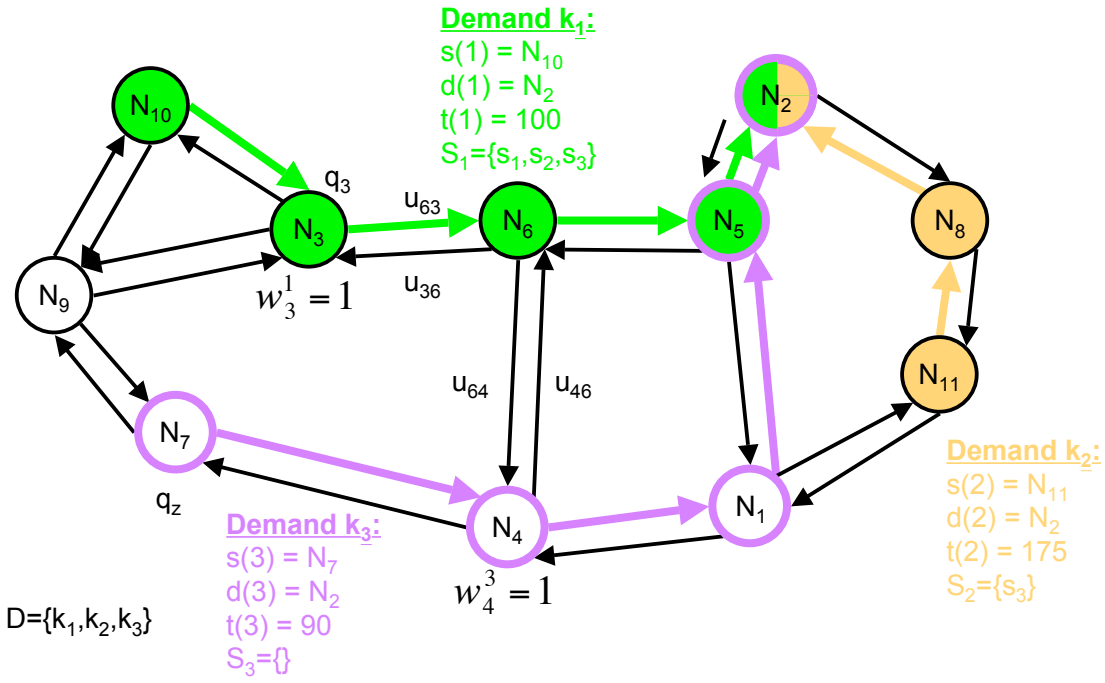



Figure 6.3: Predefined singlepath routing - security services placement possibilities

of a programmable router. Minimizing the maximum router utilization which is denoted $v$, thus improves the overall performance:

$$\min v \tag{OPT2}$$

The composition of $v$ is explained in detail in the following sections.

## 6.3 Predefined Single-Path Routing

In the following an integer linear programming model [98, 116] for finding a deployment strategy for the single-path routing scenario is introduced. A network can be modeled as a directed graph with the set of nodes $N$ representing the routers and the set of edges $E$ symbolizing the links. Figure 6.3 depicts the network graph corresponding to the network shown in Figure 7.17. A link is an unidirectional connection between two routers that is identified via the tuple start-/end-node. The bandwidth of a link $e \in E$ is denoted $u_e$. Traffic is incorporated into the optimization framework in terms of traffic demands denoted $D$. A traffic demand $k \in D$ consists of a source node $s(k)$, a destination node $d(k)$ and a value $t(k)$ which defines the volume of it. Three exemplary traffic demands $k_1$ with $s(1) = N_{10}, d(1) = N_2, t(1) = 100$, $k_2$ with with $s(2) = N_{11}, d(2) = N_2, t(2) = 175$ and $k_3$ with $s(3) = N_7, d(3) = N_2, t(3) = 90$ are depicted in Figure 6.3. The set of available services for protecting subnetworks from intrusions is denoted by $S = \{s_1, \ldots, s_m\}$. A set of security services denoted $S_k$ ($S_k \subset S$) can be assigned to a traffic demand $k$. The composition of $S_k$ depends on the specific properties of subnetworks $s(k)$ and $d(k)$ and their relationship.

In the given example, security services $S_1 = \{s_1, s_3, s_5\}$ are assigned to traffic demand $k_1$, security service $S_2 = \{s_3\}$ to traffic demand $k_2$ and no security service is assigned to traffic demand $k_3$ ($S_3 = \{\ \}$) as, for example, traffic from node $N_7$ destined to $N_2$ is sent through an *IPSec*-tunnel and hence traffic $k_3$ is encrypted and cannot be analyzed for malicious content. The security services listed in $S_k$ must be deployed on the routers such that all traffic of demand $k$ is routed through them. In this scenario one predefined path $P^k \in P$ to route traffic of demand $k$ from source $s(k)$ to destination $d(k)$ exists and as a result of this security services $S_k$ can only be deployed on routers that are part of this path. Figure 6.3 depicts the security services placement possibilities for the three exemplary traffic demands. Parameter $w_i^k$ denotes whether node $i$ is part of path $P^k$. In the given example, security services $S^1$ can only be deployed on nodes $N_{10}, N_3, N_6, N_5$ and $N_2$. Moreover, it can be seen that all subnetworks are eliminated from the graph. Traffic demands and security service requests are pushed into the corresponding gateway nodes. For example, subnetwork $N_4$ of Figure 7.17 is connected to the network via gateway router $R_2$ and hence, traffic demands $k_1$ and $k_2$ of Figure 6.3 have node $N_2$ as destination.

The processing time of security service $i$ is denoted $T_i$. $T_{base}$ represents the routing delay in a standard network router and $T_{active}$ represents the overhead necessary to decide whether a packet must be processed on the programmable router.

The MILP decides on whether a node $i$ is programmable and where to place the security services. In accordance with the situation, the problem formulation includes two decision variables. The decision whether or not a node $i \in N$ is programmable, is represented by variable $x_i \in \{0, 1\}$. This variable has the following semantics:

$$x_i = \begin{cases} 1 & \text{if node } i \text{ is programmable,} \\ 0 & \text{otherwise.} \end{cases}$$

The decision where services should be placed is modeled by the decision variable $y_{is}^k$. Variable $y_{is}^k$ is either of type binary—$y_{is}^k \in \{0, 1\}$—or continuous/real—$y_{is}^k \in [0, 1]$. In the former, the semantics of the variable is

$$y_{is}^k = \begin{cases} 1 & \text{if service } s \text{ is running on node } i \text{ for traffic demand } k, \\ 0 & \text{otherwise.} \end{cases}$$

In the latter, variable $y_{is}^k$ denotes the fraction of demand $k$ that is provided service type $s$ at node $i$. This increases the possibility of load balancing as a traffic demand $k$ must not be processed in its totality at a node $i$ by service $s \in S_k$. However, it must be considered that a flow cannot be divided into arbitrary smaller parts. A TCP flow that is fractionally analyzed by a security service $s$ at multiple nodes $i$ suffers under varying RTTs and out-of-order packet delivery. Here a traffic demand is the aggregation of many smaller flows defined by a quadruple $[sAddr, dAddr, sPort, dPort]$. Such a small flow should be the smallest traffic unit to be assigned to a tuple $node, service$. The parameters and variables which are the basis for the formulation of the optimization problem are summarized in Table 6.1 and Table 6.2.

First of all, to protect the end-systems connected to a node $n \in N$ all security service $s \in S_k$ must be placed such that all network traffic is routed through them. Consequently, to

secure all end-systems of a network the security service requests $s \in S_k$ of all traffic demands $k \in D$ must be fulfilled. This can be modeled by the following linear constraint:

$$\sum_{i \in P_k} y_{is}^k \geq 1 \qquad \forall\ i \in \mathbf{N}, \forall\ s \in \mathbf{S_k}, \forall\ k \in \mathbf{D}. \tag{6.2}$$

For placing a security service on a node, the referring node has to be a programmable. In terms of the defined variables, this requirement can be expressed as follows:

$$x_i \geq y_{is}^k \qquad \forall\ i \in \mathbf{N}, \forall\ s \in \mathbf{S_k},\ \forall\ k \in \mathbf{D}. \tag{6.3}$$

An assignment of the variables $x$ and $y$ that satisfies (6.2) and (6.3) corresponds to a deployment strategy. In order to avoid an explosion of the waiting time and operate in the loss-free region—see Figure 4.13—a *stability constraint* is added to the MILP. The idea is the following: a programmable router should be able to process the incoming packets fast enough such that its buffer does not overflow. In other words, the average packet *processing time* per packet—can be calculated using Equation (6.1)—must not exceed the average packet *interarrival time*.

A packet belonging to traffic demand $k \in \mathbf{D}$ thus has at node $i \in \mathbf{N}$ a processing time of:

$$T_{\text{base}} + T_{\text{active}} \cdot x_r i + \sum_{s \in S_k} T_s \cdot y_{is}^k$$

Assuming that all packets at the router have the same characteristics, the average processing time can now be calculated as a weighted average of the traffic demands that are served by node $i$. With parameter $w_i^k$ defining whether node $i$ is part of path $P^k$ for traffic demand $k$, the total traffic entering node $i$ is:

$$f_i = \sum_{k \in D} t(k) w_i^k \qquad \forall i \in \mathbf{N} \tag{6.4}$$

Thus, the average time between consecutive packet arrivals is $1/f_i$. The average processing time of a packet at a programmable router node $i$ is:

$$T_{base} + T_{active} + \sum_{k \in D} \frac{t(k) w_i^k}{f_i} \sum_{s \in S_k} T_s y_{is}^k$$

The bilinear product term $w_i^k y_{is}^k$ is 0 when $w_i^k = 0$, i.e., when node $i$ is not on the path for demand $k$. In that case, $y_{is}^k$ is also 0. Otherwise, when $w_i^k = 1$, the product term equals $y_{is}^k$, and thus, in both cases $w_i^k y_{is}^k = y_{is}^k$. Hence, the average processing time becomes:

$$T_{base} + T_{active} + \frac{1}{f_i} \sum_{k \in D} t(k) \sum_{s \in S} T_s y_{is}^k$$

This must be at most $1/f_i$, whence:

$$f_i \cdot (T_{base} + T_{active}) + \sum_{k \in D} t(k) \sum_{s \in S} T_s y_{is}^k \leq 1 \qquad \forall i \in \mathbf{N}$$

Hence, the stability constraint for a programmable node ($x_i = 1$) is:

$$\sum_{k \in D} t(k) w_i^k (T_{base} + T_{active}) + \sum_{k \in D} t(k) \sum_{s \in S} T_s y_{is}^k \leq 1 \qquad \forall i \in \mathbf{N}.$$

In a more general way, taking into account standard routers as well as programmable ones, the complete stability constraint is:

$$\sum_{k \in D} t(k) w_i^k (T_{base} + T_{active}) + \sum_{k \in D} t(k) \sum_{s \in S} T_s y_{is}^k \leq 1 + C(1 - x_i) \qquad \forall i \in \mathbf{N} \tag{6.5}$$

with

$$C = \sum_{k \in D} t(k)(T_{base} + T_{active} + \sum_{s \in S} T_s). \tag{6.6}$$

$C$ is a so-called *indicator*-variable [135] (aka as *big-M formulation*), it is specified such that constraint (6.5) is correct when node $i$ is a programmable router ($x_i = 1$), and is redundant otherwise ($x_i = 0$). The complete mixed integer program with the objective to minimize the number of programmable routers is summarized in MILP 1.

The idea of the second optimization model is to evenly distribute the load among nodes. The slack in (6.5)—the difference between processing time and interarrival time—corresponds to the free capacity of node $i$. If this slack vanishes or almost vanishes for any node, the delay performance of the node will degrade rapidly. Thus, the goal of the second objective function to maximize the minimum slack in (6.5) over all nodes. This is equivalent to minimizing the maximum left-hand side in (6.5). The corresponding problem formulation is given in MILP 2.

---

$$\text{minimize} \quad \textstyle\sum_{i \in N} x_i$$

subject to:

$$\begin{aligned}
\sum_{i \in P_k} y_{is}^k &\geq 1 && \forall\ i \in \mathbf{N}, \forall\ s \in \mathbf{S_k}, \forall\ k \in \mathbf{D} \\
x_i &\geq y_{is}^k && \forall\ i \in \mathbf{N}, \forall\ s \in \mathbf{S_k},\ \forall\ k \in \mathbf{D} \\
\sum_{k \in D} t(k) w_i^k (T_{base} + T_{active}) + \sum_{k \in D} t(k) \sum_{s \in S_k} T_s y_{is}^k &\leq 1 + C(1 - x_i) && \forall i \in \mathbf{N} \\
x_i &\in \{0, 1\} && \forall\ i \in \mathbf{N} \\
y_{is}^k &\geq 0 && \forall i \in \mathbf{N}, k \in \mathbf{D}, s \in \mathbf{S_k}
\end{aligned}$$

---

MILP 1: Predefined Single-Path Routing - Minimizing the Amount of Programmable Routers

Table 6.1: Predefined Single-Path Routing - Parameter Space

| Parameter | Description |
|---|---|
| N | set of nodes |
| E | set of links |
| D | set of traffic demands |
| $s(k)$ | source node of traffic demand $k$ |
| $d(k)$ | destination node of traffic demand $k$ |
| $t(k)$ | value of traffic demand $k$ [flow units] |
| $S$ | set of security services $S = \{s_1, s_2, \ldots, s_a\}$ |
| $S_k$ | set of security services assigned to traffic demand $k$ |
| $P$ | set of paths $P = \{P^1, P^2, \ldots, P^n\}$ |
| $P^k$ | predefined single-path from $s(k)$ to $d(k)$ for routing demand $k$ |
| $w_i^k$ | defines whether node $i$ is on the predefined path for demand $k$ |
| $T_{base}$ | core routing delay |
| $T_{active}$ | additional delay caused by a programmable router |
| $T_s$ | processing time of service $s$ |
| $u_e$ | bandwidth of link $e$ [flow units] |
| $f_i$ | total network traffic entering node $i$ |
| $C$ | is specified such that constraint (6.5) is correct when node $i$ is a programmable router ($x_i = 1$), and is redundant otherwise ($x_i = 0$) |

Table 6.2: Predefined Single-Path Routing - Decision Variables

| Variable | Description |
|---|---|
| $x_i$ | defines whether node $i$ is a programmable router |
| $y_{is}^k$ | specifies the fraction of demand $k$ that is provided service type $s$ at node $i$ |

## 6.4 Predefined Multipath Routing

To increase the potential for load balancing, the problem formulation of the preceding section is extended towards multipath routing, meaning that a traffic demand $k$ can be split across multiple paths from source $s(k)$ to destination $d(k)$. Figure 6.4 depicts for traffic demands $k_1, k_2$ and $k_3$ two possible paths (solid and dotted arrows) and as it can be seen—in contrast to the previous problem formulation—more possibilities to deploy the requested security services $S_k$ exist. Here $P^k = \{p1, p2, \ldots\}$ denotes the set of available paths from source $s(k)$ to destination $d(k)$ for routing demand $k$. For example, the $K$-shortest hop paths from $s(k)$ to $d(k)$ can be chosen as the set $P^k$.

The extension of the problem formulation towards multipath routing increases the number of decision variables as the optimization model must decide on:

- whether a node $i$ is programmable

minimize $v$

subject to: (6.2), (6.3)

$$\sum_{k \in D} t(k) w_i^k (T_{base} + T_{active}) + \sum_{k \in D} t(k) \sum_{s \in S} T_s y_{is}^k \leq v + C(1 - x_i) \quad \forall i \in \mathbf{N} \tag{6.7}$$

$$x_i \in \{0, 1\} \quad \forall\, i \in \mathbf{N} \tag{6.8}$$

$$y_{is}^k \geq 0 \quad \forall i \in \mathbf{N}, k \in \mathbf{D}, s \in \mathbf{S_k} \tag{6.9}$$

MILP 2: Predefined Single-Path Routing - Minimizing the Maximal Router Utilization

Table 6.3: Predefined Multipath Routing - Additional/Modified Parameters and Variables

| Parameter | Description |
|---|---|
| $P^k$ | predefined set of paths from source $s(k)$ to destination $d(k)$ for routing demand $k$ |

| Variable | Description |
|---|---|
| $z^k(P)$ | specifies the amount of traffic on path $P$ for routing demand $k$ |
| $y_{is}^k(P)$ | specifies the amount from $z^k(P)$ that is provided service type $s$ at node $i$ |

- where to place the security services and
- how much traffic of a demand $k$ to route via which path.

Like in the previous section variable $x_i \in \{0, 1\}$ decides whether a router $i$ is programmable or not. To decide on the routing a third variable $z^k(P)$ is introduced which denotes the amount of traffic on path $P$ for routing demand $k$. Also the definition of variable $y_{is}^k$ varies to the one given in the previous section: service distribution variable $y_{is}^k(P)$ specifies the amount (and not fraction) from $z^k(P)$ that is provided service type $s$ at node $i$. The variable $y_{is}^k(P)$ is defined only if node $i$ appears on path $P$. $P$ denotes either the set of nodes or the set of links on path $P$, the specific use will be clear from the context. The multipath routing version of the problem can be expressed as the polynomial size mixed integer linear program shown in MILP 3.

Constraint (6.10) guarantees that all traffic $t(k)$ of demand $k$ is routed via a predefined path $P \in P^k$ from $s(k)$ to $d(k)$. Equation (6.11) assures that a programmable router $i$ running service $s$ for traffic demand $k$ maximally processes $t(k)$ traffic units of that demand. The requirement that all traffic of a demand $k$ must be analyzed related security services assigned $S_k$ is satisfied by Constraint (6.12). Equation (6.13) ensures that no node $i$—this constraint also counts for non-programmable standard routers—must route more traffic than

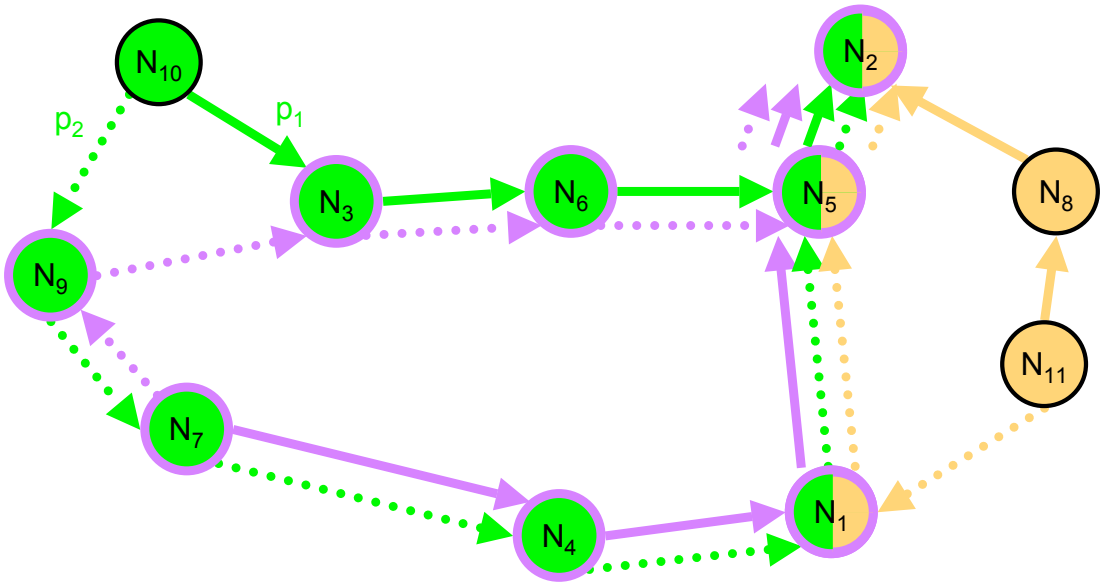



Figure 6.4: Predefined multipath routing - security services placement possibilities

it is capable to (Routing delay = $T_{\text{base}}$). Next, Equation (6.13) represents the stability constraint for programmable routers saying that a programmable router must process incoming packets fast enough such that its buffer does not overflow. The stability constraint (6.14) is a modified version of Constraint (6.5) that was introduced in the previous section. Parameter $u_e$ denotes the capacity of link $e$ and Constraint (6.15) ensures that these are taken into account when splitting the traffic demands $k \in D$ between the set of predefined paths. Equation (6.16) states that a node is either a standard router or a programmable one. Finally, Constraint (6.17) specifies that variables $z^k(P)$ and $y^k_{is}(P)$ must be greater than or equal zero.

In a manner analogous to that in the previous section, the maximum ratio of average packet processing time to the average packet inter-arrival time at each node can be minimized. The value of this ratio at a node $i$ is the left-hand-size of Constraint (6.13) if node $i$ is not a programmable router, or Constraint (6.14) if node $i$ is a programmable router. Again the maximum value of this ratio over all nodes $i \in N$ is denoted $v$. The corresponding linear program is presented in MILP 4.

## 6.5 Joint Traffic Routing and Distribution of Security Services

The scenario described in this section provides a further degree of freedom: single-path routing without consideration of predefined paths and a simultaneous optimal distribution of security services. The optimization model routes each traffic demand $k$ on a single path through the network and deploys the requested security services $S_k \in S$ on nodes in accordance to the chosen objective function. Theoretically, a security service assigned to a traffic demand $k$ can be located on any node of the network (see Figure 6.5,). The optimization

$$\text{minimize} \quad \sum_{i \in N} x_i$$

subject to:

$$\sum_{P \in P^k} z^k(P) = t(k) \qquad \forall k \in \mathbf{D} \tag{6.10}$$

$$y^k_{is}(P) \leq t(k) x_i \qquad \forall i \in \mathbf{N}, P \in \mathbf{P^k}, k \in \mathbf{D}, s \in \mathbf{S_k} \tag{6.11}$$

$$\sum_{i \in P} y^k_{is}(P) = z^k(P) \qquad \forall P \in \mathbf{P^k}, k \in \mathbf{D}, s \in \mathbf{S_k} \tag{6.12}$$

$$\sum_{k \in D} \sum_{P \ni i} z^k(P) T_{base} \leq 1 \qquad \forall i \in \mathbf{N} \tag{6.13}$$

$$\sum_{k \in D} \sum_{P \ni i} z^k(P)(T_{base} + T_{active}) + \sum_{k \in D} \sum_{P \ni i} \sum_{s \in S_k} T_s y^k_{is}(P) \leq 1 + C(1 - x_i) \qquad \forall i \in \mathbf{N} \tag{6.14}$$

$$\sum_{k \in D} \sum_{P \ni e} z^k(P) \leq u_e \qquad \forall e \in \mathbf{E} \tag{6.15}$$

$$x_i \in \{0, 1\} \qquad \forall i \in \mathbf{N} \tag{6.16}$$

$$z^k(P), y^k_{is}(P) \geq 0 \qquad \forall i \in \mathbf{N}, P \in \mathbf{P^k}, k \in \mathbf{D}, s \in \mathbf{S_k} \tag{6.17}$$

MILP 3: Predefined Multipath Routing - Minimizing the Amount of Programmable Routers

model decides:

- whether a node $i$ is programmable or not,
- where to place the security services and
- how to route the traffic demands $k \in D$ on a single-path through the network, involving the decisions:
  - what set of links $e \in E$ to use and
  - what set of nodes $i \in N$ to use.

For this purpose, the optimization model introduced in the previous sections must be modified, additional and modified variables/parameters are listed in Table 6.4. Parameter $E^+(i)$ denotes the set of outbound links of node $i$ and analogously, parameter $E^-(i)$ denotes the set of inbound links of node $i$. Like in Section 6.3, variable $x_i$ is of value 1 if node $i$ is a programmable router node and otherwise it is of value 0. Also, security service distribution variable $y^k_{is}$ is defined as in Section 6.3, specifying the fraction of demand $k$ that is provided service $s$ at node $i$. Traffic demands can be split into smaller flows allowing for load balancing. Apart form this, two additional decision variables—specifying the routing (links and

$$\text{minimize} \quad v$$

subject to: (6.10) – (6.12) and (6.15) – (6.17)

$$\sum_{k \in D} \sum_{P \ni i} z^k(P) T_{base} \leq v \qquad \forall i \in \mathbf{N} \qquad (6.18)$$

$$\sum_{k \in D} \sum_{P \ni i} z^k(P)(T_{base} + T_{active}) + \sum_{k \in D} \sum_{P \ni i} \sum_{s \in S_k} T_s y^k_{is}(P) \leq v + C(1 - x_i) \qquad \forall i \in \mathbf{N} \qquad (6.19)$$

MILP 4: Predefined Multipath Routing - Minimizing the Maximal Router Utilization

nodes)—are integrated into the problem formulation. The decision whether or not a link $e$ is used by a traffic demand $k$ is represented by variable $z^k_e \in \{0, 1\}$. The semantics of this variable is

$$z^k_e = \begin{cases} 1 & \text{if the routing of demand } k \text{ uses link } e, \\ 0 & \text{otherwise.} \end{cases}$$

Further, variable $w^k_i$ states whether a node $i$ is on the path of traffic demand $k$. Analogously, the semantics of this variable is

$$w^k_i = \begin{cases} 1 & \text{if node } i \text{ is on the routed path for traffic demand } k, \\ 0 & \text{otherwise.} \end{cases}$$

Based on the introduced parameters and variables, the optimization problem can be expressed as the polynomial size mixed integer linear program represented in MILP 5. Equations (6.20)–(6.24) contain the routing constraints of the demands $k \in D$. Flow conservation

Table 6.4: Joint Traffic Routing and Security Services Distribution - Additional/Modified Parameters and Variables

| Parameter | Description |
|---|---|
| $E^+(i)$ | set of outbound links of node $i$ |
| $E^-(i)$ | set of inbound links of node $i$ |

| Variable | Description |
|---|---|
| $x_i$ | specifies whether node $i$ is programmable |
| $y^k_{is}$ | specifies the fraction of demand $k$ that is provided service type $s$ at node $i$ |
| $z^k_e$ | specifies whether link $e$ is part of the routed path for demand $k$ |
| $w^k_i$ | specifies whether node $i$ is part of the routed path for demand $k$ |

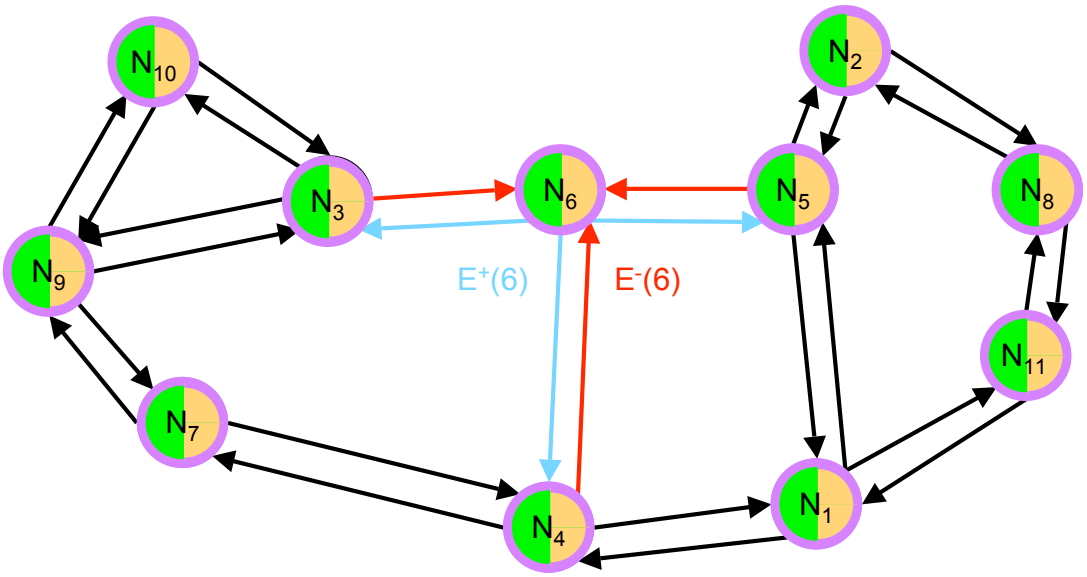



Figure 6.5: Joint traffic routing and distribution of security services

is ensured by Constraint (6.20). A traffic demand $k$ arises at its source $s(k)$ and hence, at a source node the sum of all inbound and outbound traffic flows must be $+1$:

$$\sum_{e \in E^+(i)} z_e^k - \sum_{e \in E^-(i)} z_e^k = +1 \quad \text{if } i = s(k), \forall i \in N, \forall k \in D.$$

Correspondingly, a traffic demand ends at its destination $d(k)$ and consequently, there the sum of inbound and outbound traffic flows must be $-1$:

$$\sum_{e \in E^+(i)} z_e^k - \sum_{e \in E^-(i)} z_e^k = -1 \quad \text{if } i = d(k), \forall i \in N, \forall k \in D.$$

Finally, intermediate nodes must conserve traffic demands, the sum of incoming traffic flows must equal the sum of outgoing traffic flows:

$$\sum_{e \in E^+(i)} z_e^k - \sum_{e \in E^-(i)} z_e^k = 0 \quad \text{if } i \neq d(k) \text{ or } s(k), \forall i \in N, \forall k \in D.$$

A further claim is that the resulting routes are loop-free and to guarantee this the following equations must be fulfilled. It must be enforced that the total in-degree and out-degree of links used for the routing of any traffic demand $k$ is at most 1 at any node.

$$\sum_{e \in E^-(i)} z_e^k \leq 1 \qquad \forall i \in \mathbf{N}, k \in \mathbf{D}$$

$$\sum_{e \in E^+(i)} z_e^k \leq 1 \qquad \forall i \in \mathbf{N}, k \in \mathbf{D}$$

$$\text{minimize} \quad \textstyle\sum_{i \in N} x_i$$

subject to:

$$\sum_{e \in E^+(i)} z_e^k - \sum_{e \in E^-(i)} z_e^k = \begin{cases} +1 & \text{if } i = s(k) \\ -1 & \text{if } i = d(k) \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \mathbf{N}, k \in \mathbf{D} \qquad (6.20)$$

$$\sum_{e \in E^-(i)} z_e^k \leq 1 \qquad \forall i \in \mathbf{N}, k \in \mathbf{D} \qquad (6.21)$$

$$\sum_{e \in E^+(i)} z_e^k \leq 1 \qquad \forall i \in \mathbf{N}, k \in \mathbf{D} \qquad (6.22)$$

$$p_j^k - p_i^k + (1 - eps) \geq z_e^k \qquad \forall e \in \{\mathbf{E}^+(\mathbf{i}) \cap \mathbf{E}^-(\mathbf{j})\}, k \in \mathbf{D} \qquad (6.23)$$

$$w_i^k = \begin{cases} 1 & \text{if } i = s(k) \text{ or } d(k) \\ \sum_{e \in E^-(i)} z_e^k & \text{otherwise} \end{cases} \qquad \forall i \in \mathbf{N}, k \in \mathbf{D} \qquad (6.24)$$

$$y_{is}^k \leq w_i^k \qquad \forall i \in \mathbf{N}, k \in \mathbf{D}, s \in \mathbf{S_k} \qquad (6.25)$$

$$y_{is}^k \leq x_i \qquad \forall i \in \mathbf{N}, k \in \mathbf{D}, s \in \mathbf{S_k} \qquad (6.26)$$

$$\sum_{i \in N} y_{is}^k = 1 \qquad \forall k \in \mathbf{D}, s \in \mathbf{S_k} \qquad (6.27)$$

$$\sum_{k \in D} t(k) w_i^k T_{base} \leq 1 \qquad \forall i \in \mathbf{N} \qquad (6.28)$$

$$\sum_{k \in D} t(k) w_i^k (T_{base} + T_{active}) + \sum_{k \in D} t(k) \sum_{s \in S} T_s y_{is}^k \leq 1 + C(1 - x_i) \qquad \forall i \in \mathbf{N} \qquad (6.29)$$

$$\sum_{k \in D} z_e^k t(k) \leq u_e \qquad \forall e \in \mathbf{E} \qquad (6.30)$$

$$z_e^k, w_i^k, x_i \in \{0, 1\} \qquad \forall e \in \mathbf{E}, \forall i \in \mathbf{N}, k \in \mathbf{D} \qquad (6.31)$$

$$y_{is}^k \in [0, 1] \qquad \forall i \in \mathbf{N}, k \in \mathbf{D}, s \in \mathbf{S_k} \qquad (6.32)$$

MILP 5: Joint Traffic Routing and Security Service Distribution - Minimizing the Amount of Programmable Routers

But the two constraints do not suffice to guarantee loop-free routes as a node $i$ can simultaneously be startpoint and endpoint of different links used to route a demand $k$. For example

a path via nodes $N_1, N_2, N_3, N_1$ would not violate the above conditions. Thus, another constraint is required to generate loop-free routes and for this reason the following lemma is postulated:

**Lemma 1.** *No feasible solution can have a flow on a (directed) cycle in the network.*

To respect Lemma 1 the continuos variable $p_i^k$ for each node $i$ and demand $k$ as well as the parameter $eps = \frac{1}{n}$ (n is the number of network nodes) are introduced. Now Lemma 1 can be formulated as:

$$p_j^k - p_i^k + (1 - eps) \geq z_e^k, \qquad \forall e \in \{\mathbf{E}^+(\mathbf{i}) \cap \mathbf{E}^-(\mathbf{j})\}, k \in \mathbf{D}$$

*Proof of Lemma 1.* (by contradiction)

- Suppose a feasible solution has a flow, for some demand $k$, along a cycle $N_1, \ldots, N_m, N_1$ of length $m \leq n$
- Write down the above constraint for each link on this cycle
- Adding all these $m$ constraints and using telescopic cancellation for the p(i,k) terms on the left, this results in: $m(1 - eps) \geq m \rightarrow$ **Contradiction!**
- Further it must be noted that up to $(m-1)$ links on this cycle can be used by the flow, since the inequality $m(1 - eps) \geq m - 1$ is valid ($eps = \frac{1}{m}$ and $m \leq n$). Consequently, no cycle-free feasible solution is excluded.

□

By adding the above three constraints it is guaranteed that the resulting routes are loop-free. Drawing back the attention to the formulation of the MILP, a node $i$ is on the path for traffic demand $k$ if it is either source- or destination-node:

$$w_i^k = 1, \quad \text{if } i = s(k) \text{ or } d(k), \forall i \in N, \forall k \in D.$$

Furthermore, an intermediate node $i$ is part of the routed path if it has an incident link that is used. This can be modelled as follows:

$$w_i^k = \sum_{e \in E^-(i)} z_e^k \quad \text{if } i \neq s(k) \text{ or } d(k), \forall i \in N, \forall k \in D.$$

Next, a security service $s$ assigned to traffic demand $k$ must be deployed on a programmable router $i$ that is part of the routed path:

$$y_{is}^k \leq w_i^k \qquad \forall i \in N, \forall k \in D, \forall s \in S_k$$
$$y_{is}^k \leq x_i \qquad \forall i \in N, \forall k \in D, \forall s \in S_k.$$

The former constraint models the fact that a routed demand $k$ can receive some security service at node $i$ only if node $i$ is on the path and the latter models the additional requirement

that such a node $i$ must be programmable. In addition, the optimization model must ensure that all traffic of a demand $k$ is analyzed by the corresponding security services $s \in S_k$. This is achieved by constraint:

$$\sum_{i \in N} y_{is}^k = 1 \quad \forall k \in D, s \in S_k.$$

The stability constraints are given in Equations (6.28) and (6.29). The total traffic entering node $i$ is $f_i = \sum_{k \in D} t(k) w_i^k$. Thus, the average time between consecutive packet arrivals is $1/f_i$. The stability condition for a non-programmable node $i$ is $T_{base} \leq 1/f_i$, or $f_i T_{base} \leq 1$ and consequently, the complete stability constraint is

$$\sum_{k \in D} t(k) w_i^k T_{base} \leq 1 \quad \forall i \in N$$

and analogously to Section 6.3, the stability constraint for programmable nodes is:

$$\sum_{k \in D} t(k) w_i^k (T_{base} + T_{active}) + \sum_{k \in D} t(k) \sum_{s \in S} T_s y_{is}^k \leq 1 + C(1 - x_i) \quad \forall i \in N.$$

This constraint is correct when node $i$ is a programmable router ($x_i = 1$), and is redundant otherwise ($x_i = 0$). The following equation guarantees that link capacities $u_e$ are considered by the optimization framework:

$$\sum_{k \in D} z_e^k t(k) \leq u_e \quad \forall e \in E.$$

Finally, variables $z_e^k$, $w_i^k$, $x_i$ are of type binary, implying a value of either 1 or 0 (6.31). Variable $y_{is}^k$ is like in Section 6.3 of type continuous within the specified value range $y_{is}^k \in [0, 1]$.

In accordance to the previous sections, the problem formulation for the second optimization function—minimizing the maximal router utilization—varies slightly (see MILP 6).

---

minimize $v$

subject to: (6.20)–(6.27), (6.30)–(6.32) and

$$\sum_{k \in D} t(k) w_i^k T_{base} \leq v \qquad \forall i \in \mathbf{N} \qquad (6.33)$$

$$\sum_{k \in D} t(k) w_i^k (T_{base} + T_{active}) + \sum_{k \in D} t(k) \sum_{s \in S} T_s y_{is}^k \leq v + C(1 - x_i) \qquad \forall i \in \mathbf{N} \qquad (6.34)$$

---

MILP 6: Joint Traffic Routing and Security Service Distribution - Minimizing the Maximal Router Utilization

The mixed integer linear programs outlined in this chapter can be solved using a standard (MI)LP solver like CPLEX [73].

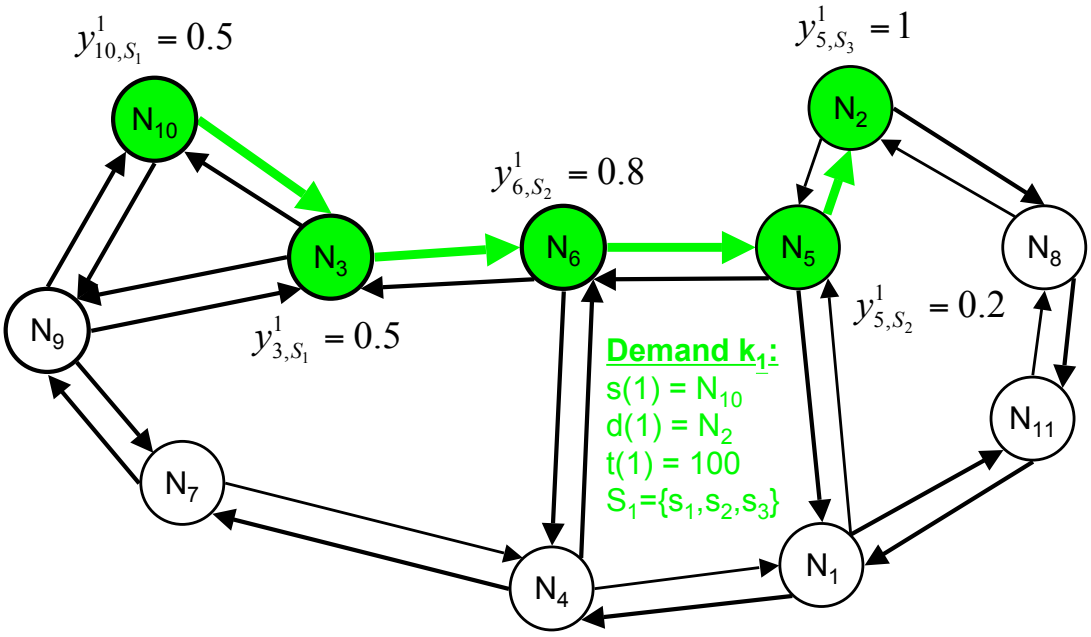



Figure 6.6: Predefined singlepath routing - sequence constraint placement of security services

## 6.6 Optimal Placement of Security Services under the Constraint of a Predefined Order

Occasionally, it can be reasonable to deploy the requested security services in accordance to a predefined sequence. For example, a TCP state keeping service which filters packets that do not belong to an existing TCP-connection should be placed in front of the security services that analyze TCP-packets for malicious content. Such a predefined order can also be used to respect the activity of current threats as security services that detect and filter fast spreading worms (or viruses, etc.) should be placed close to the edge of the corresponding network to reduce the overall amount of packets that must be processed by the security services located on inner network routers. It is emphasized that the described approach places two security services $s_1$ and $s_2$—service $s_1$ must precede service $s_2$—on different nodes. This can be modified such that the two security services $s_1$ and $s_2$ could be placed on one and the same node $i$. Then service $s_1$ must be inserted into the services processing (see Section 4.4) chains ahead of service $s_2$; in this case the placement fulfils the condition of placing service $s_1$ in front of service $s_2$.

A security service sequence is expressed with the help of a corresponding $S \times S$ matrix (see Table 6.5). In the given example, security service $S_1$ must be placed ahead of security service $S_2$ and in turn, security service $S_2$ must precede security service $S_3$. To deploy the security services under the constraint of a predefined order the optimization models—introduced in Sections 6.3 to 6.5—must be extended. The extension of the problem formulation requires the introduction of variable $a^k_{is}$ and the parameters listed in Table 6.5. Variable $a^k_{is}$ determines whether a node $i$ analyzes any packets belonging to demand $k$ with service $s$. The semantics

Table 6.5: An Exemplary Security Services Sequence ($S_1 \rightarrow S_2 \rightarrow S_3$)

| **Security Service** | $S_1$ | $S_2$ | $S_3$ |
|---|---|---|---|
| $S_1$ | 0 | 1 | 1 |
| $S_2$ | 0 | 0 | 1 |
| $S_3$ | 0 | 0 | 0 |

Table 6.6: Variables and Parameters for Order Constraint Deployment of Security Services Predefined Routing (Single-path and multipath)

| **Parameter** | **Description** |
|---|---|
| $a_{is}^k$ | specifies whether node $i$ runs service $s$ for (fraction of) demand $k$ |
| **Parameter** | **Description** |
| $o_{s_1 s_2}$ | defines whether security service $s_1$ must be placed in front of security service $s_2$ |
| $b_{i_1 i_2}^k$ | specifies whether node $i_1$ is in front of node $i_2$ on path $P$ that is predefined for demand $k$ |

of this variable is:

$$a_{is}^k = \begin{cases} 1 & \text{if } y_{is}^k > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Variable $a_{is}^k$ equals variable $y_{is}^k$ ($a_{is}^k = y_{is}^k$) in case that no fractional assignment of traffic demands is allowed $\left(y_{is}^k \in \{0,1\}\right)$. Parameter $o_{s_1 s_2}$, specified by the corresponding security services sequence table, defines whether security service $s_1$ must be deployed ahead of security service $s_2$. The semantics of the parameter is:

$$o_{s_1 s_2} = \begin{cases} 1 & \text{if security service } s_1 \text{ must precede security service } s_2, \\ 0 & \text{otherwise.} \end{cases}$$

Further, parameter $b_{i_1 i_2}^k(P)$ specifies whether node $i_1$ lies ahead of $i_2$ on path $P$ for demand $k$. Analogously, the semantics is:

$$b_{i_1 i_2}^k = \begin{cases} 1 & \text{if node } i_1 \text{ lies in front of node } i_2 \text{ on the predefined path } P \text{ for demand } k, \\ 0 & \text{otherwise.} \end{cases}$$

The following constraint is added to the optimization model to ensure the correct placement of two security services $s_1$ and $s_2$ for demand $k$ whereas service $s_1$ must be placed ahead of service $s_2$ (see Figure 6.6):

$$b_{i_1,i_2} \cdot \left(a_{i_1,s_2}^k + a_{i_2,s_1}^k\right) \leq 1 \qquad \forall i_1, i_2 \in \mathbf{P}$$

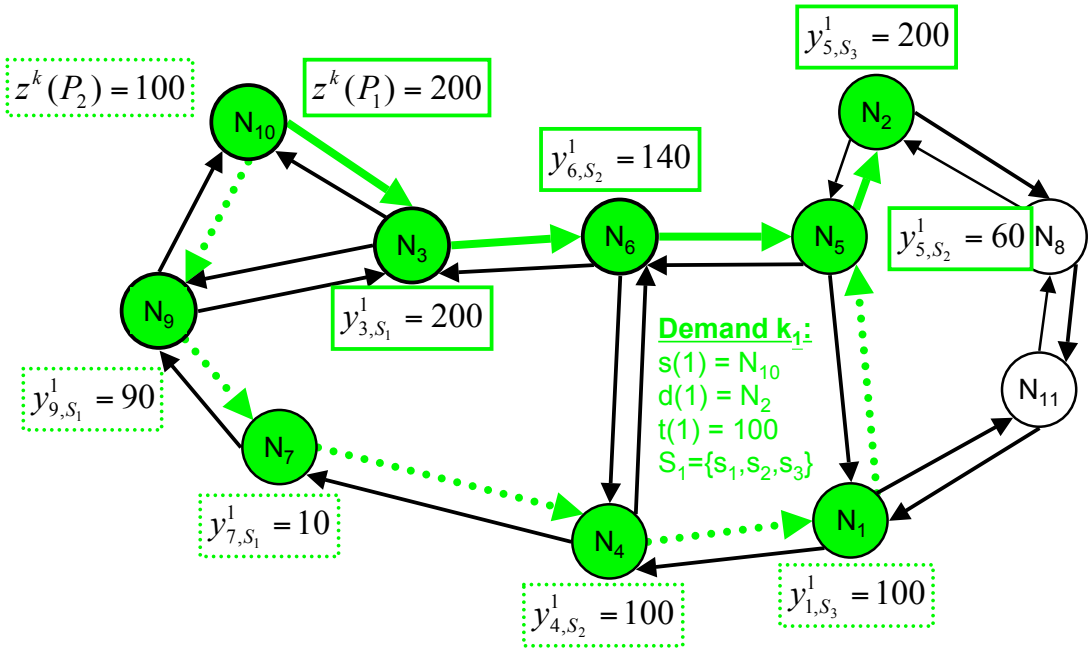



Figure 6.7: Predefined multipath routing - sequence constraint placement of security services

This equation ensures—even in case of a fractional flow assignment $\left(y^k_{is} \in [0,1]\right)$—that all packets of a demand $k$ are completely processed by security service $s_1$ before being processed by security service $s_2$. In detail, it guarantees that there is no pair of nodes $(i_1, i_2)$ on path $P$ with node $i_1$ running service $s_2$ and lying in front of node $i_2$ which runs service $s_1$. Consequently, the general constraint which is also valid for the predefined multipath scenario depicted in Figure 6.7 is:

$$b_{i_1,i_2} \cdot \left(a^k_{i_1,s_2} + a^k_{i_2,s_1}\right) \leq 2 - o_{s_1,s_2} \qquad \forall i_1, i_2 \in \mathbf{P^k}, \forall k \in \mathbf{D}, \forall s_1, s_2 \in \mathbf{S^k} \quad \text{and} \quad s_1 \neq s_2 \tag{6.35}$$

## 6.7 Varying Computational Speeds of Routers

It is a reasonable assumption that the routers in a network will not be of one type and consequently, varying amounts of security services can be deployed on the different types of routers. For example, Section 4.5 discussed the performance of routers of type *PC733* an *PC3000* and when respecting the stability constraint a *PC3000* is capable to run more security services compared to a *PC733* system. To respect the computational speeds of varying router types the MILPs presented so far must be extended. In detail, the parameters $T_{\text{base}}$, $T_{\text{active}}$, $T_s$ and C are no longer given for a complete network they are rather extended towards $T_{base,n}$, $T_{active,n}$, $T_{s,n}$ and $C_n$ to be valid for router $n$. Consequently, the Equation 6.5

is then reformulated to

$$\sum_{k \in D} t(k) w_i^k (T_{base,n} + T_{active,n}) + \sum_{k \in D} t(k) \sum_{s \in S} T_{s,n} y_{is}^k \leq 1 + C_n(1 - x_i) \qquad \forall i \in \mathbf{N}$$

with

$$C_n = \sum_{k \in D} t(k)(T_{base,n} + T_{active,n} + \sum_{s \in S} T_{s,n}).$$

## 6.8 A Remark on Fractional Service Assignments

The decision variable $y_{is}^k(P)$ in Section 6.4 is of type real and its value ranges from zero to $t(k)$ of the corresponding traffic demand. By definition, the predefined multipath routing scenario—represented by MILP 5 and MILP 4—allows to split a single security service assignment $y_{is}^k(P)$ over multiple routers of the relevant path $P$. In other words, a router A could be assigned to inspect 80% of a flow and a router B which lies on the same path is responsible to analyze the remaining 20% of the flow. In practice this is hard to realize. Either a marking mechanism must be implemented or a stochastic approach must be taken.

The approach followed here is the atomic placement of security services. A possibility to achieve atomic security service assignments would be to extend MILP 5 and MILP 4 by constraint:

$$y_{is}^k(P) == z^k(P) \quad \forall y_{is}^k(P) > 0, \qquad \forall i \in \mathbf{N}, P \in \mathbf{P^k}, k \in \mathbf{D}, s \in \mathbf{S_k}. \tag{6.36}$$

But the introduction of the above constraint tremendously increases the solution times. Thus, optimal solutions are calculated using MILP 5 and MILP 4 respectively and furthermore, these solutions are subsequently processed by the following algorithm.

1. Select all fractional security service assignments and sort them according to the load caused by them ($y_{is}^k(P) \cdot t_s$)
2. Take first fractional security service assignment
3. Search for matching and completing candidates (identical demand $k$, service $s$, path $P$)
4. Deploy complete security service on the node that will have to suffer least
5. Add additional load to the corresponding node.
6. Remove candidates from list and return to step 2.

An example is given below that shows how the algorithm merges nine fractional security service assignments into four atomic ones. The first column shows the variable names, the second column their values. The third column specifies the amount of traffic of demand $k$ that is routed via path $P$. Finally, the fourth column presents the load that is caused by the security service assignment.

The amount of 352 units network traffic is routed via path 1 from node 11 to node 8 ($z[11, 8, 1] = 352$). Furthermore, 237 units of this flow are analyzed by security service 1

on router 11—$y[1, 11, 11, 8, 1] = 237$—and the remaining 115 units are treated by security service 1 on router 8—$y[1, 8, 11, 8, 1] = 115$. The processing of 237 units of network traffic on router 1 causes a load of about $y^k_{is}(P) * t_s = 0.13$ and analogously, the processing of 115 units of network traffic by the same security service on node 8 generates a load of about 0.06.

In a next step, the best candidate to atomically deploy the security service is chosen. Therefore, the loads that have been added so far to the candidates as well as the extra load—caused by adding the other candidates' fractional assignments—are considered and compared. As a result of this Node 11 is chosen to process all 352 units of network packets with security service 1 as the sum $AdditionalLoad[11] + 0.06$ is smaller than $AdditionalLoad[8] + 0.13$ (at startup $AdditionLoad[i] = 0 \forall i \in N$). Subsequently, the candidates are removed from the list of fractional service assignments and the algorithm proceeds with the assignment $y[2, 7, 7, 2, 2]$. The resulting atomic security service assignments are listed an the end of the output.

```
                 y[s,r,o,d,p]    z[o,d,p]         y*t[s]
y#1#11#11#8#1    237.000000000   352.000000000   0.130350000
y#2#7#7#2#2      337.000000000   338.000000000   0.124690000
y#4#11#8#11#1    320.000000000   322.000000000   0.118400000
y#1#2#11#8#2     169.000000000   174.000000000   0.092950000
y#1#8#11#8#1     115.000000000   352.000000000   0.063250000
y#1#5#11#8#2     3.000000000     174.000000000   0.001650000
y#1#1#11#8#2     2.000000000     174.000000000   0.001100000
y#4#8#8#11#1     2.000000000     322.000000000   0.000740000
y#2#5#7#2#2      1.000000000     338.000000000   0.000370000

Candidate y#1#11#11#8#1 0.673295455 352.000000000
Candidate y#1#8#11#8#1 0.326704545 352.000000000
LoadAdded[11] = 0.063250000

Candidate y#2#7#7#2#2 0.997041420 338.000000000
Candidate y#2#5#7#2#2 0.002958580 338.000000000
LoadAdded[7] = 0.000370000

Candidate y#4#11#8#11#1 0.993788820 322.000000000
Candidate y#4#8#8#11#1 0.006211180 322.000000000
LoadAdded[11] = 0.063990000

Candidate y#1#2#11#8#2 0.971264368 174.000000000
Candidate y#1#5#11#8#2 0.017241379 174.000000000
Candidate y#1#1#11#8#2 0.011494253 174.000000000
LoadAdded[2] = 0.002750000

Resulting Assignments:

y#1#11#11#8#1 1.000000000
```

```
y#2#7#7#2#2 1.000000000
y#4#11#8#11#1 1.000000000
y#1#2#11#8#2 1.000000000
```

## 6.9 Summary

This section introduced three optimal security service deployment strategies which can be combined with two objective functions. The most limited deployment strategy *Pre* assumes predefined routes and the two other strategies combine the task of routing with the deployment of security services. Thereby strategy *SP* specifies a single route for each traffic demand and strategy *MP* might assign multiple routes to a commodity. In the latter, the commodity is divided by the strategy into multiple smaller flows that are routed via different paths.

The first objective function implemented aims at minimizing the number of programmable routers in a network as their architecture is more complex than the one of traditional routers. Hence, it could be the intention of a provider to minimize the number of programmable routers used in his network. The second objective function minimizes the maximum utilization of any router node in the network. In other words, the strategy fairly distributes the workload caused by doing intrusion prevention among all router nodes. Further, the strategies implemented allow to consider different router types in terms of computational speeds additionally, security services can also be optimally deployed under the constraint of a predefined order. For example, a security service A must be placed in front of security service B. Section 7.6 provides information of the time required to calculate optimal security service deployments.

# Chapter 7

# Fidran Performance Evaluation

This chapter presents the performance evaluation of the Fidran-framework and the optimal deployment strategies that were introduced in Chapter 6. To achieve realistic results the operation of Fidran was emulated in the in the *Cyber Defense Technology Experimental Research* testbed (Deter) for two network topologies: a tree- and a high-speed network. The experiments show that the intelligent deployment of security services reduces the impact of doing intrusion prevention on the network performance. The Deter-testbed and the conducted emulations of the networks are described in detail in Section 7.1.

Section 7.2 describes the generation of network traffic, respecting measurements of local-area [88], [136] and wide-area [107] network traffic which have shown that packet-switched data traffic is self-similar. Section 7.3 evaluates the results of the first scenario which is a limited networking environment, and Section 7.4 discusses the emulation of the Abilene network. Finally, Section 7.7 discusses the lessons learnt in this section.

## 7.1 Emulation: The DETER Testbed

The operation of Fidran in a network was emulated with the help of the *Cyber Defense Technology Experimental Research* testbed (DETER) [22, 26, 113] which is a shared infrastructure designed for medium-scale repeatable experiments in computer security. The testbed provides a pool of over 300 computers of varying hardware which can be used to emulate networks. Each experiment requires a configuration file defining network topology and the sequence of events that should happen in course of the experiment. The specification of the topology includes:

- the assignment of the hardware to a node (e.g. $Node_A$ is a Pentium III system at 733 MHz),
- the determination of the OS image to be activated on a node (e.g. $Node_A$ runs a Linux Fedora image),
- the definition of how the nodes are connected with each other including specification of bandwidth and delay for each link.

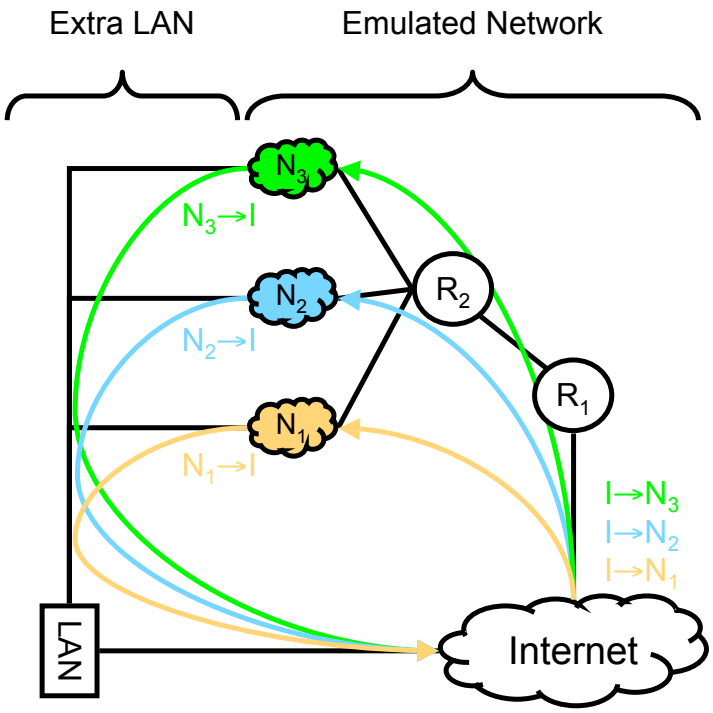



Figure 7.1: Testbed setup: Sending packets back to the sender for the purpose of using one clock to measure end-to-end delays

The DETER testbed provides links of up to 1 Gbps and to model link delays so-called shape nodes are used. The minimum possible non-zero delay is 2 ms due to the scheduling granularity of the shaping nodes. Topologies are specified in the language provided by the *ns-2 Network Simulator* [110] and so-called *program-agents* are used to schedule actions to happen on defined nodes at the specified points in time. For example, a program-agent can be used to start a traffic generator on a chosen node at a specified time. To avoid that control information is sent via a link that is part of experiment, each node of a network is by default connected to the control network .

Throughout the experiments all nodes were running a customized Fedora Core 4 operating system with a 2.6.12 kernel. Routes between nodes were setup with the native IP routing functions provided by the OS. The purpose of the experiments is to evaluate the impact of the different deployment strategies on the network performance in terms of the end-to-end delay and packet loss rate. To capture incoming and outgoing traffic on a node the tool *tcpdump* was used. By means of those files the packet loss rate can be calculated. To accurately measure end-to-end delays of packets sent from a node A to a node B well synchronized clocks are required. The DETER testbed uses the *Network Time Protocol* (NTP) to synchronize the nodes' clocks and according to the DETER manual NTP keeps all clocks within a few ms of each other. But this is not sufficient to measure end-to-end delays in the order of several milliseconds. To evade the difficulties associated with using clocks of different nodes (the sender's and the receiver's clock) all packets send from a node A to a node B are sent back

via an extra path from node B to node A.

Figure 7.1 depicts an exemplary network consisting of subnetworks ($N_1$, $N_2$ and $N_3$), the Internet and the routers ($R_1$ and $R_2$). Traffic send from the Internet to the subnets is routed via routers $R_1$ and $R_2$. Each packet that is received by a subnet is send back to the Internet via the depicted LAN. To do so the subnet must modify the source (source address translation—SNAT) and the destination address (destination address translation—DNAT) of the packet. The LAN is dimensioned such that the probability of packet loss on this part of the transmission path is very small. Of course, the sending back of a packet via the LAN adds to the overall end-to-end delay which is the difference between the sending time and the reception time measured at the Internet. But the comparison of the deployment strategies is still valid as the overhead is added to all packets.

Furthermore, the emulations run in an IEEE 802.3 [18] environment. IEEE 802.3 contains a flow control mechanism in order to optimize throughput and accordingly, the placement of the security services influences the sending rate of the subnets. It can happen that routers receive packets faster than they transmit them and then the packet queues fill up. In case that the packet queue of a router is filled to a certain threshold value, it sends a so-called $X_{off}$ message to the link partner in order to stop it from sending for a specific length of time. Further on, the MAC control sub-layer continues to transmit *PAUSE* frames with the programmed idle time as long as the threshold has been exceeded. If the queue falls under the threshold prior to the expiration of this time, another *PAUSE* frame is sent with a zero time specific to re-enable transmission, referred to as $X_{on}$. Flow control was turned off throughout the emulations.

## 7.2 The Generation of Self-Similar Network Traffic

Measurements of local-area [88], [136] and wide-area [107] network traffic have shown that packet-switched data traffic is self-similar. A self-similar object is exactly or approximately similar to a part of itself, a popular example are fractals as they can be divided into parts and each part is a reduced copy of the whole. In the context of network traffic self-similarity means that packet bursts, also known as packet trains, appear on a wide range of time scales. References [104, 127] explain how self-similar network traffic can be modelled by the superposition of a large number of 0/1 renewal processes whose *ON* and *OFF* periods are heavy tailed distributed. The main characteristic of a random variable obeying a heavy-tailed distribution is that it exhibits extreme variability and thus very large values have a non negligible probability. The Pareto distribution is a heavy tailed distribution and it has the *probability density function* (PDF):

$$P(x) = \frac{\alpha \cdot b^{\alpha}}{x^{\alpha+1}}, \quad x \geq b \tag{7.1}$$

A Pareto distribution is specified by two parameters: shape parameter $\alpha$ determines the *heaviness* of the tail and parameter b defines the minimum value of the random variates. Figure 7.2 shows exemplary density functions for Pareto distributions with a minimum value $b = 1$ and shape parameters $\alpha = \{1.1, 1.5, 1.9\}$. The principle of generating self-similar network traffic by multiplexing independent traffic sources whose *ON* and *OFF* periods are heavy tailed distributed is depicted in Figure 7.3. The figure shows the sending activity over

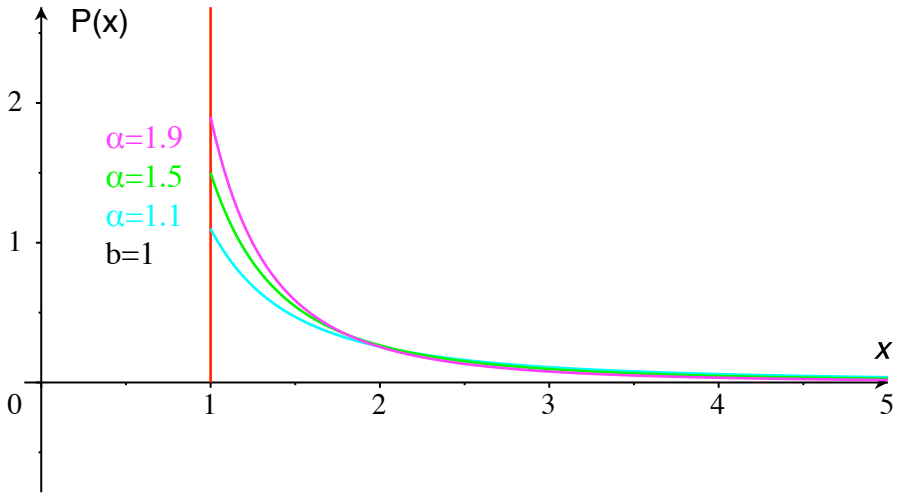



Figure 7.2: Pareto distributions with shape parameter $\alpha = \{1.1, 1.5, 1.9\}$ and minimum value $b = 1$

Table 7.1: IP packet size distribution measured by *Caida*

| IP packet size [Byte] | Packets [%] | Bytes [%] |
|---|---|---|
| 40 | 33.5 | 3.0 |
| 552 | 4.7 | 5.7 |
| 576 | 5.3 | 6.7 |
| 1500 | 16.5 | 54.6 |
| other | 40 | 30 |

the time for three traffic sources ($X_1, X_2$ and $X_3$) and the overall traffic flow ($X_1 + X_2 + X_3$) which is the result of the aggregation of the individual flows. The last graph in this figure illustrates the multiplexed traffic of the three sources. It can be seen how packet bursts occur. Typically. the Pareto distributions used to generate the *ON* and *OFF* periods of the sources are not identical. Reference [88] recommends a shape parameter $\alpha_{on} = 1.7$ and a shape parameter $\alpha_{off} = 1.2$ to generate self-similar network traffic for a LAN environment. Further, shape parameters found in WAN traces [88] are typically around $\alpha_{off} = 1.0$ and $\alpha_{on} = 2.0$.

Inspired by the ideas presented in reference [127] Glen Kramer implemented a self-similar network traffic generator [83]. A slightly modified version that was implemented to address the IP packet size distribution in the Internet was used throughout the conducted experiments. The red curve in Figure 7.4(a) shows the cumulative distribution function (cdf) for the packet sizes of 5.7 million IP packets [34] that were collected in an Internet backbone by the *Cooperative Association for Internet Data Analysis* (CAIDA), and the green curve shows

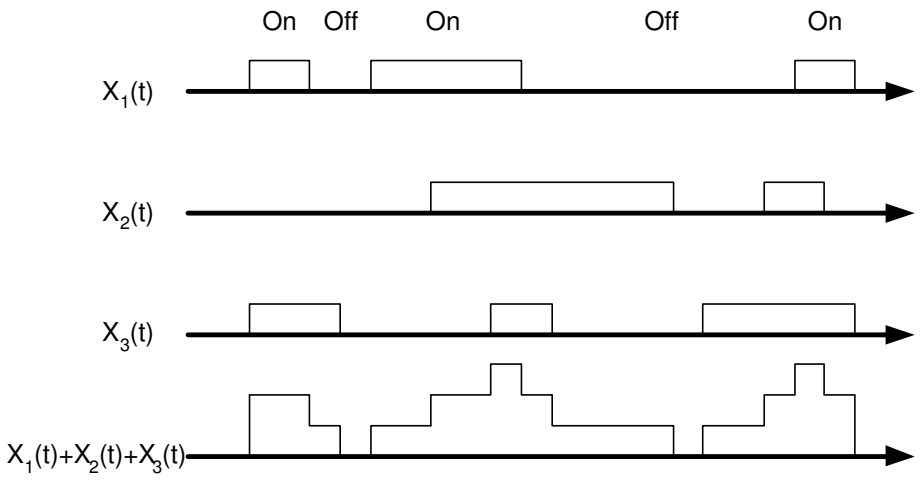
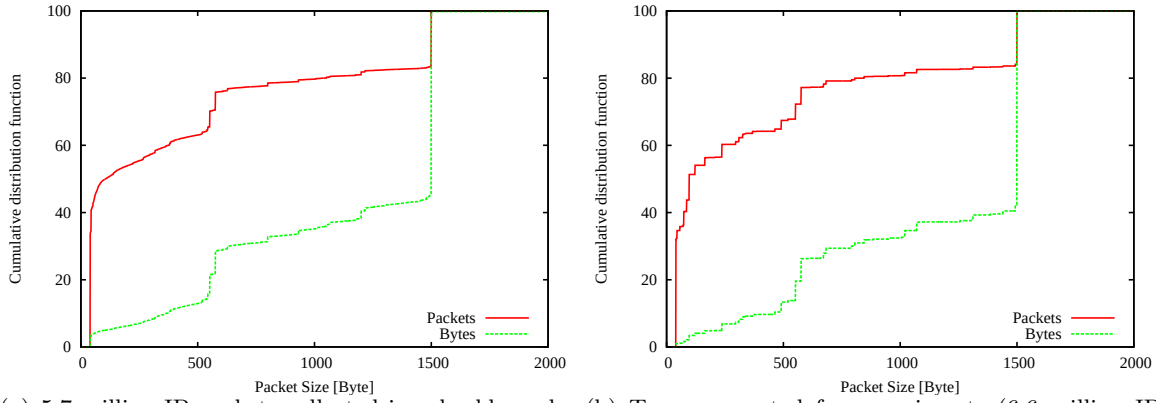



Figure 7.3: Superposition of *On*- and *Off*-renewal processes

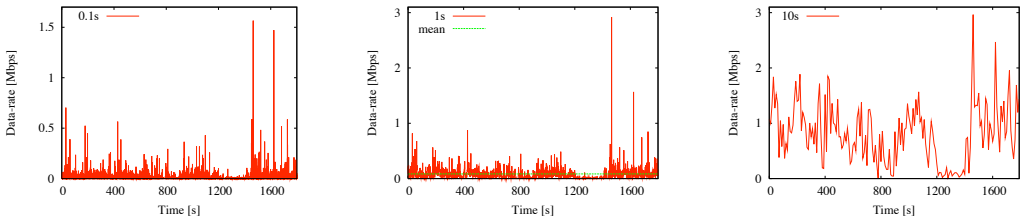
the corresponding cdf for the carried byte volume. A finding of the measurement done by Caida is that almost half of the packets are less than 44 Byte (red curve) but over half of the byte volume is carried by packets of size 1500 Byte (green curve). Table 7.1 lists the results of the mentioned traffic analysis in more detail. To model the IP packet size distribution in the Internet a flow from a source A to a destination B is the aggregated flow of five *ON* and *OFF* sources of different packet sizes. In detail, each *ON* and *OFF* source generates a flow of a fixed packet size (40, 552, 576, 1500 Byte and a randomly chosen packet size between 40 and 1500 Byte) and the according load is calculated via the desired load of the aggregated flow and the fraction of bytes carried by the respective IP packet size. For example, to generate a flow between source A and destination B of load 1 Mbps, the *ON* and *OFF* source of IP




(a) 5.7 million IP packets collected in a backbone by *Caida*

(b) Trace generated for experiments (6.6 million IP packets)

Figure 7.4: Packet size distributions




(a) Time-scale 0.1s (b) Time-scale 1s (c) Time-scale 10s

Figure 7.5: An exemplary flow at different time scales

packet size 1500 Byte produces a flow of 0.546 Mbps (= 1 Mbps · 0.546). Figure 7.4(b) depicts the cdf for the IP packet size (red curve) as well as the cdf for the byte volume carried (green curve) for a traffic trace that was produced with the presented network traffic generator for an experiment. It can be seen that the curves resemble those of Figure 7.4(a).

Figure 7.5 shows an exemplary flow that was generated with the network traffic generator (five *ON OFF* sources, $\alpha_{on} = 1.7$, $\alpha_{off} = 1.2$) introduced above. The figures depict the sending-rate for a time period of 1800 s for the time-scales 0.1, 1 and 10 s. The burstiness of the generated traffic can clearly be seen for the three time scales. Besides the sending rate over the time, Figure 7.5(b) additionally depicts the overall mean sending rate (66 Mbps) of the flow. Finally, to avoid effects of congestion and flow control mechanisms all experiments were restricted to Udp-traffic.

## 7.3 A Limited Networking Environment

A limited networking environment is a network that consists of a manageable number of identifiable systems implying that network topology as well as the security services requested by the connected end-systems can be analyzed in the automated way described in Chapter 5. The tree network depicted in Figures 7.6 was setup to get a first impression of the benefit of optimal deployment strategies. The goal of the experiment described was to assess the benefit of intelligent deployment strategies for a typical restricted networking environment for three different load scenarios and three different types of subnetworks. Intuitively, it can be assumed that the benefit of deployment strategies that incorporate the traffic rates when calculating the optimal distribution of security services increases with growing traffic-rates. For this purpose, a *low*, a *medium* and a *high* traffic scenario was modeled. Further, flows destined to subnetworks that do not require the protection of a security service will suffer to a certain degree under the operation of the intrusion prevention overlay network. The effect of heterogeneous security requirements was analyzed by specifying three types of subnetworks. The first type does not require protection by a security service, the second requests the installation of five security services and finally, the third type needs to be protected by ten security services.

As tree networks provide a single path from each sender to each receiver the optimal

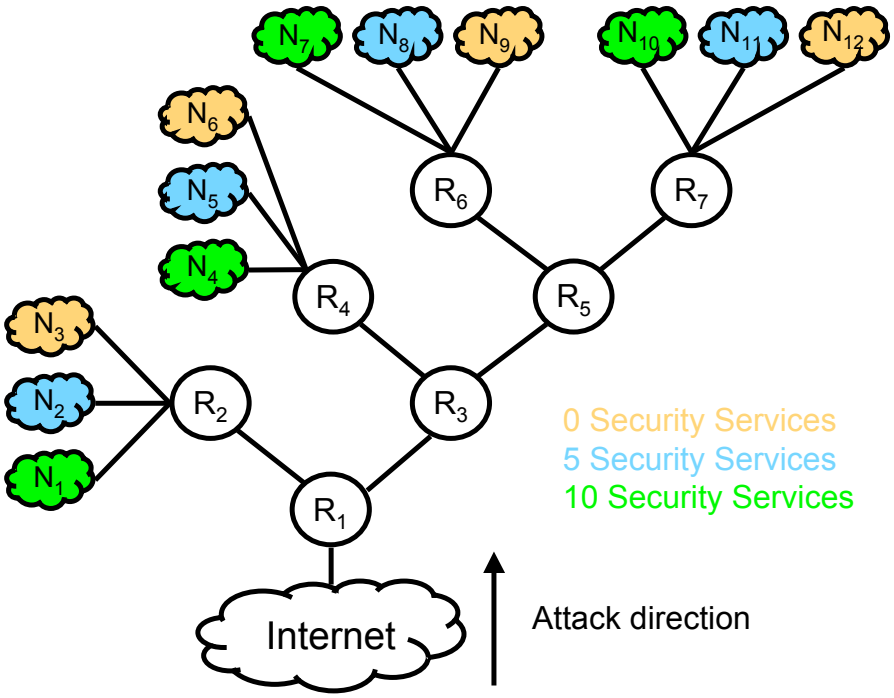
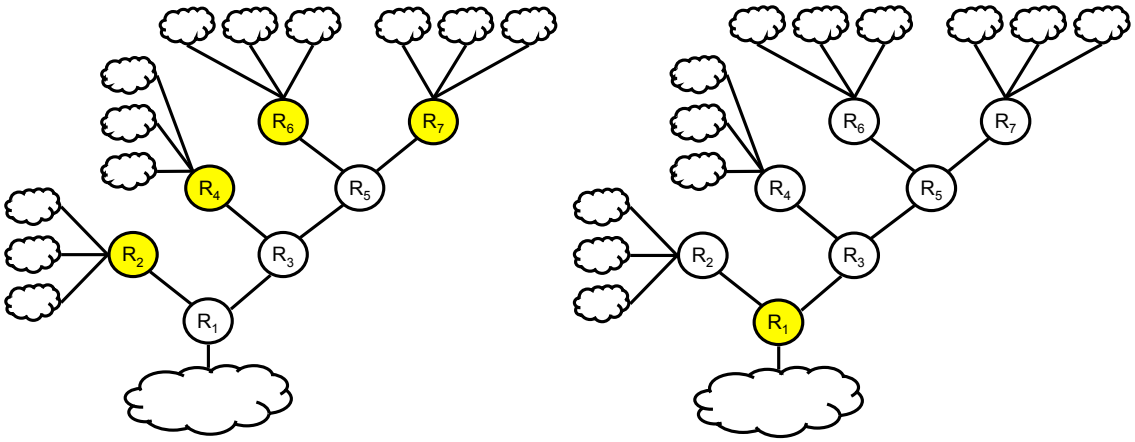



Figure 7.6: An exemplary tree network

deployment strategy for the predefined singlepath routing scenario of Section 6.3 was compared with the *Late-* and *early*-deployment strategy which are depicted in Figure 7.7. The *late*-deployment strategy places the requested security services as close as possible to the requesting subnet/end-system. As a consequence, the same security services are loaded and executed at the same time at several routers of the network, which, in that case, operate on smaller traffic flows than security services that are deployed further upstream. In contrast, the *early*-deployment strategy uses the first available router on the path from the Internet to the requesting subnet/end-system. This means that fewer routers must run the intrusion prevention framework yet with a higher load. The predefined singlepath deployment strategy distributes the security services under the objective of:

- fulfilling all security requests while minimizing the number of active routers in a network and keeping the router queues bounded (see MILP 1 in Section 6.3);
- fulfilling all security requests while minimizing the maximal workload of an active router (see MILP 2 in Section 6.3);

The network consists of twelve subnets ($N_1 \dots N_{12}$) and seven routers ($R_1 \dots R_7$). All links of the network have a bandwidth of 100 Mbps and do not cause a propagation delay. To reduce complexity, it is initially assumed that all attacks originate from the Internet as indicated by the arrow in Figure 7.6. Furthermore, a "hotspot"-scenario was assumed throughout the experiments resulting in disproportionate flows. All traffic was generated at the Internet as defined in Table 7.2: 80% of all Internet-traffic was send in equal portions to subnets




(a) S1: Late (b) S2: Early

Figure 7.7: The *early* and the *Late* deployment strategy—yellow routers represent the deployment of the security services

$N_{10}$, $N_{12}$ and $N_{12}$ and the remaining traffic was uniformly distributed among networks $N_1 - N_9$. The flows were generated with the traffic generator introduced in Section 7.2. The shape parameters $\alpha_{on}$ and $\alpha_{off}$ were set to 1.95 and 1.05. The minimum packet train size was set to 1, consequently at least one packet is sent at each *ON* period. To consider the hardware resources provided by the DETER testbed, the network was emulated on a scale of 1 : 15 which means that traffic rates were divided by 15 and services times as well as the programmable router delay $T_{\text{active}}$ were multiplied by 15. For example, regarding the *high* load scenario the overall mean traffic rate that is generated in the Internet is 18.145 Mbps. The generation of 272.175 Mbps ($= 18.145 \cdot 15$) is out of scope for the hardware provided by the DETER testbed. Figure 7.8 depicts the sending rate over the time for the flow from the Internet to network $N_{12}$. The red line shows the trace produced by the traffic generator, the green curve represents the measured sending-rate at the Internet and the blue line depicts the difference between the produced traffic trace and the traffic generated during an experiment. Referring to the blue line, it can be seen that the sending application running on the Internet node is not always able to send the packets exactly at the points in time specified by the trace file. Furthermore, the higher the traffic rates the bigger the difference between traffic trace and the actually generated traffic. This is one reason why the network is emulated on a scale of 1 : 15.

The current Snort rule database contains over 6.000 attack signatures which are stored in 48 separate files and consequently, a multitude of security services would be running on an active router in case that a security service would be implemented for each of the 48 files. Further, as mentioned, to analyze the effect of heterogeneous protection demands, three types of subnetworks requiring the protection of 0, 5 and 10 security services were specified. For the purpose of traceability only one type of security service, namely the *Frontpage* security service (see Table 4.3), was used. The service processing times $T_{\text{service}}$ of security services using the Aho-Corasick algorithm are about the same size as the complexity of the algorithm

does not depend on the number or on the length of the attack signatures. On average the *Frontpage* security service delays an IP packet—the average IP packet size is 529 Byte—by $5.85\mu s$ when running on a *PC-733* system. Analogous to the scaling down of the traffic rates, the services times $T_{\text{service}}$ have to be multiplied by 15—the security service is executed 15 times resulting in an average service time of time $87.75\mu s$.

The traffic rates listed in Table 7.2 were scaled—under the assumption of constant bit rate traffic and the deployment of the security services corresponding to the *Late* strategy—so that router $R_7$ is overloaded in case of the *high* traffic scenario. The *Late* strategy places all security services requested by networks $N_{10}$ (no security service), $N_{11}$ (5 security services) and $N_{12}$ (10 security services) on router $R_7$. In conformity to Equation 6.1 the total processing time for a packet destined to the subnet $N_{12}$ is:

$$T_{base} + T_{active} + 10 \cdot T_s(Frontpage)$$

Then, the average processing time of a packet at router $R_7$ is:

$$T_{base} + T_{active} + \frac{1}{3390}\left(1073 \cdot 0 + 1191 \cdot 5 \cdot 87.75\mu s + 1126 \cdot 10 \cdot 87.75\mu s\right) = 470\mu s.$$

The average load of router $R_7$ is 1.59 ($= 3390 \cdot 470 \cdot 10^{-6}$) and as explained in Section 6.1 a routers becomes overloaded in case that the average packet inter arrival time equals or exceeds the mean processing time. Figure 7.9 depicts the aggregated flow arriving at router $R_7$ over the time for the three traffic scenarios as well as the router capacity of router $R_7$. In the *medium* traffic scenario the router capacity of $R_7$ is utilized on average by 70% but few traffic bursts exceed the router capacity. Finally, in the *low* traffic scenario no router is overloaded at any time. Table 7.3 summarizes the parameters used for the emulation of the tree-network. Each router was capable of buffering 100 packets.

### 7.3.1 The Benefit of Optimal Deployment Strategies in a Tree-Network

In a first set of experiment runs the deployment strategies *early*, *Late* and *predefined singlepath routing*—for both objective functions presented in Section 6.2—were compared for the three traffic scenarios *low*, *medium* and *high* with each other. In the following the objective function minimizing the number of programmable routers in a network while fulfilling all security requests and keeping all router queues bounded is abbreviated as *minRouter*. Analogously the second objective function which minimizes the maximal workload of a programmable router while fulfilling all security requests and keeping all router queues bounded is referred to as *minQueue*. All router nodes were systems of type *PC-733*, the corresponding parameters are listed in Table 7.3. Further Tables 7.4 to 7.6 represent the optimal solutions calculated by MILP 1 and MILP 2 for all three traffic scenarios.

As depicted in Figure 7.7 all security services are deployed on router $R_1$ in case of the *early* strategy and when deployed pursuant to the *Late* strategy routers $R_2$, $R_4$ $R_6$ and $R_7$ are assigned to run the requested security services. The optimal solutions are calculated using average traffic rates as well as mean packet processing times as input parameters. Regarding Tables 7.4 and 7.5, it can be seen that the *minRouter* strategy places for the *low* and the *medium* traffic scenario all requested security services on router $R_1$, correspondingly, in these two cases the *minRouter* strategy equals the *early* strategy. But the difference

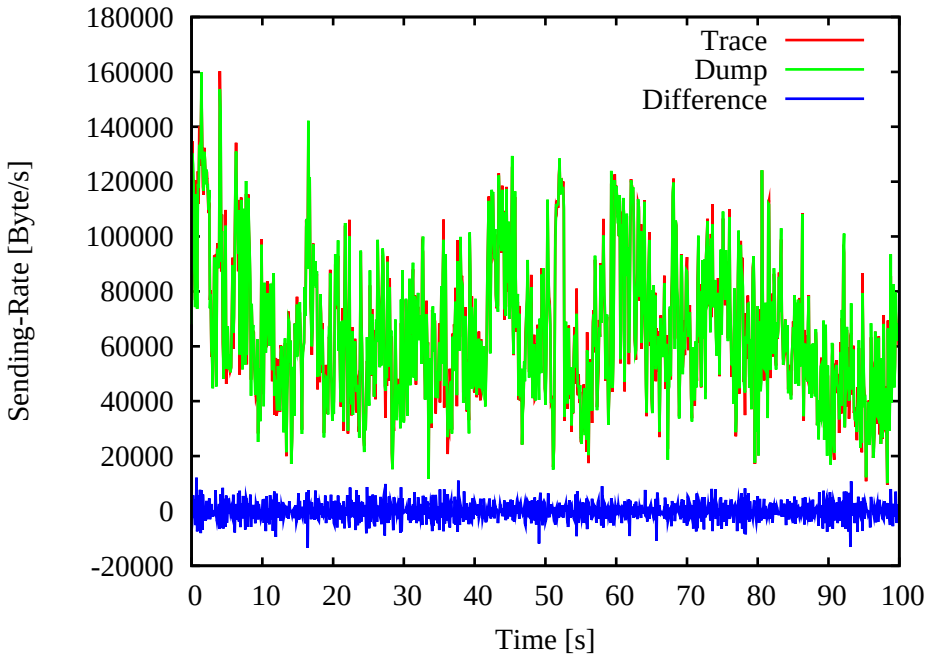



Figure 7.8: Comparison between generated trace and measured dump file for flow from Internet to $N_{12}$

Table 7.2: Traffic matrix

| Subnet | Low Load | | Medium Load | | High Load | |
|---|---|---|---|---|---|---|
| | Packets | Load [Mbps] | Packets | Load [Mbps] | Packets | Load [Mbps] |
| $N_1$ | 18830 | 0.038 | 25479 | 0.175 | 59861 | 0.428 |
| $N_2$ | 20799 | 0.044 | 25025 | 0.172 | 59940 | 0.395 |
| $N_3$ | 19722 | 0.045 | 25969 | 0.179 | 60474 | 0.431 |
| $N_4$ | 17384 | 0.023 | 26358 | 0.183 | 62323 | 0.439 |
| $N_5$ | 20384 | 0.047 | 25987 | 0.167 | 57762 | 0.421 |
| $N_6$ | 18852 | 0.047 | 24178 | 0.161 | 60813 | 0.428 |
| $N_7$ | 20032 | 0.046 | 26652 | 0.191 | 62092 | 0.433 |
| $N_8$ | 16164 | 0.048 | 26375 | 0.182 | 61119 | 0.435 |
| $N_9$ | 21132 | 0.048 | 26970 | 0.189 | 52246 | 0.387 |
| $N_{10}$ | 266534 | 0.596 | 321341 | 2.335 | 648304 | 4.541 |
| $N_{11}$ | 259873 | 0.587 | 305735 | 2.150 | 695383 | 5.041 |
| $N_{12}$ | 265399 | 0.620 | 331250 | 2.321 | 670445 | 4.766 |

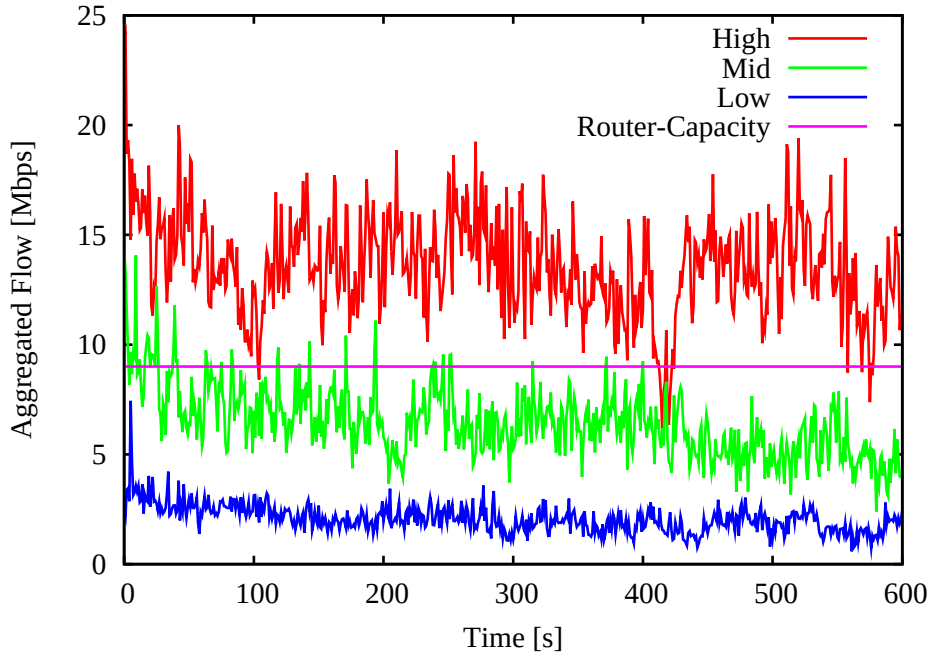



Figure 7.9: Scaling of the traffic scenarios: Aggregated Flow arriving at router $R_7$

Table 7.3: Delay and traffic parameters of the LAN scenario

| Parameter | | Value |
|---|---|---|
| Average IP packet size | | 529 Byte |
| Service delay | $T_{\text{service}}$ | $90.660\mu s$ |
| Routing delay | $T_{\text{base}}$ | $19.880\mu s$ |
| Active router delay | $T_{\text{active}}$ | $4.395\mu s$ |
| FIDRAN queue length | | 100 |
| Number of requested services | | 0, 5, 10 |
| Traffic shape | $\alpha_{on}$ | 1.95 |
| | $\alpha_{off}$ | 1.05 |
| Traffic sources per packet size | | 10 |
| Experiment duration | *low* | 1800 s |
| | *medium* | 600 s |
| | *high* | 600 s |

Table 7.4: Optimal security services deployment for the *low* traffic scenario

| | $N_1$ | $N_2$ | $N_3$ | $N_4$ | $N_5$ | $N_6$ | $N_7$ | $N_8$ | $N_9$ | $N_{10}$ | $N_{11}$ | $N_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $R_1$ | 10 | 5 | - | 10 | 5 | - | 10 | 5 | - | 10 | 5 | - |
| $R_2$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_3$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_4$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_5$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_6$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_7$ | - | - | - | - | - | - | - | - | - | - | - | - |

(a) Strategy: minRouter

| | $N_1$ | $N_2$ | $N_3$ | $N_4$ | $N_5$ | $N_6$ | $N_7$ | $N_8$ | $N_9$ | $N_{10}$ | $N_{11}$ | $N_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $R_1$ | - | - | - | - | - | - | - | - | - | 2 | 1 | - |
| $R_2$ | 10 | 5 | - | - | - | - | - | - | - | - | - | - |
| $R_3$ | - | - | - | - | - | - | - | - | - | 3 | 1 | - |
| $R_4$ | - | - | - | 10 | 5 | - | - | - | - | - | - | - |
| $R_5$ | - | - | - | - | - | - | - | - | - | 3 | 1 | - |
| $R_6$ | - | - | - | - | - | - | 10 | 5 | - | - | - | - |
| $R_7$ | - | - | - | - | - | - | - | - | - | 2 | 2 | - |

(b) Strategy: minQueue

between both strategies becomes evident for the *high* traffic scenario. In contrary to the *minRouter* strategy the *early* strategy does not consider traffic rates. According to Table 7.6 the *minRouter* strategy uses three routers to optimally deploy the security services for the *high* traffic scenario. The reason for this is depicted in Figure 7.10. The left column of the figure depicts the theoretical workload of the routers—equaling the right side of Equation (6.5)—based on the outcome of MILP 1. In addition, the right column of the figure shows the mean packet dropping rates obtained from the experiments carried out. Now, considering the workloads of router $R_1$ for the *early* strategy and for the three traffic scenarios it can be seen that stability constraint is fulfilled for the *low* and *medium* traffic scenario. But in case of the *high* traffic scenario router $R_1$ is overloaded ($> 2$) and consequently, the *minRouter* strategy relocates security services from router $R_1$ to routers $R_3$ and $R_7$.

To minimize the maximal workload of a router the *minQueue* strategy, in contrary to the other strategies, uses as many as possible routers. For all three traffic scenarios the strategy places the security services requested by subnets $N_1$, $N_2$, $N_4$, $N_5$, $N_7$ and $N_8$ on the corresponding last hop routers $R_2$, $R_4$ and $R_6$. The security services requested by networks $N_{10}$ and $N_{11}$ which receive a good portion of all Internet traffic are distributed over four routers, namely routers $R_1$, $R_3$, $R_5$ and $R_7$. Looking at the workloads associated with the *minQueue* strategy in Figure 7.10 it can be seen that for all three traffic scenarios the workloads are well balanced and smaller than one.

Under the assumption of constant-bit-rate traffic, a router starts to drop packets in case that the packet processing time is larger than the packet inter arrival time. A workload of one means that the packet processing time equals the packet inter arrival time. In real networks traffic rates and packet processing times vary due to the self-similar nature of network traffic and due to diverse packet characteristics, but the correlation between theoretical workload and measured packet drop rate is still evident in Figure 7.10. According to the presented results a router is more likely to drop packets the more its workload approaches /exceeds one.

Table 7.5: Optimal security services deployment for the *medium* traffic scenario

| | $N_1$ | $N_2$ | $N_3$ | $N_4$ | $N_5$ | $N_6$ | $N_7$ | $N_8$ | $N_9$ | $N_{10}$ | $N_{11}$ | $N_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $R_1$ | 10 | 5 | - | 10 | 5 | - | 10 | 5 | - | 10 | 5 | - |
| $R_2$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_3$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_4$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_5$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_6$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_7$ | - | - | - | - | - | - | - | - | - | - | - | - |

(a) Strategy: minRouter

| | $N_1$ | $N_2$ | $N_3$ | $N_4$ | $N_5$ | $N_6$ | $N_7$ | $N_8$ | $N_9$ | $N_{10}$ | $N_{11}$ | $N_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $R_1$ | - | - | - | - | - | - | - | - | - | 3 | - | - |
| $R_2$ | 10 | 5 | - | - | - | - | - | - | - | - | - | - |
| $R_3$ | - | - | - | - | - | - | - | - | - | 3 | 1 | - |
| $R_4$ | - | - | - | 10 | 5 | - | - | - | - | - | - | - |
| $R_5$ | - | - | - | - | - | - | - | - | - | 3 | 1 | - |
| $R_6$ | - | - | - | - | - | - | 10 | 5 | - | - | - | - |
| $R_7$ | - | - | - | - | - | - | - | - | - | 1 | 3 | - |

(b) Strategy: minQueue

Table 7.6: Optimal security services deployment for the *high* traffic scenario

| | $N_1$ | $N_2$ | $N_3$ | $N_4$ | $N_5$ | $N_6$ | $N_7$ | $N_8$ | $N_9$ | $N_{10}$ | $N_{11}$ | $N_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $R_1$ | 10 | 5 | - | 10 | 1 | - | 9 | 5 | - | 1 | 4 | - |
| $R_2$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_3$ | - | - | - | - | 4 | - | 1 | - | - | 1 | - | - |
| $R_4$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_5$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_6$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_7$ | - | - | - | - | - | - | - | - | - | 8 | 1 | - |

(a) Strategy: minRouter

| | $N_1$ | $N_2$ | $N_3$ | $N_4$ | $N_5$ | $N_6$ | $N_7$ | $N_8$ | $N_9$ | $N_{10}$ | $N_{11}$ | $N_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $R_1$ | - | - | - | - | - | - | - | - | - | 3 | 1 | - |
| $R_2$ | 10 | 5 | - | - | - | - | - | - | - | - | - | - |
| $R_3$ | - | - | - | - | - | - | - | - | - | 3 | 1 | - |
| $R_4$ | - | - | - | 10 | 5 | - | - | - | - | - | - | - |
| $R_5$ | - | - | - | - | - | - | - | - | - | 3 | 1 | - |
| $R_6$ | - | - | - | - | - | - | 10 | 5 | - | - | - | - |
| $R_7$ | - | - | - | - | - | - | - | - | - | 1 | 2 | - |

(b) Strategy: minQueue

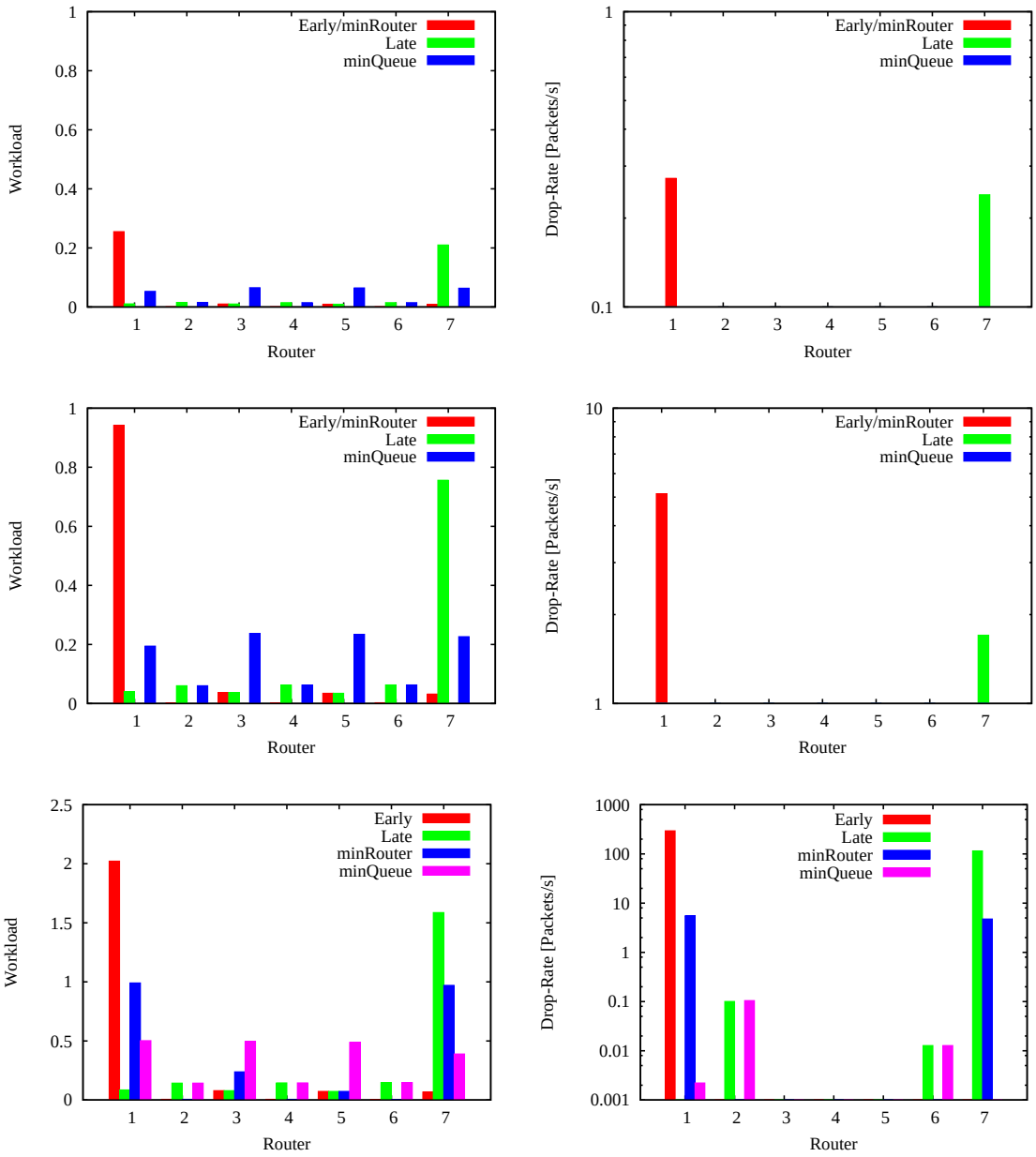



Figure 7.10: Left column depicts the theoretical average workload of the routers as an outcome of the MILPs and the right column shows the associated mean dropping rates for the traffic scenarios *low*, *medium* and *high* for the tree-network

For the traffic scenarios *medium* and *high* the routers $R_1$ and $R_7$ have the highest utilization as well as the highest drop rates. Furthermore, the packet loss rate increases tremendously for workloads larger than one. For example, routers $R_1$ and $R_7$ have a load of 0.99 and 0.97 for the *high* traffic scenario and the *minRouter* deployment strategy. The corresponding packet loss rates are 5.53 $\frac{packets}{s}$ and 4.77 $\frac{packets}{s}$. Under the same traffic conditions router $R_1$ ($R_7$) has a workload of 2.02 (1.59) in case that the security services are deployed pursuing the *early* (*Late*) strategy. The related drop rates are 885.97 $\frac{packets}{s}$ and 346.93 $\frac{packets}{s}$ and consequently, the relation between workload and drop rate is nonlinearly.

However, a small portion of packets may also be dropped in case of small workloads. For example, router $R_1$ in case of the *early* deployment strategy and router $R_7$ in case of the *Late* deployment strategy drop a small amount of packets although their workloads are smaller than one in case of the *low* traffic scenario. This can be explained with the help of Figure 7.9 which depicts the aggregated flows for all traffic levels from the Internet to router $R_7$. Considering the blue curve, representing the aggregated flow for the *low* traffic scenario, the router is capable to handle all arriving packets on average, but few packet bursts exceed its capacity leading to packet loss.

In the following the impact of the deployment strategies on the packets' end-to-end-delays is analyzed. Figure 7.11 depicts the flow-specific average end-to-end delays as well as the flow-specific average packet drop rates for each emulated network scenario. The results for the *low* traffic scenario show that the *early* and the *minRouter* strategy cause the highest end-to-end-delays. Referring to the delay caused by the operation of an intrusion prevention overlay network the *minQueue* strategy performs best. But overall it can be said that the differences between the deployment strategies are considerably small for the *low* traffic scenario. The reason for this is that the packet loss rate even for the *early* deployment strategy is negligible and as a result, it can be inferred that in this scenario the time $T_{\text{waiting}}$—the time that a packet must wait before being served—is very small. Drawing the attention towards the traffic scenarios *medium* and *high* the differences between the security services deployment strategies increase but still the *early* strategy causes the largest end-to-end-delays. Additionally, the results clearly indicate a correlation between packet loss rates and end-to-end-delays. The higher the packet loss rate the longer it takes a packet to reach its destination. The reason for this is the waiting time $T_{\text{waiting}}$ which increases with growing traffic volumes. For example, the contribution of the delay $T_{\text{waiting}}$ to the overall end-to-end-delay is visible for the flows addressed to subnets $N_{10}$, $N_{11}$ and $N_{12}$ for the *Late* deployment strategy. Packets destined to the former two subnets are inspected by ten and five security services respectively resulting in theoretical minimum end-to-end-delays of:

$$e2e_{N_{10}} = 10 \cdot 90.660\mu s + 1 \cdot 4.395\mu s + 4 \cdot 19.880\mu s = 990.5\mu s$$

$$e2e_{N_{11}} = 5 \cdot 90.660\mu s + 1 \cdot 4.395\mu s + 4 \cdot 19.880\mu s = 537.2\mu s$$

In practice, end-to-end-delays of $2263\mu s$ and $1627\mu s$ were measured for the *low* traffic scenario. These increase to $5300\mu s$ and $4308\mu s$ for the *medium* traffic scenario and in case of the *high* traffic scenario it takes a packet $31668\mu s$ and $30178\mu s$ on average to reach subnets $N_{10}$ and $N_{11}$ respectively. The theoretical calculated values ignore the time periods that are needed to send and receive a packet. Further, although the link propagation delays are set to zero, the emulation of them will add a minimal overhead to the delay of each packet. However

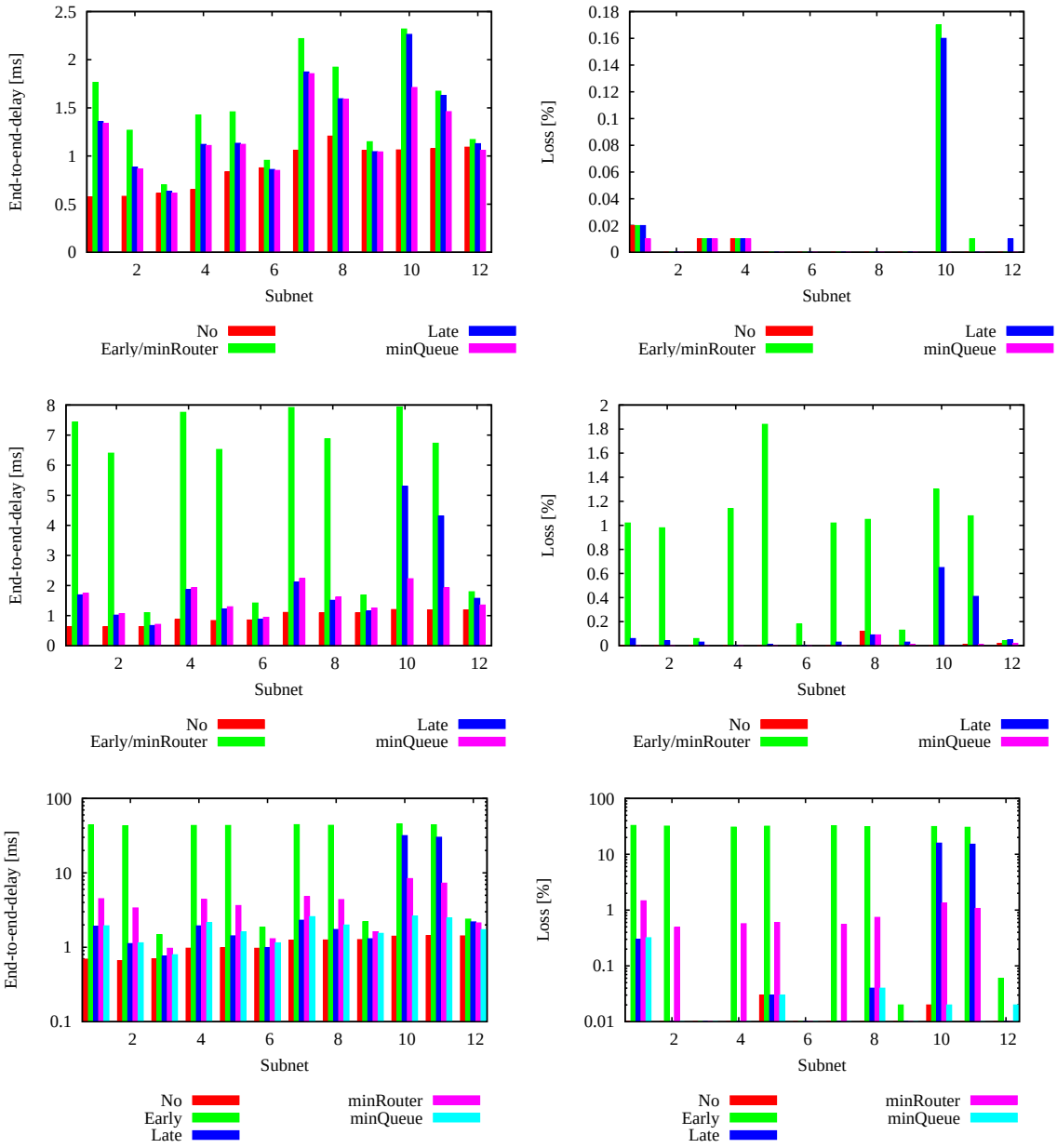



Figure 7.11: The figures in the left column depict the flow-specific end-to-end delays and the figures on the right side show the flow-specific drop-rate for the traffic scenarios *low*, *medium* and *high* for the tree-network

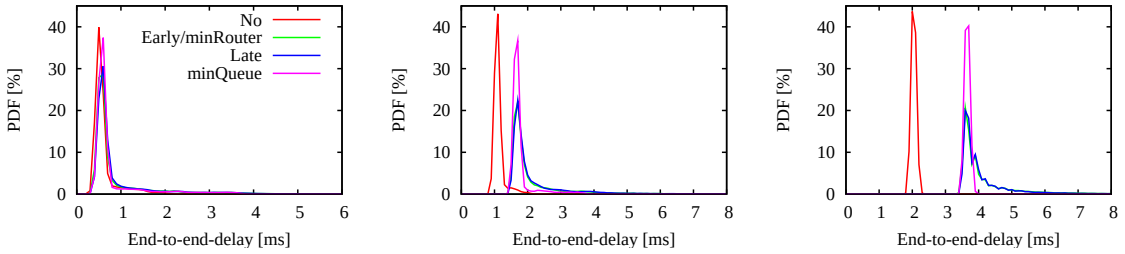
the discrepancies between the measured end-to-end-delays are mainly caused by the waiting times $T_{\text{waiting}}$. But when applying the *minQueue* deployment strategy delays of $2626\mu s$ and $2478\mu s$ were measured for the same flows in case of the *high* traffic scenario. The *minQueue* strategy reduces the waiting time as it minimizes the maximal workload of all Fidran routers or in other words, it minimizes the maximal average waiting time $T_{\text{waiting}}$ of all programmable routers. The effect is clearly visible for the *high* traffic scenario as end-to-end-delays as well as packet loss are significantly smaller compared to the other strategies.

Recapitulating, the benefit of the optimal deployment strategies *minRouter* and *minQueue*—compared to the intuitive strategies *early* and *Late*—becomes more apparent with increasing traffic rates. Both optimal strategies show that they reduce the impact—in terms of packet loss and end-to-end-delay—of an intrusion prevention overlay network on the network performance. Finally, packets that must not be analyzed by any security services are additionally delayed by the Fidran framework. In the given example, this counts for the flows addressed to networks $N_3$, $N_6$, $N_9$ and $N_{12}$ and according to Figure 7.11 the related packet loss rates are negligible. Further, the figure also shows the measured end-to-end-delays in case that no Fidran system was running on a router of the network (deployment strategy *no*). Considering the end-to-end-delays of the flows to subnets $N_3$, $N_6$, $N_9$ and $N_{12}$ it can be seen that the additional delay added by the Fidran framework to the overall end-to-end-delay is rather small.

A finding of the performance evaluation of the Fidran prototype conducted in Section 4.5 was that the processing times of the implemented security services depend on the size of the packets. But the results shown in Figures 4.11 and 4.12 only count for packets of a constant size that have to be inspected by a single security service. Hence the next paragraph analyzes how the size of an IP packet affects the overall end-to-end-delay of a flow that has to traverse multiple routers to reach its destination and which in addition must be analyzed by multiple security services. Figure 7.12 depicts for each deployment strategy the PDF of the end-to-end-delays of packets of size 40, 5xx and 1500 Byte that were send in case of the *low* traffic scenario to subnet $N_{10}$. The label *5xx* includes the packet sizes 552 and 576 Byte. Further, the *low* traffic scenario was chosen to minimize the influence of the waiting time $T_{\text{waiting}}$ on the overall end-to-end-delay. The following findings can be inferred from the figure:

- the larger a packet the longer is it delayed and
- the *minQueue* strategy efficiently reduces the time that packets must wait to be served.

The larger a packet the longer it takes the packet to reach its destination whether or not a Fidran system is running. For instance, on average it lasts 0.61 ms till a packet of size 40 Byte completes the way from the Internet to subnet $N_{10}$ in case that no router is running the Fidran system (*no*-strategy). However, under the same conditions the average travel time for a packet of size 5xx Byte is 1.1 ms and a packet of size 1500 Byte actually requires 2.0 ms to complete the same distance. The reason for this is that it takes more time to send and receive large packets compared to small ones. Moreover, the operation of the Fidran framework aggravates the effect of the packet size on the end-to-end-delay. On average the time period of 0.83 ms elapses till an IP packet of size 40 Byte reaches network $N_{10}$ in case that the security services are deployed according to the *minQueue* strategy. In the same scenario packets of size 5xx Byte and 1500 Byte require 1.77 ms and 3.66 ms respectively to arrive at the destination.




(a) 40 Byte packets (b) 5xx Byte packets (c) 1500 Byte packets

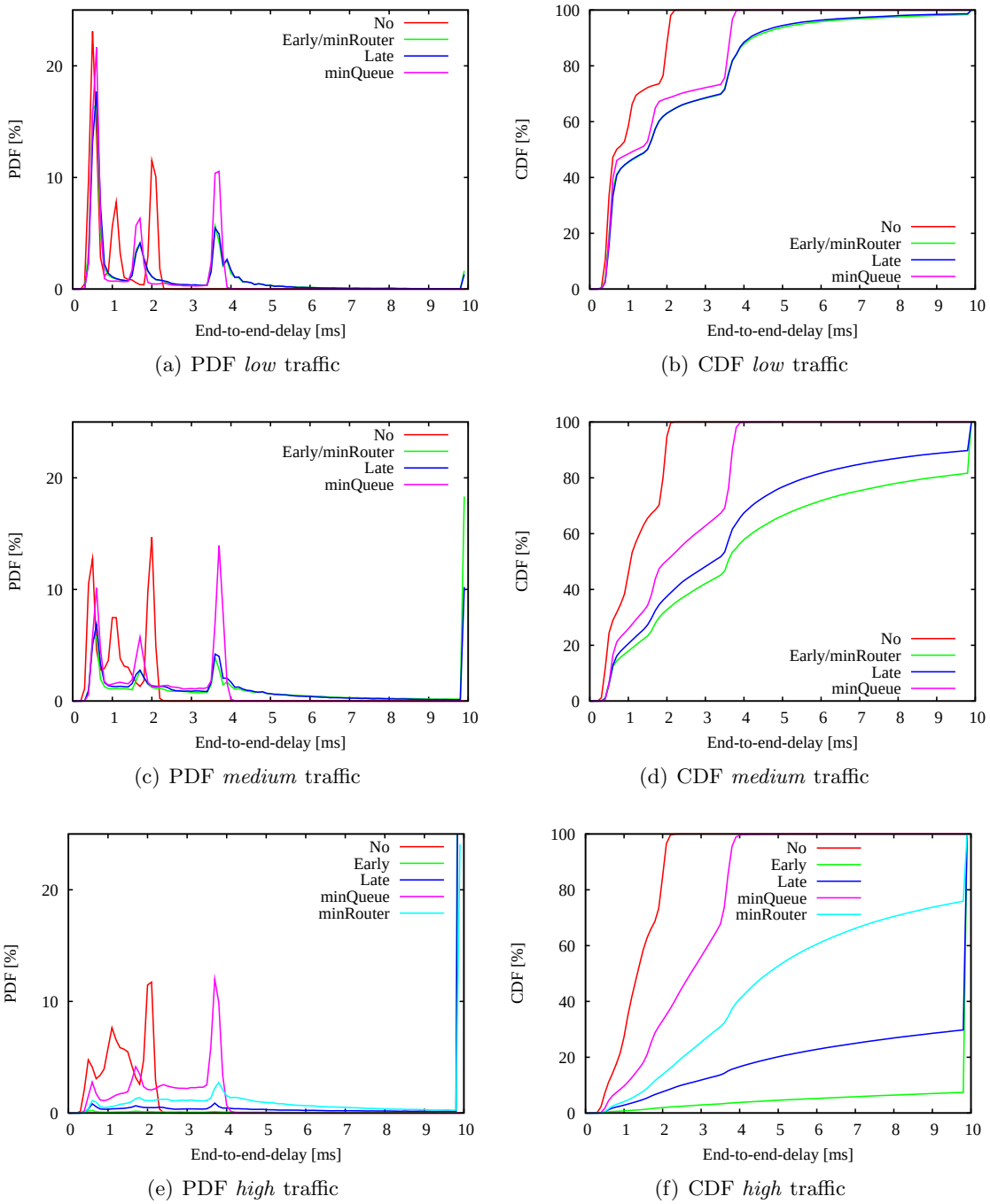
Figure 7.12: Probability density functions of measured end-to-end-delays—flow addressed to subnet $N_{10}$ and *low* traffic scenario—for packets of size 40, 5xx and 1500 Byte

The second finding says that the *minQueue* strategy efficiently reduces the time that packets must wait to be served. Considering the graphs in Figure 7.12(c) which depict the results of the strategies *early*, *Late* and *minRouter* it can be seen that some packets require more time than 4 ms to arrive at subnet $N_{10}$ and this is not the case for the *minQueue* strategy. In addition, the PDFs related to the *minQueue* strategy have for all packet sizes the highest peaks compared to the other deployment strategies. In other words, the wider the shape of a graph the longer do packets wait to be served.

Figures 7.13 and 7.14 depict the *cumulative distribution functions* (CDF) as well as the PDFs of the end-to-end-delays measured at subnets $N_{10}$ and $N_{12}$ for all traffic scenarios. The two flows were chosen because the paths from the Internet to subnets $N_{10}$ and to subnet $N_{12}$ are the longest of the network. In addition, the flow destined to subnet $N_{10}$ must be analyzed by ten security services whereas packets addressed to $N_{12}$ must not traverse any security service. Accordingly, it is assumed that the contrast between both flows has noticeable implications on the corresponding end-to-end-delays.

Figures 7.13(a), 7.14(c) and 7.14(e) depict the PDFs of the delays measured for the flow between the Internet and network $N_{10}$. Each curve has three peaks and as shown in Figure 7.12 packets of size 40, 5xx and 1500 Byte respectively can be assigned to them. Furthermore, the figures reveal the impact of the traffic scenario on the end-to-end-delays. The more traffic is transported through the network the more increases the average end-to-end-delay. This can also be seen in Figures 7.13(b), 7.13(d) and 7.13(f) as the slopes of the depicted CDFs decrease for increasing traffic volumes. In addition, the figures show that the impact of the deployment strategies increases for growing traffic rates. The difference between the results in case that no router is running the Fidran system and the case that security services are located in the network increases for growing traffic volumes. The figure shows the benefit of the developed optimal strategies which perform better compared to the strategies *early* and *Late*.

Figure 7.14 shows the impact of the Fidran framework on a flow that must not be analyzed by security services. Again the overhead caused by Fidran increases for growing traffic volumes and again strategies *minRouter* and *minQueue* have a smaller influence compared to strategies *early* and *Late* on the end-to-end-delay in case of the *high* traffic scenario. To explain this it must be kept in mind that single processor systems were used throughout the




(a) PDF *low* traffic

(b) CDF *low* traffic

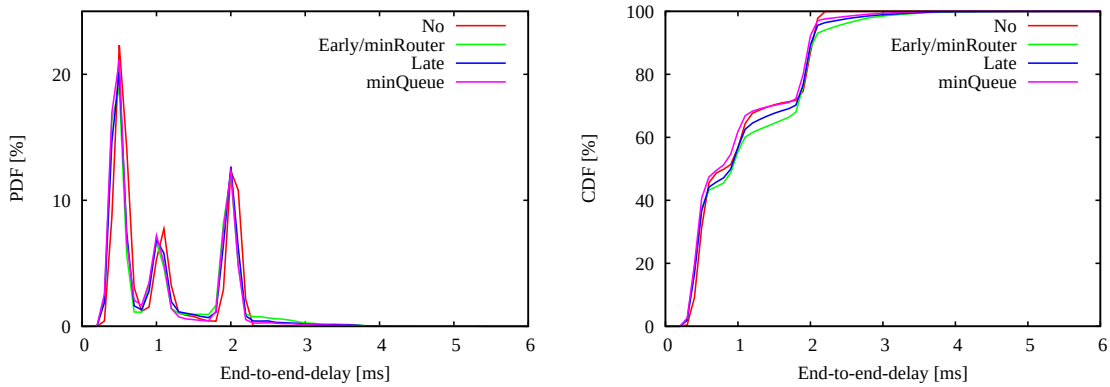
(c) PDF *medium* traffic

(d) CDF *medium* traffic

(e) PDF *high* traffic

(f) CDF *high* traffic

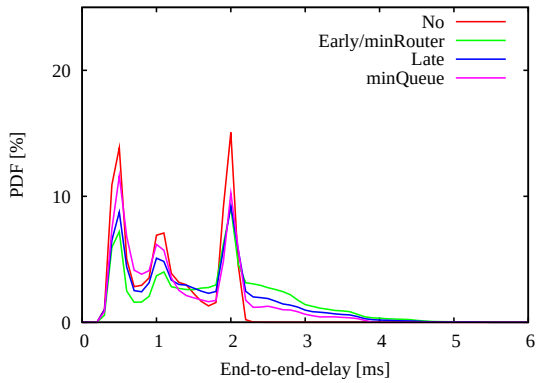
Figure 7.13: PDFs and CDFs of the end-to-end-delays measured at subnet $N_{10}$ for traffic scenarios *low*, *medium* and *high*.



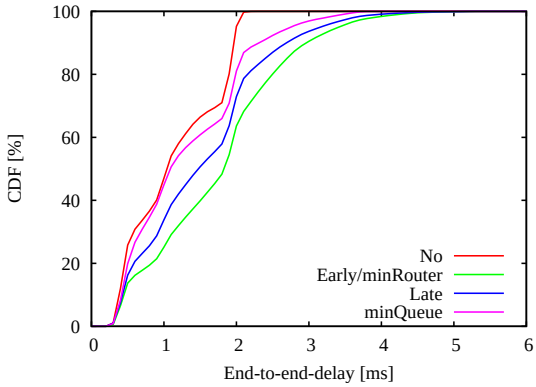

(a) PDF *low* traffic



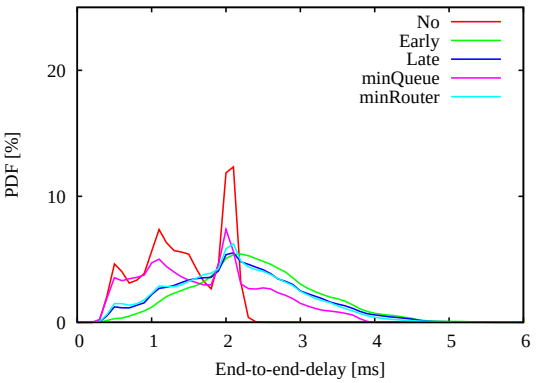

(b) CDF *low* traffic



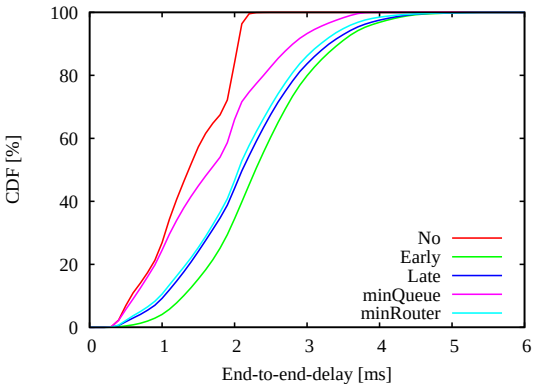

(c) PDF *medium* traffic




(d) CDF *medium* traffic




(e) PDF *high* traffic




(f) CDF *high* traffic

Figure 7.14: PDFs and CDFs of the end-to-end-delays measured at subnet $N_{12}$ for traffic scenarios *low*, *medium* and *high*.

experiments. The following happens in case that a Fidran system serves a packet when another one arrives at a NIC of that router. The arriving packet triggers the execution of an hard interrupt which have priority over soft interrupts as well as kernel threads. Accordingly, the processor halts the execution of the active Fidran kernel thread and then it executes the requested hard interrupt routine which transfers the packet from the NIC buffer to the kernel space. Subsequently the processor schedules the appropriate soft interrupt which is responsible for the further processing of the packet and it reactivates the interrupted Fidran thread. The newly arrived packet stays in the kernel buffer until the soft interrupt routine becomes active. Consequently, the probability that a packet must wait to be passed to the Fidran system is higher the higher the workload of the latter. As the *minQueue* strategy minimizes the maximal workload over all routers and hence it also reduces the overall waiting time. In case of a multi-processor system, one processor can be assigned to handle the arriving packets whereas the remaining processors perform the security analysis.

To validate this, the behavior of a Fidran router was examined for the case that the packet arrival rate exceeds the router's capacity. Again the testbed depicted in Figure 4.6 was setup—using systems of type PC3000—and the security service deployed on the router delays all packets by 0.25 ms. Further, traffic was send at constant rates from the sender to the receiver and all IP packets were of size 1500 Byte. Figure 7.15 depicts the measured delays $t_{system}$, $t_{fidran}$, $t_{base}$ over the offered data-rate as well as the theoretical delay that is caused by processing 100 packets with security service s (processing time $t_s$). The delay $t_{system}$ is the overall time that a packet stays on the router whereas $t_{fidran}$ represents the period of time that a packet remains inside the Fidran system, corresponding to tracepoints T2 and T5 depicted in Figure 4.5. Delay $t_{base}$ is the core routing delay which in this case equals the difference between the delays $t_{system}$ and $t_{fidran}$. Finally, the last curve represents the theoretical period of time that it takes to empty a filled waiting queue (buffer size 100 packets) under the assumption that the packets must be inspected by security service s and no further packets arrive. Referring to Figure 7.15, the router is working to capacity at a traffic-rate of about 31 Mbps and consequently, at that point there is a sudden increase of the delays $t_{system}$ and $t_{fidran}$. For higher traffic rates, the Fidran waiting queue is filled and consequently, both delays are bounded by the length of that queue. Further, the discrepancy between the delay $t_{fidran}$ and the purple curve is caused by the effect described above, but according to the figure the waiting time is still dominating the overall delay $t_{system}$.

### 7.3.2 Programmable Routers of Heterogeneous Performance

To emulate the cooperation of routers of varying computational speeds, routers $R_1$, $R_3$ and $R_5$ of the tree network depicted in Figure 7.6 were substituted by routers of type *PC3000*. The security service requirements were the same as in the previous section and the deployment strategies *minRouter* and *minQueue* calculated for the *high* traffic scenario are listed in Table 7.7. To facilitate the identification of the routers of type *PC3000* routers $R_1$, $R_3$ and $R_5$ are underlined in the table. Intuitively, the substitution of the routers affects strategies *early*, *minRouter* and *minQueue* as they use at least one of these routers to inspect the network traffic. In contrast the *Late* strategy might not benefit of the substitution.

A first finding when comparing Table 7.7 with Table 7.6 is that for both scenarios the results calculated for the deployment strategy *minRouter* are identical. In both scenarios all

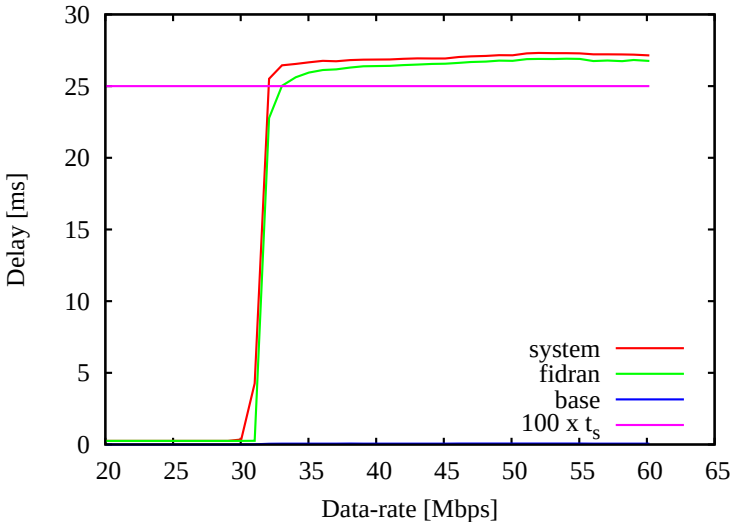



Figure 7.15: Delays caused by a PC3000 system

Table 7.7: Optimal security services deployment for the *high* traffic scenario and heterogeneous routers

| | $N_1$ | $N_2$ | $N_3$ | $N_4$ | $N_5$ | $N_6$ | $N_7$ | $N_8$ | $N_9$ | $N_{10}$ | $N_{11}$ | $N_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $R_1$ | 10 | 5 | - | 10 | 5 | - | 10 | 5 | - | 10 | 5 | - |
| $R_2$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_3$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_4$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_5$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_6$ | - | - | - | - | - | - | - | - | - | - | - | - |
| $R_7$ | - | - | - | - | - | - | - | - | - | - | - | - |

(a) Strategy: minRouter

| | $N_1$ | $N_2$ | $N_3$ | $N_4$ | $N_5$ | $N_6$ | $N_7$ | $N_8$ | $N_9$ | $N_{10}$ | $N_{11}$ | $N_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $R_1$ | 3 | - | - | 1 | 2 | - | 1 | 2 | - | 2 | 2 | - |
| $R_2$ | 7 | 5 | - | - | - | - | - | - | - | - | - | - |
| $R_3$ | - | - | - | - | - | - | - | - | - | 4 | 1 | - |
| $R_4$ | - | - | - | 9 | 3 | - | - | - | - | - | - | - |
| $R_5$ | - | - | - | - | - | - | - | - | - | 4 | 1 | - |
| $R_6$ | - | - | - | - | - | - | 9 | 3 | - | - | - | - |
| $R_7$ | - | - | - | - | - | - | - | - | - | - | 1 | - |

(b) Strategy: minQueue

security services are placed on router $R_1$. But regarding the results for the *minQueue* strategy it can be seen that security services are relocated: security services are removed from slower routers and inserted on routers of higher performance. For example, in the previous section routers $R_2$, $R_4$ and $R_6$ were assigned to run 15 security services (Table 7.6), but by upgrading routers $R_1$, $R_3$ and $R_5$ three security services are relocated from each of the routers to router $R_1$.

Figure 7.16 depicts the results for the heterogeneous network setup. The workloads cal-

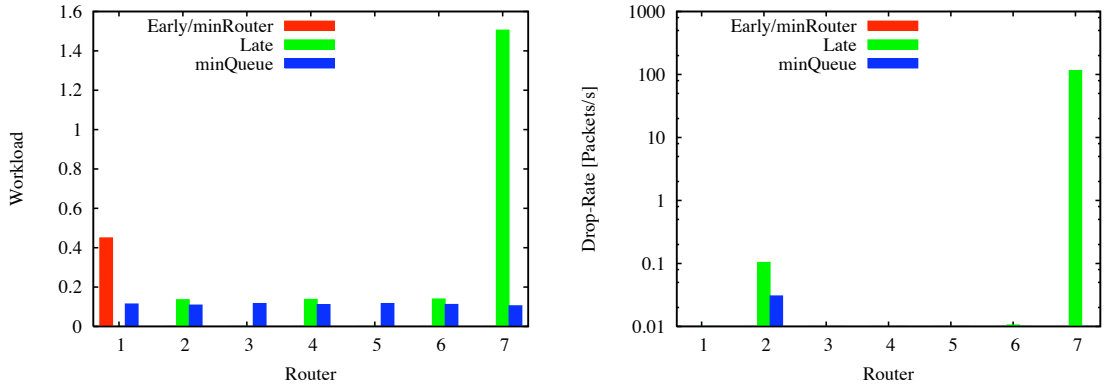
culated for every strategy and router are illustrated in Figure 7.16(a). Furthermore, the absolute number of packets dropped per second by the Fidran systems, the flow specific end-to-end-delay as well as the flow specific packet loss rates are shown for every strategy in the remaining figures. A comparison of the results depicted in Figure 7.16 with the results presented in the previous section, approves the assumption that the *Late* strategy does not benefit of the substitution of the routers. Differently the remaining strategies are able to use the improved performance of routers $R_1$, $R_3$ and $R_5$ resulting in a reduction of packet loss and end-to-end-delays. For the strategies *early* and *minRouter* almost no packet loss is observed. Drawing the attention towards the *minQueue* strategy there the maximal workload $v$ is reduced by the substitution of the three routers from $v_{pc733} = 0.399085$ to $v_{heterogeneous} = 0.116541$. But the packet drop rate measured for the *minQueue* strategy is slightly higher than for strategies *early* and *minRouter* respectively. In detail, the Fidran system running on router $R_2$ which is of type *PC733* drops on average 0.03 packets per second. The objective of the *minQueue* strategy is to reduce the maximal workload of all routers. A router's workload of 0.2 means that on average the router is occupied at 20% of any given time period. Consequently, during 80% of that interval the router is idle and it has free capacity to process packets. But it must be kept in mind that the free capacity of a router of type *PC733* is faster exhausted, for example through the occurrence of a traffic peak, than the free-capacity of *PC3000* router.

Summarizing, this section showed the benefit of optimally deploying security services in a tree network. The deployment strategies introduced in Section 6.3 were assessed for the traffic scenarios *low*, *medium* and *high*. The experiments demonstrated that the impact of doing intrusion prevention in terms of packet loss and end-to-end-delay can be reduced by the intelligent distribution of security services. Furthermore, the penalty for flows that do not require any protection is within acceptable limits and finally, it was shown that it is possible to consider routers of heterogeneous performance.

## 7.4 A High-Speed Networking Environment: The Abilene Network

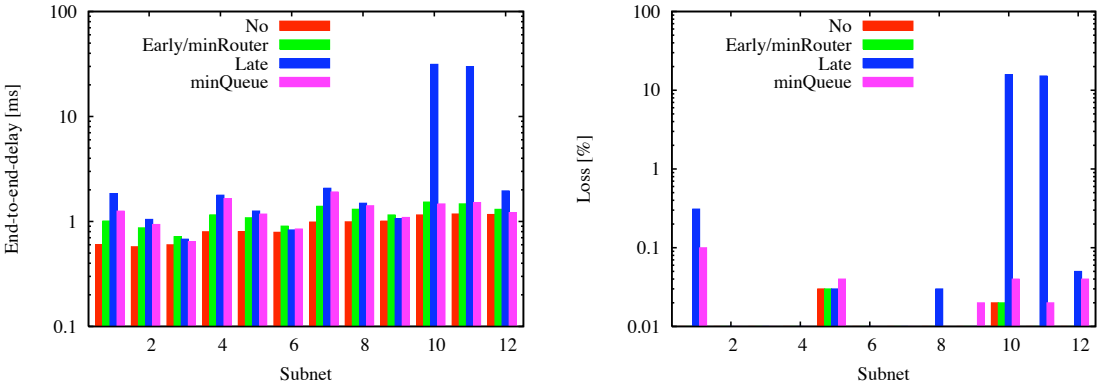
The *Abilene* network, depicted in Figure 7.17, was the second evaluated scenario. Abilene is a research IP backbone connecting multiple universities across the US which makes real world data—traffic flows and link capacities—available on the project's web-site [11]. Abilene is an example for the deployment of Fidran in a high-speed networking environment like a backbone. The propagation delay for each link was specified by dividing the distance from start node to end node by the speed of light.

In the network each subnet sends data to all other subnets resulting in an overall number of 30 traffic flows. Table 7.8 represents the traffic matrix, the column index specifies the source and the row index the destination. To avoid effects of congestion and flow control mechanisms all experiments were restricted to Udp-traffic. To consider the hardware resources provided by the DETER testbed, the network was emulated on a scale of $1 : 100$ which means that traffic rates were divided by 100 and accordingly the delays were multiplied by 100. To generate the traffic each subnet is supplied with an UDP sender for each destination which generates self-similar traffic as described previously. The durations of the *ON* and *OFF*




(a) Theoretical workload

(b) Packets dropped per second by FIDRAN




(c) Flow-specific end-to-end-delays

(d) Flow-specific packet loss

Figure 7.16: The results for the LAN network with heterogeneous routers for the *high* traffic scenario

states are Pareto-distributed with parameters: $\alpha_{ON} = 1.7$, $\alpha_{OFF} = 1.2$ and $location = 1$. Each experiment lasted 300 s and included the sending of over 850,000 packets and each experiment was conducted with three different traces.

The experiments described in following examine the benefit of combining the process of routing traffic demands with the placement of the security services. In the previously discussed tree network scenarios the routes were predefined. Consequently, singlepath routing as well as multipath routing in combination with the deployment of security services means a further degree of freedom in the optimization.

The security services listed in Table 4.2 were used throughout the conducted experiments. In a first series of experiments it was assumed that each flow in the network is analyzed by the first six security services and in a second series each flow is inspected by all seven security services. The security services processing times were adapted as explained to the Deterlab network environment.

Table 7.9 lists the deployment strategies that are compared in this section. The *Late*

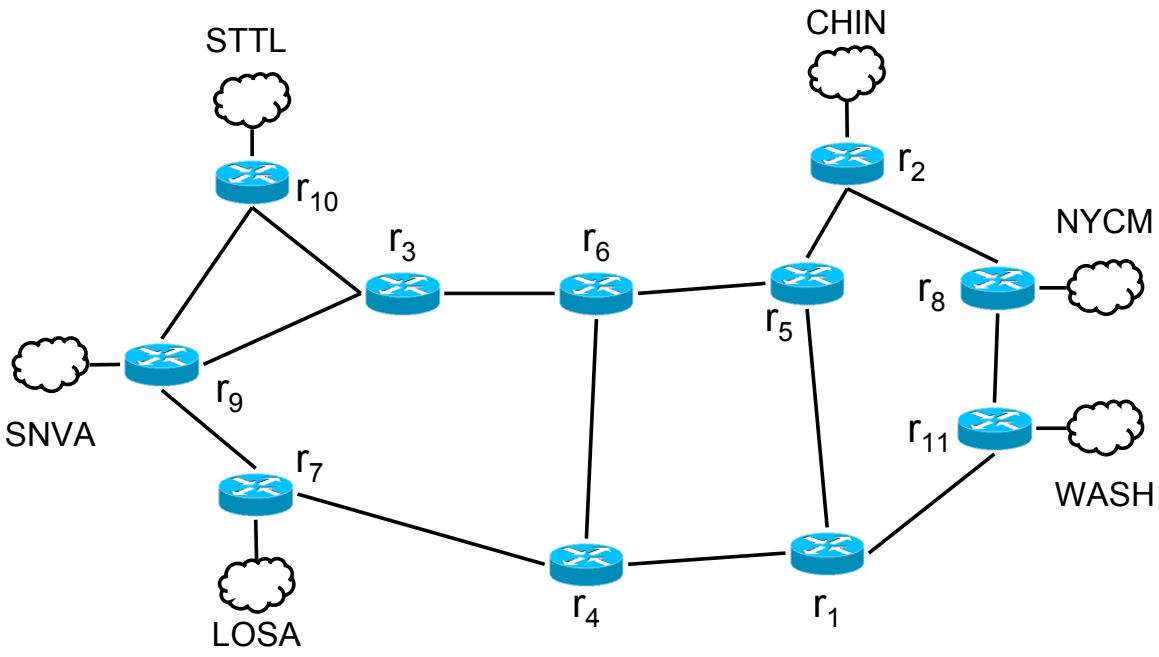



Figure 7.17: The Abilene network topology

strategy uses the shortest paths calculated by the Dijkstra algorithm and it places the requested security services as close as possible to the requesting subnets. Furthermore, the performance of the solutions obtained of the MILPs introduced in Chapter 6 with the objective of minimizing the maximum router utilization are studied and compared. In the context of predefined singlepath routing scenario also the Dijkstra algorithm is used to calculate the predefined paths. The *Pre* strategy (for predefined) optimally distributes the security services among the routers in the shortest path.

### 7.4.1 Abilene Network: Securing each Commodity by Six Security Services

In this scenario each demand $k$ is analyzed by six security services $S_k = \{S_1 \dots S_5\}$. To better understand the differences in the service placement, Tables 7.10 show the amount of service assignments for each destination-router pair that is the sum of services multiplied by the flow or flow-fraction processed by each router. In total a number of 180 service assignments must be made as five flows are sent to each subnet. Fractional flow assignments happen in case of the multipath routing strategy *MP* which splits given commodities into multiple smaller flows.

The *Late* strategy places the security services as close as possible to the requesting subnets on the shortest paths. It neither considers the volume routed via a path nor does it regards the load that is caused by the deployment of security services. Accordingly, security services are only deployed on six out of eleven routers. The *Pre* strategy distributes the requested

Table 7.8: The Abilene Traffic Matrix [Mbps]

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 35.53 | 6.77 | 3.75 | 8.37 | 14.77 |
| LOSA | 113.58 | X | 51.50 | 10.30 | 26.26 | 58.90 |
| NYCM | 71.82 | 64.68 | X | 10.44 | 31.91 | 108.35 |
| SNVA | 5.68 | 34.09 | 3.29 | X | 55.06 | 2.13 |
| STTL | 66.26 | 27.79 | 21.84 | 9.02 | X | 15.63 |
| WASH | 93.45 | 75.20 | 176.86 | 8.22 | 36.30 | X |

Table 7.9: The Scenarios Emulated

| Scenario | Description |
|---|---|
| Late | Security services are deployed as close as possible to the requesting subnet/end-system on the shortest paths |
| Pre | Security services are optimally distributed among the routers in the shortest path |
| SP | Security services are optimally distributed among the routers in a single path obtained from joint optimization |
| MP | Security services are optimally distributed among the routers of multiple paths obtained from joint optimization |

Table 7.10: Service assignments under the constraint that each commodity must be analyzed by **6** security services

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| $R_1$ | x | x | x | x | x | x |
| $R_2$ | 30 | x | x | x | x | x |
| $R_3$ | x | x | x | x | x | x |
| $R_4$ | x | x | x | x | x | x |
| $R_5$ | x | x | x | x | x | x |
| $R_6$ | x | x | x | x | x | x |
| $R_7$ | x | 30 | x | x | x | x |
| $R_8$ | x | x | 30 | x | x | x |
| $R_9$ | x | x | x | 30 | x | x |
| $R_{10}$ | x | x | x | x | 30 | x |
| $R_{11}$ | x | x | x | x | x | 30 |

(a) *Late* strategy

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| $R_1$ | x | 5 | x | x | x | 6 |
| $R_2$ | 11 | x | 1 | x | x | 3 |
| $R_3$ | 2 | x | 1 | x | 1 | x |
| $R_4$ | x | 1 | x | x | x | x |
| $R_5$ | 2 | 1 | x | x | x | x |
| $R_6$ | 7 | x | x | x | x | x |
| $R_7$ | x | 14 | 2 | 6 | 3 | x |
| $R_8$ | 1 | x | 6 | x | x | 6 |
| $R_9$ | 6 | 4 | 10 | 24 | 6 | 6 |
| $R_{10}$ | 1 | 5 | 5 | x | 20 | 6 |
| $R_{11}$ | x | x | 5 | x | x | 3 |

(b) *Pre* strategy

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| $R_1$ | x | 9 | x | x | x | 5 |
| $R_2$ | 8 | x | 6 | x | x | x |
| $R_3$ | 10 | x | 6 | 4 | x | 5 |
| $R_4$ | x | 2 | 3 | x | x | x |
| $R_5$ | 1 | 5 | 1 | 2 | 1 | 3 |
| $R_6$ | 4 | x | 6 | 6 | 13 | x |
| $R_7$ | x | 3 | 5 | 1 | x | 5 |
| $R_8$ | 4 | x | 1 | x | x | 3 |
| $R_9$ | 1 | 7 | x | 14 | 5 | 5 |
| $R_{10}$ | 2 | 4 | x | 3 | 11 | 1 |
| $R_{11}$ | x | x | 2 | x | x | 3 |

(c) *SP* strategy

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| $R_1$ | 3.05 | 1.00 | 0.10 | x | 0.29 | 3.17 |
| $R_2$ | 6.65 | x | 6.00 | x | x | 6.00 |
| $R_3$ | 8.63 | 2.20 | 7.36 | 1.00 | 5.73 | 0.06 |
| $R_4$ | 0.75 | 4.05 | 2.55 | 1.00 | 0.76 | 4.00 |
| $R_5$ | 4.67 | 3.26 | 2.04 | 1.00 | 2.00 | 0.49 |
| $R_6$ | 2.25 | 4.28 | 2.97 | 4.00 | 3.00 | 2.33 |
| $R_7$ | 1.37 | 7.11 | 2.70 | x | 0.47 | 3.00 |
| $R_8$ | x | x | 2.90 | x | x | 1.53 |
| $R_9$ | 1.63 | 5.88 | 0.45 | 17.00 | 3.81 | 5.02 |
| $R_{10}$ | 1.00 | 2.22 | 0.03 | 6.00 | 13.94 | 2.87 |
| $R_{11}$ | x | x | 2.90 | x | x | 1.53 |

(d) *MP* strategy

security services among the routers of the shortest paths. This leaves little flexibility for the optimization of service placement. The shortest paths between neighboring subnets often consists of a considerably small number of hops. For example, the security services for the commodities between NYCM and WASH—the largest and the third largest commodity—can be deployed on two routers. This can easily result in resource shortages. Finally, strategies *SP* and *MP*, combine the task of routing traffic through the network with the task of placing security services in the network. As a result of this both strategies deploy security services on all routers of the network.

The resulting path lengths in terms of hops are given in Tables 7.11. Strategies *Late* and *Pre*—in contrast to strategies *SP* and *MP*—use symmetric routes for the commodities between two subnets. In context of the multipath routing strategy the number of routers is counted over all paths used by the corresponding commodity but a hop that lies on multiple paths is only considered once. For example, the demand from CHIN to LOSA is routed via paths:

- $\underline{R_2} \rightarrow \underline{R_5} \rightarrow \underline{R_6} \rightarrow \underline{R_4} \rightarrow \underline{R_7}$
- $R_2 \rightarrow R_5 \rightarrow R_6 \rightarrow \underline{R_3} \rightarrow \underline{R_9} \rightarrow R_7$

Table 7.11: Path lengths under the constraint that each commodity is secured by **6** security services

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | x | 6 | 2 | 5 | 5 | 3 |
| LOSA | 6 | x | 7 | 2 | 3 | 4 |
| NYCM | 2 | 7 | x | 6 | 6 | 2 |
| SNVA | 5 | 2 | 6 | x | 2 | 7 |
| STTL | 5 | 3 | 6 | 2 | x | 7 |
| WASH | 3 | 4 | 2 | 7 | 7 | x |

(a) *Late* and *Pre* strategy

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | x | 5 | 2 | 5 | 5 | 4 |
| LOSA | 6 | x | 6 | 2 | 3 | 4 |
| NYCM | 2 | 6 | x | 6 | 6 | 2 |
| SNVA | 5 | 2 | 6 | x | 2 | 5 |
| STTL | 5 | 3 | 6 | 2 | x | 6 |
| WASH | 3 | 4 | 7 | 5 | 6 | x |

(b) *SP* strategy

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | x | 7 | 2 | 5 | 5 | 3 |
| LOSA | 9 | x | 10 | 2 | 3 | 4 |
| NYCM | 5 | 5 | x | 6 | 6 | 10 |
| SNVA | 5 | 2 | 6 | x | 2 | 5 |
| STTL | 5 | 3 | 8 | 2 | x | 8 |
| WASH | 5 | 9 | 5 | 5 | 9 | x |

(c) *MP* strategy

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | x | 2 | 1 | 1 | 1 | 1 |
| LOSA | 2 | x | 2 | 1 | 1 | 1 |
| NYCM | 2 | 1 | x | 1 | 1 | 3 |
| SNVA | 1 | 1 | 1 | x | 1 | 1 |
| STTL | 1 | 1 | 2 | 1 | x | 3 |
| WASH | 2 | 3 | 2 | 1 | 3 | x |

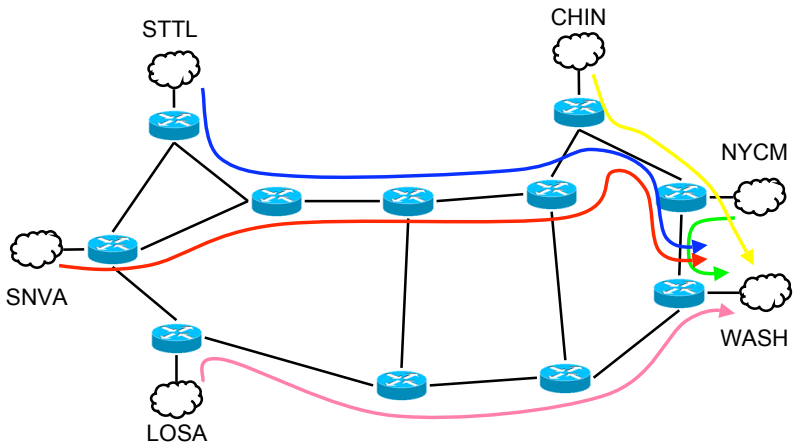
(d) *MP* strategy paths used

Accordingly, the commodity is routed over seven—to point them out they are underlined—unique nodes. Table 7.11(d) states how many paths are used to transmit the individual commodities.

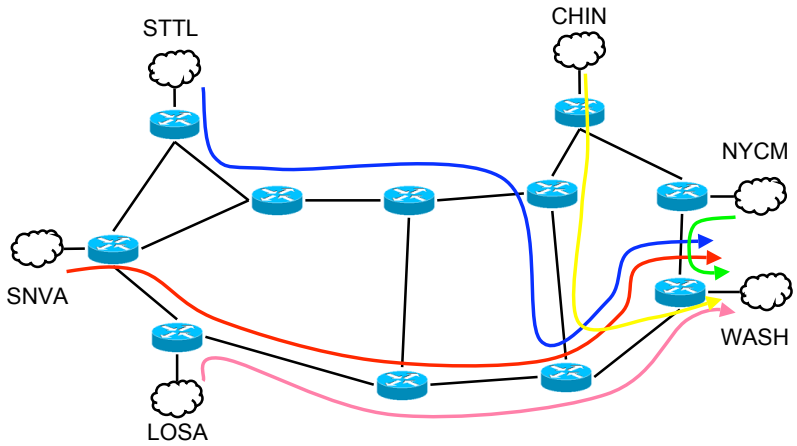
On average a commodity traverses 4.47 hops in case of the *Late* and *Pre* strategy. The singlepath routing strategy generates an average path length of 4.37 hops. 5.37 unique hops are passed on average by a flow in the multipath scenario. To specify the average number of hops per megabit routed the weighted average defined in Equation (7.2) is calculated for each strategy.

$$\frac{t(k)hops(k)}{\sum_{k\in D} t(k)} \tag{7.2}$$

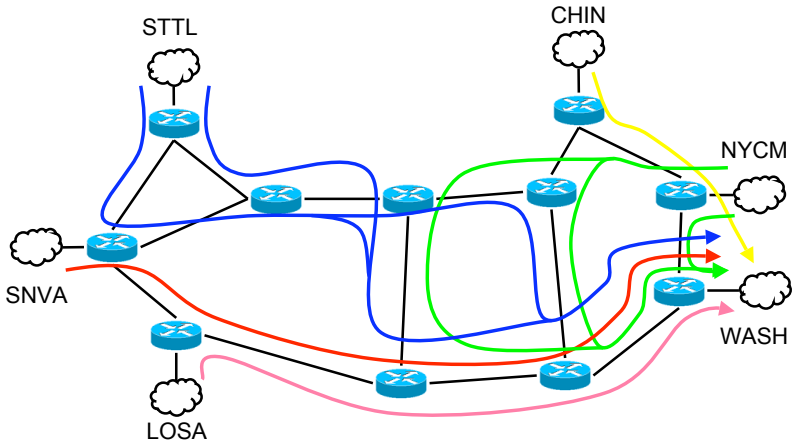
The outcome is that on average a megabit traverses 3.96, 4.50 and 6.11 hops for strategies *Late/Pre*, *SP* and *MP*. The load caused by the security services can better be balanced the higher the value. Combining the task of routing with the task of deploying security services increases the number of hops per megabit transmitted. The singlepath strategy achieves this by creating longer paths for large commodities and the multipath strategy split commodities into multiple smaller flows which are routed over different paths; referring to Table 7.11(d) the 30 commodities are divided into 45 flows. The effect of the strategies is illustrated in Figure 7.18 which shows the routes for the commodities destined for WASH for the strategies discussed. The shortest path routing does not consider the load due to the security services and additionally, it must be kept in mind that in case of the shortest path algorithm the calculated routes are symmetric. Hence, the resulting configuration of the network leaves little flexibility for the optimization of the service placement, especially for large commodities like the one from Wash to NYCM. In contrast the singlepath strategy routes the same demand over seven hops and the multipath strategy splits the commodity into two smaller flows.




(a) Predefined routing strategies *Late* and *Pre*




(b) Singlepath strategy *SP*




(c) MP strategy *MP*

Figure 7.18: Routes for commodities towards WASH

Table 7.12 shows the overall statistics for the scenario emulated. The first column represents for each strategy the measured relative packet-loss and the second column shows the packet loss caused by Fidran. Further, in the third column the average end-to-end-delay and the corresponding 95% confidence interval of a packet is given. Referring to the table it can be stated that doing intrusion prevention in the network increases packet loss and end-to-end-delay. Moreover, almost all packet loss is caused by Fidran but the benefit of the strategies *Pre*, *SP* and *MP* is visible. Although the resulting paths are longer from a link delay perspective, the strategies introduced perform better as packet loss and end-to-end-delays are reduced. Strategy *Late* causes 2.1% of all packets to be dropped and on average it takes a packet 34.75 ms to reach its destination. 1.4% of all packets are lost when applying the *Late* strategy and a packet loss of 1.1% is caused by strategies *SP* and *MP*. For all three optimal deployment strategies it can be said that on average packets reach their destination ten milliseconds earlier than in case of the *Late* strategy.

Table 7.13 and Table 7.14 compare the deployment strategies in more detail by specifying flow-specific drop-rates as well as flow-specific end-to-end-delays. Throughout the tables the three largest commodities (**WASH-NYCM**, **LOSA-CHIN** and **NYCM-WASH**) are written bold and the three smallest commodities (SNVA-WASH, SNVA-NYCM and SNVA-CHIN) are written underlined. Router $R_2$ is the bottleneck in context of the *Late* strategy. It inspects all traffic destined to CHIN and according to the traffic matrix 7.8 node CHIN receives the largest volume of network traffic of all subnets. This is illustrated in more detail by Figure 7.19. The figure on the left depicts the routers' packet processing rates and the figure on the right depicts the routers' packet drop rates. Router $R_2$ can be identified as bottleneck when employing strategy *Late* and its impact on the corresponding flow drop rates is given in Table 7.13(b). The drop-rates of the flows addressed to CHIN are significantly higher than for the remaining strategies. In addition, the bottleneck caused by router $R_2$ also affects the average end-to-end-delays of the corresponding flows as illustrated in Table 7.14. On average a packet needs about 53 ms to get from LOSA to CHIN in case of the *Late* strategy. When applying a more advanced security services deployment strategy, this delay can be reduced by over 20 ms.

Next, the number of flows with a packet loss higher than 2.5% are counted for each deployment strategy. For the *Late* strategy ten commodities were counted and three for the *Pre* strategy. However, it must be remarked that strategy *Pre* causes a packet loss of 8.1%—this is the highest drop rate measured in this scenario—for the traffic demand between

Table 7.12: Overall statistics for scenario Abilene under the constraint that each commodity must be secured by **6** security services

| Strategy | Loss [%] | Fidran Loss [%] | End-to-end-delay [ms] |
|---|---|---|---|
| No | 0 | 0 | 13.412 ±0.019 |
| Late | 2.100 | 2.100 | 34.756 ±0.059 |
| Pre | 1.400 | 1.300 | 24.803 ±0.038 |
| SP | 1.100 | 1.000 | 25.300 ±0.034 |
| MP | 1.100 | 1.000 | 23.665 ±0.032 |

Table 7.13: Flow specific packet loss [%] for scenario Abilene under the constraint that each commodity must be secured by **6** security services; the three largest/smallest commodities are emphasized/underlined

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0 | 0 | 0 | 0 | 0 |
| LOSA | **0** | X | 0 | 0 | 0 | 0 |
| NYCM | 0 | 0 | X | 0 | 0 | **0** |
| SNVA | 0.100 | 0.200 | 0 | X | 0.300 | 0 |
| STTL | 0 | 0.100 | 0 | 0 | X | 0 |
| WASH | 0.100 | 0 | **0.100** | 0.100 | 0 | X |

(a) Strategy *No*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 2.800 | 0.800 | 0 | 0 | 0.100 |
| LOSA | **2.700** | X | 1.900 | 0.300 | 0.700 | 1.400 |
| NYCM | 2.800 | 2.400 | X | 0 | 1.300 | **1.200** |
| SNVA | 3.600 | 4.300 | 3.100 | X | 1.000 | 0 |
| STTL | 2.500 | 2.400 | 1.100 | 1.100 | X | 0.600 |
| WASH | 3.600 | 2.800 | **2.900** | 0 | 0.900 | X |

(b) Strategy *Late*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.600 | 0.600 | 0 | 0 | 0.100 |
| LOSA | **0.700** | X | 0.600 | 0.700 | 0.300 | 1.100 |
| NYCM | 1.000 | 1.700 | X | 0.100 | 1.500 | **1.200** |
| SNVA | 0.400 | 3.600 | 1.100 | X | 1.000 | 0 |
| STTL | 0.300 | 2.100 | 8.100 | 1.100 | X | 4.300 |
| WASH | 0.700 | 2.000 | **2.300** | 0 | 0.800 | X |

(c) Strategy *Pre*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.900 | 0.200 | 0 | 0 | 0.100 |
| LOSA | **0.800** | X | 0.500 | 0.400 | 0.500 | 1.100 |
| NYCM | 0.600 | 2.000 | X | 0.200 | 1.200 | **0.800** |
| SNVA | 0 | 3.600 | 0.100 | X | 1.300 | 0 |
| STTL | 0.800 | 2.000 | 0 | 1.600 | X | 0.700 |
| WASH | 0.700 | 1.600 | **1.600** | 0 | 1.300 | X |

(d) Strategy *SP*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.300 | 0.200 | 0 | 0 | 0 |
| LOSA | **0.700** | X | 0.500 | 0.600 | 0.100 | 1.000 |
| NYCM | 0.500 | 1.800 | X | 0 | 1.300 | **1.000** |
| SNVA | 0.700 | 3.300 | 0.200 | X | 1.200 | 0 |
| STTL | 0.400 | 1.800 | 0.100 | 1.400 | X | 0.400 |
| WASH | 0.600 | 2.300 | **1.600** | 0.300 | 1.100 | X |

(e) Strategy *MP*

STTL and NYCM. Further, strategies *SP* and *MP* reduce the amount of flows with a drop rate higher than 2.5% to a single commodity. Next, the results given in Table 7.14 show that the proposed security service deployment strategies reduce the impact of doing intrusion prevention in the network on a packet's end-to-end-delay. Finally, Table A.1 in Appendix A stores for each commodity and for each strategy the flow specific absolute packet drop rate (95 % confidence interval). The table gives an impression of the relation between relative and absolute packet loss.

In this scenario the performance of the three optimal deployment strategies, in terms of packet loss and end-to-end-delay, is comparable. In the next section the number of requested security services is increased there each flow must be analyzed by all seven security services of Table 4.2.

Table 7.14: Flow specific end-to-end-delay [s] for scenario Abilene under the constraint that each commodity must be secured by **6** security services

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 20.68 ±0.01 | 5.59 ±0.01 | 17.54 ±0.04 | 18.49 ±0.03 | 7.96 ±0.01 |
| LOSA | **20.66 ±0.01** | X | 25.76 ±0.01 | 3.79 ±0.01 | 9.06 ±0.01 | 19.32 ±0.01 |
| NYCM | 5.81 ±0.00 | 25.71 ±0.01 | X | 22.77 ±0.03 | 23.12 ±0.01 | **2.55 ±0.00** |
| SNVA | 17.65 ±0.03 | 3.93 ±0.01 | 22.68 ±0.05 | X | 6.09 ±0.01 | 25.21 ±0.09 |
| STTL | 19.28 ±0.01 | 9.27 ±0.02 | 24.29 ±0.02 | 6.18 ±0.03 | X | 26.25 ±0.02 |
| WASH | 8.84 ±0.01 | 19.91 ±0.01 | **3.59 ±0.01** | 25.56 ±0.04 | 26.35 ±0.02 | X |

(a) Strategy *No*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 37.57 ±0.33 | 27.85 ±0.70 | 22.72 ±0.31 | 28.02 ±0.46 | 20.54 ±0.36 |
| LOSA | **53.19 ±0.26** | X | 51.40 ±0.31 | 9.56 ±0.42 | 20.87 ±0.35 | 32.26 ±0.22 |
| NYCM | 37.84 ±0.34 | 44.11 ±0.21 | X | 28.26 ±0.17 | 33.36 ±0.14 | **14.18 ±0.10** |
| SNVA | 48.78 ±1.04 | 21.27 ±0.24 | 50.31 ±1.41 | X | 18.98 ±0.20 | 36.67 ±0.91 |
| STTL | 53.92 ±0.35 | 26.84 ±0.28 | 49.84 ±0.51 | 11.65 ±0.32 | X | 39.47 ±0.25 |
| WASH | 44.06 ±0.27 | 38.94 ±0.20 | **37.39 ±0.18** | 33.82 ±0.29 | 39.71 ±0.15 | X |

(b) Strategy *Late*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 30.22 ±0.20 | 16.17 ±0.43 | 25.25 ±0.56 | 26.08 ±0.35 | 24.73 ±0.48 |
| LOSA | **27.89 ±0.10** | X | 35.16 ±0.15 | 10.55 ±0.41 | 19.61 ±0.28 | 27.82 ±0.17 |
| NYCM | 17.33 ±0.18 | 33.94 ±0.13 | X | 30.11 ±0.29 | 30.88 ±0.11 | **10.11 ±0.07** |
| SNVA | 29.50 ±0.46 | 13.16 ±0.14 | 29.19 ±0.47 | X | 17.75 ±0.17 | 31.10 ±0.54 |
| STTL | 28.12 ±0.14 | 18.22 ±0.19 | 32.96 ±0.27 | 12.91 ±0.35 | X | 33.59 ±0.18 |
| WASH | 23.48 ±0.15 | 30.41 ±0.12 | **29.06 ±0.15** | 35.47 ±0.34 | 37.41 ±0.12 | X |

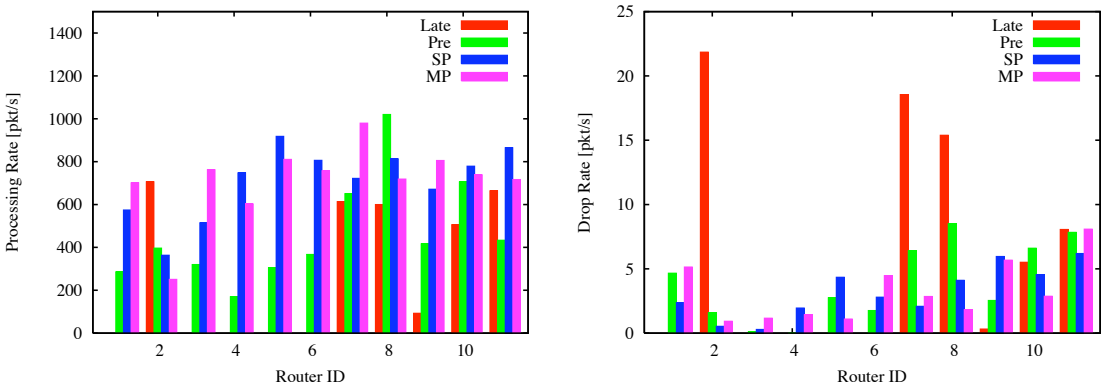
(c) Strategy *Pre*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 28.56 ±0.19 | 10.14 ±0.22 | 25.45 ±0.47 | 23.04 ±0.17 | 16.87 ±0.20 |
| LOSA | **30.53 ±0.10** | X | 30.51 ±0.11 | 10.49 ±0.40 | 18.51 ±0.28 | 25.86 ±0.13 |
| NYCM | 12.77 ±0.12 | 33.54 ±0.11 | X | 29.69 ±0.24 | 29.72 ±0.09 | **8.64 ±0.06** |
| SNVA | 25.21 ±0.34 | 13.90 ±0.14 | 28.98 ±0.37 | X | 17.23 ±0.15 | 26.39 ±0.33 |
| STTL | 32.89 ±0.15 | 19.75 ±0.18 | 31.94 ±0.18 | 15.48 ±0.31 | X | 35.22 ±0.20 |
| WASH | 17.30 ±0.09 | 30.39 ±0.12 | **42.02 ±0.10** | 30.65 ±0.31 | 33.27 ±0.11 | X |

(d) Strategy *SP*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 28.04 ±0.16 | 11.63 ±0.30 | 24.58 ±0.45 | 24.74 ±0.25 | 14.52 ±0.25 |
| LOSA | **32.11 ±0.09** | X | 28.86 ±0.11 | 8.84 ±0.33 | 16.46 ±0.26 | 27.12 ±0.13 |
| NYCM | 15.96 ±0.13 | 28.69 ±0.12 | X | 28.94 ±0.22 | 30.45 ±0.11 | **14.44 ±0.08** |
| SNVA | 27.68 ±0.36 | 11.71 ±0.10 | 28.44 ±0.37 | X | 15.47 ±0.14 | 28.22 ±0.67 |
| STTL | 32.17 ±0.15 | 18.90 ±0.16 | 34.64 ±0.17 | 13.05 ±0.29 | X | 37.43 ±0.17 |
| WASH | 19.89 ±0.12 | 37.25 ±0.11 | **19.06 ±0.09** | 30.31 ±0.29 | 34.34 ±0.10 | X |

(e) Strategy *MP*




(a) Packet processing rates

(b) Packet drop rates

Figure 7.19: Router statistics for scenario Abilene under the constraint that each commodity must be secured by **6** security services

### 7.4.2 Abilene Network: Securing each Commodity by Seven Security Services

The scenario emulated in this section is identical to the one discussed previously, but the fact that each commodity must now be analyzed by seven security services. In the following the scenario is analogously analyzed as the previous.

Table 7.15 shows for each emulated strategy the number of service assignments per router. The corresponding paths lengths are given in Table 7.16 and Table 7.16(b) reflects how many paths are used by strategy *MP* to route a specific commodity. The average path length remains 4.47 hops for strategies *Late* and *Pre*. Strategy *SP* generates routes of an average length of 4.20 hops and strategy *MP* requires 4.97 hops per commodity. However when calculating the average number of hops that each megabit traverses as defined in Equation (7.2), the outcome is 3.96 for strategies *Late* and *Pre*. The singlepath and multipath deployment strategies achieve average values of 4.09 and 5.05. Comparing them to those that were achieved in the previous scenario (4.50 for strategy *SP* and 6.11 for strategy *MP*) it can be expected that the impact of all strategies on the network performance increases.

The assumption is evidenced by Table 7.17 which represents the overall statistics for this scenario in terms of the packet loss, the packet loss caused by FIDRAN and the average end-to-end-delay of all packets. By increasing the protection requirements from six services to seven, packet loss rates and end-to-end-delays increase:

- *Late*: packet loss from 2.1 % to 3 % and end-to-end-delay from 34.76 ms to 42.13 ms,
- *Pre* from 1.4 % to 2.4 % and end-to-end-delay from 24.80 ms to 30.15 ms,
- *SP* from 1.1 % to 2.1 % and end-to-end-delay from 25.30 ms to 27.38 ms,
- *MP* from 1.1 % to 1.4 % and end-to-end-delay from 23.67 ms to 25.73 ms.

Table 7.15: Service assignments under the constraint that each commodity must be analyzed by **7** security services

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| $R_1$ | x | x | x | x | x | x |
| $R_2$ | 35 | x | x | x | x | x |
| $R_3$ | x | x | x | x | x | x |
| $R_4$ | x | x | x | x | x | x |
| $R_5$ | x | x | x | x | x | x |
| $R_6$ | x | x | x | x | x | x |
| $R_7$ | x | 35 | x | x | x | x |
| $R_8$ | x | x | 35 | x | x | x |
| $R_9$ | x | x | x | 35 | x | x |
| $R_{10}$ | x | x | x | x | 35 | x |
| $R_{11}$ | x | x | x | x | x | 35 |

(a) *Late* strategy

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| $R_1$ | x | 5 | x | x | x | 7 |
| $R_2$ | 11 | x | 6 | x | x | 5 |
| $R_3$ | 1 | x | 1 | x | 1 | x |
| $R_4$ | x | 2 | x | x | x | x |
| $R_5$ | 8 | x | 1 | 1 | 1 | x |
| $R_6$ | 9 | x | 5 | 12 | 5 | 1 |
| $R_7$ | 2 | 12 | 4 | 1 | x | x |
| $R_8$ | 1 | x | 3 | x | x | 7 |
| $R_9$ | x | 9 | 4 | 21 | 7 | 7 |
| $R_{10}$ | 1 | 7 | 6 | x | 21 | 6 |
| $R_{11}$ | 2 | x | 5 | x | x | 2 |

(b) *Pre* strategy

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| $R_1$ | 2 | 3 | 2 | 1 | x | 2 |
| $R_2$ | 6 | x | 6 | x | x | 1 |
| $R_3$ | 10 | x | 1 | 3 | 11 | 3 |
| $R_4$ | x | 8 | 1 | 1 | x | 6 |
| $R_5$ | 5 | 1 | 1 | 1 | 2 | x |
| $R_6$ | 5 | 2 | 3 | 1 | 3 | 1 |
| $R_7$ | 1 | 6 | 5 | x | x | 3 |
| $R_8$ | 1 | x | 6 | x | x | 3 |
| $R_9$ | 3 | 11 | 6 | 26 | 6 | 3 |
| $R_{10}$ | 1 | 3 | 4 | 2 | 13 | 2 |
| $R_{11}$ | 1 | 1 | x | x | x | 11 |

(c) *SP* strategy

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| $R_1$ | 3.05 | 1.00 | 0.10 | x | 0.29 | 3.17 |
| $R_2$ | 6.65 | x | 6.00 | x | x | 6.00 |
| $R_3$ | 8.63 | 2.20 | 7.36 | 1.00 | 5.73 | 0.06 |
| $R_4$ | 0.75 | 4.05 | 2.55 | 1.00 | 0.76 | 4.00 |
| $R_5$ | 4.67 | 3.26 | 2.04 | 1.00 | 2.00 | 0.49 |
| $R_6$ | 2.25 | 4.28 | 2.97 | 4.00 | 3.00 | 2.33 |
| $R_7$ | 1.37 | 7.11 | 2.70 | x | 0.47 | 3.00 |
| $R_8$ | x | x | 2.90 | x | x | 1.53 |
| $R_9$ | 1.63 | 5.88 | 0.45 | 17.00 | 3.81 | 5.02 |
| $R_{10}$ | 1.00 | 2.22 | 0.03 | 6.00 | 13.94 | 2.87 |
| $R_{11}$ | x | x | 2.90 | x | x | 1.53 |

(d) *MP* strategy

Table 7.16: Path lengths for the optimal deployment strategies under the constraint that each commodity must be secured by **7** security services

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | x | 5 | 2 | 5 | 5 | 3 |
| LOSA | 5 | x | 5 | 2 | 3 | 4 |
| NYCM | 2 | 5 | x | 6 | 6 | 2 |
| SNVA | 5 | 2 | 6 | x | 2 | 5 |
| STTL | 5 | 3 | 6 | 2 | x | 6 |
| WASH | 4 | 4 | 5 | 5 | 6 | x |

(a) *SP* strategy

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | x | 5 | 2 | 5 | 5 | 4 |
| LOSA | 5 | x | 7 | 2 | 3 | 6 |
| NYCM | 5 | 10 | x | 7 | 11 | 2 |
| SNVA | 6 | 2 | 6 | x | 2 | 5 |
| STTL | 5 | 3 | 6 | 2 | x | 6 |
| WASH | 5 | 6 | 5 | 5 | 6 | x |

(b) *MP* strategy

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | x | 1 | 1 | 1 | 1 | 1 |
| LOSA | 1 | x | 1 | 1 | 1 | 2 |
| NYCM | 2 | 3 | x | 1 | 2 | 1 |
| SNVA | 2 | 1 | 1 | x | 1 | 1 |
| STTL | 1 | 1 | 1 | 1 | x | 1 |
| WASH | 2 | 2 | 2 | 1 | 1 | x |

(c) *MP* strategy: number of paths used to route commodity

Strategies *Late* and *Pre* suffer most under the requirement that each commodity must be additionally inspected by a seventh security service. When employing the *Late* strategy 0.9 % more packets are dropped and the average end-to-end-delay increases by 7.37 ms. Also strategy *Pre* cannot cope with the increased security requirements. Packet loss increases by 1 % and on average it takes a packet 5.35 ms longer to reach its destination. However the singlepath strategy drops 1 % more packets but the average end-to-end-delay increases only by 2.08 ms. Finally, strategy *MP* deals best with the increased security requirements as packet loss increases and delay increase by 0.3 % and 2.06 ms respectively.

Next, the flow specific packet losses given in Table 7.13 are examined and again the commodities are counted for which a packet loss higher than 2.5 % was measured. This is the case for nine commodities when deploying the security services in conformity to strategy *Late*. The highest packet drop rate of 7.80 % is observed for the traffic demand from NYCM to

Table 7.17: Overall statistics the Abilene scneario under the constraint that each commodity must be secured by **7** security services

| Strategy | Loss [%] | Fidran Loss [%] | End-to-end-delay [ms] |
|---|---|---|---|
| *No* | 0 | 0 | 13.208 ±0.019 |
| *Late* | 3.000 | 3.000 | 42.130 ±0.071 |
| *Pre* | 2.400 | 2.400 | 30.151 ±0.044 |
| *SP* | 2.100 | 2.100 | 27.379 ±0.036 |
| *MP* | 1.400 | 1.400 | 25.733 ±0.036 |

Table 7.18: Flow specific packet losses [%] for scenario Abilene under the constraint that each commodity must be secured by **7** security services; the three largest/smallest commodities are emphasized/underlined

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0 | 0 | 0 | 0 | 0 |
| LOSA | **0** | X | 0 | 0 | 0 | 0 |
| NYCM | 0.100 | 0 | X | 0 | 0 | **0** |
| SNVA | 0 | 0 | 0 | X | 0 | 0 |
| STTL | 0 | 0 | 0 | 0 | X | 0 |
| WASH | 0.100 | 0.100 | **0.100** | 0 | 0 | X |

(a) Strategy *No*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.200 | 0.900 | 0 | 0 | 0.500 |
| LOSA | **5.700** | X | 0.500 | 0.300 | 3.300 | 3.100 |
| NYCM | 7.800 | 1.200 | X | 0 | 1.500 | **2.900** |
| SNVA | 1.700 | 3.700 | 2.400 | X | 1.200 | 0 |
| STTL | 4.000 | 0.800 | 0.800 | 1.200 | X | 0.100 |
| WASH | 5.400 | 2.200 | **3.100** | 0 | 0.700 | X |

(b) Strategy *Late*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0.800 | 0.900 | 0 | 0 | 1.400 |
| LOSA | **2.200** | X | 0.200 | 0.100 | 3.400 | 2.500 |
| NYCM | 4.100 | 1.200 | X | 0 | 1.400 | **4.700** |
| SNVA | 0 | 3.500 | 0 | X | 1.200 | 0 |
| STTL | 0.200 | 0.100 | 0.200 | 1.300 | X | 0 |
| WASH | 2.600 | 1.700 | **4.200** | 1.000 | 0.700 | X |

(c) Strategy *Pre*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0.400 | 0.100 | 0 | 0 | 0.100 |
| LOSA | **2.000** | X | 0.400 | 0.400 | 3.400 | 2.700 |
| NYCM | 2.400 | 1.300 | X | 0.100 | 1.100 | **2.600** |
| SNVA | 0.400 | 3.800 | 0.200 | X | 1.600 | 0 |
| STTL | 0.300 | 0.200 | 0.100 | 1.000 | X | 0 |
| WASH | 3.500 | 4.500 | **3.100** | 2.300 | 0.600 | X |

(d) Strategy *SP*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0.700 | 0.600 | 0 | 0 | 0.100 |
| LOSA | **2.000** | X | 0 | 0.300 | 2.800 | 2.400 |
| NYCM | 2.500 | 0.800 | X | 0 | 1.200 | **2.300** |
| SNVA | 0 | 3.500 | 2.600 | X | 0.800 | 0 |
| STTL | 0.200 | 0 | 0.200 | 1.000 | X | 0 |
| WASH | 0.600 | 1.400 | **1.700** | 0.300 | 1.100 | X |

(e) Strategy *MP*

CHIN. When choosing strategy *Pre* still seven flows have a packet drop rate higher than the threshold value specified. This time the highest packet loss rate of 4.70 % is observed for the traffic demand between NYCM and WASH. Also seven flows were counted when applying the singlepath strategy. Here the maximum drop rate of 4.50 % is monitored for the flow from WASH to LOSA. Finally, only three commodities are counted for strategy *MP* and the highest packet loss rate of 3.50 % was measured for the flow from SNVA to LOSA.

Table A.2 in Appendix A stores for each commodity and for each strategy the flow specific absolute packet drop rate (95 % confidence interval). The table gives an impression of the relation between relative and absolute packet loss.

Figure 7.20 depicts for each emulated strategy the router statistics in terms of packet processing rate and packet dropping rate. According to Figure 7.20(b) and analogously to the preceding scenario router $R_2$ is the bottleneck when choosing strategy *Late*. This correlates to the results shown in Table 7.18(b) as for all flows destined to CHIN high packet loss rates were observed. Switching to strategy *Pre*, there routers $R_8$ and $R_{11}$ attract the attention. According to Table 7.15(b) router $R_8$ runs 11 security services and router $R_{11}$ nine. Both routers are assigned to inspect the commodities between NYCM and WASH (both directions)

Table 7.19: Flow specific end-to-end-delay [ms] for scenario Abilene under the constraint that each commodity must be secured by **7** security services

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 20.50 ±0.01 | 5.58 ±0.01 | 17.52 ±0.04 | 18.46 ±0.03 | 7.92 ±0.01 |
| LOSA | **20.68 ±0.01** | X | 26.08 ±0.02 | 3.77 ±0.01 | 9.07 ±0.01 | 19.11 ±0.01 |
| NYCM | 6.51 ±0.01 | 26.86 ±0.02 | X | 23.59 ±0.04 | 24.41 ±0.02 | **3.23 ±0.01** |
| SNVA | 17.45 ±0.03 | 3.91 ±0.01 | 22.78 ±0.05 | X | 6.09 ±0.01 | 25.25 ±0.09 |
| STTL | 18.46 ±0.01 | 9.03 ±0.01 | 23.54 ±0.02 | 5.76 ±0.01 | X | 25.46 ±0.02 |
| WASH | 8.91 ±0.01 | 20.05 ±0.01 | **3.66 ±0.01** | 25.66 ±0.04 | 26.73 ±0.02 | X |

(a) Strategy *No*

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 41.67 ±0.32 | 33.26 ±0.79 | 23.21 ±0.35 | 27.76 ±0.38 | 26.40 ±0.46 |
| LOSA | **63.85 ±0.34** | X | 54.87 ±0.37 | 10.13 ±0.45 | 20.04 ±0.28 | 38.19 ±0.24 |
| NYCM | 55.47 ±0.40 | 51.11 ±0.24 | X | 30.19 ±0.18 | 37.39 ±0.19 | **29.95 ±0.15** |
| SNVA | 53.87 ±1.21 | 24.65 ±0.26 | 48.45 ±1.25 | X | 19.68 ±0.20 | 42.62 ±1.29 |
| STTL | 61.31 ±0.42 | 28.45 ±0.39 | 51.24 ±0.49 | 11.54 ±0.32 | X | 41.90 ±0.45 |
| WASH | 55.95 ±0.33 | 43.10 ±0.23 | **40.60 ±0.19** | 34.73 ±0.33 | 41.22 ±0.21 | X |

(b) Strategy *Late*

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 30.44 ±0.19 | 23.27 ±0.54 | 22.85 ±0.30 | 25.93 ±0.35 | 27.75 ±0.46 |
| LOSA | **30.62 ±0.11** | X | 39.28 ±0.21 | 16.22 ±0.57 | 18.29 ±0.25 | 28.31 ±0.16 |
| NYCM | 33.46 ±0.24 | 44.04 ±0.18 | X | 32.30 ±0.31 | 33.33 ±0.15 | **29.59 ±0.14** |
| SNVA | 22.04 ±0.23 | 21.33 ±0.20 | 29.91 ±0.47 | X | 20.03 ±0.20 | 31.36 ±0.56 |
| STTL | 27.98 ±0.16 | 18.41 ±0.27 | 31.47 ±0.20 | 14.24 ±0.39 | X | 33.21 ±0.25 |
| WASH | 31.97 ±0.18 | 31.65 ±0.14 | **29.90 ±0.15** | 36.66 ±0.39 | 37.73 ±0.17 | X |

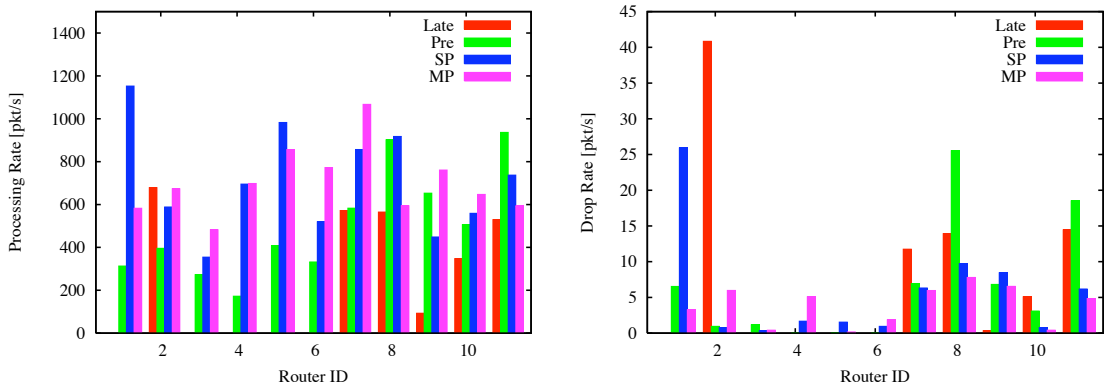
(c) Strategy *Pre*

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 27.30 ±0.15 | 11.26 ±0.23 | 24.86 ±0.43 | 25.24 ±0.29 | 17.49 ±0.28 |
| LOSA | **26.75 ±0.11** | X | 31.39 ±0.14 | 11.29 ±0.47 | 17.86 ±0.24 | 30.07 ±0.14 |
| NYCM | 18.17 ±0.16 | 34.14 ±0.13 | X | 32.27 ±0.35 | 31.59 ±0.12 | **20.59 ±0.11** |
| SNVA | 24.14 ±0.38 | 15.54 ±0.16 | 29.42 ±0.51 | X | 19.28 ±0.18 | 33.50 ±0.89 |
| STTL | 26.44 ±0.15 | 17.56 ±0.25 | 30.33 ±0.17 | 13.99 ±0.41 | X | 34.81 ±0.28 |
| WASH | 30.77 ±0.14 | 41.88 ±0.16 | **32.59 ±0.09** | 40.92 ±0.42 | 33.34 ±0.14 | X |

(d) Strategy *SP*

| To → | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 29.07 ±0.18 | 13.69 ±0.34 | 23.66 ±0.33 | 25.06 ±0.23 | 19.33 ±0.28 |
| LOSA | **29.75 ±0.12** | X | 33.58 ±0.11 | 9.95 ±0.45 | 15.86 ±0.19 | 28.93 ±0.13 |
| NYCM | 22.41 ±0.19 | 37.88 ±0.12 | X | 41.83 ±0.36 | 37.19 ±0.14 | **19.72 ±0.11** |
| SNVA | 26.41 ±0.34 | 12.53 ±0.12 | 32.63 ±0.54 | X | 15.36 ±0.13 | 30.96 ±0.88 |
| STTL | 25.66 ±0.10 | 16.05 ±0.17 | 30.89 ±0.16 | 11.96 ±0.31 | X | 31.04 ±0.21 |
| WASH | 23.83 ±0.11 | 31.66 ±0.11 | **23.83 ±0.11** | 40.51 ±0.42 | 39.44 ±0.15 | X |

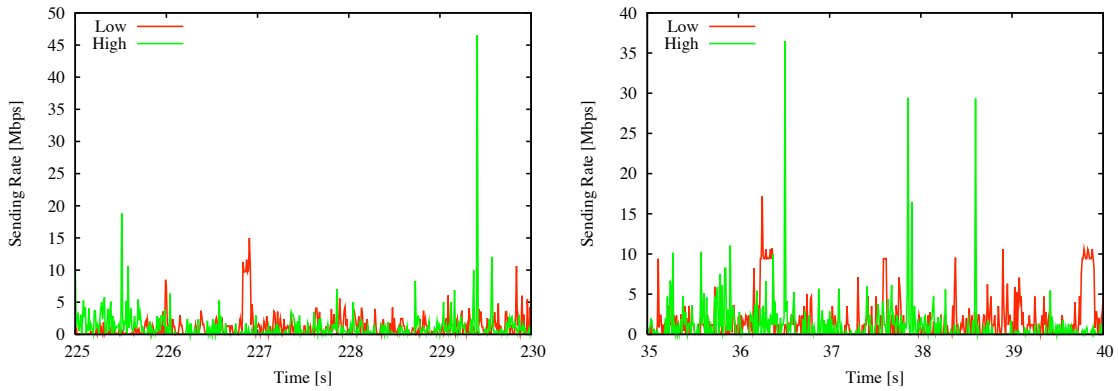
(e) Strategy *MP*




(a) Packet processing rates (b) Packet drop rates

Figure 7.20: Router statistics for scenario Abilene under the constraint that each commodity must be secured by **7** security services

as well as the commodities between CHIN and WASH (both directions) and referring to Table 7.18(c) many packets of these flows get dropped. Consequently, they cause the bigger part of all packet loss. When employing strategy *SP* router $R_1$ becomes overloaded. The router runs ten security services which it provides to seven different commodities. But overall the strategy reduces the packet loss. Finally, the workload that is caused by doing intrusion prevention is balanced out very well by strategy *MP*.




(a) Commodity from WASH to NYCM (b) Commodity from LOSA to CHIN

Figure 7.21: The impact of the bandwidth on the traffic generation process

Table 7.20: Measured sending-rates of the Abilene nodes

| | High | | | Low | | |
|---|---|---|---|---|---|---|
| Node | Mean [bit/ms] | Standard deviation | 95 % c.i. | Mean [bit/ms] | Standard deviation | 95 % c.i. |
| CHIN | 656 | 4550 | 16.48 | 565 | 2386 | 8.64 |
| LOSA | 2379 | 8451 | 30.61 | 2380 | 4849 | 17.56 |
| NYCM | 2697 | 8480 | 30.72 | 2700 | 5066 | 18.35 |
| SNVA | 933 | 8257 | 29.91 | 936 | 7361 | 26.66 |
| STTL | 1369 | 11175 | 40.48 | 1384 | 10113 | 36.63 |
| WASH | 3626 | 19599 | 70.99 | 3609 | 18617 | 67.44 |

(a) Scenario with 6 security services

| | High | | | Low | | |
|---|---|---|---|---|---|---|
| Node | Mean [bit/ms] | Standard deviation | 95 % c.i. | Mean [bit/ms] | Standard deviation | 95 % c.i. |
| CHIN | 665 | 4131 | 14.96 | 609 | 2462 | 8.92 |
| LOSA | 2300 | 8756 | 31.72 | 2335 | 4663 | 16.89 |
| NYCM | 2634 | 16585 | 60.07 | 2637 | 15301 | 55.42 |
| SNVA | 928 | 8044 | 29.14 | 929 | 7348 | 26.62 |
| STTL | 1271 | 6266 | 22.70 | 1318 | 3529 | 12.79 |
| WASH | 3779 | 20556 | 74.46 | 3472 | 15335 | 55.55 |

(b) Scenario with **7** security services

## 7.5 Self-Similar Network Traffic of Smaller Bandwidth

In this section the Abilene-scenarios of the previous section—each commodity must be analyzed by six and seven security services respectively—are re-emulated with another set of traffic traces. So far the Abilene network was emulated on a scale of 1:100 (see Section 7.4) which means that a link of bandwidth 9.92 Gbps was emulated by a link of bandwidth 99.2 Mbps. Further, the traffic generation process described in Section 7.2 regards the link capacities that are defined for the Abilene network. To limit the size of the packet bursts while considering the traffic rates defined in Table 7.8 the link capacities were downscaled by a factor of 10 throughout the network traffic generation process. In the following the traffic traces that were originally generated are indicated *high* and the set of traces generated while considering smaller link capacities is labelled *low*. The effect is depicted for two exemplary commodities in Figure 7.21. Figure 7.21(a) showing the sending rates of node WASH towards node NYCM. The green curve represents the *high* network trace and the red one is the outcome of the modified traffic generation process. Figure 7.21(b) depicts the analogous curves for the commodity from LOSA to CHIN. Besides the traffic generation process, the emulation setup of the Abilene network was not any further modified.

Using the lower link capacities for the traffic generation results in flows with smaller packet bursts as depicted for the two exemplary traffic demands. In addition, Table 7.20 compares the sending rates that were measured for each node throughout the emulations. The nodes' mean sending-rates are for traffic scenarios high and low about the same size but standard deviations as well as confidence intervals are smaller for the latter and accordingly, there the traffic streams are more steadily.

Table 7.21: Overall statistics for scenario Abilene under the constraint that each commodity must be secured by **6** security services for both types of generated self-similar traffic

| | High | | | Low | | |
|---|---|---|---|---|---|---|
| Strategy | Loss [%] | FIDRAN Loss [%] | End-to-end-delay [ms] | Loss [%] | FIDRAN Loss [%] | End-to-end-delay [ms] |
| No | 0 | 0 | 13.412 ±0.019 | 0 | 0 | 13.163 ±0.019 |
| Late | 2.100 | 2.100 | 34.756 ±0.059 | 1.900 | 1.900 | 33.837 ±0.059 |
| Pre | 1.400 | 1.300 | 24.803 ±0.038 | 0.900 | 0.800 | 23.341 ±0.035 |
| SP | 1.100 | 1.000 | 25.300 ±0.034 | 0.600 | 0.600 | 24.469 ±0.033 |
| MP | 1.100 | 1.000 | 23.665 ±0.032 | 0.600 | 0.600 | 22.565 ±0.030 |

### 7.5.1 Protecting each Commodity with Six Security Services

Initially, the scenario where each traffic demand is analyzed by six security services was re-emulated using the traffic traces generated in the preceding section. The emulation setup—routes as well as service deployment—was identical as described in Section 7.4.1. The overall statistics for this scenario and for both sets of traffic traces are given in Table 7.21. The results labeled *high* are the same as in Table 7.12.

A finding is that packet loss is reduced when generating the network traffic in conformity to the *low* traffic traces (see Table 7.21). The reason for this is that packets bursts cause the FIDRAN queues to fill up and consequently, the larger a packet burst the more packets will be dropped. The effect is illustrated for strategy *SP* and the commodity from WASH to NYCM in Figure 7.22. The left figure depicts the sending rate—the red curve—of node WASH as well as the reception rate—green curve—of LOSA over the time for the *high* traffic traces. In addition, the blue curve shows the number of packets that are dropped (y-axis on the right) over the time. The right figure depicts the corresponding curves for the *low* traffic traces. Packets are either delayed—as they have to wait to be served—or dropped—as the queues are already filled—when the green and red curve do not match each other. The figures show that the fitting between the red and the green curve is better when using the *low* traffic traces and consequently, the packet loss is smaller in that case. Table 7.22 represents the flow specific packet drop rates.

In the next section the scenario with seven security services is re-emulated.

### 7.5.2 Protecting each Commodity with Seven Security Services

Analogously to the preceding section the scenario described in Section 7.4.2 at which each commodity must be analyzed by seven security services, is re-emulated using the traffic traces described above. Average packet loss and end-to-end-delay are given for both traffic scenarios in Table 7.23. This time, both packet loss and end-to-end-delay increase tremendously when using the *low* traffic traces for the emulation of the scenario.

Table 7.24 points to the bottleneck as it stores the flow specific packet losses for each security service deployment strategy. First of all for each strategy the highest packet loss rate is observed for the commodity from WASH to NYCM and further, for each deployment strategy the following routers can be identified as bottleneck:

- *Late*: router $R_8$ causes over 80 % of all packet loss,

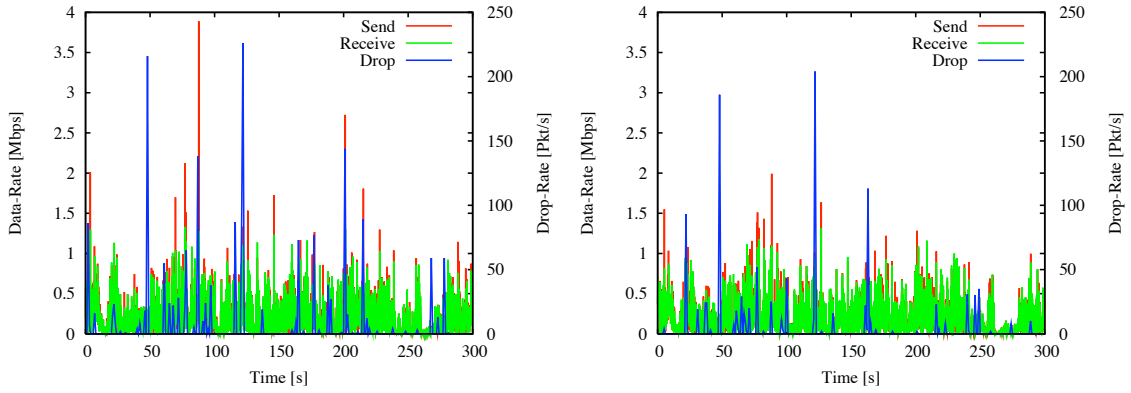
- *Pre*: routers $R_8$ and $R_{11}$ cause over 90 % of all packet loss,
- *SP*: router $R_1$ causes over 90 % of all packet loss,
- *MP*: router $R_{11}$ causes over 90 % of all packet loss.

The implications of a bottleneck router can be described as follows. For example, when employing strategy *SP* the bottleneck router $R_1$ is configured to serve seven commodities. These can easily be identified in Table 7.24(d) as they all have packet loss rates larger than 5 %. Furthermore, all routers that were identified as bottleneck, except router $R_{11}$ for strategy *Pre*, have in common that they are running security service 1 for the traffic demand from WASH to NYCM. The commodity is analyzed in more detail in Figure 7.23. There, sending-, reception- and drop-rates are depicted for both traffic traces and strategy *SP*. The difference between sending and reception rate is clearly higher for the *low* traffic scenario. As a consequence, also the resulting packet loss is tremendously higher.

Recapitulating it can be stated that an outcome of the modified traffic generation process is the emergence of at least one router that is overloaded independent of the security service deployment strategy applied. The reason for this are the reduced variances of the sending rates which leads to more constant traffic flows and which eliminates periods of times of low traffic volumes. Hence, the queues of the bottleneck routers are constantly filled resulting in high drop rates.

## 7.6 Time Required to Calculate Optimal Deployment Strategies

Table 7.25 shows the time needed to calculate the optimal distribution of the security services for the Abilene scenarios. All deployment strategies were solved to optimality on a Pentium IV 3.2 GHz machine using the CPLEX optimization software version 10.1.1. According to the




(a) High (b) Low

Figure 7.22: Sending and reception rate as well as packet loss rate of the traffic demand from WASH to NYCM, the commodity is secured by **6** security services and strategy *SP* is applied

Table 7.22: Flow specific packet losses [%] for scenario Abilene under the constraint that each commodity must be secured by **6** security services/using the *low* traffic traces; the three largest/smallest commodities are emphasized/underlined

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0 | 0 | 0 | 0 | 0 |
| LOSA | **0** | X | 0 | 0 | 0 | 0 |
| NYCM | 0 | 0 | X | 0 | 0 | **0** |
| SNVA | 0.200 | 0.200 | 0 | X | 0.200 | 0 |
| STTL | 0 | 0 | 0 | 0 | X | 0 |
| WASH | 0.200 | 0 | **0.100** | 0 | 0.100 | X |

(a) Strategy *No*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0.400 | 0.200 | 0.300 | 0 | 3.200 |
| LOSA | **3.000** | X | 1.700 | 0 | 0 | 1.500 |
| NYCM | 2.000 | 1.200 | X | 0 | 1.400 | **1.000** |
| SNVA | 5.200 | 2.900 | 0.700 | X | 1.200 | 0.100 |
| STTL | 4.600 | 0.400 | 1.300 | 0.500 | X | 0.800 |
| WASH | 4.500 | 1.200 | **2.600** | 0 | 0.400 | X |

(b) Strategy *Late*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0.100 | 0 | 0.400 | 0.100 | 0.800 |
| LOSA | **0.300** | X | 0.700 | 0.100 | 0 | 0.700 |
| NYCM | 0.300 | 0.300 | X | 0.100 | 1.300 | **0.700** |
| SNVA | 0.200 | 2.800 | 0 | X | 1.100 | 0 |
| STTL | 0.400 | 0.400 | 2.300 | 1.100 | X | 3.700 |
| WASH | 1.100 | 0.300 | **1.700** | 0.100 | 0.400 | X |

(c) Strategy *Pre*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0 | 0 | 0.700 | 0 | 0.400 |
| LOSA | **0.500** | X | 0.200 | 0 | 0 | 0.500 |
| NYCM | 0.300 | 0.100 | X | 0.300 | 1.200 | **0.500** |
| SNVA | 0.400 | 2.500 | 0 | X | 0.900 | 0 |
| STTL | 0.900 | 0.300 | 0 | 0.900 | X | 0.200 |
| WASH | 1.000 | 0.600 | **1.000** | 0.100 | 0.700 | X |

(d) Strategy *SP*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0 | 0.100 | 0.500 | 0 | 1.900 |
| LOSA | **0.400** | X | 0.300 | 0 | 0 | 0.700 |
| NYCM | 0.100 | 0.200 | X | 0.600 | 1.300 | **0.700** |
| SNVA | 0.200 | 2.100 | 0 | X | 0.800 | 0 |
| STTL | 0.700 | 0.300 | 0 | 1.300 | X | 0.200 |
| WASH | 0.900 | 0.600 | **1.200** | 0.100 | 0.600 | X |

(e) Strategy *MP*

Table the solution times for both Abilene scenarios are about the same size. Strategy *Pre* is specified to optimality in less than 10 s whereas Strategy *SP* requires already 700 s and specifying the multipath strategy takes about 2700 s. But it must be considered that the bigger part of the solution time is required to improve good solutions to optimality. In context of the *SP* strategy, the solutions found after 60 s have a gap of about 1.5 % to optimality and the solutions provided after 1000 s for strategy *MP* have a gap of about 5 % to optimality.

In consideration of strategy *MP*, the solution time strongly depends on the number of paths provided to the problem. For example, the presented optimal solutions were calculated using 250 predefined routes. Hence, to assess the mentioned influence, the number of paths were restricted per commodity to 5 and 3 paths resulting in an overall amount of 150 paths and 90 paths. It takes about 500 s and 150 s respectively to calculate the corresponding solutions.

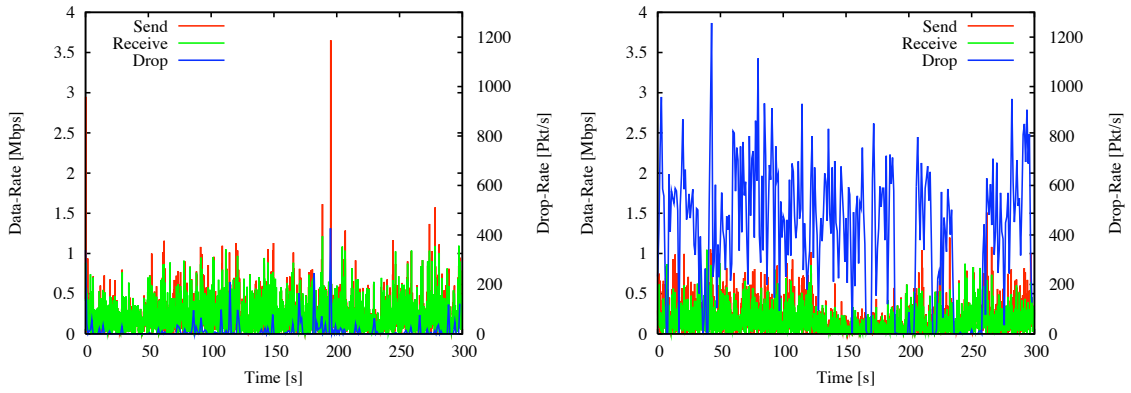
Based on these figures it can be assumed that the optimal deployment strategies for bigger networks can be specified within acceptable periods of time.

Table 7.23: Overall statistics for scenario Abilene under the constraint that each commodity must be secured by **7** security services for both types of generated self-similar traffic

| | High | | | Low | | |
|---|---|---|---|---|---|---|
| Strategy | Loss [%] | Fidran Loss [%] | e2e [ms] | Loss [%] | Fidran Loss [%] | e2e [ms] |
| *No* | 0 | 0 | 13.208 ±0.019 | 0.200 | 0 | 10.833 ±0.016 |
| *Late* | 3.000 | 3.000 | 42.130 ±0.071 | 15.600 | 15.600 | 47.799 ±0.066 |
| *Pre* | 2.400 | 2.400 | 30.151 ±0.044 | 14.300 | 14.300 | 33.301 ±0.038 |
| *SP* | 2.100 | 2.100 | 27.379 ±0.036 | 14.100 | 14.100 | 31.355 ±0.035 |
| *MP* | 1.400 | 1.400 | 25.733 ±0.036 | 10.500 | 10.500 | 30.417 ±0.035 |

## 7.7 Summary

This section evaluated the performance of the optimal deployment strategies developed. Two different network topologies were emulated for varying traffic scenarios. In Section 7.3 the benefit of the Fidran architecture in combination with strategy *Pre* was evaluated. A finding was that the penalty for commodities that must not be protected is within acceptable limits. Furthermore, the results show that the impact on the network performance can be reduced by intelligently deploying the security services. In Section 7.4 the Abilene network was emulated for two types of security requirements as well as two types of self-similar traffic. The emulations demonstrated that the optimal strategies reduce packet loss and end-to-end-delay. Furthermore, strategy *MP* better compensates than strategies *SP* and *Pre* the impact caused by the security services. Comparing the two latter strategies with each other, the singlepath strategy performs better in terms of packet loss and end-to-end-delay.




(a) High (b) Low

Figure 7.23: Sending and reception rate as well as packets dropped of the traffic demand from WASH to NYCM, commodity is inspected by **7** security services and strategy *SP* is applied

Table 7.24: Flow specific packet losses [%] for scenario Abilene under the constraint that each commodity must be secured by **7** security services/Low traffic traces; the three largest/ smallest commodities are emphasized/underlined

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0 | 0 | 0 | 0 | 0 |
| LOSA | **0** | X | 0 | 0 | 0 | 0 |
| NYCM | .100 | 0 | X | 0 | 0 | **.100** |
| SNVA | 0 | 0 | 0 | X | 0 | 0 |
| STTL | 0 | 0 | 0 | 0 | X | 0 |
| WASH | 0 | 0 | **.100** | 0 | 0 | X |

(a) Strategy *No*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 2.300 | 10.600 | 0 | 0 | 1.300 |
| LOSA | **7.000** | X | 18.200 | 0 | 1.500 | 2.500 |
| NYCM | 12.900 | 1.100 | X | 0 | 1.100 | **2.700** |
| SNVA | 3.700 | 3.500 | 8.900 | X | 1.300 | 0 |
| STTL | 6.900 | .500 | 10.500 | .500 | X | .100 |
| WASH | 8.600 | 1.600 | **36.200** | 0 | .900 | X |

(b) Strategy *Late*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.900 | 5.100 | 0 | .100 | 5.900 |
| LOSA | **.700** | X | 8.200 | .100 | 1.400 | 1.900 |
| NYCM | 13.300 | .700 | X | 0 | 1.100 | **16.400** |
| SNVA | 0 | 3.400 | 0 | X | 1.200 | 0 |
| STTL | .600 | .200 | 0 | .900 | X | 0 |
| WASH | 11.700 | .600 | **32.100** | .100 | .600 | X |

(c) Strategy *Pre*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.400 | 0 | 1.600 | 0 | .800 |
| LOSA | **.400** | X | 11.100 | 0 | 1.000 | 5.200 |
| NYCM | 2.300 | 7.000 | X | 0 | 1.000 | **2.500** |
| SNVA | 0 | 3.000 | 0 | X | 1.000 | .300 |
| STTL | .400 | .200 | 0 | .600 | X | 0 |
| WASH | 22.200 | 24.400 | **30.300** | 18.700 | .700 | X |

(d) Strategy *SP*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.500 | 0 | 0 | 0 | .400 |
| LOSA | **.600** | X | 8.200 | .200 | .700 | 1.200 |
| NYCM | 2.200 | .800 | X | 0 | 1.600 | **8.800** |
| SNVA | 0 | 2.700 | 0 | X | .900 | 0 |
| STTL | .300 | .300 | 0 | .700 | X | 0 |
| WASH | .400 | .700 | **26.300** | .500 | .900 | X |

(e) Strategy *MP*

Table 7.25: Time needed to specify optimal deployment strategies

| | Solution time for Abilene scenarios [s] | |
|---|---|---|
| Strategy | 6 Services | 7 Services |
| *Pre* | $< 10$ | $< 10$ |
| *SP* | $\sim 700$ | $\sim 700$ |
| *MP* | $\sim 2700$ | $\sim 2700$ |

# Chapter 8

# Conclusions

At the beginning of this thesis the need for a flexible intrusion prevention overlay network that relieves users from continuously maintaining their systems was analyzed. Recent developments show that communication networks cannot be secured by sporadic and uncoordinated security devices like firewalls at users and cooperates sites. The reasons behind this trend originate from multiple developments. The number of DNS-registered hosts in the Internet keeps Taking those trends into account, it can hardly be expected that all users and administrators will be able to keep their system(s) secure. Furthermore, fixing security holes as soon as patches become available can hardly be done in time on all end systems. Further, Chapter 3 discussed existing approaches to intrusion prevention and it addition, it showed that current systems do not address the above listed aspects. Thus, in order to relieve end-users and administrators from continuously having to deal with today's massive amount of security challenges, the protection of end systems should be done in the network. For this purpose, a flexible overlay network of intrusion prevention systems running on top of an active networking environment named Fidran was developed.

The concept followed by Fidran is introduced in Chapter 4. The main principle is called demand-driven intrusion prevention which uses the fact that attacks require the existence of one or multiple concrete vulnerabilities to succeed. For example, the *Code Red* attack exploits a buffer overflow that exists in certain versions of Microsoft's *Internet Information Server* (IIS). In the approach followed, an intrusion prevention service provides protection against attacks exploiting a concrete vulnerability. Flows towards an end-system are only analyzed by intrusion prevention services that protect against attacks that could actually harm it.

Recapitulating the proposed Fidran architecture comprises three functional parts:

1. the first functional part (not for high-speed deployment) includes the intelligence to analyze the network to be protected:
   - identification of the network topology,
   - discovery of the hosts that are running and
   - hosts profiling, identification of operating system and running applications.
2. the second functional part provides the intrusion prevention framework that actually allows to dynamically deploy intrusion prevention services on programmable nodes in the network.

3. the third functional part decides which intrusion prevention services are deployed on which programmable routers.

In a limited networking environment Fidran is capable to gather network knowledge which is used to limit security checks. The network analysis process is described in detail in Chapter5. Summarizing, it can be said that in analyzes:

- the topology of the network,
- the reachable end-systems:
  - distance between Internet and end-system,
  - running OS and applications,
  - amount of traffic that is destined to an end-system.

The case study conducted in Chapter 5 demonstrates that the approach to automatically gather network information reduces the amount of security checks that are performed. As a consequence, this reduces also the overall false-positive rate.

In the proposed concept a security service provides protection to a concrete application or it either prevents attacks from exploiting a concrete vulnerability. Security services can be inserted and removed from a Fidran system at runtime. Furthermore, a local security policy specifies which traffic flows must be inspected by which security services. Consequently, for each router the following degrees of freedom exist:

- decision whether or not a router is active.
- choice of protection services to be integrated.
- specification of which traffic must be analyzed by which protection services.

To assess our concept we implemented a Fidran prototype including a set of seven pattern matching security services. A performance evaluation of the prototype verified the router system model that was introduced in Section 6.1, and router specific parameters like $T_{\text{base}}$ and $T_{\text{active}}$ were measured.

To decide on what security services to deploy on which router an optimization framework was formulated. We differentiate between scenarios:

- Deploying security service along predefined paths:
  - single-path
  - multipath
- Joint traffic single-path routing and security service distribution

Predefined routing implies that at least one path from any source to any destination is specified—as a result the routing tables are set. In a single-path routing environment, a single path exists between any two subnetworks. In contrast, in a multipath routing environment multiple paths exist between subnetworks, allowing for load-balancing. Finally, in the last scenario the network topology is given but no paths between the subnetworks are defined.

Hence, the optimization framework specifies traffic routing—a single path from each source to each destination—and the distribution of the requested security services.

Each deployment strategy can be combined with one of two objective functions:

- the minimization of the number of programmable routers used in a network while fulfilling all security requests and keeping all router queues bounded;
- the minimization of the maximal workload of a programmable router while fulfilling all security requests and keeping all router queues bounded.

The first objective is to minimize the number of programmable routers and the idea for the second optimization model is to evenly distribute the load among routers.

The benefit of the proposed intrusion prevention overlay network was shown in Chapter 7. There a limited tree-network as well as the Abilene network were emulated for varying sets of self-similar network traffic traces. The generation process of the traffic traces considered the IP packet size distribution discussed in Section 7.2. Each emulation conducted verified that the intelligent deployment of intrusion prevention services reduces the impact on the network performance in terms of packet loss rate and end-to-end-delay. The proposed framework allows to consider routers of varying computational capabilities as well as the placement of security services under the constraint of a predefined order. Commodities that do not require any protection—for example encrypted commodities—are only slightly penalized.

The emulation of the Abilene scenarios—topology and traffic matrix base on a real network—showed the effect of combining the task of routing with the task of deploying security services. Strategies *SP* and *MP* achieved better results that strategy *Pre* which places the services on the routers of the predefined paths. The scenario under consideration showed that the joint optimization of single path routing and service placement is a big improvement with respect to optimal service placement over routes calculated with the Dijkstra algorithm, since the latter does not take the additional router load due to security processing into account.

The singlepath strategy tends to generate long paths to disburden heavy loaded routers. In contrast the multipath strategy splits huge flow into smaller ones and reroutes these over different paths. Both solutions show that they balance the load well. However, it must be considered that a flow cannot be divided into arbitrary smaller parts. A TCP flow transmitted via multiples paths suffers under varying RTTs and out-of- order packet delivery. Furthermore, some attacks span more than one packet, and might not be recognized by services that see only a fraction of the attacking traffic. A flow in a high-speed network is the aggregation of many smaller flows

Summarizing, the presented optimization framework for joint traffic routing and security service distribution allows to deploy security functions on routers in high-speed networks according to a chosen objective. The framework is also well suited to study the impact of topology changes, like capacity increases, new nodes, new links, on network performance.

# Appendix A

# Additional Results

Table A.1: Flow specific packet loss rates [pkt/s] for scenario Abilene under the constraint that each commodity must be secured by **6** security services using the **high** traffic traces; the three largest/smallest commodities are emphasized/underlined

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | - | - | - | - | 0.01 ±0.002 |
| LOSA | **-** | X | - | - | - | - |
| NYCM | - | 0.01 ±0.001 | X | - | 0.03 ±0.004 | **-** |
| SNVA | - | 0.17 ±0.020 | - | X | 0.28 ±0.032 | - |
| STTL | - | 0.09 ±0.011 | 0.03 ±0.004 | - | X | - |
| WASH | 0.23 ±0.027 | 0.01 ±0.001 | **0.37 ±0.042** | - | 0.01 ±0.001 | X |

(a) Strategy *No*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 2.19 ±0.250 | 0.19 ±0.022 | - | 0.01 ±0.002 | 0.03 ±0.004 |
| LOSA | **6.64 ±0.761** | X | 2.51 ±0.287 | 0.08 ±0.009 | 0.41 ±0.047 | 1.69 ±0.193 |
| NYCM | 4.00 ±0.458 | 4.39 ±0.503 | X | - | 2.47 ±0.284 | **6.02 ±0.691** |
| SNVA | 0.55 ±0.062 | 4.94 ±0.567 | 0.23 ±0.027 | X | 1.25 ±0.144 | - |
| STTL | 3.37 ±0.385 | 2.22 ±0.254 | 0.55 ±0.062 | 0.24 ±0.028 | X | 0.34 ±0.039 |
| WASH | 7.34 ±0.841 | 4.83 ±0.553 | **11.80 ±1.354** | - | 1.40 ±0.161 | X |

(b) Strategy *Late*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.25 ±0.143 | 0.14 ±0.017 | - | 0.01 ±0.002 | 0.04 ±0.005 |
| LOSA | **1.73 ±0.198** | X | 0.89 ±0.102 | 0.16 ±0.018 | 0.20 ±0.023 | 1.39 ±0.159 |
| NYCM | 1.45 ±0.166 | 3.11 ±0.356 | X | 0.03 ±0.006 | 2.88 ±0.331 | **5.98 ±0.686** |
| SNVA | 0.07 ±0.009 | 4.21 ±0.483 | 0.09 ±0.011 | X | 1.25 ±0.144 | - |
| STTL | 0.49 ±0.056 | 1.94 ±0.223 | 0.10 ±0.011 | 0.24 ±0.028 | X | 0.36 ±0.041 |
| WASH | 1.39 ±0.160 | 3.32 ±0.380 | **8.94 ±1.026** | - | 1.25 ±0.143 | X |

(c) Strategy *Dijkstra*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.53 ±0.175 | 0.06 ±0.007 | 0.01 ±0.002 | 0.02 ±0.002 | 0.05 ±0.006 |
| LOSA | **2.02 ±0.231** | X | 0.67 ±0.077 | 0.11 ±0.013 | 0.27 ±0.031 | 1.37 ±0.157 |
| NYCM | 1.00 ±0.114 | 3.58 ±0.411 | X | 0.05 ±0.007 | 2.27 ±0.261 | **4.20 ±0.482** |
| SNVA | 0.02 ±0.002 | 4.12 ±0.473 | 0.02 ±0.003 | X | 1.52 ±0.175 | 0.01 ±0.002 |
| STTL | 1.10 ±0.126 | 1.84 ±0.211 | 0.05 ±0.006 | 0.34 ±0.039 | X | 0.40 ±0.045 |
| WASH | 1.30 ±0.149 | 2.69 ±0.308 | **6.40 ±0.734** | - | 1.90 ±0.217 | X |

(d) Strategy *Single-Path*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.03 ±0.118 | 0.06 ±0.007 | - | - | - |
| LOSA | **1.92 ±0.220** | X | 0.68 ±0.078 | 0.15 ±0.017 | 0.10 ±0.012 | 1.27 ±0.146 |
| NYCM | 0.81 ±0.092 | 3.36 ±0.385 | X | 0.02 ±0.002 | 2.46 ±0.283 | **4.79 ±0.550** |
| SNVA | 0.12 ±0.014 | 3.87 ±0.444 | 0.03 ±0.004 | X | 1.47 ±0.168 | - |
| STTL | 0.64 ±0.073 | 1.65 ±0.189 | 0.08 ±0.009 | 0.30 ±0.035 | X | 0.21 ±0.024 |
| WASH | 1.17 ±0.134 | 3.86 ±0.442 | **6.43 ±0.738** | 0.08 ±0.009 | 1.57 ±0.179 | X |

(e) Strategy *Multipath*

Table A.2: Flow specific packet loss rates [pkt/s] for scenario Abilene under the constraint that each commodity must be secured by **7** using the **high** traffic traces; the three largest/ smallest commodities are emphasized/underlined

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | - | - | - | - | 0.01 ±0.002 |
| LOSA | - | X | - | - | - | - |
| NYCM | 0.13 ±0.014 | 0.01 ±0.001 | X | - | 0.01 ±0.002 | **0.16 ±0.018** |
| SNVA | - | - | - | X | - | - |
| STTL | - | - | 0.03 ±0.004 | - | X | 0.01 ±0.001 |
| WASH | 0.19 ±0.022 | 0.20 ±0.023 | **0.76 ±0.088** | 0.01 ±0.001 | 0.04 ±0.004 | X |

(a) Strategy *No*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.19 ±0.137 | 0.21 ±0.024 | - | - | 0.18 ±0.020 |
| LOSA | **11.51 ±1.319** | X | 0.52 ±0.060 | 0.09 ±0.010 | 1.76 ±0.201 | 4.51 ±0.516 |
| NYCM | 11.60 ±1.323 | 1.81 ±0.207 | X | - | 1.31 ±0.151 | **9.79 ±1.123** |
| SNVA | 0.27 ±0.031 | 4.32 ±0.496 | 0.18 ±0.021 | X | 1.46 ±0.167 | - |
| STTL | 5.41 ±0.618 | 0.51 ±0.059 | 0.44 ±0.051 | 0.25 ±0.029 | X | 0.06 ±0.007 |
| WASH | 11.00 ±1.260 | 3.78 ±0.434 | **12.44 ±1.427** | - | 0.58 ±0.067 | X |

(b) Strategy *Late*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0.83 ±0.095 | 0.22 ±0.025 | - | 0.01 ±0.001 | 0.51 ±0.057 |
| LOSA | **4.12 ±0.473** | X | 0.23 ±0.027 | 0.05 ±0.005 | 1.84 ±0.211 | 3.67 ±0.421 |
| NYCM | 6.18 ±0.705 | 1.77 ±0.203 | X | 0.01 ±0.002 | 1.23 ±0.141 | **15.84 ±1.817** |
| SNVA | - | 4.02 ±0.462 | - | X | 1.40 ±0.160 | - |
| STTL | 0.38 ±0.044 | 0.11 ±0.013 | 0.16 ±0.018 | 0.27 ±0.031 | X | - |
| WASH | 5.15 ±0.590 | 2.73 ±0.313 | **16.74 ±1.920** | 0.25 ±0.029 | 0.57 ±0.066 | X |

(c) Strategy *Dijkstra*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0.44 ±0.050 | 0.05 ±0.005 | - | - | 0.06 ±0.007 |
| LOSA | **3.64 ±0.418** | X | 0.45 ±0.051 | 0.10 ±0.012 | 1.80 ±0.207 | 3.99 ±0.457 |
| NYCM | 3.53 ±0.403 | 1.89 ±0.216 | X | 0.03 ±0.004 | 1.05 ±0.120 | **8.94 ±1.026** |
| SNVA | 0.07 ±0.009 | 4.27 ±0.490 | 0.02 ±0.003 | X | 1.79 ±0.205 | 0.01 ±0.001 |
| STTL | 0.43 ±0.049 | 0.15 ±0.018 | 0.10 ±0.012 | 0.23 ±0.026 | X | 0.02 ±0.002 |
| WASH | 7.27 ±0.832 | 7.67 ±0.878 | **12.60 ±1.446** | 0.57 ±0.065 | 0.55 ±0.063 | X |

(d) Strategy *Single-Path*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0.74 ±0.084 | 0.14 ±0.016 | - | - | 0.07 ±0.008 |
| LOSA | **3.89 ±0.447** | X | 0.02 ±0.003 | 0.08 ±0.010 | 1.50 ±0.173 | 3.52 ±0.403 |
| NYCM | 3.82 ±0.436 | 1.26 ±0.144 | X | 0.02 ±0.002 | 1.13 ±0.130 | **8.06 ±0.925** |
| SNVA | 0.01 ±0.001 | 4.00 ±0.459 | 0.20 ±0.023 | X | 1.02 ±0.117 | - |
| STTL | 0.27 ±0.031 | 0.03 ±0.003 | 0.13 ±0.015 | 0.22 ±0.025 | X | - |
| WASH | 1.26 ±0.144 | 2.53 ±0.290 | **6.97 ±0.800** | 0.09 ±0.011 | 0.87 ±0.099 | X |

(e) Strategy *Multipath*

Table A.3: Flow specific packet loss rates [pkt/s] for scenario Abilene under the constraint that each commodity must be secured by **6** security services /using the **low** traffic traces; the three largest/smallest commodities are emphasized/underlined

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | - | - | - | - | - |
| LOSA | - | X | - | - | - | - |
| NYCM | - | - | X | - | - | **0.01 ±0.001** |
| SNVA | 0.01 ±0.001 | 0.05 ±0.006 | - | X | 0.14 ±0.016 | - |
| STTL | - | - | - | - | X | - |
| WASH | 0.42 ±0.048 | 0.10 ±0.011 | **0.44 ±0.050** | 0.01 ±0.002 | 0.12 ±0.014 | X |

(a) Strategy *No*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0.31 ±0.035 | 0.03 ±0.004 | 0.06 ±0.007 | 0.03 ±0.004 | 1.03 ±0.118 |
| LOSA | **7.27 ±0.833** | X | 2.24 ±0.257 | - | 0.03 ±0.004 | 1.87 ±0.215 |
| NYCM | 2.86 ±0.327 | 1.83 ±0.210 | X | 0.03 ±0.004 | 0.95 ±0.108 | **4.81 ±0.551** |
| SNVA | 0.78 ±0.087 | 3.35 ±0.384 | 0.07 ±0.008 | X | 1.39 ±0.160 | 0.01 ±0.007 |
| STTL | 7.34 ±0.840 | 0.31 ±0.036 | 0.76 ±0.087 | 0.11 ±0.012 | X | 0.44 ±0.050 |
| WASH | 10.40 ±1.191 | 1.90 ±0.217 | **10.44 ±1.197** | - | 0.61 ±0.069 | X |

(b) Strategy *Late*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0.10 ±0.011 | 0.02 ±0.002 | 0.06 ±0.007 | 0.04 ±0.005 | 0.28 ±0.032 |
| LOSA | **0.96 ±0.110** | X | 0.97 ±0.111 | 0.05 ±0.006 | 0.05 ±0.006 | 0.92 ±0.106 |
| NYCM | 0.54 ±0.061 | 0.56 ±0.064 | X | 0.09 ±0.010 | 0.88 ±0.101 | **3.50 ±0.401** |
| SNVA | 0.04 ±0.005 | 3.26 ±0.374 | - | X | 1.33 ±0.153 | - |
| STTL | 0.66 ±0.076 | 0.26 ±0.030 | - | 0.21 ±0.024 | X | 0.01 ±0.001 |
| WASH | 2.48 ±0.284 | 0.54 ±0.062 | **6.73 ±0.772** | 0.04 ±0.004 | 0.61 ±0.069 | X |

(c) Strategy *Dijkstra*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0.04 ±0.005 | - | 0.10 ±0.011 | - | 0.15 ±0.018 |
| LOSA | **1.32 ±0.151** | X | 0.37 ±0.043 | 0.02 ±0.003 | 0.04 ±0.004 | 0.69 ±0.079 |
| NYCM | 0.46 ±0.052 | 0.17 ±0.020 | X | 0.19 ±0.021 | 0.80 ±0.091 | **2.73 ±0.314** |
| SNVA | 0.07 ±0.008 | 2.93 ±0.337 | - | X | 1.09 ±0.125 | - |
| STTL | 1.48 ±0.170 | 0.25 ±0.029 | 0.03 ±0.004 | 0.18 ±0.020 | X | 0.13 ±0.014 |
| WASH | 2.33 ±0.267 | 0.98 ±0.112 | **3.81 ±0.437** | 0.01 ±0.002 | 0.97 ±0.111 | X |

(d) Strategy *Single-Path*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 0.05 ±0.006 | 0.02 ±0.003 | 0.08 ±0.009 | 0.04 ±0.004 | 0.62 ±0.071 |
| LOSA | **0.97 ±0.111** | X | 0.47 ±0.054 | - | 0.01 ±0.001 | 0.84 ±0.096 |
| NYCM | 0.25 ±0.029 | 0.38 ±0.044 | X | 0.39 ±0.044 | 0.88 ±0.100 | **3.29 ±0.377** |
| SNVA | 0.04 ±0.005 | 2.48 ±0.285 | 0.01 ±0.002 | X | 0.98 ±0.112 | - |
| STTL | 1.23 ±0.141 | 0.24 ±0.028 | 0.03 ±0.003 | 0.25 ±0.029 | X | 0.12 ±0.014 |
| WASH | 1.91 ±0.219 | 0.93 ±0.106 | **4.44 ±0.509** | 0.02 ±0.002 | 0.92 ±0.106 | X |

(e) Strategy *Multipath*

Table A.4: Flow specific packet loss rates [pkt/s] for scenario Abilene under the constraint that each commodity must be secured by **7** security services /using the **low** traffic traces; the three largest/smallest commodities are emphasized/underlined

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | - | - | - | - | - |
| LOSA | - | X | - | - | - | - |
| NYCM | 0.24 ±0.028 | 0.07 ±0.008 | X | 0.01 ±0.001 | 0.02 ±0.002 | **0.50 ±0.057** |
| SNVA | - | - | - | X | - | - |
| STTL | - | - | - | - | X | - |
| WASH | 0.11 ±0.012 | 0.05 ±0.005 | **1.53 ±0.176** | 0.02 ±0.002 | 0.01 ±0.001 | X |

(a) Strategy *No*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.70 ±0.194 | 1.54 ±0.168 | - | 0.02 ±0.004 | 0.38 ±0.044 |
| LOSA | **22.06 ±2.529** | X | 17.42 ±1.991 | - | 0.30 ±0.035 | 3.66 ±0.420 |
| NYCM | 19.41 ±2.218 | 1.69 ±0.193 | X | - | 0.94 ±0.108 | **9.15 ±1.050** |
| SNVA | 0.50 ±0.055 | 4.06 ±0.465 | 0.75 ±0.078 | X | 1.49 ±0.171 | - |
| STTL | 11.08 ±1.267 | 0.34 ±0.039 | 5.84 ±0.661 | 0.11 ±0.013 | X | 0.05 ±0.006 |
| WASH | 17.46 ±1.995 | 2.91 ±0.333 | **485.85 ±55.805** | - | 0.58 ±0.066 | X |

(b) Strategy *Late*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.46 ±0.167 | 0.76 ±0.083 | - | 0.02 ±0.003 | 1.73 ±0.193 |
| LOSA | **2.46 ±0.282** | X | 7.95 ±0.913 | 0.05 ±0.006 | 0.31 ±0.036 | 2.86 ±0.328 |
| NYCM | 20.03 ±2.285 | 1.02 ±0.117 | X | 0.02 ±0.003 | 0.96 ±0.110 | **55.23 ±6.332** |
| SNVA | - | 3.88 ±0.445 | - | X | 1.48 ±0.169 | 0.01 ±0.007 |
| STTL | 0.99 ±0.113 | 0.17 ±0.020 | 0.04 ±0.005 | 0.17 ±0.020 | X | 0.02 ±0.002 |
| WASH | 23.58 ±2.692 | 1.18 ±0.135 | **431.43 ±49.553** | 0.03 ±0.004 | 0.49 ±0.056 | X |

(c) Strategy *Dijkstra*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.06 ±0.121 | 0.02 ±0.002 | 0.12 ±0.015 | 0.02 ±0.003 | 0.25 ±0.028 |
| LOSA | **1.38 ±0.159** | X | 10.80 ±1.237 | 0.02 ±0.003 | 0.12 ±0.014 | 7.59 ±0.867 |
| NYCM | 3.54 ±0.405 | 10.22 ±1.165 | X | 0.01 ±0.002 | 0.87 ±0.099 | **8.49 ±0.974** |
| SNVA | 0.01 ±0.001 | 3.46 ±0.397 | 0.01 ±0.002 | X | 1.20 ±0.138 | 0.02 ±0.005 |
| STTL | 0.67 ±0.077 | 0.18 ±0.020 | 0.01 ±0.001 | 0.12 ±0.013 | X | 0.02 ±0.002 |
| WASH | 44.73 ±5.108 | 42.80 ±4.893 | **407.10 ±46.759** | 4.60 ±0.510 | 0.62 ±0.071 | X |

(d) Strategy *Single-Path*

| → To | CHIN | LOSA | NYCM | SNVA | STTL | WASH |
|---|---|---|---|---|---|---|
| CHIN | X | 1.13 ±0.129 | 0.02 ±0.003 | 0.01 ±0.001 | 0.01 ±0.001 | 0.14 ±0.016 |
| LOSA | **2.00 ±0.229** | X | 7.96 ±0.914 | 0.05 ±0.006 | 0.09 ±0.011 | 1.88 ±0.215 |
| NYCM | 3.45 ±0.395 | 1.18 ±0.135 | X | 0.02 ±0.002 | 1.40 ±0.160 | **29.71 ±3.406** |
| SNVA | - | 3.19 ±0.366 | 0.01 ±0.001 | X | 1.12 ±0.128 | 0.01 ±0.003 |
| STTL | 0.56 ±0.064 | 0.22 ±0.025 | 0.05 ±0.006 | 0.15 ±0.017 | X | 0.02 ±0.003 |
| WASH | 0.90 ±0.103 | 1.14 ±0.131 | **352.78 ±40.520** | 0.11 ±0.013 | 0.74 ±0.084 | X |

(e) Strategy *Multipath*

# List of Figures

# Bibliography

[1] Aide - advanced intrusion detection environment. http://sourceforge.net/projects/aide.

[2] Common Vulnerabilities and Exposures. http://cve.mitre.org/cve/downloads/allcves.html.

[3] Computer emergency response team (cert) - statistics. www.cert.org/stats.

[4] LIDS - Linux Intrusion Detection System. http://www.lids.org.

[5] md5deep. http://md5deep.sourceforge.net/.

[6] OSSEC HIDS - Open Source HIDS. http://www.ossec.net/.

[7] Prelude. http://www.prelude-ids.org/.

[8] Snare - system intrusion analysis reporting environment. http://www.intersectalliance.com/projects/Snare/.

[9] Tripwire. http://www.tripwire.com.

[10] Trusted computer system evaluation criteria. http://csrc.nist.gov/secpubs/rainbow/std001.txt.

[11] The Abilene Network. `http://abilene.internet2.edu`.

[12] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. In *Communications of the ACM*, volume 18, pages 333–340, 1975.

[13] D. Alexander, W. Arbaugh, M. Hicks, P. Kakkar, A. Keromytis, J. Moore, C. Gunder, S. Nettles, and J. Smith. The switch ware active network architecture. *IEEE Network Special Issue on Active and Controllable Networks*, 12:29–36, June 1998.

[14] Edward Amoroso. *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response*. Intrusion Net Books, 1999.

[15] K. G. Anagnostakis, S. Ioannidis, S. Miltchev, J. Ioannidis, Michael B. Greenwald, and J. M. Smith. Efficient packet monitoring for network management. In *Proceedings of IFIP/IEEE Network Operations and Management Symposium (NOMS) 2002*, April 2002.

[16] James P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Co, 1980.

[17] K. Egevang andP. Francis. The ip network address translator (nat). RFC 1631, May 1994.

[18] ANSI. *Information processing systems: local area networks — Part 3. Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications.* ANSI/IEEE Std 802.3-1990 edition. International standard ISO/IEC 8802-3, IEEE product number: SH13482 edition, 1992.

[19] M. Asaka, S. Okazawa, and S. Goto. A Method of Tracing Intruders by Use of Mobile Agents. In *9th Annual Conference of the Internet Society (INET)*, San Jose, CA, USA, 1999.

[20] S. Axelsson. The base-rate fallacy and its implications for the intrusion detection security. In *Proc. of 6th ACM Conference on Computer and Communications Security*, Singapore, 1999.

[21] Christian Bachmeir and Peter Tabery. PIRST-ONs: a service architecture for embedding and leveraging active and programmable networks technology. In *IEEE 10th International Conference on Software, Telecommunications and Computer Networks, SoftCOM*, October 2002.

[22] R. Bajcsy, T. Benzel, M. Bishop, B. Braden, C. Brodley, S. Fahmy, S. Floyd, W. Hardaker, A. Joseph, G. Kesidis, K. Levitt, B. Lindell, P. Liu, D. Miller, R. Mundy, C. Neuman, R. Ostrenga, V. Paxson, P. Porras, C. Rosenberg, J. Tygar, S. Sastry, D. Sterne, and S. Wu. Cyber defense technology networking and evaluation. *Communications of ACM*, 47(3):58–61, March 2004.

[23] J. S. Balasubramaniyan, J. O. Garcia-Fernandez, D. Isacoff, E. Spafford, and D. Zamboni. An architecture for intrusion detection using autonomous agents. In *ACSAC '98: Proceedings of the 14th Annual Computer Security Applications Conference*. IEEE Computer Society, 1998.

[24] Jay Beale, James C. Foster, Jeffrey Posluns, Ryan Russell, and Brian Caswell. *Snort 2.0 Intrusion Detection*. Syngress, 2003.

[25] Steven M. Bellovin. A Technique for Counting Natted Hosts. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 267–272, New York, NY, USA, 2002. ACM Press.

[26] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab. Experience with DETER: a testbed for security research. In *Testbeds and Research Infrastructures for the Development of Networks and Communities TRIDENTCOM*, March 2006.

[27] Stephen P. Berry. Shoki. http://shoki.sourceforge.net/.

[28] BITKOM. Private Computernutzung steigt in Deutschland auf 70 Prozent. `http://bitkom.de/49795_49655.aspx`.

[29] Herbert Bos and Kaiming Huang. Towards software-based signature detection for intrusion prevention on the network card. In *Proceedings of Eighth International Symposium on Recent Advances in Intrusion Detection (RAID2005)*, Seattle, WA, September 2005.

[30] R. S. Boyer and J. S. Moore. A fast string search algorithm. *Communications of ACM*, 20(10):762–772, 1977.

[31] R. Braden. Requirements for Internet Hosts – Communication Layers. RFC 1122, Oct. 1989.

[32] H. K. Browne, W. A. Arbaugh, J. McHugh, and W. L. Fithen. A trend analysis of exploitations. In Francis M. Titsworth, editor, *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 214–231, Los Alamitos, CA, May 14–16 2001. IEEE Computer Society.

[33] David Brumley, James Newsome, Dawn Song, Hao Wang, and Somesh Jha. Towards automatic generation of vulnerability-based signatures. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 2–16, Washington, DC, USA, 2006. IEEE Computer Society.

[34] Caida. Packet sizes and sequencing. `http://www.caida.org/analysis/learn/packetsizes/`, March 1998.

[35] K. L. Calvert, S. Bhattacharjee, E. Zegura, and J. Sterbenz. Directions in active networks. *IEEE Communications Magazine*, 36(10):72–78, Oct 1998.

[36] Andrew T. Campbell, Herman G. De Meer, Michael E. Kounavis, Kazuho Miki, John B. Vicente, , and Daniel Villela. A survey of programmable networks. *ACM SIGCOMM Computer Communications Review*, 29(2):7–23, April 1999.

[37] CERT. W32/blaster worm. http://www.cert.org/advisories/CA-2003-20.html, August 2003.

[38] William R. Cheswick and Steven M. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley Publishing Company, 1994.

[39] Ramkumar Chinchani and Eric van den Berg. A fast static analysis approach to detect exploit code inside network flows. In Valdes and Zamboni [132], pages 284–308.

[40] Cisco. NetRanger. http://www.cisco.com/univercd/cc/td/doc/product/iaabu/netrangr/.

[41] K. G. Coffman and A. M. Odlyzko. Internet growth: is there a "moore's law" for data traffic? pages 47–93, 2002.

[42] Internet Systems Consortium. Isc internet domain survey. http://www.isc.org/index.pl?/ops/ds/, 2005.

[43] E. Crawley, R. Nair, B. Rajagopalan, and H. Sandick. A Framework for QoS-based Routing in the Internet. RFC 2386, Aug. 1998.

[44] R. K. Cunningham, R. P. Lippmann, D. J. Fried, S. L. Garfinkel, I. Graf, K. R. Kendall, S. E. Webster, D. Wyschogrod, and M. A. Zissman. Evaluating intrusion detection systems without attacking your friends: The 1998 darpa intrusion detection evaluation. In *ID*, 1999.

[45] F. Cuppens. Managing alerts in a multi-intrusion detection environment. In *ACSAC '01: Proceedings of the 17th Annual Computer Security Applications Conference*, page 22, Washington, DC, USA, 2001. IEEE Computer Society.

[46] F. Cuppens and Alexandre Miège. Alert correlation in a cooperative intrusion detection framework. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 202, Washington, DC, USA, 2002. IEEE Computer Society.

[47] O. Dain and R. Cunningham. Fusing heterogeneous alert streams into scenarios.

[48] H. Debar, D. Curry, and B. Feinstein. The Intrusion Detection Message Exchange Format. IETF Internet-Draft, Feb. 2006.

[49] H. Debar and A. Wespi. Aggregation and correlation of intrusion-detection alerts. In *RAID '00: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, pages 85–103, London, UK, 2001. Springer-Verlag.

[50] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Comput. Networks*, 31(9):805–822, 1999.

[51] Dorothy E. Denning. An intrusion-detection model. *IEEE Trans. Softw. Eng.*, 13(2):222–232, 1987.

[52] R. Deraison, H. Meer, R. Temmingh, C. v. d. Walt, R. Alder, J. Alderson ando A. Johnston, and G. A. Theall. *Nessus Network Auditing*. Syngress, 2004.

[53] R. Doms. Dynamic host configuration protocol. RFC 2131, Mar. 1997.

[54] Steven Northcutt et al. Shadow. http://www.nswc.navy.mil/ISSEC/index.html, 1998.

[55] W. La Cholter et al. IBAN: Intrusion Blocker based on Active Networks. In *Proc. of Dance 2002*, 2002.

[56] Jeff Forristal and Greg Shipley. Vulnerability Assessment Scanners. *Network Computing*, Jan. 2001.

[57] Thomas Fuhrmann, Till Harbaum, Marcus Schöller, and Martina Zitterbart. Amnet 2.0: An improved architecture for programmable networks. In James P. G. Sterbenz, Osamu Takada, Christian F. Tschudin, and Bernhard Plattner, editors, *IWAN*, volume 2546 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 2002.

[58] Fyodor. Remote os detection via tcp/ip stack fingerprinting. http://www.insecure.org/nmap/nmap-fingerprinting-article.html.

[59] Fyodor. The Art of Port Scanning. *Phrack Magazine*, 7, 1997.

[60] E. Gerbier. Afick (another file integrity checker). http://afick.sourceforge.net/.

[61] Coretez Giovanni. Fun with packets: Designing a stick. http://www.eurocompton.net/stick/papers/Peopledos.pdf.

[62] AN Security Working Group. Security architecture for active nets. online: ftp://ftp.tislabs.com/pub/activenets/secarch2.ps, July 1998.

[63] AN Working Group. Architectural framework for active networks, 1998.

[64] Joshua Haines, Lee Rossey, Rich Lippmann, and Robert Cunnigham. Extending the 1999 evaluation. In *DISCEX*, Anaheim, CA, USA, June 2001.

[65] M. Handley, C. Kreibich, and V. Paxson. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Usenix Security Symposium*, Washington, DC, USA, 2001.

[66] A. Hess and g. Schäfer. Isp-operated protection of home networks with fidran. In *Proc. of first IEEE Consumer Communications and Networking Conference (CCNC'2004)*, Las Vegas, Nevada, USA, Jan. 2004.

[67] A. Hess and G. Schäfer. A flexible and dynamic access control policy framework for an active networking environment. In *Proc. of Kommunikation in Verteilten Systemen (KiVS 2003)*, pages 321–333, Leipzig, Germany, February 2003.

[68] A. Hess and G. Schäfer. Realizing a flexible access control mechanism for active nodes based on active networking technology. In *Proc. of 2004 IEEE International Conference on Communications (ICC 2004)*, Paris, France, June 2004.

[69] A. Hess, M. Schoeller, G. Schäfer, M. Zitterbart, and A. Wolisz. A dynamic and flexible access control and resource monitoring mechanism for active nodes. In *Proc. of OpenArch 2002*, pages 11–16, New York City,New York, USA, June 2002. Short Paper.

[70] R. N. Horspool. Practical fast searching in strings. *Softw., Pract. Exper.*, 10(6):501–506, 1980.

[71] J. Howard. *An Analysis of Security Incidents on the Internet.* PhD thesis, Carnegie Mellon University, 1998.

[72] Thomas H.Ptacek and Timothy N.Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. January 1998.

[73] ILOG CPLEX Division. *CPLEX 9.0 Reference Manual*, 2003. `http://www.cplex.com`.

[74] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith. Implementing a distributed firewall. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 190–199. ACM Press, 2000.

[75] ISS. RealSecure. http://www.iss.net/support/documentation.

[76] G. Jakobson and M. D. Weissman. Alarm correlation. *IEEE Network Magazine*, 7:52–59, Nov. 1993.

[77] R. Janakiraman, M. Waldvogel, and Q. Zhang. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *IEEE WET ICE Workshop on Enterprise Security*, Linz, Austria, June 2003.

[78] Trevor Jim, Greg Morrisett, Dan Grossman, Michael Hicks, James Cheney, and Yanling Wang. Cyclone: A Safe Dialect of C. In *USENIX Annual Technical Conference*, pages 275–288, Montery, CA, USA, Jun. 2002.

[79] Klaus Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security*, November 2003.

[80] Klaus Julisch and Marc Dacier. Mining intrusion detection alarms for actionable knowledge. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 366–375, New York, NY, USA, 2002. ACM Press.

[81] M. Kodialam, T. V. Lakshman, and Sudipta Sengupta. Configuring networks with content filtering nodes with applications to network security. In *IEEE INFOCOM*, 2005.

[82] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[83] Glen Kramer. Synthetic traffic generation. `http://wwwcsif.cs.ucdavis.edu/~kramer/research.html`.

[84] C. Kruegel and W. Robertson. Alert Verification Determining the Success of Intrusion Attempts. In *1st Workshop on the Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, Dortmund, Germany, Jul. 2004.

[85] C. Kruegel, F. Valeur, and G. Vigna. *Intrusion Detection and Correlation: Challenges and Solutions*, volume 14 of *Advances in Information Security*. Springer, 2005.

[86] Christopher Krügel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. Polymorphic worm detection using structural information of executables. In Valdes and Zamboni [132], pages 207–226.

[87] Network Research Group Lawrence Berkeley National Laboratory. ARPWatch. http://www-nrg.ee.lbl.gov/.

[88] Will E. Leland, Murad S. Taqq, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of Ethernet traffic. In Deepinder P. Sidhu, editor, *ACM SIGCOMM*, pages 183–193, San Francisco, California, 1993.

[89] U. Lindqvist and E. Jonsson. How to systematically classify intrusions. In *IEEE Symposium on Security and Privacy*, Oakland, California, USA, 1997.

[90] Avivah Litan. Phishing Attacks Escalate, Morph and Cause Considerable Damage. `http://www.gartner.com`, December 2007.

[91] Point Topic Ltd. World broadband statistics: Q3 2006. http://www.point-topic.com, 2006.

[92] Andrew Mackie, Jensenne Roculan, Ryan Russell, and Mario Van Velzen. Nimda worm analysis. Technical report, SecurityFocus, Sep. 2001.

[93] Robert Mathonet, Herwig Van Cotthem, and Leon Vanryckeghem. Dantes: An expert system for real-time network troubleshooting. In *IJCAI*, pages 527–530, 1987.

[94] Peter Mell, Vincent Hu, Richard Lippmann, Josh Haines, and Marc Zissman. An overview of issues in testing intrusion detection systems. Technical report, National Institute of Standards and Technology ITL and Massachusetts Institute of Technology Lincoln Laboratory, 2003.

[95] H.D. Moore. The metasploit project. http://www.metasploit.com.

[96] Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Ducassé. M2d2: A formal data model for ids alert correlation. In *RAID*, pages 115–127, 2002.

[97] S. Murphy, E. Lewis, R. Puga, R. Watson, and R. Yee. Strong security for active networks. In *IEEE Open Architectures and Network Programming*, pages 63–70, 2001.

[98] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. J. Wiley, New York, 1988.

[99] Peter G. Neumann and Donn B. Parker. A summary of computer misuse techniques. In *12th National Computer Security Conference*, pages 396–407, Oct. 1989.

[100] NFR. Network Flight Recorder. http://www.nfr.com/.

[101] Peng Ning, Yun Cui, and Douglas S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pages 245–254, New York, NY, USA, 2002. ACM Press.

[102] Stephen Northcutt, Lenny Zeltser, Scott Winters, Karen Kent Frederick, and Ronald W. Ritchey. *Network Perimeter Security*. New Riders, 2002.

[103] Kevin Novak. VA Scanners Pinpoint Your Weak Spots. *Network Computing*, Jun. 2003.

[104] K. Park, G. Kim, and M. E. Crovella. *Self-similar network traffic and performance evaluation*, chapter 14, pages 349–366. John Wiley & Sons, Inc., 2000.

[105] Samuel Patton, William Yurcik, and David Doss. An achilles heel in signature-based ids: Squealing false positives in snort. In *Proc. of fourth International Symposium on Recent Advances in Intrusion Detection (RAID)*, Davis, California, USA, Oct. 2001.

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

[107] Vern Paxson and Sally Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[108] Tadeusz Pietraszek. Using adaptive alert classification to reduce false positives in intrusion detection. In *In Proc. of 7th Recent Advances in Intrusion Detection (RAID)*, pages 102–124, Sophia Antropolis, France, Sep. 2004.

[109] Phillip A. Porras and Alfonso Valdes. Live traffic analysis of TCP/IP gateways. In *Internet Society's Networks and Distributed Systems Security Symposium*, March 1998.

[110] The Vint Project. The ns-2 Network Simulator. `http://www.isi.edu/nsnam/ns/doc/index.html`.

[111] Psionic. PortSentry - Sentry Tools. http://sourceforge.net/projects/sentrytools/.

[112] E. Rescorla. Security holes...who cares? In *12th USENIX Security Symposium*, pages 75–90, Washington, DC, USA, Aug. 2003.

[113] Cyber Defense Technology Experimental Research. The deter testbed: Overview. `http://www.isi.edu/deter/docs/testbed.overview.htm`, Oct. 2004.

[114] J. Reynolds and J. Postel. Assigned Numbers. RFC 1700, Oct. 1994.

[115] G. Schäfer. *Security in Fixed and Wireless Networks: An Introduction to securing data communications*. John Wiley & Sons, 2004.

[116] Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, Berlin, 2002.

[117] Beverly Schwartz, Alden W. Jackson, W. Timothy Strayer, Wenyi Zhou, R. Dennis Rockwell, and Craig Partridge. Smart packets: applying active networks to network management. *ACM Trans. Comput. Syst.*, 18(1):67–88, 2000.

[118] Umesh Shankar and Vern Paxson. Active mapping: Resisting nids evasion without altering traffic. In *IEEE Symposium on Security and Privacy*, pages 44–61, 2003.

[119] Joel Snyder. Taking aim. *Information Security*, Jan. 2004.

[120] Sophos. Sophos annual security report. `http://www.sophos.com/pressoffice/news/articles/2005/12/toptensummary05.htm`, July 2005.

[121] Lance Spitzner. Iding remote hosts, without them knowing - passive fingerprinting. 2000.

[122] William Stallings. *Network and Internetwork Security Principles and Practice*. Prentice Hall, 1995.

[123] S. Staniford, G. Grim, and R. Jonkman. Flash worms: Thirty seconds to infect the internet. `http://www.silicondefense.com/flash`.

[124] Stuart Staniford, James A. Hoagland, and Joseph M. McAlerney. Practical automated detection of stealthy portscans. *J. Comput. Secur.*, 10(1-2):105–136, 2002.

[125] S. Staniford-Chen, B. Tung, and D. Schnackenberg. The common intrusion detection framework. In *In Proc. of the second Information Survivability Workshop*, Orlando, Florida, USA, Oct. 1998.

[126] Symantec. NetProwler. http://www.symantec.com/region/can/eng/product/np/.

[127] Murad S. Taqqu, Walter Willinger, and Robert Sherman. Proof of a fundamental result in self-similar traffic modeling. *SIGCOMM Comput. Commun. Rev.*, 27(2):5–23, 1997.

[128] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80–86, 1997.

[129] David L. Tennenhouse and David J. Wetherall. Towards an active network architecture. *Computer Communication Review*, 26(2), 1996.

[130] Birger Toedtmann and Erwin P. Rathgeb. Anticipatory distributed packet filter configuration for carrier-grade ip-networks. In *IFIP TC6 Networking*, pages 928–941, Coimbra, Portugal, May 2006.

[131] Alfonso Valdes and Keith Skinner. Probabilistic alert correlation. In *RAID '00: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, pages 54–68, London, UK, 2001. Springer-Verlag.

[132] Alfonso Valdes and Diego Zamboni, editors. *Recent Advances in Intrusion Detection, 8th International Symposium, RAID 2005, Seattle, WA, USA, September 7-9, 2005, Revised Papers*, volume 3858 of *Lecture Notes in Computer Science*. Springer, 2006.

[133] Ke Wang, Gabriela Cretu, and Salvatore J. Stolfo. Anomalous payload-based worm detection and signature generation. In Valdes and Zamboni [132], pages 227–246.

[134] David J. Wetherall, John V. Guttag, and David L. Tennenhouseing. Ants: A toolkit for building and dynamically deploying network protocols. In *IEEE OPENARCH*, San Francisco, USA, April 1998.

[135] H.P. Williams. *Model Building in Mathematical Programming*, volume 3rd edition. Wiley, New York, USA, 1990.

[136] Walter Willinger, Murad S. Taqqu, Robert Sherman, and Daniel V. Wilson. Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 5(1):71–86, 1997.

[137] Adam Wolisz, Christian Hoene, Berthold Rathke, and Morten Schläger. Proxies, Active Networks, Re-configurable Terminals: The Cornerstones of Future Wireless Internet. In *IST Mobile Commuications Summit*, pages 795–803, Galway, Ireland, October 2000.

[138] B. Wotring, B. Potter, and M. J. Ranum. *Host Integrity Monitoring Using Osiris and Samhain*. Syngress, 2005.

[139] Xipeng Xiao and L. M. Ni. Internet qos: a big picture. *Network, IEEE*, 13(2):8–18, 1999.

[140] V. Yegneswaran, P. Barford, and S. Jha. Global intrusion detection in the domino overlay system. In *The 11th Annual Network and Distributed System Security Symposium*, San Diego, CA, feb 2004.

[141] William Yurcik. Controlling intrusion detection systems by generating false positives: Squealing proof-of-concept. In *LCN*, pages 134–135. IEEE Computer Society, 2002.

[142] M. Zalewski. p0f. http://lcamtuf.coredump.cx/p0f.shtml.