Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2008

MSc THESIS

Efficient Pre-filtering Techniques for Packet Inspection

Angelos Arelakis

Abstract

CE-MS-2008-15

Network Security is a significant issue nowadays. The information flow is enormous and the attacks have been substantially evolved. Every single packet of the flow must be scanned in deep and checked with all known attack rules (Deep Packet Inspection) to determine whether it is malicious. However, the task of Deep Packet Inspection requires a significant amount of processing, creating a bottleneck to the network. Packet Pre-filtering divides this task into two stages. The first stage (Pre-filtering stage) inspects the packet using a set of subrules and therefore needs less processing. This set is the result of preprocessing the initial rules where a smaller portion of every single rule is selected. In addition, this set of subrules must be efficient enough so that the least possible rules are needed to be processed in the second stage, achieving smaller implementation cost and/or smaller latency. This thesis proposes five techniques which accommodate Pre-filtering to meet these requirements. The three of them are the extraction techniques and create the set of subrules. Each subrule has a header and a part of the content (static pattern) or of the PCRE (type of regular expression). The extraction techniques are: the First Content Prefix which extracts the prefix of the first content of each rule, the PCRE Prefix which exploits the PCRE and extracts a prefix of it, and the Unique Part Rule which creates a set of unique subrules, extracting part of the content(s). Two more techniques have also been proposed. The Rule Correlation correlates the subrules (of the Pre-filtering stage) with similar characteristics to exclude them from the first stage of processing, achieving smaller latency. Secondly, Smart Rule Reuse optimizes the second stage of processing by exploiting the temporal locality of the activated rules between consecutive packets. All the techniques were evaluated using SNORT Network Intrusion Detection System and real attack traffic traces. The most efficient extraction technique is the Unique Part Rule (selected part length to 8 bytes), because only 2 rules on average are activated per packet while the maximum number of them, which indicates the required number of resources in the second stage, is approximately 64. The Rule Correlation achieves to correlate about 1700 rules out of the 9000 rules when used in combination with Unique Part Rule technique, achieving smaller latency or fewer resources in the first stage, while the Smart Rule Reuse uses rules activated by previous packets and hence avoids memory accesses so that the second stage of processing has lower latency.

Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

Efficient Pre-filtering Techniques for Packet Inspection

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Angelos Arelakis
born in Thessaloniki, Greece

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Efficient Pre-filtering Techniques for Packet Inspection

by Angelos Arelakis

Abstract

Network Security is a significant issue nowadays. The information flow is enormous and the attacks have been substantially evolved. Every single packet of the flow must be scanned in deep and checked with all known attack rules (Deep Packet Inspection) to determine whether it is malicious. However, the task of Deep Packet Inspection requires a significant amount of processing, creating a bottleneck to the network. Packet Pre-filtering divides this task into two stages. The first stage (Pre-filtering stage) inspects the packet using a set of subrules and therefore needs less processing. This set is the result of preprocessing the initial rules where a smaller portion of every single rule is selected. In addition, this set of subrules must be efficient enough so that the least possible rules are needed to be processed in the second stage, achieving smaller implementation cost and/or smaller latency. This thesis proposes five techniques which accommodate Pre-filtering to meet these requirements. The three of them are the extraction techniques and create the set of subrules. Each subrule has a header and a part of the content (static pattern) or of the PCRE (type of regular expression). The extraction techniques are: the First Content Prefix which extracts the prefix of the first content of each rule, the PCRE Prefix which exploits the PCRE and extracts a prefix of it, and the Unique Part Rule which creates a set of unique subrules, extracting part of the content(s). Two more techniques have also been proposed. The Rule Correlation correlates the subrules (of the Pre-filtering stage) with similar characteristics to exclude them from the first stage of processing, achieving smaller latency. Secondly, Smart Rule Reuse optimizes the second stage of processing by exploiting the temporal locality of the activated rules between consecutive packets. All the techniques were evaluated using SNORT Network Intrusion Detection System and real attack traffic traces. The most efficient extraction technique is the Unique Part Rule (selected part length to 8 bytes), because only 2 rules on average are activated per packet while the maximum number of them, which indicates the required number of resources in the second stage, is approximately 64. The Rule Correlation achieves to correlate about 1700 rules out of the 9000 rules when used in combination with Unique Part Rule technique, achieving smaller latency or fewer resources in the first stage, while the Smart Rule Reuse uses rules activated by previous packets and hence avoids memory accesses so that the second stage of processing has lower latency.

Laboratory	:	Computer Engineering
Codenumber	:	CE-MS-2008-15

Committee Members :

Advisor:	Assistant Prof. Georgi Gaydadjiev, CE, TU Delft
Advisor:	Dr. Ioannis Sourdis, CE, TU Delft

Chairperson:	Associate Prof. Koen Bertels, CE, TU Delft
Member:	Assistant Prof. Christian Doerr, NAS, TU Delft

to the people I love...

σε αυτούς που αγαπώ...

Acknowledgements

I am thankful to Ioannis Sourdis and Georgi Gaydadjiev for their guidance and support during my research study for this thesis. I am also grateful to them for their help, guidance and encouragement during my Master. I would like also to thank all of my professors for the role they have played to my thinking and my career as an engineer.

I am grateful to my parents and my sister, Stella, for their daily and patient support and encouragement all these years. It would not be easy to do my Master and have so great fun in Delft without having so great friends and colleagues. Many thanks to George (Ricky) and Maria, Stavros, George (Emerald), George (Tom Cruz) and the other guys from Greece, Iran, Italy and the entire world.

Finally, I would like to thank our former CE administrator, Bert Meijs and the CE secretary, Lidwina Tromp for their valuable technical and administrative help during all these years.

Angelos Arelakis
Delft, The Netherlands
December 15, 2008

Contents

List of Figures

List of Tables

List of Acronyms

BEPE	Best Effort Processing Engines
CAM	Content Addressable Memories
DFA	Deterministic Finite Automata
DIDS	Distributed Intrusion Detection System
DoS	Denial of Service
FCP	First Content Prefix
FTP	File Transfer Protocol
GTSE	Guaranteed Throughput Specialized Engines
HEX	Hexadecimal
HIDS	Host-based Intrusion Detection Systems
HTTP	Hypertext Transfer Protocol
HW	Hardware
ICMP	Internet Control Message Protocol
IDES	Intrusion Detection Expert System
IDS	Intrusion Detection Systems
KGB	Komityet Gosudarstvennoy Bezopasnosty
LRU	Least Recently Used
MRU	Most Recently Used
NFA	Non-deterministic Finite Automata
NIDS	Network Intrusion Detection Systems
NIPS	Network Intrusion Prevention Systems
PCRE	Perl Compatible Regular Expression
PE	Processing Elements
PINE	Packet INspection Engine
PP	PCRE Prefix
regex	regular expression
SID	Signature ID
SMTP	Simple Mail Transfer Protocol
SW	Software
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UP	Unique Part (Rule)
UPwC	Unique Part (Rule) with Correlation rules
UPnC	Unique Part (Rule) no (without) Correlation rules
URI	Uniform Resource Identifier

Introduction 1

If a firewall is a doorman, a NIDS is an undercover KGB agent
-Gianni Tedesco

The widespread use of Computers and Internet is obvious in the last decades and even more popular in the last years. They play an important role in our life and the reason is one and simple: Information. It is not a coincidence that the term Information Era is used for this age. Millions of gigabytes of information flow every day over the Internet. Computers are organized in networks in order to transfer data and the Internet is considered the best means through which fast flow of information takes place. Information can be public or private and in both cases, information security is a very critical issue. In the case of public information (news, digital encyclopedias, etc), the webservers must be shielded against aspiring hackers. However, information security is more important in systems that keep private information, like personal computers where personal data (credit card numbers, photos, personal documents) is stored, systems that rely on computers like ATMs, air traffic control systems, power systems, systems responsible for storing and transporting healthcare data, credit card processing systems and systems that keep data for national security.

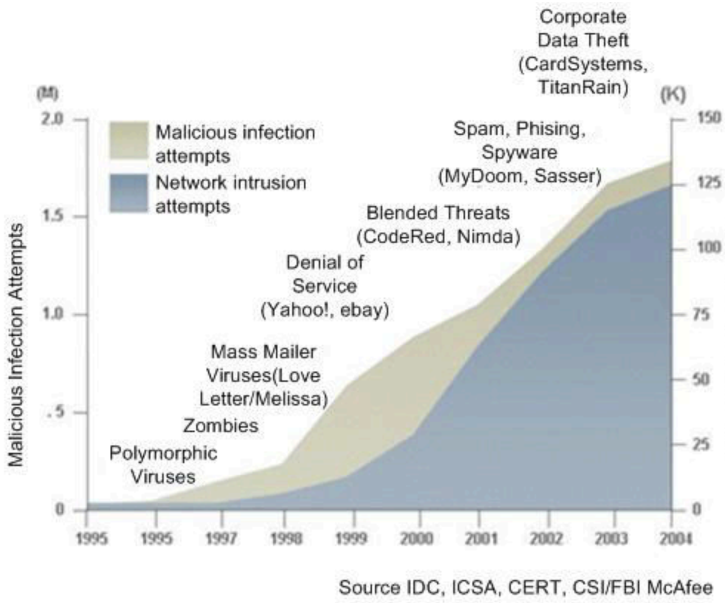


Figure 1.1: The figure depicts the number of network intrusion attempts and the number of malicious infection attempts from 1994 to 2004. The attacks are more and more sophisticated and this is shown by the evolution of the attacks through the years (Source: [8]).

Many attacks, which are motivated by financial, political or even military objectives, take place every day making information and network security a hot issue. Figure 1.1 depicts the number of attack attempts (network and malicious) from 1994-2004. The network intrusion attempts have increased exponentially since 1998 while the same figure shows how the attacks have been evolved year by year. Another figure, Figure 1.2, depicts the average daily number of attacks per month of 2007 that were blocked by SecureWorks. Looking at this figure, someone can conclude that most of the attacks have as a target health-care data rather than banks or other conventional targets and this verifies the concern that the target nowadays can be anyone [2].

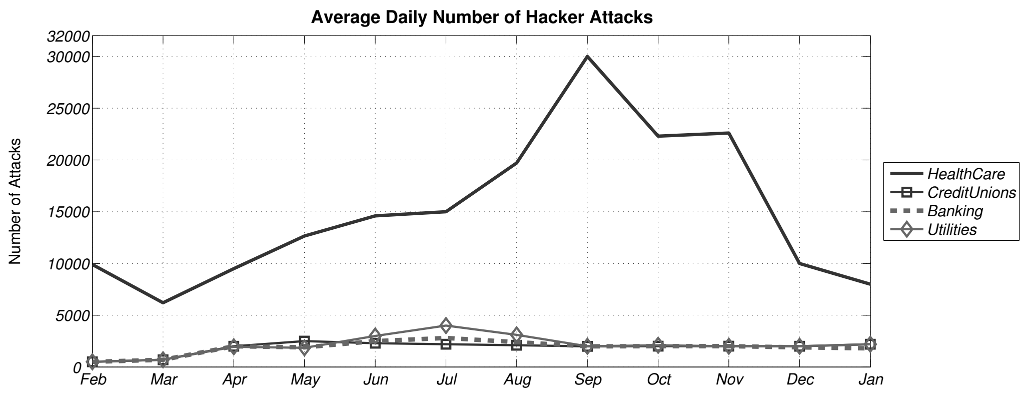


Figure 1.2: Average daily number of hacker attacks blocked by SecureWorks in 2007. (Source: [17]).

The problem on the network security starts with the unreliability of the Operating Systems (OS) regarding the security issues [2]. All the OS and especially Microsoft Windows have several security vulnerabilities. This fact and taking into account the fact that MS Windows is currently used more than any other OS is the first starting point for the potential intruders. Another significant problem may be the bad configuration of the network [2]. These problems along with the importance of information and network security created the need for security systems.

There are many kinds of security systems but are mainly divided into two categories based on the time instance they handle the attack. In other words, they can be active (handle the attack when it is happening) or passive (handle the attack after it has happened). The attacks have been significantly evolved and for that reason active security systems are needed. The main requirements of these systems is that they must be fast and must not depend on user's decision (automated). Network Intrusion Detection Systems (NIDS) have mainly this concept. They scan every single packet using thousands of rules in order to detect the potential intrusion, attempting to match it with an intrusion description (intrusion detection rule). This is a highly computational task due to the large number of rules which must be processed and their "complexity" because of their possible sophisticated description. A significant amount of research has been performed on the field of NIDS which attempts to accelerate this task. One of the proposed approaches is the Multi-stage packet scanning where the packet inspection is divided in many stages. The Packet Pre-filtering follows exactly this idea.

Packet Pre-filtering, which was proposed by Sourdis in [32] and [30], is based on the observation that a single incoming packet may not match all the intrusion detection rules. It divides the packet inspection into two stages. The idea is to extract a small portion of each rule creating a set of subrules and use this in the first phase of scanning. After a lightweight processing during the Pre-filtering stage (first stage), only the rules which have been activated by the incoming packet are sent to the next (second) stage, for a more sophisticated packet processing to be fully matched against the packet and determine whether it is a threat or not. The number of activated rules affects the performance and the implementation cost of the second stage and for that reason, the proper selection of the rule portion may reduce the number of the activated rules. This thesis proposes extraction techniques which attempt to tackle exactly the problem of the efficient selection of the rules' portion and other techniques, to improve a NIDS that uses Pre-filtering.

The rest of the introductory chapter is organized in three sections. In the first one, a brief presentation of security systems and especially of Network Intrusion Detection Systems (NIDS) along with the Deep Packet Inspection, which is the main task of NIDS, is provided so that the reader will obtain a small background to understand the problem which is stated in section 1.2. Section 1.3 gives the goals of this thesis and its contributions. Finally, the chapter concludes with section 1.4 which presents the thesis overview.

1.1 NIDS and Deep Packet Inspection

It was previously stated why network security is a critical issue. Security systems have been created to provide information and network security by filling the holes of OS and by bypassing the misconfigurations of the networks. There are many different kinds of security systems and they are classified basically into two categories: 1) defence security systems (or active and are Firewalls, Intrusions Detection Systems) and 2) malicious software elimination programs (or passive and are Antivirus, Anti-spyware). Since the goals of this thesis are related to a certain defence security system (NIDS), a brief description will be made only about defence security systems.

One of the most famous defence security systems is the Firewall. The Firewall is defined as a piece of hardware or a software program which functions in a networked environment to prevent some communications forbidden by the pre-defined security policy. Its basic task is to control the network traffic between different zones of trust, which are a) the Internet (a zone with no trust) and b) an internal network (a zone with high trust). The ultimate goal is to provide controlled connectivity between zones of differing trust levels through the enforcement of a security policy and connectivity model based on the least privilege principle. However, human intervention is required to decide how to screen traffic and "instruct" the Firewalls to accept or deny incoming packets. Besides, a poorly/incorrectly configured Firewall may reduce the system's effective immunity to attacks. In addition, most of today's security threats are content-based (spamming, email spoofing, worms) and cannot be handled by firewalls. For these reasons Network Intrusion Detection Systems (NIDS) must be used as well to provide more efficient network security. “If a firewall is a doorman, a NIDS is an undercover KGB agent. He silently gathers intelligence and can spot any enemy even if the door security has already let them in” [36].

NIDS are at the first line of defence and work along with the firewalls. They are automated because they do not depend on human’s decisions. However, they must perform Deep Packet Inspection (DPI) in all the traffic packets that traverse the monitored network segment to determine whether they are hostile or not. In DPI, every single incoming packet is firstly classified based on its header field (protocol, source and destination IP, source and destination ports) and the packet’s payload (body) is scanned in deep to determine whether known attack patterns or signatures exist inside it. The combination of header and payload scan determines whether a packet is an intrusion packet or not. Figure 1.3 depicts a potential intrusion packet which hides the threat inside its payload. Consequently, DPI is the only sophisticated way to determine if a packet is malicious or not.

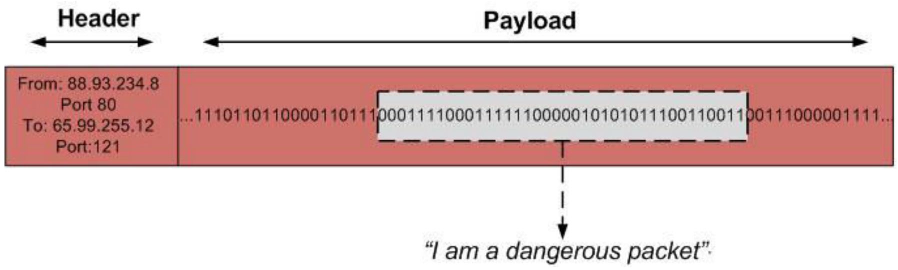


Figure 1.3: A packet has mainly two fields: the header which contains information about source and destination and the payload which contains the data. The attacks have been evolved and the potential threat is hidden inside the packet’s information. For that reason, the entire payload must be scanned in deep to determine whether it is malicious or not.

The NIDS belong to a more general category of security systems which is called Intrusion Detection Systems (IDS), the goal of which is to assist automating the process of intrusion detection. Other popular IDS are the Host-based Intrusion Detection Systems (HIDS) and the Anomaly-based Intrusion Detection Systems (Anomaly IDS), [2]. The HIDS monitor system logs for basic events (i.e. failed login attempts) and kernel messages to find activities that may be potentially hostile but require the installation of program agents on the systems making the administration of them more complex. On the other hand, the Anomaly IDS try to find the problem without knowing specifically the source of it; they detect the intrusions based on heuristics. It is obvious that NIDS are the most efficient IDS especially because of their straightforward concept.

In conclusion, there are many different types of security systems, from which the defense security systems are now considered the most important due to the fact that they can handle better the newly evolved attacks. NIDS are considered the most sophisticated among them because they do not require the administration by a human factor as HIDS or Firewalls, hence making the intrusion detection more automated and they are based on specific rules and not on heuristics like Anomaly IDS. However, their main disadvantage is that Deep Packet Inspection (DPI) must be performed in order to determine if the packet details match any known rule of intrusion. Finally, it must be mentioned here that when referring to a NIDS in this thesis and from now on is a Network Intrusion Detection and Prevention System (NIDS and NIPS).

1.2 Problem Statement

In the previous section, it was explained why NIDS are considered the most efficient IDS. However, it was stated that DPI is needed in order to determine whether a packet is a threat or not because the threat is hidden inside the packet data, as Figure 1.3 depicted. The problem starts from the fact that networks today are becoming faster and faster. Every six months the network bandwidth is doubled [32]. As a consequence, DPI and therefore NIDS must have *high processing throughput* so that they will be really fast and will not be the bottleneck for the network [32]. However, most today's NIDS perform DPI in a brute-force way as it is also shown in Figure 1.4.

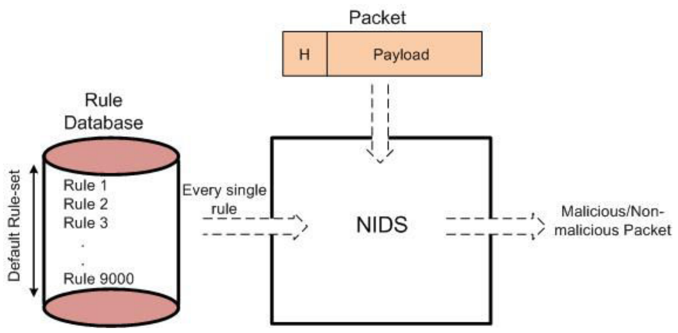


Figure 1.4: The brute-force Deep Packet Inspection performed by most of NIDS. The incoming packet is scanned in deep and every single rule of the database is checked against it. Thus, DPI needs a lot of processing power.

Another important characteristic of NIDS is the *implementation cost*. NIDS can be software programs that run on the General Purpose Processor of the inspected machine or can be a hardware unit that does this specific work. In both cases, implementation cost must be as low as possible. The previous two characteristics are affected significantly by the *scalability* and *flexibility* of the system. A NIDS is a system that will always be updated with new rules to support more and more threats. For this reason, a NIDS must be scalable and flexible in order its implementation cost not to increase and its performance not to be diminished significantly, when the number of rules increases.

An additional problem is related to the *complexity* of the rules. In the past, NIDS rules were much simpler regarding the payload match. Most of the rules had only specific patterns and the purpose was to match the patterns. Nowadays, many rules contain patterns, regular expressions (regex) and payload restrictions like *within*, *offset*, *distance*, *depth*, etc. The regular expressions can describe more than one patterns giving to the NIDS flexibility, while the latter fields set restrictions on where in the packet payload a found content is valid, making rule matching more specific. However, it can be easily realized that these features require significantly more processing and make rule matching a highly computationally intensive task. In addition, year by year or, even worse, month by month intrusions are getting more sophisticated and this requests more sophisticated detection and prevention rules, as a consequence even more computational power is going to be required.

The Multi-stage packet inspection attempts to tackle these problems and alleviate the overheads by dividing DPI into multiple stages. Packet Pre-filtering is one of the proposed ideas that belong to the field of Multi-stage packet inspection and was proposed by Sourdis in [32] and [30]. Figure 1.5 depicts a NIDS that uses Pre-filtering, where DPI is divided into two stages to perform it in a more efficient way. The concept of Packet Pre-filtering is to extract a small portion of each and every rule creating a set of subrules and match the incoming packet against this set instead of the entire rule-set. The set of subrules is kept simple in order to require significantly lower computational power to be processed against the incoming packet. Only the activated rules (by the Pre-filtering phase), which are expected to be a few tens, are sent to the second phase. This phase is more sophisticated and performs the full match of the initial rules against the incoming packet to determine whether it is a malicious or not. The problem is: select a portion of the rule in order to have a correct rule so that there are no false positives; the portion must be small and simple so that a lightweight processing will be needed and efficient enough so that only a few tens of them will be activated and sent to the next (more sophisticated) phase to be fully processed.

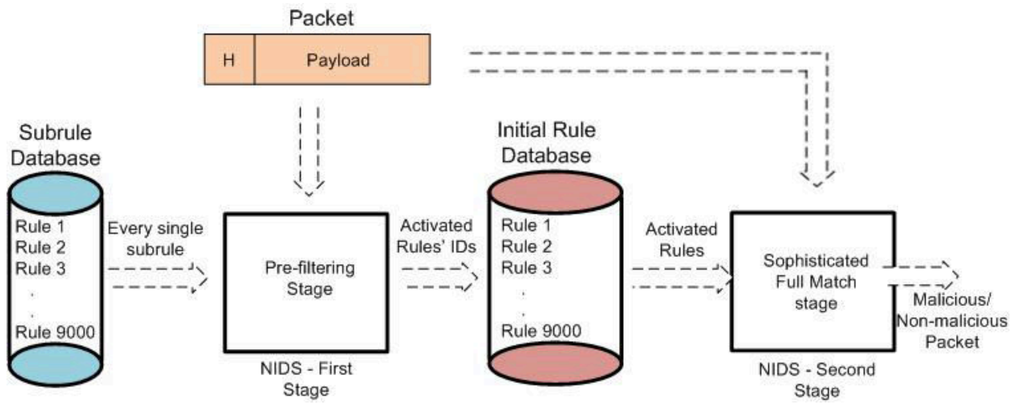


Figure 1.5: NIDS and thus DPI are divided in two stages. In the first, the packet is checked using a set of subrules, which resulted by extracting a small portion of every rule of the original rule-set. The activated rules' IDs are sent to the next stage to access the database that keeps the original rules. Only the activated rules are matched against the incoming packet in a sophisticated way.

Many questions can arise regarding which portion of the rules is better to be selected that will lead to an efficient Packet Pre-filtering stage:

- Are we going to use both header and payload parts of the rule (a rule has header and payload as exactly the packet) and which specific fields of the rule descriptions will be used?
- If a rule contains many same fields (e.g. many static patterns) which one will be selected and based on which criteria?

- What should be the length of the extracted portion?

These are only some of the questions that can arise. This thesis studied thoroughly the concept of Packet Pre-filtering and the above problems in order to discover techniques which extract efficient portions of the rules and other techniques to improve Pre-filtering.

1.3 Thesis Goals and Contributions

In the previous section, it was explained what are the requirements of today's NIDS and the problems that arise. Many ideas have been proposed so far on how to alleviate some of these problems and one of them is Packet Pre-filtering. However, the problem of the Packet Pre-filtering is that a portion of each rule must be extracted so that every single incoming packet will be matched with a set of subrules and the following two requirements must be met:

- the processing task must be lightweight, and
- the fewer the activated rules per incoming packet in the Pre-filtering stage, the better for the required resources of the second stage, if the NIDS follows the parallel processing model (hardware NIDS), or the better the performance if the NIDS follows the sequential processing model (software NIDS).

The goal of this thesis is to find efficient Pre-filtering techniques which extract a portion of each rule of the entire rule-set and other techniques so that the Pre-filtering meets the above requirements and furthermore, is improved and enhanced. The contributions of this thesis are:

- **Extraction techniques**: Extensive study was performed in order to determine which rule's fields are going to be used and how small must be the selected portion. The selected fields are the content (static pattern) and the PCRE (type of regular expression) from the rule's payload part and the whole header part. Three extraction techniques are proposed:
    1. **First Content Prefix**: It extracts the prefix of the content (the first content if more than one per rule).
    2. **PCRE (Perl Compatible Regular Expression) Prefix**: It extracts the prefix of the PCRE field which is a certain type of a regular expression (regex) which was selected for this thesis. Generally, regular expressions (regexes) have a specific syntax and a single one regex is able to describe more than one dangerous contents.
    3. **Unique Part Rule**: It extracts small part(s) from the content description(s) so that the extracted rule-set consists of unique subrules. Using this technique, the number of activated rules per packet that are sent to the second phase of the NIDS is substantially reduced compared to the previous approaches.

- **Rule Correlation**: The idea of correlating rules is proposed and constitutes another contribution of this thesis. This technique achieves to correlate rules based on their internal features and can be used in combination with any extraction technique. In the thesis, this technique is used along with the Unique Part Rule extraction technique and is applied to these rules, of which a portion cannot be extracted and give a unique subrule.

- **Smart Rule Reuse**: This technique improves the second stage of a hardware NIDS that uses Pre-filtering and is based on the idea that there are same activated rules between consecutive packets. Thus, it monitors the Processing Elements (PEs) of the second stage to be aware of which one processes which rule so that if there are common activated rules between two or more consecutive packets, it will not be needed to re-download these specific rules onto the PEs but only the new ones.

- **Evaluation using the SNORT NIDS**: The above techniques have been implemented and the three first have been applied on real rules (SNORT rule-set). A final contribution is that all the above techniques have been evaluated loading the respective extracted sets of subrules on a real open-source NIDS which is SNORT, [26], using real attack traffic input traces.

The goals and the contributions of this thesis were presented. Next section provides with the overview of this thesis.

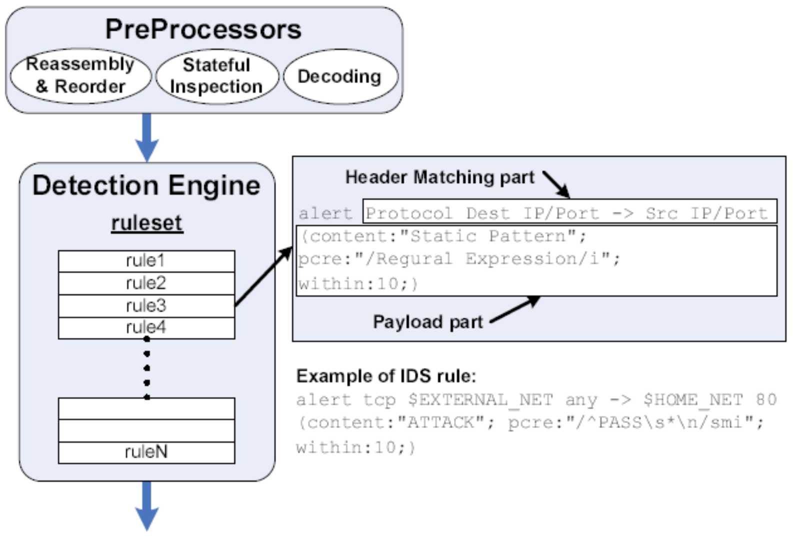
1.4 Thesis Overview

The introductory chapter ends with the thesis overview. The thesis is organized as follows. In Chapter 2, the concept of Packet Pre-filtering will be discussed thoroughly along with a brief description of the related work on the field of multi-stage NIDS. Also this chapter gives a more extensive description of NIDS and discusses the SNORT NIDS that was selected to implement the proposed techniques as mentioned in the previous section.

Chapter 3 described the proposed Packet Pre-filtering techniques. This chapter discusses the idea of each one of them and the algorithms that were constructed. Furthermore, Chapter 4 evaluates our Packet Pre-filtering techniques of Chapter 3 using the measurements taken in SNORT. A comparison between the techniques is presented. Finally, Chapter 5 concludes this thesis summarizing its contributions, conclusions and future works.

2 Background

Network Intrusion Detection Systems are considered as the most efficient IDS due to the fact that they have more efficient concept and are more automated comparing them to the other IDS types. The concept behind a NIDS is to perform Deep Packet Inspection which scans the packet in deep and compares the details of the header and the payload of the packet to the respective fields of the rules of the intrusion rule-set that is normally stored in a database.



```
alert Protocol Dest IP/Port -> Src IP/Port
(content:"Static Pattern";
pcre:"/Regural Expression/i";
within:10;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 80
(content:"ATTACK"; pcre:"/^PASS\s*\n/smi";
within:10;)
```

Figure 2.1: A typical NIDS. It consists mainly of two parts: 1) the Pre-processing unit that performs tasks like reassembly, decoding, etc and 2) the Detection Engine. The Detection Engine contains the database of the know intrusion rules and consists of the header matching part and the payload matching part. (Source: [32]).

A typical NIDS is depicted in Figure 2.1 and mainly, consists of two parts: the Pre-processors and the Detection Engine which are described below:

- *Pre-processors*, [32]: The initial tasks that must be performed, before DPI takes place in the Detection Engine, are fulfilled in the pre-processing unit. Pre-processors are working in a higher level than the Detection Engine. They try to decode various kinds of traffic like Telnet, HTTP, SMTP, FTP, that are encoded differently. They perform also the stateful inspection task which tries to discover abnormalities, like Denial of Service (DoS) attacks, in a higher level. The goal of the DoS attacks is to destroy the functionality of the system and prevent it from working properly or working at all. Examples of DoS attacks are buffer overflows (bomber email, etc) or protocol attacks. Finally, Pre-processors perform the task of reassembling and reordering packets because many attacks span on multiple packets.

- *Detection Engine*: The Detection Engine is the part of the NIDS that implements the DPI. Every packet has a header and a payload to be scanned, thus the detection engine is also divided in two parts: 1) Header matching part and 2) Payload matching part.

Header matching part: The header of the packet contains information about the protocol, the address of origin of the packet, the destination address and the source and destination port. The address and port can also be a range instead of a specific number, [32]. In the past, the Header Matching part was based on the brute-force search as its task was to match explicitly every header part of a rule with the header information of the packet. In the recent years, the fact that more attacks use more sophisticated techniques like range IP addresses and ports, forces Header matching part to be more sophisticated by using packet header classification. There are many proposed algorithms and techniques for header packet classification in [41], [14], [35] and other that are implemented also in reconfigurable hardware ([12], [20], [28] and [29]).

Payload matching part: The payload of the packet is the actual data that it transfers. A packet is a vehicle of attack, if a specific sequence or sequences of tokens are contained inside its payload. The payload part of an Inspection Engine analyzes the payload part of the rules and try to find whether one or more of the rules' descriptions match the packet. The most common technique was to match literally the content (static pattern) of a rule with the packet payload. However, static pattern matching is not sufficient. Regular expressions can also be employed to describe malicious packet contents. Regular expressions describe dynamically many sequences of tokens that can exist inside a packet. Additionally, there are also other fields like *offset*, *within*, *distance*, *depth*, *byte_jump*, which are called payload restrictions and are described in more detail in section 2.3. These fields put extra constraints regarding the placement of malicious contents inside packet's payload and make pattern matching more accurate. However, all these new features make payload matching more computationally intensive.

A significant amount of research has been done in order to design efficiently units that perform these tasks with high performance and low implementation cost maintaining the bandwidth of the network. A different and relatively new concept is the Multi-stage packet inspection. The Multi-stage packet inspection approach attempts to solve the problem of high computations during packet inspection by dividing it into multiple stages. Packet Pre-filtering idea, which was proposed by Sourdis in [32] and [30], follows this concept.

Packet Pre-filtering: A potential NIDS that uses Packet Pre-filtering consists of two stages. The first stage is the Pre-filtering stage which is loaded with a lightweight set of subrules. This set is resulted by preprocessing the initial rule-set and extracting a small portion of each rule of it, based on the observation that every single incoming packet may match only a few tens of rules and not the entire rule-set. Consequently, the second stage of that NIDS is a more sophisticated Full Match engine that fully matches only the activated rules from the first phase, against the incoming packet. Figure 2.2 depicts a NIDS that uses Packet Pre-filtering. Packet Pre-filtering is loaded with the set of subrules which requires less processing but it is sufficient to exclude the majority of the intrusion detection rules from further processing. This thesis attempts to develop efficient techniques which extract the set of subrules and other techniques to accommodate Pre-filtering to meet the above requirements and generally to improve and enhance its concept.

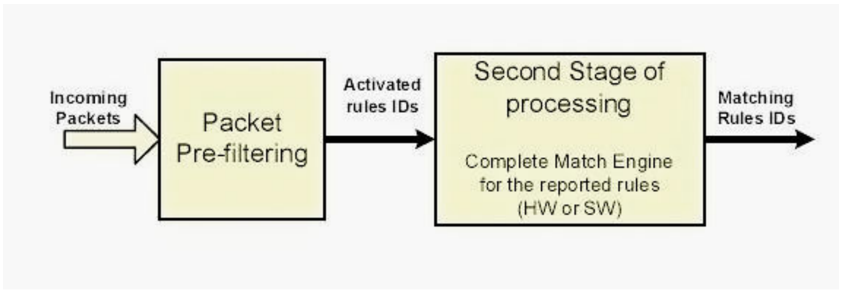

Incoming Packets
Packet Pre-filtering
Activated rules IDs
Second Stage of processing
Complete Match Engine for the reported rules (HW or SW)
Matching Rules IDs

Figure 2.2: A two-phase NIDS that uses the Pre-filtering idea. The Pre-filtering stage is loaded with a lightweight but still sophisticated subrule-set extracted by the original one, according to a specific technique. The IDs of the activated rules of the Pre-filtering phase are sent to the second phase where the respective full rules will be matched against the incoming packet. (Source: [32]).

This chapter discusses briefly the related work in section 2.1 and describes thoroughly the concept of Pre-filtering in section 2.2. The work of this thesis is based significantly on this concept, as it was previously mentioned. In addition, section 2.3 describes why SNORT was selected among the other NIDS for this thesis and describes its main characteristics. Finally, the chapter concludes with a small summary in section 2.4.

2.1 Related Work

Significant research has been done in the field of NIDS especially the last 10 years. Many algorithms, software programs and hardware units have been proposed which attempt to detect intrusions using sophisticated ways maintaining the performance or/and keeping low the implementation cost. In addition, there are few proposed ideas which split the pattern matching into multiple stages but almost none, to the best of the author's of this thesis knowledge, follow the concept of multi-stage packet inspection. The difference between multi-stage packet inspection and multi-stage pattern matching is that the former include both header and payload fields of the rules, while the latter is limited to the static patterns.

One related work was proposed by Dharmapurikar et al. in [11] and [4]. The authors utilized Bloom Filters to predict if specific patterns are matched by an input packet and use only the predicted patterns for full matching in the next phase. The first phase of the system consists of parallel Bloom Filters. A Bloom filter is a data structure that stores a set of patterns compactly by computing multiple hash functions on each member of the set. With this technique, a database of patterns is queried for the membership of a particular one per packet. However, Bloom Filters allow false positives and the next phase must analyze and match more predicted patterns than the actual number. Additional related works that follow the concept of multi-step matching are [25] by Ramaswamy et al. and [24] by Rabin. They propose the approximate fingerprint, where fingerprints of pattern prefixes are constructed. A fingerprint is actually a small tag for a larger pattern. Then, the packet payload is examined to probabilistically determine whether it contains a known attack pattern. The Rabin fingerprints which are the firstly introduced fingerprints result by computing the modulo function of a pattern and a predetermined irreducible polynomial. The result of using approximate fingerprints is that a (small) fraction of the whole set of patterns (about 25%) is excluded from further matching in the next phase.

Another related work that follow the idea of multi-phase payload matching is [19], proposed by Markatos et al., where a two step approach is followed. In the first step, it is searched if the packet payload contains all the possible patterns of a rule in arbitrary positions and if so, full pattern match of the candidate rules is performed in the next step using conventional pattern matching algorithms (Aho-Corasick [1], Boyer-Moore [9], Gusfield [15], Wu-Mander [45]). However, their proposed idea is limited only to static pattern matching. Another disadvantage is that many rules can have more than one contents (patterns) inside one single rule and their pattern matching will need a lot of processing. For example, a SNORT rule of July 2008 has in average 4 static patterns. The same authors have also proposed Piranha, in [3], where again payload matching is divided into two steps. In the first step, a different than the original rule-set is used like in our Pre-filtering. Every rule's content is represented by its less frequent 4-byte substring and if the incoming packet's payload contains this sequence of characters, only the matched rules are fully matched in the second phase of Piranha. Again their study is limited only in static patterns and furthermore, many rules may contain smaller than 4 bytes contents. However, Piranha achieves 28% higher speed than SNORT v2.2 and 73% less memory.

This thesis attempt to exploit the Pre-filtering by extracting a set of small and simple subrules so that only few of them will be activated and need to be fully matched in the second stage, instead of approximately 7000 (75%) rules which are needed using the SNORT rule-set of July 2008 and Rabin's or Ramaswamy's fingerprints. In addition, our proposed Pre-filtering techniques are not only limited to static patterns like all the previously mentioned related works, but exploit also the rules' header and one of the proposed techniques attempts to exploit also regular expressions (PCRE). Finally, all the proposed extraction techniques attempt to create a set of subrules which is loaded into the Pre-filtering stage, so that the least possible processing will be needed and the smallest number of activated rules will be sent to the next phase.

All the related works that were presented until now discussed and attempted to exploit the idea of multi-stage pattern matching. Most of the researchers propose innovative ideas on how to construct an efficient first stage of a multi-stage pattern matching NIDS. However, it is important the second, third, etc stages to be also efficient since they affect the total performance/implementation cost of the NIDS. For that reason, sophisticated algorithms must be used instead of the brute-force search, considering also the fact that rules are getting more and more sophisticated and/or complex, and the system's performance may degrade even if the number of rules, which must be examined, is small.

Many known algorithms can be used for the Full Match stages. Aho-Corasick in [1], Boyer-Moore in [9], Gusfield in [15], Knuth-Morris-Pratt in [18], Shift_OR in [5] and Wu-Mander in [45] are some of the proposed algorithms. All of them can be used in both software and hardware NIDS. In addition, there are many proposed hardware units that can be used in the later stages of hardware multi-stage NIDS to efficiently fully match a static pattern or a regular expression. Static patterns can be efficiently matched using 1) Content Addressable Memories (CAM) and discrete comparators (some of the proposed are Gokhale et al. in [13], Baker et al. in [6] and Sourdis et al. in [34]) and 2) Hashing (some of the proposed are the Perfect Hashing Memory proposed by Sourdis et al. in [31] and a Hashmem architecture based on CRC polynomial hash functions [22] and [23]). Finally, the most known techniques for efficient regular expression matching are the Deterministic Finite Automata (DFA, some of the proposed are: Moscola et al. in [21] and Baker et al. in [7]) and the Non-deterministic Finite Automata (NFA, some of the proposed are: Sidhu et al. in [27], Sourdis et al. in [33]). All the presented ideas of this paragraph had been proposed in one-stage NIDS where the whole packet (header and payload) is proceeded for matching from the very beginning. However, they can be used in the later processing stages of NIDS that uses Pre-filtering or one of the previously proposed multi-stage pattern matching ideas.

In conclusion, all the multi-stage pattern matching proposed ideas share the same concept: reduce the number of candidate rules that must be fully matched. There are several different ideas which are all based on the same concept. However, Packet Pre-filtering is not limited to static patterns but exploit regular expression as well and the header of the packet. It has not false positives and the number of the activated rules per incoming packet can be just a few tens. These characteristics motivated this thesis to discover efficient techniques that attempt to enhance and improve Packet Pre-filtering.

2.2 Packet Pre-filtering

Packet Pre-filtering follows the concept of the multi-stage packet inspection and divides the Deep Packet Inspection into two stages (phases). It is based on the observation that a single incoming packet will fully or partially match just a few tens of intrusion detection rules out of the thousand ones, and was proposed by Sourdis in [32]. It must be noted here that the fact of matching a few tens of rules during Packet Pre-filtering is assisted also by header matching because a number of rules will be excluded due to a header mismatch.

In phase one, the incoming packet enters the Pre-filtering engine where a small portion of each rule of the intrusion detection rule-set has been selected according to a specific technique. The Pre-filtering engine chooses the activated rules that were partially matched during the first phase and sends them to the second phase of the packet inspection engine. In the second phase, only these activated rules are matched fully against the incoming packet. A typical two-phase NIDS that uses Packet Pre-filtering is depicted in Figure 2.3.

The Packet Pre-filtering engine is shown in more detail on the bottom of this figure.

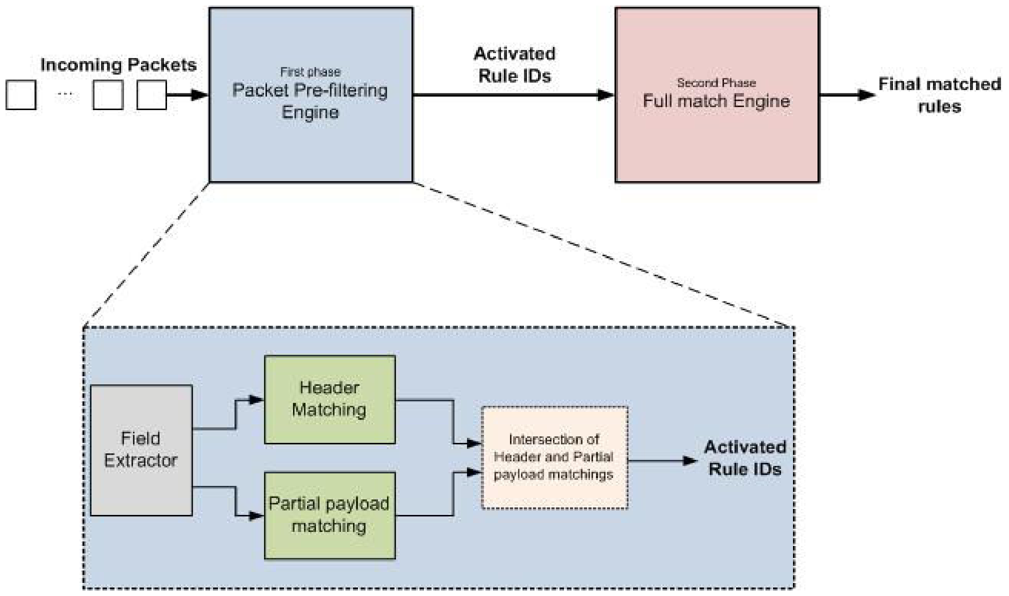


Figure 2.3: A two-phase NIDS that uses the Pre-filtering idea. The Packet Pre-filtering engine consists of the field extractor, the header and payload matching units and a unit that finds the intersection of the output activated rules from the previous matching units.

It consists of the field extractor, the header and payload matching units and a unit that finds the intersection of the activated rules that were output from the previous matching units. The packet enters into the Field extractor and its header field and the payload fields which are needed in the partial payload matching unit are extracted from the packet. The header field enters into the header matching unit while the other fields in the payload matching one. Each unit of them outputs the rules' IDs (if any) that was matched and the intersection of them will be selected to be proceeded to the second stage.

The challenging part of the Pre-filtering is to find a efficient way to extract a small portion of each rule of the original intrusion detection rule-set so that the number of activated rules in the Pre-filtering stage would be as small as possible and the internal design of it would be trivial, with low implementation cost and with high performance. This section is divided in two subsections. In the first one, the theoretical analysis for Packet Pre-filtering, as it was proposed by Sourdis in [32], is briefly presented. Afterwards, subsection 2.2.2 present system cases that can use Packet Pre-filtering.

2.2.1 Theoretical Analysis for Packet Pre-Filtering

This section presents the theoretical analysis for Packet Pre-filtering, as it was performed by Sourdis in [32]. The concept of Packet Pre-filtering is to use a sophisticated and tailored rule-set so that the number of activated rules per packet is the minimum possible. For that reason, the target of the theoretical analysis is to bound the probability of reaching the maximum number of activated rules and find also the probability to overcome it.

The upper bound of this probability is given by the following equation (Equation 2.1), where m is the length of a substring used by Pre-filtering and n is the length of the packet's payload.

$$P_{all} = \prod_{i=0}^{MAX\_rules} p(m, n - m * i), n > m * MAX\_rules \tag{2.1}$$

The probability p is given by the Equation 2.2. The probability c is the probability to find any character used by Pre-filtering in the packet's payload while m and n have already been defined above.

$$p(m, n) = 1 - (1 - c^m)^{n-m+1} \tag{2.2}$$

Consequently, according to Equations 2.1 and 2.2, and based on the value of c and MAX_rules it can be found how big is the possibility to match more than MAX_rules. It is a good point to mention that in the theoretical analysis only portions of static patterns are selected for the Packet Pre-filtering phase. The selected portion can be either 1) the prefix of a static pattern or 2) a part/parts of the static pattern(s). In the first case, the total latency may be smaller because only the remaining part of the rule is needed to be matched in the second phase of processing. A possible disadvantage is that many rules may have the same prefix and consequently, more rules are activated. On the other hand, the second case is more flexible but the whole pattern will be needed to be checked in the second phase of processing leading to presumably more latency.

Two scenarios are described by the author for the evaluation of the theoretical analysis:

1. *Uniform traffic scenario*: In this scenario, uniform (random) traffic was selected. The probability to find each character is equal: $c = \frac{1}{256}$, where 256 is the total number of characters.

2. *Pessimistic traffic scenario*: In this scenario, the probability c changed from equal, to the probability of the most often character. It is 0.1 and was selected among several probabilities from other related works for the worst case scenario.

In both scenarios, the number of MAX_rules was selected to be 32 or 64. The subrules, which are loaded on the Pre-filtering stage, were extracted using the prefix of the static pattern. Also, the packet payload size was selected to be larger than usual (512 bytes-1 MB). Figure 2.4 gives the chart of probability as a function of packet's payload size for different c probabilities, different prefix lengths and different numbers of MAX_rules. It is obvious from the figure and the Equations 2.1 and 2.2 above that if the number of MAX_rules or the number of prefix characters (m) increases, the probability decreases and when the payload size of the packet increases, the probability increases as well. In the first traffic scenario (uniform) the biggest probability to overcome MAX_rules is for the case of prefix of 4 characters with $MAX\_rules = 32$ and was found to be a little less than 10^{-100}. On the other hand, for the pessimistic traffic scenario, it was found that for m=4 the probability is below 10^{-3} for big payload packet sizes (16KB) and is smaller than 10^{-6} for m=6.

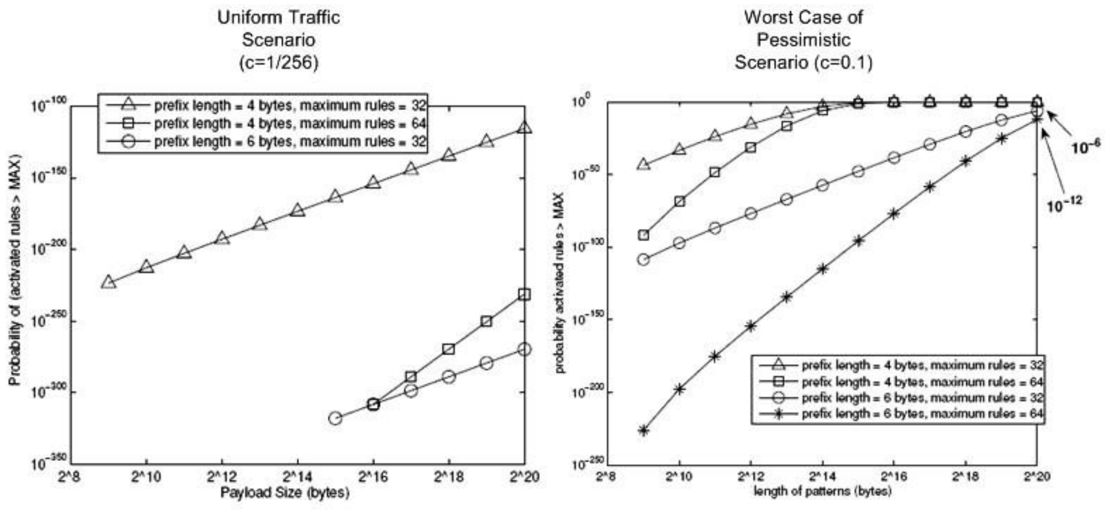


Figure 2.4: The figures show the probability of overcoming the MAX_rules during Pre-filtering as a function of packet payload size. The left figure is for uniform traffic while the right one is for the worst case ($c = 0.1$) for the pessimistic traffic scenario.(Source: [32])

In conclusion, Packet Pre-filtering is an idea that attempts to reduce the number of activated rules. It was seen that reducing the MAX_rules number of activated rules increases the probability to overcome it. It would not be wise to increase the MAX_rules to reduce the probability. Furthermore, the big packet payload increases the probability but it cannot be changed. Finally, increasing the size of the portion of the subrules may be unacceptable in terms of the performance of Pre-filtering stage. Consequently, efficient techniques must be found in order to extract (small) portions of the rules in order the maximum number of activated rules to be kept in low levels and the probability to reach and overcome this number would be as small as possible.

2.2.2 Integration of Packet Pre-Filtering

The idea of packet pre-filtering can be easily applied to both software (SW) and hardware (HW) NIDS. Software NIDS process the rules sequentially in a multi-threaded fashion on one or more, if available, processing cores. Packet pre-filtering will decrease the number of sequential full matches since only a few tens of rules will have to be checked. Thus, and taking into account that the partial match will perform a lightweight processing and consequently have very small latency, the total latency of the processing of a NIDS per packet will be smaller. On the other hand, in hardware, it is possible to match many rules in parallel using multiple Processing Elements (PE). Thus, the problem of area cost is much more significant than speed. By applying Packet Pre-filtering, the number of PEs in phase two will be much smaller while the cost of the first phase will not be significant since a small portion of each rule is extracted.

Sourdis proposed, in [32], a packet inspection engine which is called PINE, where the role of Packet Pre-filtering is more clear and is stated why the second stage engine

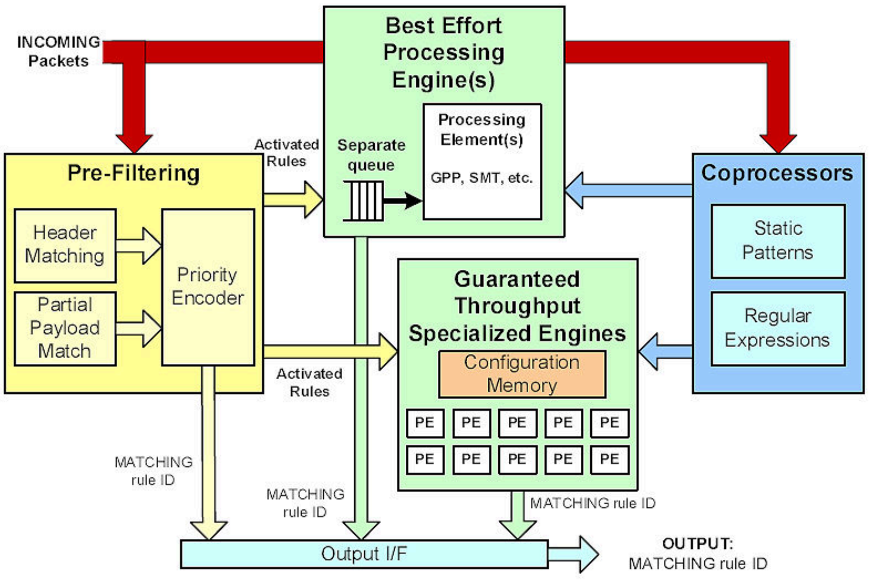


Figure 2.5: The Packet INspection Engine (PINE), proposed by Sourdis. PINE is a two phase NIDS that uses the Packet Pre-filtering. Packet Pre-filtering is the first stage unit, while the second phase unit contains the Best Effort Processing Engines (BEPE) and the Guaranteed Throughput Specialized Engines (GTSE). The coprocessors that implement the fully static pattern matching and full examination and match of regular expressions are centralized and do the job that is given to the Processing Elements. (Source: [32]).

depends significantly on the engine of the first phase. PINE is depicted in Figure 2.5. The Packet Pre-filtering unit is implemented in a hardware coprocessor. The second phase of processing is performed by the Best Effort Processing Engines (BEPE) and Guaranteed Throughput Specialized Engines (GTSE). The GTSE has a specific number of PEs and each one of them can process an activated rule that was proceeded by the Pre-filtering unit. It must be noted here, that the PEs do not perform payload matching because payload matching is performed by the centralized coprocessors (there are for both static pattern matching and regular expressions). In case a packet activates more rules than the number of PEs in GTSE, it is sent to BEPE in order to fully match all the activated rules. The BEPE unit exists for the few cases that more than a normal number of rules is activated and in order not to drop the specific packet or to create a bottleneck inside the engine, the packet is queued and processed in the BEPE. On the other hand, other packets that activate few rules are processed in the GTSE, maintaining the processing throughput and consequently the network bandwidth.

In summary, Packet Pre-filtering can be used in both software and hardware NIDS achieving lower latency in the software NIDS and smaller implementation cost and better performance in hardware NIDS. PINE is a proposed hardware NIDS that accommodates Pre-filtering. The Full Match engine consists of two processing blocks based on the number of activated rules. If an incoming packet activates fewer rules than the number of PEs, it enjoys the Guaranteed Throughput SEs, otherwise it is sent to BEPE.

2.3 SNORT

Currently, many NIDS have been created and are used. SNORT [26], Bleeding [43], Firestorm [36], Untangle [39], Bro [44] are some of them. SNORT, Untangle and Bro are open-source while Firestorm is free-software. Currently, the most widely used open source NIDS is SNORT because it has more features than the other ones making it a more complete system; it is lightweight and its pattern matching algorithm is relatively fast (for SW implementation) and it may have an easier way to describe intrusions and write intrusion rules. For these reasons, SNORT was selected among the other NIDS for the evaluation goals of this thesis.

SNORT NIDS, which was created by Martin Roesch, is capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, OS fingerprinting attempts, and many more [42]. In addition, SNORT can run in many modes [26]. In sniffer mode, if someone wants to view the packets' details (header only or header and data) onto the screen. In packet logger mode, it stores the packet into the disk. The packets are recorded in log files which can have binary format or plain ASCII text format. Another mode which is the mode that was used for the purposes of this thesis is the NIDS mode where a traffic trace, which has a tcpdump format, is read by SNORT using the snort.conf configuration file. In this mode, the user can select among several options about alerting (fast, full, none and other). Instead of the NIDS mode, the inline mode can be used where the SNORT obtain the packets through iptables and use new rule types to make decisions like pass, drop, reject. Finally, SNORT has an interesting module, which is called performance profiling, that was used in this thesis. Performance profiling gives statistics about the number of times a rule was matched or alerted, processing time that was needed, etc.

In summary, the SNORT NIDS was selected for the experimental purposes of this thesis. It is open-source and full of characteristics many of which were used during the experimental phase. This section is divided in two subsections. The first one presents briefly the SNORT preprocessors while the second one shows in detail the structure of the SNORT rules on which chapter 3 is based on.

2.3.1 SNORT preprocessors

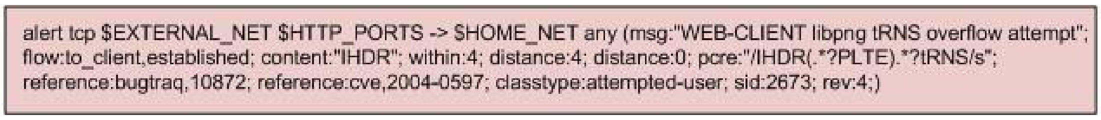
SNORT consists of the Preprocessors and the Detection Engine as every typical NIDS, as it was mentioned at the beginning of this chapter. In SNORT, the Preprocessors run before the Detection Engine but after the decoding of a packet. SNORT has Preprocessors for FTP, Telnet, HTTP, SMTP decoding for user applications as referred in the preprocessing part at the introduction of this chapter.

Among the other Preprocessors, Frag3, Stream4, Flow and Stream5 are considered the most important. The Frag3 preprocessor performs the defragmentation of packets using fast data structures. Another characteristic of Frag3 is the target-based analysis where the SNORT tries to find an intrusion target in a network, based, for example, on the IP defragmentation on this specific target. This characteristic is totally new for the NIDS.

Stream4 provides TCP and UDP stream reassembly and stateful analysis to SNORT. Currently, it handles 8192 simultaneous TCP connections and can be scaled to handle 100,000. The Flow tries to keep in pace SNORT's mechanisms during stateful inspection. Finally, Stream5 can be used as an alternative to Stream4 and Flow. Except the characteristics of Stream4 and Flow, it can provide also target-based analysis and anomaly detection which are not, usually, operational characteristics of NIDS.

2.3.2 SNORT rules

As it was mentioned at the beginning of this chapter, the Detection Engine of every typical NIDS and thus SNORT consists of the header matching unit and the payload matching unit. It is interesting to discuss now the format of IDS rules that are used in SNORT because it is a significant topic of this thesis, since some of the proposed Pre-filtering techniques, which are explained in Chapter 3, are evaluated being applied to SNORT rules.



```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"WEB-CLIENT libpng tRNS overflow attempt";
flow:to_client,established; content:"IHDR"; within:4; distance:4; distance:0; pcre:"/IHDR(.*?PLTE).*?tRNS/s";
reference:bugtraq,10872; reference:cve,2004-0597; classtype:attempted-user; sid:2673; rev:4;)
```

Figure 2.6: An example rule taken from SNORT rule-set version 2.8 of July 2008.

An example of SNORT rule is shown in Figure 2.6. As every typical IDS rule, it consists of two parts: header and payload options. The header contains the rule's actions, protocol, source and destination IP, source and destination port and the direction/bidirection operator. Rule's actions can be alert, pass, drop, log, activate but alert is used most. The protocol can be TCP, UDP, IP and ICMP and more protocols (ARP, IGRP, etc) are going to be added in the future. The source and destination IP addresses are mainly ranges and not specific addresses in most of rules. For example, in Figure 2.6, the source IP is $EXTERNAL_NET while the destination IP is $HOME_NET. $EXTERNAL_NET and $HOME_NET are initialized in the configuration file snort.conf based on the network properties. Similarly the source and destination ports are configured. Ports are specified in many ways as well: by numbers, ranges or negation. Finally, the direction/bidirection operator describes the direction of the traffic that the rule applies to.

The rule's payload options part contains much more information than header. It is the part that is used to analyze and check the payload of the packet. It contains features just for the system's organization (general rule option) like sid, msg, classtype, reference, rev, flow, etc and other parts like payload (content, PCRE, within, distance, offset, byte), non-payload (dsize, icmp_id, icmp_seq, itype) and post-detection. The last one is out of the scope of this thesis. Starting from the general rule options, the most important of them are:

- *msg*: The msg is the message that is printed during alerting or logging.

- *reference*: The reference field gives the opportunity to include external attack identification systems.
- *classtype*: The classtype keyword is used to categorize a rule as detecting an attack that is part of a more general type of attack class.
- *gid*: The generator id describes which part of SNORT generated a single event. For example, if it is one it specifies that the event came from the rule subsystem, while if it is greater or equal to 100 it has been designated for preprocessors.
- *SID*: Every rule must have a unique Signature ID (SID), in order to be identified.
- *rev*: The rev keywork is used to uniquely identify revisions of SNORT rules.

The most important Payload options are:

- *content*: The content is the static pattern. It is the most usual and the most important field of the payload part of a rule. In SNORT, the content field is matched to the packet's payload. It can contain characters, numbers, symbols, and even hexadecimal values inside pipe operators.
- *uricontent*: It is like content but it searches the normalized request Uniform Resource Identifier (URI) field. In other words, it searches for things that come normalized out of the URI buffer. The URI is a compact string of characters that is used to identify or name an Internet resource.
- *depth*: The depth keyword specifies how far into a packet SNORT should search for the specified pattern. Content keyword must precede depth keyword.
- *offset*: The offset keyword specified where to start searching for a specific content inside a packet. Content must precede again offset field.
- *within*: The within keyword makes sure that at most N bytes are between pattern matches. It is used in conjunction with the distance field.
- *distance*: The distance keyword specifies how far into a packet SNORT should ignore before starting to search for the specified pattern relative to the end of the previous pattern match.
- *byte_test*: It is capable of testing binary values or converting representative byte strings to their binary equivalent. It usually tests a byte field against a specific value using an operator like <, >, =, bitwise AND, bitwise OR, etc.
- *byte_jump*: The byte jump keyword allows rules to be written in that way that skip over specific portions of length-encoded protocols and perform detection in very specific locations.
- *PCRE*: This field allows rules to be written in Perl Compatible Regular Expressions (PCRE). PCRE have a specific syntax that is described in section 3.1 of Chapter 3. PCRE can describe more than one patterns making payload matching more flexible but also more computationally intensive.

Finally, the fields of the non-payload part of the rule are stated. The *dsize* field tests the size of the packet's payload to detect abnormalities like buffer overflow. The *itype* field make the engine check for a specific icmp_id while the icmp_seq checks for a specific ICMP sequence value and is used only for ICMP protocol packets.

Table 2.1: This table shows the number and the ratio (in %) of rules that contain only content, only PCRE, both content and PCRE fields or just a header field for the SNORT rule-set, of July 2008 (version 2.8).

Case	Number of Rules	Ratio (in %)
Content only	2921	32.68%
PCRE only	8	0.09%
Both Content and PCRE	5883	65.83%
Just header	125	1.4%
Total # Rules	8937	100%

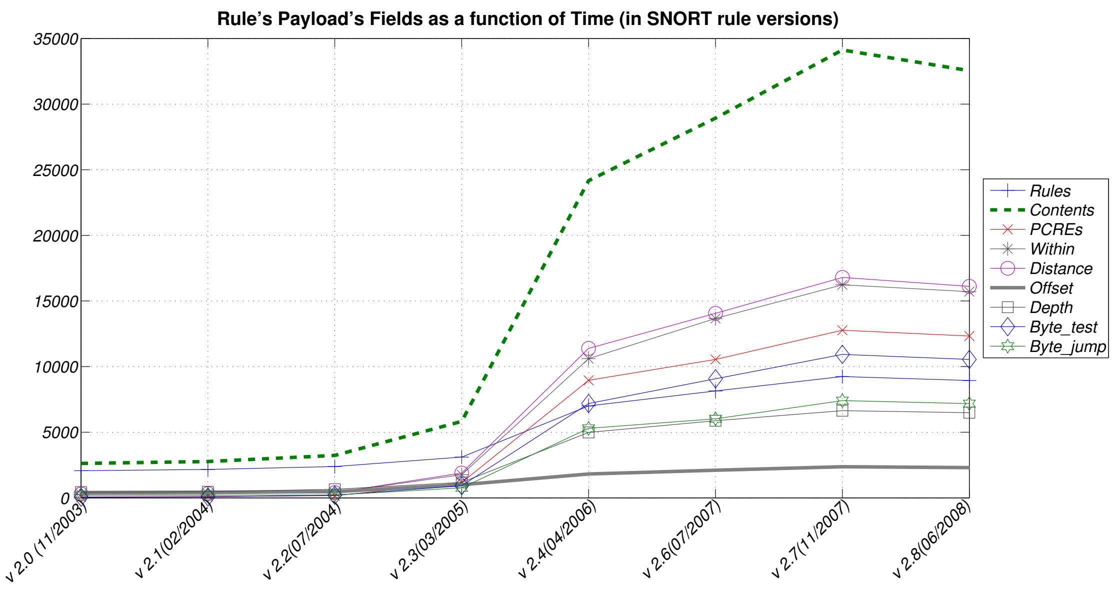


Figure 2.7: How the rule's payload fields have increased during time. Time is given in terms of SNORT rule versions. It is obvious how much workload SNORT has in order to check every rule for each incoming packet especially in the latest rule versions (after v2.6).

Table 2.1 shows the number of rules that have only header, only content, only PCRE or both fields for the rule-set of 2008 that was used during this thesis. This rule-set consists of 8937 rules. The 98.5% of the rules have both header and payload fields to be matched while about 66% of them has both content and PCRE fields. In addition, Figure 2.7 presents the number of total rules and the usage of each payload field for every SNORT rule-set from 2003 to 2008. Someone can observed that until 2005, the number of rules was around 2500 and the payload fields were few. However, after 2005, the number of rules has been almost doubled, reaching 7000 rules in 2006. All the fields and especially the number of contents have increased significantly. Finally, a small decrease can be observed in all the fields and the total number of rules, going from version 2.7 to 2.8. Perhaps, some rules were merged or others might be proved incorrect. From this figure and for the rule-set of July of 2008 which is the most recent, it can be determined how much workload the SNORT has and how much computation is needed in order to check all these fields for one single incoming packet. For example, there are around 32000 contents (static patterns) in 8804 rules (the rules with header and payload) that contain content which means almost 4 contents per rule on average. In addition, almost 50% of contents must be matched using *distance* and *within* fields making the needed computation very high.

In conclusion, it is now more obvious why payload matching is considered highly computational intensive. If the Content (static pattern) alone could efficiently describe malicious packets, the processing would be lightweight. By using the payload restrictions, payload matching is more precise but more computational power is needed. Furthermore, PCREs are from their nature more computationally intensive since they can describe many different strings. Except these fields, there are also some other fields like sid, msg, etc which are used in the internal structure of SNORT. Finally, efficient Packet Pre-filtering requires to select those fields which will provide a small number of activated rules and simultaneously low processing in the Pre-filtering stage.

2.4 Summary

In this chapter, the background on which this thesis is based, was presented. At the beginning, a typical NIDS was presented and its main operations were stated and described. The problem of the computationally intensive tasks that must be performed by NIDS especially, during payload matching was stated. Afterwards, a brief description of the Packet Pre-filtering was performed.

In section 2.1, it was briefly described the related research on the field of multi-stage pattern matching. The above does not exploit header and is limited only to static patterns. The Packet Pre-filtering, on the other hand, belongs to the field of multi-stage packet inspection and attempts to exploit more rule's features. In all the related works, the pattern matching is divided into two phases and several ideas are proposed on how to use a different rule-set version which requires less processing and only few rules will be the activated to be fully analyzed against the incoming packet in the second phase. However, all of the related works are limited to employ static patterns. On the other hand, the Packet Pre-filtering provides a more generic concept and every rule's field can be used in order to find a small portion of every single rule of the rule-set to create the set of subrules, for the Pre-filtering stage. A brief reference was made also to algorithms and hardware units that could be used to implement the second phase of a NIDS that uses multi-stage packet inspection.

Section 2.2 presented the idea of Packet Pre-filtering. Packet Pre-filtering is the first step of processing where a small portion of each rule is selected and analyzed against every single incoming packet. The activated rules of phase one are sent to the next phase to be fully matched with the packet. In this section, the theoretical analysis of Pre-filtering was also presented. It proves that the probability to overcome the maximum number of rules is very small. Furthermore, the theoretical analysis gave the motivation to find efficient Pre-filtering techniques that keeps the maximum number low or otherwise, keeps the probability of reaching or overcoming this number in very low levels. Finally, it was briefly discussed which systems and how, can accommodate Pre-filtering.

Finally, in section 2.3, it was mentioned why SNORT was selected among the several NIDS for the experimental purposes of this thesis. The main rule's characteristics were presented and extra discussion was made on the payload options of rules that cause the high computations in SNORT and generally in a typical NIDS. The section concluded with some quantitative details about the SNORT rule-set of July 2008 that was used for the purposes of this thesis and generally how the usage of payload fields has increased in the past 5 years.

Packet Pre-filtering Techniques 3

Deep Packet Inspection is the main task of Network Intrusion Detection Systems. It classifies the packets based on their header fields and the packet's body (payload) is scanned deeply and compared with known attack signatures to determine whether it is malicious or not. For that reason, DPI is considered high computationally intensive and may create a significant bottleneck in the network bandwidth. The Packet Pre-filtering ([32], [30]) preserves the high processing throughput and the network bandwidth by dividing the DPI into two stages. The idea behind Packet Pre-filtering is to use a small portion from every rule of the rule-set and check this against the packet. This requires more lightweight processing and is performed in the Pre-filtering stage (first stage). Afterwards, the second stage performs the full match of only the activated rules from the first phase against the incoming packet to determine whether it is hostile or not. Consequently, the goal is to select a small and efficient part of the rules and create a set of subrules (loaded in the Pre-filtering stage) which requires low processing and the number of the activated rules is the smallest possible. In section 2.2.1, it was shown that Pre-filtering is efficient, if the probability to overcome the number of maximum activated rules is kept very low, having a reasonably small maximum number of them. Thus, the subrules must differ with each other in order for the number of activated rules per packet to be as few as possible. To summarize, the Pre-filtering requirements are:

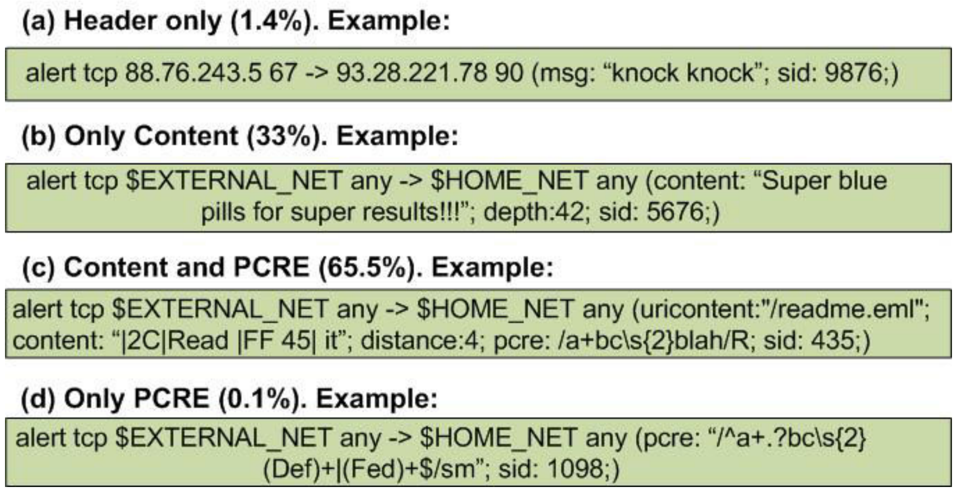
1. The Pre-filtering stage must be loaded with a simple subrule-set which requires relatively low computational power.

2. This set of subrules must preserve the efficient characteristics of the initial rule-set so that the number of activated rules per packet must be the lowest possible.

The goal of this thesis is to develop efficient techniques which are going to locate and select (extract) this rule's portion for every single rule of the whole rule-set so that Pre-filtering not only meets the above requirements but also is improved and enhanced. Every rule contains a lot of different features (fields). Section 2.3 presented all the possible SNORT rule's options for the payload description. It was observed, especially in Figure 2.7, which fields increase the processing requirements of payload matching. Thus, all the rule's options must be studied, according also to the above requirements, to find which ones are more suitable to be utilized to create the set of subrules for the Pre-filtering stage. It is a good point to mention that the SNORT rule-set was selected to be the representative set of rules that is going to be used for the purposes of this thesis, due to the fact that is considered the most complete NIDS rule-set and is used by the majority of the researchers.

In this thesis, both header and payload fields of the rules are included in the set of subrules. The whole header part was selected (protocol, source and destination IP, source and destination port for TCP/UDP rules). It does not contain many fields and hence does not require heavy computations. From the payload part, the fields which require high computational power must be excluded. These fields are mainly the payload restrictions: *distance*, *within*, *offset*, *depth*, *byte_test* and *byte_jump*. The remained rule's payload options are the content and the PCRE. Furthermore, the fields *msg*, *classtype*, *reference*, *sid* and *rev* were selected from the general part, while from the non-payload part of the rule *dsize*, *itype*, *icmp_seq* and *icmp_id* were also selected. The last nine fields require very low processing, while the last three are especially used by rules that describe ICMP packets.

As it was said, only the content and the PCRE remained from the payload rule options. The content describes one specific string and is from its nature a feature that requires lightweight processing. On the other hand, a PCRE can describe many different static patterns. A PCRE has a whole grammar and syntax like a natural language and for this reason, its processing may be extremely heavy sometimes. The proposed Pre-filtering extraction techniques use the above two features in order to extract the set of subrules, providing to the Pre-filtering stage high processing throughput and the lowest possible number of activated rules. The latter implies also that the latency and implementation costs in the second phase will be kept also low. Consequently, the proposed extraction techniques must provide a set of subrules that requires low processing power and simultaneously preserves the sophisticated characteristics of the rules.


(a) Header only (1.4%). Example:

```
alert tcp 88.76.243.5 67 -> 93.28.221.78 90 (msg: "knock knock"; sid: 9876;)
```

(b) Only Content (33%). Example:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (content: "Super blue
                pills for super results!!!"; depth:42; sid: 5676;)
```

(c) Content and PCRE (65.5%). Example:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (uricontent:"/readme.eml";
content: "|2C|Read |FF 45| it"; distance:4; pcre: /a+bc\s{2}blah/R; sid: 435;)
```

(d) Only PCRE (0.1%). Example:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (pcre: "/^a+.?bc\s{2}
                    (Def)+|(Fed)+$/sm"; sid: 1098;)
```

Figure 3.1: Examples of rules of the SNORT rule-set of July 2008 that have only header, or have header and payload and from the payload they have only content, or only PCRE, or both content and PCRE. The distribution of the rules, of the specific set of July 2008, is given based on these cases along with a rule example. Case (a) is trivial and has no payload. Rule of case (b) describes a packet's payload which must contain the string that is inside the double quotes, in its first 42 bytes. An interesting case is case (c), where the URIcontent must be matched and after 4 bytes (this is implied by distance), the second content must be matched. If it is also matched then SNORT has to determine if the packet's payload is described by the PCRE field. Case (d) is an example of one rule that contains only PCRE.

During this thesis, it was decided to examine the efficiency of Pre-filtering when a portion of one specific field (content or PCRE) of every single rule is extracted. Thus, the strategy is to examine each time only one of these two fields. However, it is impossible to apply this strategy for every rule of SNORT rule-set of July 2008 because there are some rules that contain only header (1.4%), rules that contain only content (around 33%) or rules that contain only PCRE field (0.1%). Fortunately, most of the rules contain both header and payload and its payload consists of both content and PCRE fields. Figure 3.1 depicts 4 examples of rules of the above cases. The proposed extraction techniques are three:

- First Content Prefix (content-based): Extract the prefix of the content (the first content if the rule contains more than one).
- PCRE Prefix (regular expression (regex)-based): Extract the prefix of the PCRE.
- Unique Part Rule (content-based): Extract a part of the content(s) so that every single subrule, which is loaded to the Pre-filtering stage, is unique.

Figure 3.2 depicts a general view of the two content-based Pre-filtering techniques. The first one (Figure 3.2(a)) creates the subrule, using the rule's header and extracting a prefix of the content (the first content if a rule has more than one). On the other hand, the second technique (Figure 3.2(b)) makes the subrule extracting a part of one or more contents so that the combination of this part and the rule's header will provide a unique subrule. The PCRE Prefix technique creates the set of subrules similarly to First Content Prefix with the main difference that the prefix of the PCRE is taken instead of a simple static pattern.

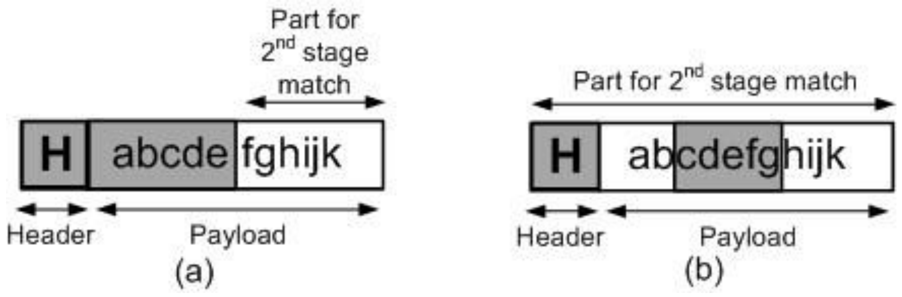


Figure 3.2: A general view of the two content-based techniques. The subrule is the grey part of the initial rule. The first case (a) is the First Content Prefix technique. The rule's part which does not participate to the subrule will be the only part that will be used in the Full Match stage. The second case (b) depicts the Unique Part Rule technique, where a part(s) of one or more content(s) is selected so that the combination of this part and the rule's header will provide a unique subrule. However, the whole rule must be processed in the Full Match stage.

The strategy that is followed when extracting the set of subrules, for the Pre-filtering stage, is summarized below:

1. Rules that have only header part are used unchanged.

2. If the selected technique is one of the content-based (First Content Prefix or Unique Part Rule), extract the portion of the content according to it. If the selected technique is the Unique Part Rule and unique parts are not found for some rules, the Rule Correlation technique utilizes these rules. For the rules that do not contain content, extract the portion of the PCRE using the PCRE Prefix technique.

3. If the selected technique is the PCRE Prefix, extract the prefix of the PCRE. For the rules that do not contain PCRE, extract the portion of the content using the First Content Prefix technique. This technique was selected to be the alternative when the PCRE prefix extraction is impossible.

This chapter is organized in seven sections. In section 3.1, important details of the content and PCRE (grammar and syntax rules) fields that are exploited by the extraction techniques are provided to the readers to give them a background before explaining the proposed techniques. Section 3.2 describes the first proposed technique which is the First Content Prefix. Afterwards, section 3.3 discusses the extraction technique for PCREs. In addition, section 3.4 presents the Unique Part Rule extraction technique which attempts to create a set consisted of unique subrules. Sections 3.5 and 3.6 propose two more techniques. They are not extraction techniques but they optimize significantly both stages of a NIDS that uses Pre-filtering. At the end, the chapter concludes with a small summary.

3.1 Background on Content and PCRE

The content (static pattern) field is the most important feature of an inspection rule. It is a specific sequence of characters, numbers or symbols and describes a possible attack that is hidden in the payload part of an incoming packet. An example of a typical content feature is the following: "connected2". However, it may contain hexadecimal digits inside pipe operators (i.e. $|FF$ 09|) or a combination of the former tokens and hexadecimal digits like the following: "conne|00 01 FF 23|2ct".

In addition, there are three different types of contents that can be found inside a SNORT rule:

1. *content*: It describes a malicious payload content. An example is depicted in Figure 3.1, case b.

2. *URIcontent*: This type of content is used to match strings that come out of the URI buffer in a normalized form, as it was explained in section 2.3.2. An example is shown in Figure 3.1, case c.

3. *negated content*: It describes a non-malicious payload content. The content describes a sequence of tokens that can be a potential attack but negated content is a safe sequence of tokens and a packet may be malicious if this specific content is not found in its payload. Negated contents are, normally, used along with normal contents in one rule to avoid potential false positives or in other words, to distinguish safe packets that may contain an intrusion pattern by lack, from attacking packets.

It is worth noting that many contents (static patterns) can exist inside a rule. These are used along with the payload restrictions like *within*, *offset*, etc, which were described in section 2.3.2, to identify specific patterns in specific positions. One example is shown in Figure 3.1, case c. URIcontent must be matched and after 4 bytes (this is implied by distance), the second content must be matched. If it is also matched then SNORT has to determine if the packet's payload is described by the PCRE field. In the same example, the rule contains also more than one contents. Many rules can contain more than one contents inside one rule. As it was also mentioned in section 2.3.2, a SNORT rule of July 2008 contains on average 4 static patterns (contents).

On the other hand, a regular expression is able to describe more than one malicious static patterns providing the NIDS more flexibility with the drawback of higher computational power. It can be stated that given an input string $T[1..n]$ of an alphabet Σ, which is a finite set of tokens, and a regex R of the same alphabet, which describes a set of strings $S(R) \subseteq \Sigma*$, then matching the regex R is to determine whether $T \in S(R)$ [32]. There are many kinds of regular expressions and every single one of them has a specific syntax. It was mentioned that the Perl Compatible Regular Expressions (PCRE) were used among the other regex kinds for the purposes of this thesis. It is worthy to present the main syntactic rules of PCREs in detail because they play an important role to understand why extracting the prefix of PCREs is a difficult procedure and extracting a unique part is almost impossible. The PCRE library was created by Philip Hazel [16]. More and more features are included in it year by year. The features of the PCRE syntax are classified into many categories based on their common cause and are:

- *ASCII characters*: All ASCII characters are included in this category except meta-characters, quantifiers, etc and one instance of them is matched every time.
- *Meta-characters*: Each one of them has a special meaning:
  - ".": is called dot operator. It matches any character except "new line".
  - "^": is called wedge. It matches the regex that follows, only at the beginning of the string.
  - "$": is called dollar. It matches the regex that precedes, only at the end of the string.
  - "–": is called dash. It is used to specify ranges of numbers, letters, etc inside character classes.
  - "\": is the Backslash. It is the escape character and is explained below.
  - "|": is called Union and is used in cases like i.e. regex1|regex2 where either regex1 or regex2 is matched.
  - "()": is Capture buffer. Everything inside parentheses is captured (if matched) so that operators like quantifiers can be applied.
  - "[]": is Character class. It can match one of the characters if it used like [abc], one of a range of characters if a dash is used inside like [a-z], or it matches

any of the characters except the specified characters (negated match) when used like [^abc].

- *Quantifiers*: They are used in order to match more instances. Each one has a special usage. The regex that is to be matched precedes the quantifiers. The quantifiers are:
  - "$*$": is called Kleene Star. It matches the regex, zero or more times.
  - "$+$": is called Plus. It matches the regex, one or more times.
  - "?": is called Question. It matches the regex, zero or one time.
  - "$\{N\}$": is called Exactly. It matches the regex, exactly N times.
  - "$\{N,\}$": is called At Least. It matches the regex, N or more times.
  - "$\{N,M\}$": is called Between. It matches the regex, between N and M times.
  - "any of the above quantifiers followed by "?" is called Non-Greedy. It matches the regex, as many times as indicated by the first quantifier but not greedily.
- *Escape Sequences*: The Escape Sequence is the combination of the Backslash and a special letter(s) or a meta-character after it. Every combination with a special letter has a special meaning. The most usual cases are:
  - "$\backslash n$": matches "newline".
  - "$\backslash t$": matches "tab".
  - "$\backslash r$": matches "return".
  - "$\backslash x$": and a pair of HEX digits matches the ASCII character of numeric value indicated by the HEX digits.
  - "$\backslash u$": and 4 HEX digits matches the ASCII character of numeric value indicated by the long HEX character.
  - "$\backslash C$": and a small or capital letter matches the control character.
  - "$\backslash\ followed\ by\ any\ meta-character$": escapes the meta-character and returns its literal value.
- *Character Classes and other Special Escapes*: The usage of them is similar to Escape Sequences: Some cases are:
  - "$\backslash d$": matches any digit in range 0-9.
  - "$\backslash D$": matches a non-digit character.
  - "$\backslash s$": matches whitespace character.
  - "$\backslash S$": matches a non-whitespace character.
  - "$\backslash w$": matches a "word" character (letter and digits).
  - "$\backslash W$": matches a non-"word" character.

  - "\ *any number*": is called Backreference and it has the same value as the corresponding, by the number, Captured buffer that was matched before.

- *Extended Patterns*: This category contains the Look Around assertions, Conditional Assertions and some other general assertions. Look Around assertions match/do not match the regex without consuming it. Typical examples are the positive look ahead assertion, $regex1(? = regex2)$, where it matches the regex1 followed by regex2 without consuming regex2 and the negative look ahead assertion, $regex1(?!regex2)$, where it matches the regex1 if it is not followed by regex2. Conditional assertions, like (?(?=regex) then | else) continue matching with normal regex if look ahead succeeds, or otherwise with else regex. Finally, there are other more general assertion cases that is used by PCREs. One of the most usual is $(? : regex)$ which captures the regex but it can not be used for backreference.

- *Modifiers*: The modifiers set at compile time flags for the regex. The regex value is inside the // symbols. On the other hand, the modifiers are after these symbols. Two kinds of modifiers are used: 1) PCRE modifiers (s, m, i) and 2) SNORT modifiers (R, U, B). Firstly, the PCRE modifier "s" changes the function of dot character to include also the match of "new line". Modifier "m" modifies ^ and $ to match the regex after and before the newline respectively. Finally, modifier "i" makes the regex case insensitive. Regarding the SNORT modifiers, R makes the regex to be matched relative to the end of the last pattern match. As a result, R requires the previous content to have been matched. Modifier "U" matches the regex to the decoded URI buffer like URIcontent, while "B" changes the SNORT not to use the decoded buffers for matching the regex.

The alternative for finding and extracting unique parts of PCRE is limited significantly by the above rules. It is very difficult, if not impossible, to find a correct unique part taking into account the meta-characters, quantifiers, backreferences etc. Consequently, it was decided to study the way of extracting the Prefix of the PCRE. It is explained in detail in section 3.3, how this is done and according to which policies. What it should be additionally mentioned is that the whole PCRE is parsed and the Abstract Syntax Tree (AST) is generated in order to correctly and more efficiently extract the prefix of the PCRE. The AST is actually a tree representation of the syntax of an input string according to some syntax rules. In our case, the input string is the regex (the sequence of tokens between the // symbols of the whole PCRE) and the syntax rules are the above ones.

In conclusion, the content is considered the most significant feature of an inspection rule. Although it describes just one malicious pattern that may exist inside the packet's payload in contrast with regex, it is more preferable due to the fact that it is less computationally intensive than the regex. In SNORT specifically, it is insisted that despite the fact that a regex can describe the same threat that is also described by a combination of contents and payload restrictions, it is more preferable to describe it using the latter way (the combination of content and the other fields). Finally, this section presented some important background details for the Content and the PCRE fields which are going to be exploited by the following 3 extraction techniques.

3.2 First Content Prefix Extraction Technique

The first approach is to extract the prefix of the content. It is the most straightforward idea in the field of text processing because of its simplicity. The most important issue that arises is which is the selected content from which the prefix is extracted in case a rule contains more than one contents. More specifically, the SNORT rule-set of July 2008 contains 3258 rules (37%) which have one content and 5546 rules (63%) with more than one contents. Regarding the firstly mentioned issue, the first content of the rule (if it has more than one) is selected to be used for the prefix extraction. Generally speaking, the proposed First Content Prefix technique is the simple and brute-force method which does not have any sophisticated characteristic on extracting the portion. The idea is to select the first content (if there are more) of the rule and just take the prefix of it. This approach requires little processing in order to extract a prefix for every rule of the rule-set and additionally, only the remaining part of the rule will have to be matched in the second stage. A simple example is depicted in Figure 3.3b. The initial pattern was “My door is open” and the 6-byte prefix of it is “My doo” because the white space is also considered as a character. In the case of a content with smaller length than the requested length of the portion, the whole content is selected.

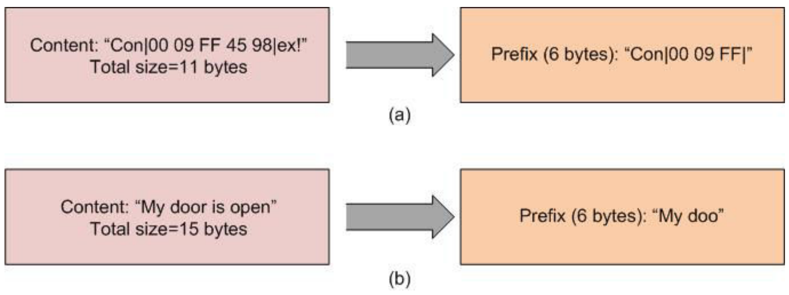

Content: “Con|00 09 FF 45 98|ex!”
Total size=11 bytes
Prefix (6 bytes): “Con|00 09 FF|”
(a)
Content: “My door is open”
Total size=15 bytes
Prefix (6 bytes): “My doo”
(b)

Figure 3.3: Examples of extracting the prefix of a content. In both cases, the prefix length was selected to 6 bytes. In (a), the content consists of ASCII characters along with HEX digits inside pipe operators. In (b), the extraction of the prefix of the content is more straightforward.

If the content consists of just numbers, characters or symbols, it is straightforward to know how many characters the static pattern has, and trivial enough to extract the prefix, as it was shown in Figure 3.3b. However, if the static pattern contains hexadecimal (HEX) digits, the prefix extraction must be performed carefully, since there is danger to extract an *incorrect* pattern. The HEX digits are inside pipe operators in pairs and two pairs are distinguished with white spaces. It is also known that a pair of HEX digits forms an ASCII character (with size of a byte). Thus, in order to count the total size of a static pattern which contains a sequence of HEX digits, the number of HEX pairs inside the pipe operators must be counted, without counting the white spaces between them, and add it to the size of the rest of the pattern. During prefix extraction, if the prefix finishes in the middle of a field inside pipe operators, a pipe operator is added to the extracted portion after the last selected pair of HEX digits and this is the prefix of the content. Figure 3.3a depicts an example of taking the prefix of this case (combination of HEX digits and normal tokens). The prefix length was selected to six bytes. Until the first pipe operator, the counter counted 3 bytes. Thus, we need 3 more characters to have the prefix. The size of the HEX sequence is 5 pairs or in other words 5 symbols. The prefix stops in the third pair (6 characters in total).

In conclusion, the First Content Prefix technique is an approach which does not use any sophisticated mechanism in order to extract the prefix. It is very trivial and it requires the minimum time in processing the whole rule-set because each rule is examined until finding the first content (if more) and take its prefix. In addition, only the remaining part of the initial rule should be matched in the Full Match stage. However, if the selected content has smaller length than the requested length of the portion, the whole content is selected, even though there may be more and longer contents inside a rule. This is one drawback of this technique. Another disadvantage of using this technique is that many subrules, for the Pre-filtering stage, may be the same. This may happen in cases where two rules with the same header but with different, initially, payload parts may have the same prefix of their first content.

3.3 PCRE Prefix Extraction Technique

The regular expression (regex) is a string that can describe many particular different patterns based on the syntax of the specific regex, providing more flexibility but with more needed computations. Simultaneously, a potential drawback of the regex is that too many packets may activate the rule that uses regexes. The regex field of a SNORT rule is specifically a Perl Compatible Regular Expression (PCRE). PCREs, as every regex, have a specific syntax. The most important features of it, divided in categories based on their cause, were presented in detail in section 3.1. These syntactic rules play an important role for this Pre-filtering technique for regular expressions because they limit the extraction technique alternatives but at the same time help to design and implement one of them.

The purpose of this approach is to extract a prefix of the first PCRE field of the rule, in a similar way as in First Content Prefix. However, extracting the prefix of the PCRE is not easy at all, compared to the previous extraction technique, due to the nature of the PCREs and generally of regexes, and the limits it introduces. More specifically, the idea of PCRE Prefix extraction technique is that given a PCRE A that describes the strings "p_1, p_2, p_3, ..., p_N", it is possible to find a prefix of the PCRE A which describes the strings "p'_1, p'_2, p'_3, ..., p'_N, p'_{N+1}, p'_{N+2}, ..., p'_M", where p'_1 is the prefix of p_1, p'_2 is the prefix of p_2, etc and "p'_{N+1}, p'_{N+2}, ..., p'_M" are extra strings that are not matched by the initial PCRE (PCRE A). For example, the PCRE "ab+xy{2}" describes the strings: "abxyy", "abbxyy", "abbbxyy". A 3-byte prefix of it is "ab+x" and describes the strings "abx", "abbx" and "abbbx" which are prefixes of the previous strings and additional strings like "abxe", "abbxa", etc which are not described by the initial PCRE. All the rules of the regex syntax, which were described in section 3.1, must be taken into consideration so that correct extractions of regular expressions are performed, otherwise it is possible that the extracted regex may be matched by every packet or may not be matched by potential threats leading to an unsafe NIDS.

The selected Pre-filtering technique for PCREs is to take the prefix of it and not a unique part as proposed in Unique Part Rule later. It may be possible to extract a unique part from a PCRE but generally, searching for unique parts inside PCREs is limited by the nature of them, or in other words by factors like quantifiers, parentheses, character classes and other as presented in section 3.1. Even if a part was found, the needed procedure would have required a significant amount of processing. In order to extract the prefix, the syntactic rules of PCREs, which were presented in section 3.1, have been studied thoroughly and in order to construct the extraction rules.

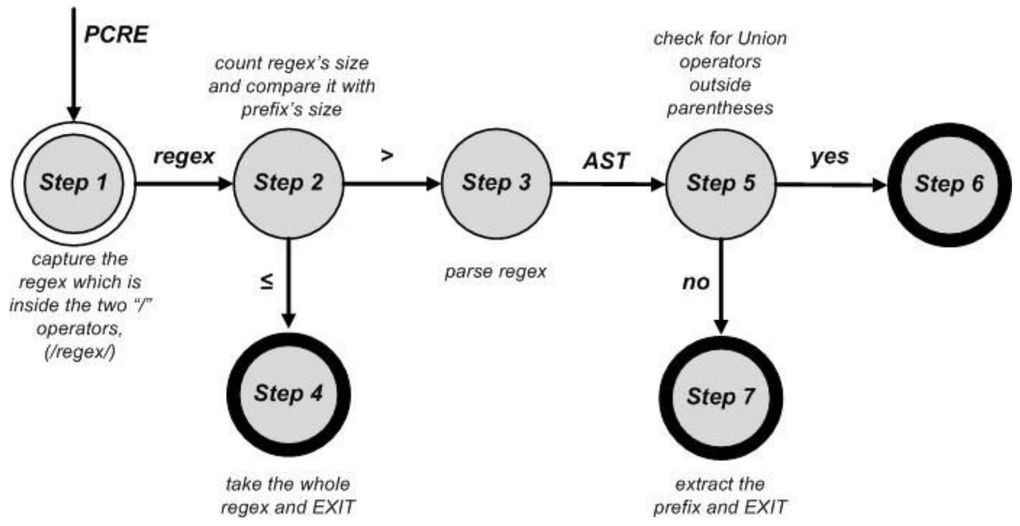
The idea of extracting the prefix from a PCRE, first of all, requires from the rule to contain a PCRE field because the resulted extracted subrule will contain only a PCRE field among the other payload fields. The first matter that must be checked from the PCRE is the modifiers and especially whether the "R" modifier exists. It is reminded that the "R" modifier requests from SNORT to match the PCRE if and only if the previous content had been matched. This means that the previous content must be also included into the subrule. The problem is, actually, that in these cases the PCRE describes very simple strings, like i.e. /.{37}/, which means match any character exactly 37 times. However, the goal of this technique is to exploit PCREs and determine whether they can be efficiently used alone as a subrule's payload field and not along with contents. Thus, in this special case, the prefix of a next PCRE is taken (if exists), otherwise the First Content Prefix technique is applied for this rule, as an alternative. The other modifiers do not play a role for the prefix extraction but only for compile-time matters and are used along with the PCRE's prefix without, of course, considering them as part of the prefix.

Another important aspect that should have been studied and taken into account during prefix extraction is which symbols should be counted in order to track how much of the PCRE is still needed (based on the indicated length of prefix) to be extracted. In content, all the symbols are "equal" (in terms of counting size) except the pipe operator and the HEX digits, which should have been counted in pairs. However, the symbols are not "equal" in PCREs. Meta-characters (except dot operator) and quantifiers should not be considered as normal symbols because they have a specific role and do not match a character. Thus, it was decided, to count only the following symbols in order to know how much of the PCRE is remained to be extracted:

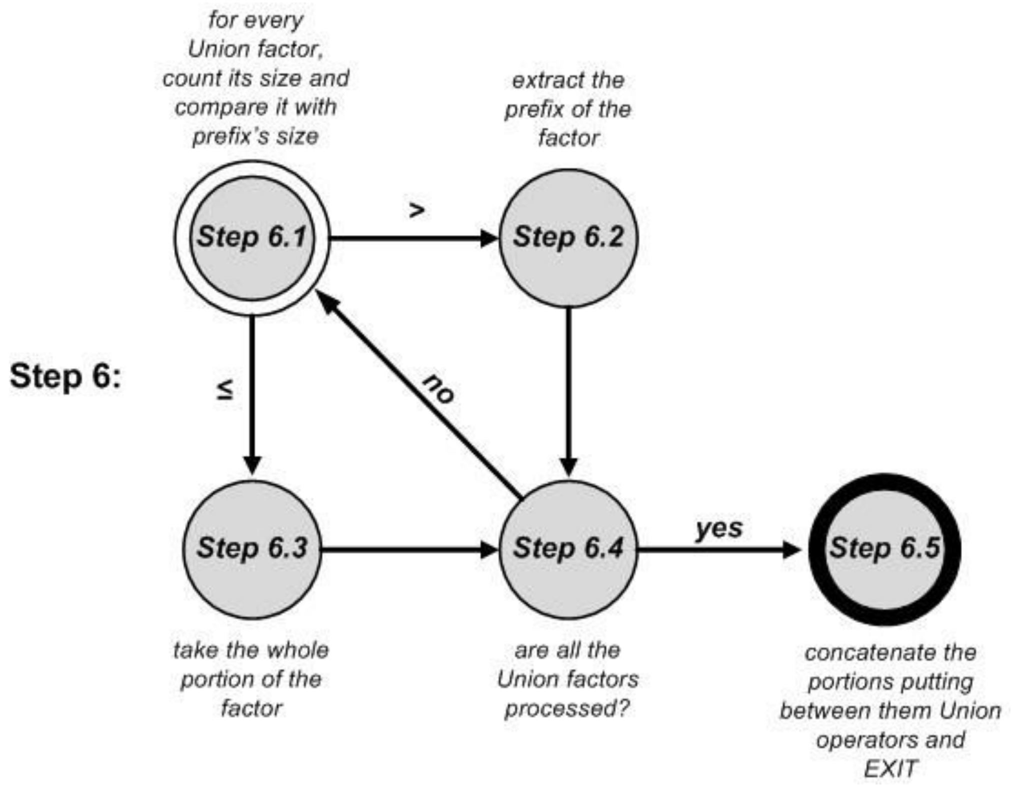
- all the symbols that follow the backslash,
- the dot operator,
- the backreferences, and
- all the ASCII characters (letters, numbers, other tokens). The ASCII character that is indicated by the numeric value of normal HEX or long HEX is considered one read character and not 3, e.g. $\backslash xFF$, or 5 in e.g. $\backslash uAFE4$.

It must be noted that these symbols are referred in the rest of this section also as "countable" symbols.

The procedure that is followed in order to extract a prefix, given a PCRE that does not have the "R" modifier, consists of several steps and it is depicted in Figure 3.4(a). The first step is to capture the regex without taking the symbols "/" and the modifiers.


PCRE
Step 1
capture the regex which is inside the two "/" operators, (/regex/)
regex
Step 2
count regex's size and compare it with prefix's size
>
≤
Step 4
take the whole regex and EXIT
Step 3
parse regex
AST
Step 5
check for Union operators outside parentheses
yes
Step 6
no
Step 7
extract the prefix and EXIT

(a) The extraction of the PCRE prefix is performed following the steps. It contains one starting node and the procedure can be terminated in three different steps (for three different cases).


Step 6:
for every Union factor, count its size and compare it with prefix's size
Step 6.1
>
Step 6.2
extract the prefix of the factor
≤
no
Step 6.3
take the whole portion of the factor
Step 6.4
are all the Union factors processed?
yes
Step 6.5
concatenate the portions putting between them Union operators and EXIT

(b) The step 6 of the top chart is analyzed further at this chart. It is especially for regular expression cases which contain Union operators but outside parentheses. This case is special because different string cases may be matched by one regular expression and for that reason a prefix of each factor of Union must be extracted. An example of this case is: the "$(abcde) \mid (xyzbn) \mid (polfg)$".

Figure 3.4: The procedure which must be followed in order to extract a PCRE prefix is shown at the top chart (a), while the bottom chart (b) zooms on step 6 of the procedure (top chart) and analyzes it.

In the 2nd step, the size of regex is counted by counting the "countable" symbols. If the size of the regex is larger than the specified prefix length, the next step is step 3, otherwise it is step 4, where the whole regex is taken and procedure is terminated. In step 3, the regex is parsed and the Abstract Syntax Tree (AST) is generated and is input in step 5. This step checks the AST for Union operators (|) which must be outside parentheses. If this does not hold, then step 7 is executed, where the prefix is extracted and the procedure is terminated. If it holds, then step 6, which is analyzed further in Figure 3.4(b), is executed. In step 6, the regex has one or more Union operators outside parentheses. That means that the regex describes different string cases and whichever factor of the Union operator can be matched. For that reason, the prefix of every regex's factor (element) must be extracted and this is done in this step. For example, consider that we want to extract the prefix (of length 4) from the following PCRE: /(abcde)|(xyzop)|(mnbvc)/. It was mentioned that Union meta-character means that either abcde, xyzop or mnbvc will be matched. Thus, the correct prefix would be to take a prefix of 4 bytes for each element and is: /(abcd)|(xyzo)|(mnbv)/.

Steps 6 and 7 of the PCRE Prefix procedure (Figure 3.4(a)) are the steps where the prefix is extracted. The AST form of the regex eases significantly the prefix extraction. The function that extracts the prefix starts reading the regex in AST form and counts the "countable" symbols until the total prefix length is reached. The extraction must be performed according to the extraction rules that were found and are proposed below, in order to extract a correct prefix. Before starting explaining the most important of the extraction rules, we define some terms which are useful for the rest of the section. The "Atom" can be a captured buffer, an extended pattern, a dot operator, an ASCII character, an escape sequence and a character class. The "AtomQ" is an "Atom" that is followed by a quantifier.

The extraction rules, when an "Atom" is found, are summarized below:

1. If the "Atom" is: *dot operators, ASCII characters or escape sequences*, it is read normally until reaching the prefix length. Then, the prefix has been extracted and the extraction procedure is terminated.

2. If the "Atom" is: *one captured buffer or more*, it is correct to read and extract it like the previous case. A simple example is depicted in Figure 3.5. Looking on the right of the Figure, it is obvious that the first three strings are prefixes of the first three payloads described by the initial regex on the left. Furthermore, a payload which contains the string "ababb" activates the rule with the regex's prefix while it would not activate the initial rule. Actually, this is the point of Pre-filtering: using a smaller portion of rule, more packets can activate it.

3. If the "Atom" is: *folded captured buffers*, the number of open parentheses is tracked so that if the prefix extraction finishes before reaching the close parentheses in the AST, they are added automatically. For example, consider the case of "/(xy+(zwdfr)a?\s*b)|(abdef)/" and that a 4-byte prefix is requested. It is a Union case but the interesting part is the first factor of the Union operator. The factor's prefix would be "(xy+(zw" and two closed parentheses must be also added in order the prefix to be correct.

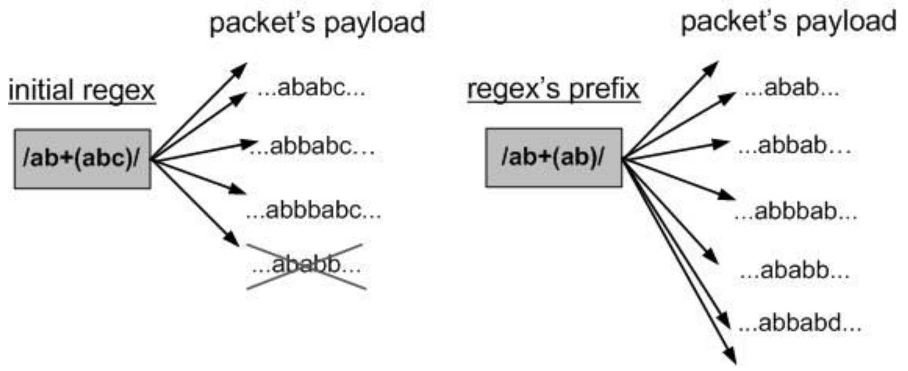


Figure 3.5: Examples of a initial regex, its 4-byte prefix and the packet payloads they describe. The initial PCRE describes the patterns: "ababc", "abbabc", "abbbabc", etc. The extracted PCRE describes "abab", "abbab", "abbbab", etc which are all prefixes of the previous patterns and can also describe other patterns like "ababb", etc, which would not be described by the initial PCREs.

4. If the "Atom" is: *assertion case:* (? : *regex*), the rules for the captured buffer are applied too.

5. If *a Union is found inside the captured buffer*, the whole captured buffer is taken in the prefix. This is a rare case. It was observed that is more usual to use Union outside parentheses. In a simple example, if we had the regex "/ab(def|try)bl*\d{2}./", the 4-byte prefix is "/ab((def)|(tru))/" according to the above rule instead of "/ab((de)|(tr))/". This was selected because it requires many computations to extract prefixes from Union operators inside parentheses considering the fact they are more complex than the presented example.

6. If the "Atom" is *character class* or *Look Around assertion*, the whole is taken because it could be wrong to count the symbols inside it as being a normal string and take just the prefix.

When an "AtomQ" is found, the extraction rules are totally different and depend significantly on what type of quantifier follows. It would not be efficient to apply the same extraction rules that hold for "Atom". This is explained better if we see the example of Figure 3.6. The initial regex is "/ab(defz)+xy/" and describes the patterns "abdefzxy", "abdefzdefzxy", etc. If the extracted prefix of the regex was "/ab(de)+/" (using the extraction rule for "Atom"), it would match strings like "abde" which is the prefix of the previous, and extra strings like "abdeyufzxy", "abdefzklo", "abdedexyv", "abdededexy", etc. Using our proposed extraction rule for "AtomQ", the whole "AtomQ" is taken and the selected regex's prefix would be "/ab(defz)+/", which is activated by packets with payload "abdefz", "abdefzdefz" that are the prefixes of the matched strings in the initial case but by less extra packets (with possible payload:"abdefzno", "abdefzdefztyi", etc but not "abdeyufzxy") than the previous (prefix) case. The proposed technique is based on the observation that the regexes describe naturally many static patterns and the quantifiers are the ones that increase this number significantly. The Pre-filtering goal is not just to take a small portion of the rule but this portion would be sufficient so that

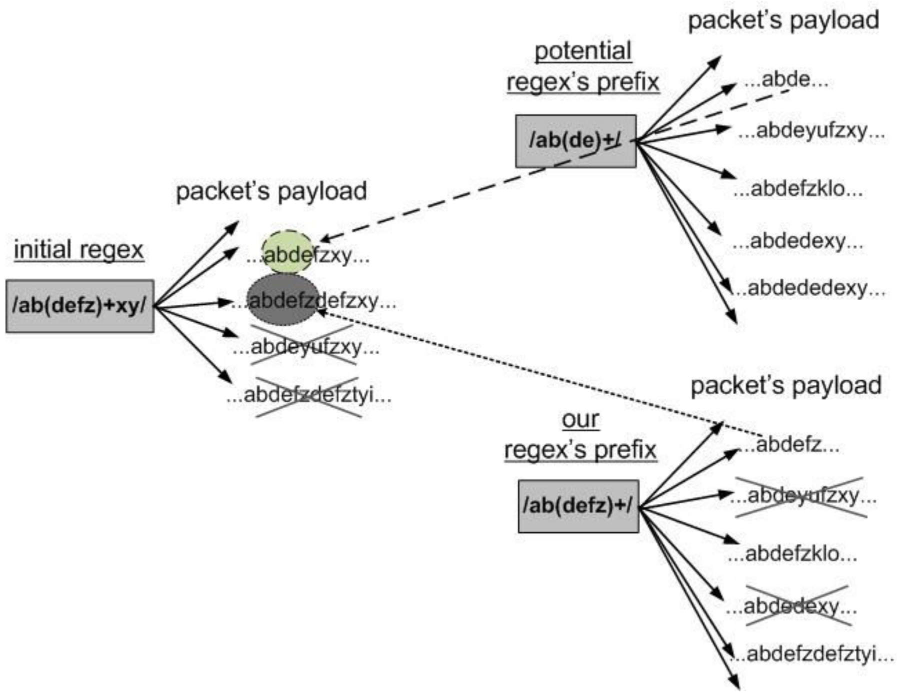


Figure 3.6: Example of taking the 4-byte prefix of a regex which contain "AtomQ" using the respective extraction rules for "Atom" and the proposed extraction rule for "AtomQ". A rule with regex "/ab(defz)+xy/" is activated by packets with payload "abdefzxy", "abdefzdefzxy", etc. If the extracted prefix of the regex was "/ab(de)+/", it would match strings like "abde" which is the prefix of the strings of the initial case, and extra strings like "abdeyufzxy", "abdefzklo", "abdedexyv", "abdededexy" etc. However, according to our proposed extraction rules for "AtomQ", the prefix is "/ab(defz)+/" which is activated by less packets ("abdefz", "abdefzdefz", "abdefzno", etc) than the previous case but it has 6 bytes instead of 4.

the minimum number of rules would be activated and sent to the second stage of the NIDS.

The idea behind the prefix extraction for "AtomQ" cases is that different extraction rules can be applied taking into account the combination of the type of the quantifier and the position of the "AtomQ" inside the regex (at the start, middle or end). Firstly, the "AtomQs" can be categorized into two categories based on the type of the quantifier: The "AtomQs" with quantifiers "∗" and "?", which actually include the zero times match, are in Category 1, while the ones with quantifiers "+" and "{...}" lie in category 2. It must be mentioned here that if the "?" is a second quantifier which follows any of the former quantifiers, the "AtomQ's" category, as specified by the previous quantifier, is maintained. The "AtomQ" can also be categorized in three other categories based on its position on the regex. Table 3.1 summarizes all these cases.

If the "AtomQ" belongs to *category 1* and it is at the *beginning* or at the *end* of the regex, is excluded from the extraction procedure because better prefix may be found and taken from the rest of the PCRE making the match more accurate and the Pre-filtering more efficient. The previous claim is based on the fact that sometimes, an "AtomQ"

Table 3.1: This table summarizes the "AtomQ" cases based on the quantifier and the position (start, middle, end) of the AtomQ inside the regex. The extraction rules for taking the prefix of an "AtomQ" are formed, according to these cases.

Category 1		
start	middle	end
regex1∗regex2	regex1regex2∗regex3	regex1regex2∗
regex1∗?regex2	regex1regex2∗?regex3	regex1regex2∗?
regex1?regex2	regex1regex2?regex3	regex1regex2?
regex1??regex2	regex1regex2??regex3	regex1regex2??
Category 2		
start	middle	end
regex1+regex2	regex1regex2+regex3	regex1regex2+
regex1+?regex2	regex1regex2+?regex3	regex1regex2+?
regex1{...}regex2	regex1regex2{...}regex3	regex1regex2{...}
regex1{...}?regex2	regex1regex2{...}?regex3	regex1regex2{...}?

of the category 1 may not be matched at all (zero times). Thus, if it is excluded and the rest of the PCRE is used for the extraction procedure, the extracted prefix will be again correct since it will match all the strings that were matched by the original PCRE and it may provide more accurate matching, describing as few as possible extra packets payloads. However, this cannot be applied for "AtomQs" of category 2 because "AtomQ" will be matched at least one time. For instance, assume that we have the regex "/(nam)∗abxy/". This matches "abxy", "namabxy", "namnamabxy", etc, as it is also shown in Figure 3.7. If the extracted prefix was "/(nam)∗a/", it would match "nama", "namnama", etc and also the string "a". But string, and more correctly one character (here character 'a'), can be found with very high probability in the payload of all the incoming packets making the respective rule to be activated always, leading to a strong inefficiency. However, if the extracted prefix was "/abxy/" all the payloads with strings matched by the initial regex, would have been matched along with fewer other extra payloads. In a second example (Figure 3.8), the regex "/(nam)+abxy/" matches "namabxy", "namnamabxy", "namnamnamabxy", etc. The rule with extracted prefix "/(nam)+a/" is activated by payloads which contain the strings "nama", "namnama", etc and other of course payloads with strings that were not matched by the initial regex but are not so many as they would be in the case of "/(nam)∗a/". Finally, the whole "AtomQ" is taken into the prefix if it is in the middle of the regex (infix) regardless of the type of the quantifier. It would lead to a wrong prefix extraction if it was excluded as in the previous cases. For example, the regex "/xy(name)∗ab/" matches the strings "xyab", "xynameab", "xynamenameab", etc and if the extracted prefix was /xyab/ it would match only the first from the previous strings leading to an incorrect regex. The extraction rules above are applied for all the possible cases of "AtomQ".

The extraction rules were implemented in Perl programming language in order to extract the prefix from the PCRE. The language Perl was used because it is very strong on text processing. The grammar was created based on the rules of section 3.1 and the

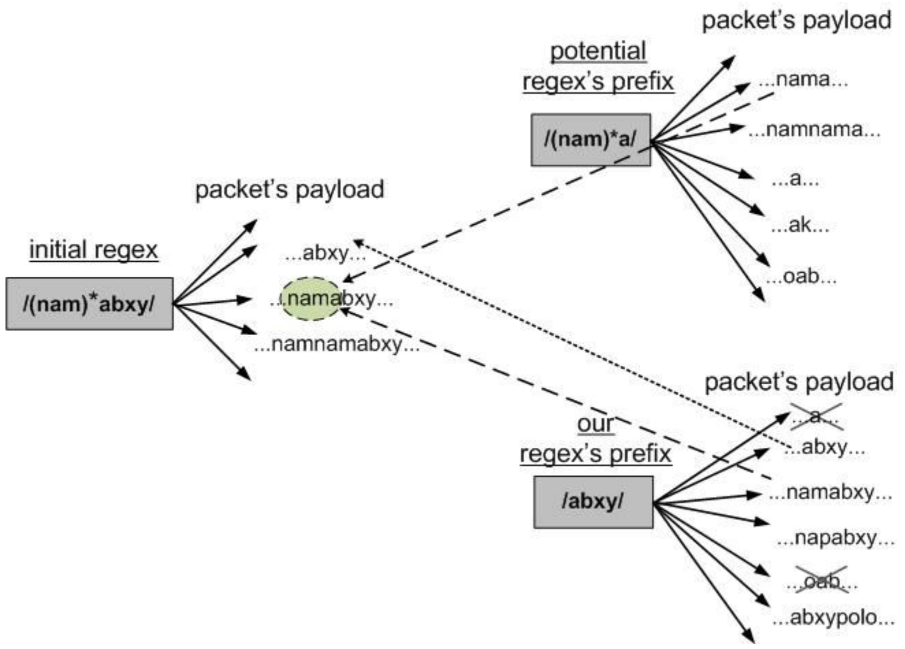


Figure 3.7: Example of extracting a regex prefix when the regex starts with an "AtomQ" with kleene star (category 1). The initial regex is activated by payloads which contain the strings "abxy", "namabxy", "namnamabxy". If the 4-byte prefix was "/(nam)∗a/", it would be activated by payloads with "nama", "namnama", etc but also by the payloads which contain character 'a'. And the probability to find a single character inside a packet's payload may be very high. According to our proposed extraction rules, the prefix is "/abxy/".

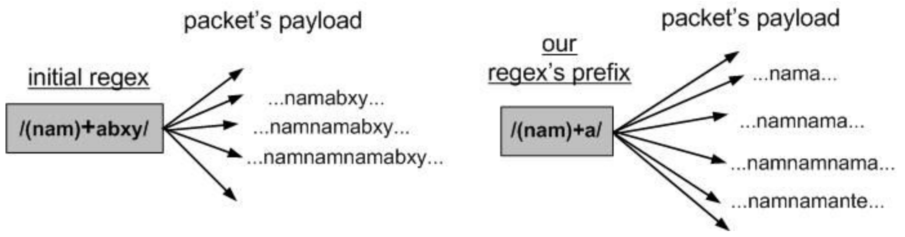
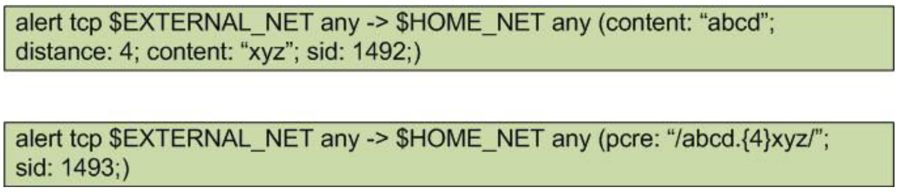


Figure 3.8: Example of extracting a regex prefix when the regex starts with an "AtomQ" with plus (category 2). This case is different than Figure 3.7. The selected 4-byte prefix is "/(nam)+a/" which is activated by payloads which contain the strings "nama", "namnama" which are prefixes of the activated payloads for the initial regex and of course, prefixes from other payloads.

RecDescent parser, which was created by Damian Conway [10], was used for the parsing of the regex and the AST generation. The AST is used in order to extract the prefix of the PCRE according to the extraction rules that were described above in this section.

To conclude with the PCRE Prefix technique, it was thoroughly discussed why extracting a portion of a PCRE is not as trivial as in content. The syntactic rules of PCRE limit the extraction alternatives. However, the idea of prefix extraction was found and the proposed extraction rules were presented and explained in detail. Without these



```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (content: "abcd";
distance: 4; content: "xyz"; sid: 1492;)
```

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (pcre: "/abcd.{4}xyz/";
sid: 1493;)
```

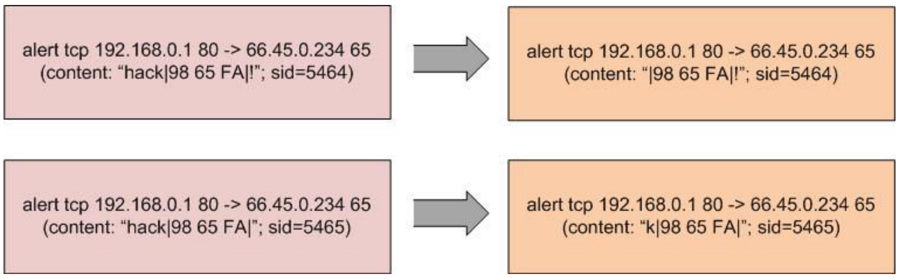
Figure 3.9: Two rules which describe the same payload. The first one using content and distance, while the second using PCRE. Both rules are activated by packet the payload of which has the string: abcdXXXXxyz, where X (capital x) can be any token.

rules, the extracted prefix would be either wrong or it would make Pre-filtering technique inefficient. Finally, it is good to be noted that in SNORT, the PCREs are usually matched complementally to content. Someone can suggest to write regexes instead of using contents along with payload restrictions like *offset*, *within*, *distance*, etc. Figure 3.9 depicts two rules which describe the same string with a different way: one uses content and the other a PCRE. However, SNORT programmers insist on the fact that if a string can be expressed using static pattern, it should be done in that way than with regexes. The last gives a good explanation for the presented results of chapter 4, when PCRE Prefix technique is used.

3.4 Unique Part Rule Extraction Technique

Section 3.2 presented the first proposed extraction technique which is based on the content field. It is simple and the extraction of the prefix can be done fast for the entire rule-set with the potential danger of having same subrules (initially with the same header and after extraction with the same payload description). Consequently, all of them are activated together. Techniques like the previous one may be proved a total failure in the concept of Packet Pre-filtering because too many rules may be matched in the first phase. They will not be thousands of rules but even one to two hundred activated rules is a lot of processing in the Full Match stage. On the other hand, section 3.3 proposed the second extraction technique which utilizes the PCRE. Extracting the PCRE prefix is limited by the PCRE syntax rules and it is expected, due to the nature of regexes, that in this case the number of activated rules will be significant. For these reasons, a more efficient approach is going to be proposed. It extracts the portion based on the rule's content so that processing of it during the Pre-filtering stage is lightweight and the minimum number of rules is activated. The proposed technique of this section is the Unique Part Rule which achieves substantially better results than the other extraction techniques, as it is shown in Chapter 4.

The proposed idea is totally different than Prefix extraction (Content or PCRE). The main idea of the Unique Part Rule technique is that it searches for a part of a static pattern (prefix, infix, suffix) or more that will give a *unique* subrule among the other so that the specific subrule will describe one and only intrusion. It must be noted that, in this idea, uniqueness is not based only on different parts of contents but also on probably same parts of contents but different header parts of these rules. An example is shown in Figure 3.10. In this case, both rules have the same header and their contents differ in last token. If the First Content Prefix extraction technique was used, the extracted rules of Pre-filtering stage would be exactly the same because the prefix “hack” would have been selected for both of them. On the other hand, using the Unique Part Rule the suffix of the content has been selected for both cases, providing two different subrules.



```
alert tcp 192.168.0.1 80 -> 66.45.0.234 65
(content: “hack|98 65 FA|!”; sid=5464)

alert tcp 192.168.0.1 80 -> 66.45.0.234 65
(content: “|98 65 FA|!”; sid=5464)

alert tcp 192.168.0.1 80 -> 66.45.0.234 65
(content: “hack|98 65 FA|”; sid=5465)

alert tcp 192.168.0.1 80 -> 66.45.0.234 65
(content: “k|98 65 FA|”; sid=5465)
```

Figure 3.10: Example of extracting a unique part for two rules that have the same header part and their contents differ in just one token. Length was selected to 4 bytes.

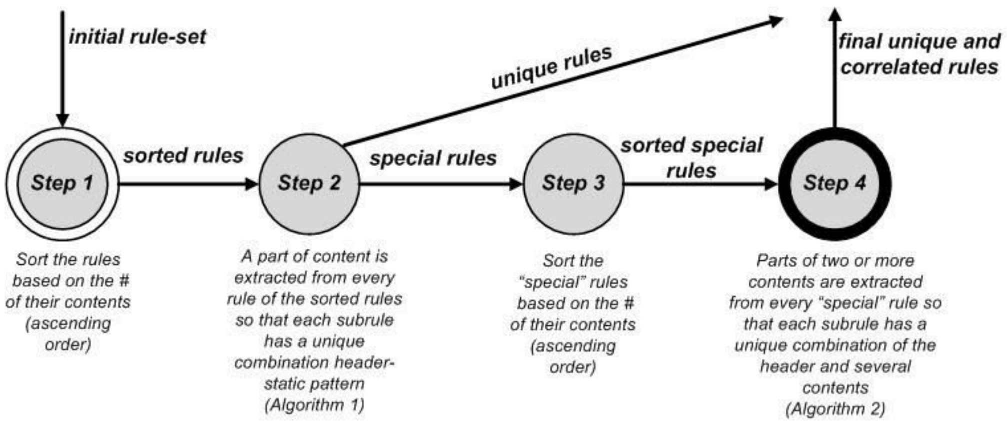

initial rule-set
unique rules
final unique and correlated rules
Step 1
sorted rules
Step 2
special rules
Step 3
sorted special rules
Step 4
Sort the rules based on the # of their contents (ascending order)
A part of content is extracted from every rule of the sorted rules so that each subrule has a unique combination header-static pattern (Algorithm 1)
Sort the “special” rules based on the # of their contents (ascending order)
Parts of two or more contents are extracted from every “special” rule so that each subrule has a unique combination of the header and several contents (Algorithm 2)

Figure 3.11: The operation of Unique Part Rule extraction technique is divided in 4 steps. The input is the entire initial rule-set and the output is the unique subrules along with some rules which are exploited by the Rule Correlation technique.

The operation of Unique Part Rule technique is divided in 4 steps as Figure 3.11 shows. The steps are explained in more detail below:

1. *Sort all the rules in ascending order based on the number of their contents*: The rules are sorted in an ascending way starting from the rules that contain the minimum number of contents. This is done so that rules with one content are processed before rules with more than one contents because more unique parts can be found in the latter rules than the former ones.

2. *Find a unique combination of header and a part of one content for every rule of the sorted rule-set (Algorithm 1)*: The entire sorted rule-set is processed and from every

rule, a part of only one content is extracted so that the resulted subrule has a unique combination of header and static pattern (Algorithm 1). The outcome of Algorithm 1 is the first unique rules, the Special rules and the first correlated rules. The unique rule is the subrule for which a unique combination of the initial rule's header and a part of only one of its static patterns (if more than one) was found. The Special rule is a rule for which a non-unique combination was found but has more than one contents and will be processed again, later in step 3. Finally, the correlated rules are rules with one static pattern for which a non-unique combination was found and are utilized by the Rule Correlation technique.

3. *Sort the Special rules like step 1*: The Special rules are sorted in ascending order based again on the number of their contents.

4. *Find a unique combination of header and parts of two or more contents for every rule of the sorted Special rules (Algorithm 2)*: The sorted rules are processed using Algorithm 2 which provides the rest of unique rules and other correlated rules. This algorithm processes every Special rule, which has been output from step 3, and attempts to find a unique subrule searching at the beginning (phase A) for unique combinations of parts of two contents and the rule's header. If this fails, phase B of Algorithm 2 is applied where search for unique combinations of the header and more than two contents' parts is performed. The output is the found unique rules while the other correlated rules are those rules for which a unique combination was not found in phase A and cannot be proceeded to phase B due to the lack of more than two contents or rules for which both phases have been failed.

The first and the third step are straightforward. The rest of this section describes Algorithms 1 and 2.

Step 2: Find a unique combination header-content's part for every rule of the sorted rule-set (Algorithm 1)

The idea behind Algorithm 1 of the Unique Part Rule technique is that a part of the content is taken and is compared with every selected, until now, static pattern part (that leads to a unique rule) to check whether it exists or is unique. If it exists into the list, the header parts of the rules that have the same content part are compared and if they also match, then another part of the same content or of another content (if possible) is taken and the whole procedure is repeated, until a unique content part or different headers of rules that have the same part exist. If a unique portion of a rule with more than one contents is not found, then the rule is named "Special rule" and is processed later by Algorithm 2.

At the beginning of Algorithm 1 and for each rule, its contents are sorted based on their size so that the bigger contents will be at the beginning of the rule. This decreases the needed amount of computations because more different (and thus unique) parts can be found in a large content. The flag "success" is used in the algorithm to notify when a unique rule was found so that part search must be stopped for this rule. In line 10, a temporally found part matches a part of a unique rule but the header parts of the two rules mismatch and thus, this rule is unique. On the other hand, in line 14, the

Algorithm 1 This algorithm extracts a part of content for each rule so that each subrule has a unique combination header-static pattern.

Input: The sorted list of rules of first step and the desired part length.
Output: The Unique rules. The Special rules which will be processed by Algorithm 2 and the correlated rules.

```
1: $success = 0$
2: for all rules do
3:     sort the contents based on their sizes in descending order
4:     for all contents do
5:         repeat
6:             take part (of specified length) of the content starting from the suffix
7:             if part exists in the list of unique content parts (thus, non unique part) then
8:                 compare headers
9:                 if headers mismatch then
10:                    $success = 1$
11:                    Exit loop
12:                end if
13:            else
14:                $success = 1$
15:                Exit loop
16:            end if
17:        until all possible parts
18:        if $success == 1$ then
19:            Exit Loop
20:        end if
21:    end for
22:    if $success == 1$ then
23:        store unique rule and save its content and header parts into the respective lists of unique content parts and header parts
24:    else
25:        if $the\ number\ of\ contents > 1$ then
26:            store the whole rule in the Special rules
27:        else
28:            store the whole rule along with the correlated unique rule, in the correlated rules
29:        end if
30:    end if
31: end for
```

"success" flag is asserted when a content part, which is not in the list of content parts of unique rules, was found and thus, a unique rule was found and again the search must be terminated. Otherwise, if the "success" flag is not asserted, the whole procedure for all possible parts of one content and for all contents is repeated. The part of the content is extracted based on the user defined length and starting from the last part of the content (suffix). For example, if the content is "*doorop*|00|*en*" and the user defined length is 4 then the first part will be "*p*|00|*en*" and if the "success" flag is not asserted, in the second iteration of "repeat-until" loop, the part "*op*|00|*e*" will be selected, etc. Of course, and similarly to the First Content Prefix extraction technique, the HEX digits are counted in pairs.

At the end of the algorithm 1 (lines 22-30), the "success" flag is evaluated. If, truly, a unique rule is found, it is stored in the subrules and the header and content parts are saved in the list of parts of unique rules which, as already mentioned, is used during the search above. If a unique rule is not found, either it will be processed later (Algorithm 2 is applied in step 4) if it contains more than one contents, or it will be considered as a correlated rule. A rule of the last category will be utilized by the Rule Correlation technique. This rule has been correlated with one unique rule (probably among many), with which it has the same header and content part. Rule Correlation is further explained in section 3.5.

At the end of Algorithm 1, there are three categories of output rules: 1) Unique rules (ready to be uploaded in the Pre-filtering phase), 2) Special rules which will be processed by Algorithm 2 and 3) Correlated rules which are going to be used according to Rule Correlation technique which is described in section 3.5. Consequently, the next step is to process the Special rules.

Step 4: Find a unique combination header-(two or more) contents' parts for every rule of the sorted Special rules (Algorithm 2)

The Special rules have already been sorted in step 3. Also, the Special rules have already their contents ordered (from bigger to smaller), from the previous algorithm. It is good to remind the reader that Special rules are rules which contain more than one contents. That is because in Algorithm 2, the idea is to extract parts (of predefined length) from more than one contents. The idea is that firstly, parts from two whichever contents are extracted in order to find a unique rule and if we run out of all the possible combinations, parts from more than two contents (if possible) are taken.

Algorithm 2 consists actually of two parts that search for a unique rule. The first part (phase A) takes place from line 3 to 29 while the second one (phase B) takes place from line 30 to 45. In phase A, we try to extract parts from two whichever contents of the processed rule. The search starts from the first content and all the combinations between this and the rest of the contents after it, are checked. If the search is not successful, the combinations of the second content and the rest of them after it, are examined, and etc. After having taken the part of one content, a part is taken among all the parts ("repeat-until" loop iteration) of another content (among all possible, second "for" loop) and the two parts are concatenated into one in order to search if the specific combination is already used by a unique extracted Special rule. List "B" contains all the concatenated parts of already extracted unique Special rules. The "success" flag is also used here in the same way as in Algorithm 1.

If the first part of the algorithm failed to provide a unique rule (actually subrule) using a combination of two whichever content parts, then phase B is applied which is

Algorithm 2 This algorithm takes as input the Special rules (output from the Algorithm 1) and extracts a part for more than one of its contents so that the combination of parts of several contents and the header make the rule unique. (Algorithm continues in the next page)

Input: The Special rules of Algorithm 2, sorted in the same way as input rules of Algorithm 1.
Output: The Unique Special rules and the correlated rules.

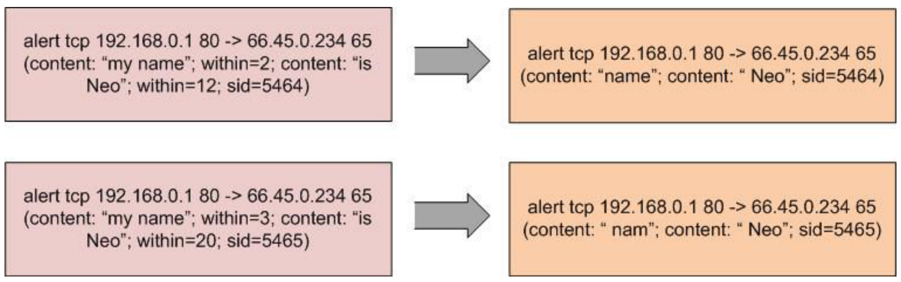
```
for all Special rules do
    $success = 0$
    for each content i do                                              ▷ (Phase A)
        $part1 = Take\_Part(i^{th}\ content)$
        $stored\_parts[i] = part1$
        for each content j after $i^{th}$ content do
            repeat
                take part of $j^{th}$ $content$ and store it in part2
                $part = concatenate(part1, part2)$
                if part exists in the list B of unique parts then
                    compare headers
                    if headers mismatch then
                        $success = 1$
                        Exit loop
                    end if
                else
                    $success = 1$
                    Exit loop
                end if
            until all possible parts
            $stored\_parts[j] = part2$
            if $success == 1$ then
                Exit loop
            end if
        end for
        if $success == 1$ then
            Exit loop
        end if
    end for
    if ($success == 0$) AND ($the\ number\ of\ contents > 2$) then     ▷ (Phase B)
        $part = concatenate(stored\_parts[1], stored\_parts[2])$
        for $i = 3$ to last stored part, of stored_parts[], do
            $part = concatenate(part, stored\_parts[i])$
            if part exists in the list B of unique parts then
                compare headers
                if headers mismatch then
                    $success = 1$
                    Exit loop
                end if
            else
                $success = 1$
                Exit loop
            end if
        end for
    end if
    if $success == 1$ then
        save the concatenated multi-content part and rule's header in list B,
        insert all the content parts separately in list A if they do not exist,
        store the unique Special rule with the multiple content parts separated
    else
        store the whole rule along with the correlated unique rule, in the correlated rules
    end if
end for

function TAKE_PART(C)                              ▷ C is the content argument
    repeat
        take part (of predefined length) of C starting from suffix
        if part exists in the list A of unique parts then
            compare headers
            if headers mismatch then
                Exit loop
            end if
        else
            Exit loop
        end if
    until all possible parts
    return part  ▷ if, unsuccessfully, non unique part found, the last extracted one is returned
end function
```

also the last opportunity for finding a unique portion for the current rule. In this part, combinations of more than two content parts will be used and for that reason, the rule must contain more than two of them. The list "stored_parts" has, from phase A, a specific part for each and every content. The first two content parts are concatenated (this is not unique) and in each iteration a new content part is concatenated to the previous part until a unique rule found (if possible) and "success" flag is asserted.

If a unique Special rule was found ("success" flag was asserted) from either case, it is stored in the unique Special rules and its concatenated part is stored in list "B" while its contents parts are stored separately in list “A”, if they are not exist. The contents of the extracted unique rule are printed separately and not as one concatenated content part because this would be totally wrong. For example, consider the first case of Figure 3.12. If the content parts were used as one content part then the extracted rule 5464 would have the content “name Neo” which match other packets than packets that contain the patterns “name” and “ Neo” separately. Thus, the concatenation of the content parts could lead to mistakes that can cause potential threats to pass as safe from the NIDS. The concatenated part is used only for comparisons to determine whether a specific combination of content parts has been already used by previously extracted unique subrules.

Finally, the function “Take_Part”, actually, implements the extraction of part based on the user-defined length and the content, as shown in lines 55-68. It works similarly to “take part” of Algorithm 1 except the fact that for every part, the list “A” of unique parts is searched for a possible match and if so, the header parts of the respective rules are also compared. If all the parts of this content are used already, the last part (it is actually the prefix) is returned. The list “A” contains all the content parts (separated and not concatenated) of the unique Special rules.



```
alert tcp 192.168.0.1 80 -> 66.45.0.234 65
(content: “my name”; within=2; content: “is
Neo”; within=12; sid=5464)
```

```
alert tcp 192.168.0.1 80 -> 66.45.0.234 65
(content: “name”; content: “ Neo”; sid=5464)
```

```
alert tcp 192.168.0.1 80 -> 66.45.0.234 65
(content: “my name”; within=3; content: “is
Neo”; within=20; sid=5465)
```

```
alert tcp 192.168.0.1 80 -> 66.45.0.234 65
(content: “ nam”; content: “ Neo”; sid=5465)
```

Figure 3.12: Example of making two subrules from two Special rules. Suppose that all the possible parts of their contents had been used by unique rules during execution of Algorithm 1. Length was selected to 4 bytes.

Figure 3.12 depicts an example of creating unique subrules from step 4. Suppose that all the possible combinations of parts of contents are used by unique rules which were extracted by the Algorithm 1. It is assumed also that the Special rules of Figure 3.12 are the first Special rules which are processed by Algorithm 2. Rule 5464 is extracted quite easily taking the suffixes of the two contents. The concatenated content part is stored in list B and the two separated contents are stored in list “A”. For rule 5465, “Take_Part” function is called for the first content. The part “name” is selected but it exists also in list “A” (from rule 5464). The headers (of the rules of the matched contents) are also the same and thus the next part which is “ nam” is selected and returned (it does not exist in list “A”). Then, the second loop starts executing. There is only one more content. The part “ Neo” is concatenated to “ nam” and gives part “ nam Neo” which does not exist in list B, and thus successfully leads to an extracted portion of rule 5465.

In both algorithms 1 and 2, when there is failure to find a unique rule the whole rule is placed into a special category which is called “correlated rules” along with one unique subrule (its ID). The issue of how these rules are used is totally related with the idea of the Rule Correlation technique and is going to be explained thoroughly in section 3.5. However, it must be explained how the specific unique rule was selected among, probably, many. A list of all the unique rules, which have been used as correlated with a non-unique, has been created and keeps track of their status (status of unique rules) about how many times they have been used as correlated until now. Each time it is found (in both algorithms) that a rule matches a unique rule in both header and content parts during its extraction process, the information of this unique rule is copied from the previous list and is added to a temporal list of possible unique rules that may be used as correlated in case the currently processing rule is not unique. At the end, and if the processing rule is not indeed unique, the ID of the unique rule that was tracked the least times is selected and the counter of the respective unique rule in the centralized list is incremented by one.

To conclude with this section, the idea of the Unique Part Rule technique was described in detail. The strategy which must be followed was presented. In addition, the technique's algorithms were presented and explained thoroughly. Algorithm 1 is applied first and attempts to extract a part of a rule's static pattern so that a unique combination of this and the rule's header may exist. The majority of the unique subrules are extracted from this algorithm, as it is also shown in Chapter 4. The rules with more than one contents for which non-unique combinations were found, are processed by Algorithm 2 which attempts to find unique combinations of the rule's header and parts of more than one contents. The Special unique rules are extracted from this algorithm. Also there is a number of rules that are named as correlated and are going to be utilized by the Rule Correlation technique which is discussed in section 3.5. It was understood why extracting a unique portion of the rule is more sophisticated than simply taking the prefix of the first content. Of course, it is a method that needs more preprocessing than the First Content Prefix technique but significantly less preprocessing than PCRE Prefix technique. However, it provides an efficient set of subrules, the processing of which during the Pre-filtering stage is lightweight like in First Content Prefix technique and less number of rules is expected to be activated than using the other extraction techniques.

3.5 Rule Correlation

Generally, the correlation as a term is defined as a statistical technique that can show whether and how strongly pairs of independent variables are related. The variables here are the rules. The main idea of it is to find an efficient way to correlate the rules in order to improve the processing of the Pre-filtering stage and find as fast as possible the activated rules for a single incoming packet. Thus, some rules (perhaps all) are correlated with each other and afterwards, they are divided in small partitions where in each one of them there is one rule "leader". If this rule is matched then all the rules that are correlated to this one (all the partition's rules), are considered as activated and are sent together to the second phase without needing to partially match them too. The issue is: according to which criteria a rule is selected as a rule "leader". Two types of Rule Correlation (Quantitative and Qualitative) are proposed and explained later in this section, but only one of them (Qualitative) follows exactly the main idea of Rule

Correlation.

A significant amount of research has been done at the area of correlation in NIDS but very little, to the best of the author's knowledge, about the way that was described before. Most of researchers [37], [40] and [46] are targeting for techniques and strategies of correlating alerts in order to construct attack graphs hence, intrusion scenarios which can ease the intrusion detection. Thus, they try to correlate alerts (not rules) with each other using specific fields of the alerted packets like the source and destination address and port numbers. They try to find similarities on these fields between a current alert packet and a previous one, the number of times they were matched, etc [46]. Other, like [40], use the previous characteristics and probabilistic theory to achieve their goal and provide more realistic alert correlation. The drawback of this kind of correlation is that the correlation is based significantly on the experiment and additionally to the quality and amount of the intrusion tests that were used for the experiments.

In our case, the goal is to correlate the rules themselves. Two types of Rule Correlation are defined: the Quantitative and the Qualitative and they are described below:

The **Quantitative** Rule Correlation is very similar to the alert correlation that was discussed in the previous paragraph. The idea of this kind of Rule Correlation is to use one of the proposed Pre-filtering techniques (sections 3.1 - 3.4) and make a lot of experiments using attack traces to find how many and which rules are activated in order to correlate them. Using this data, an undirected weighted correlation graph can be constructed which shows which rules are correlated with each other and how often (weight). Afterwards, a min-cut algorithm can be applied to this graph and break it into smaller subgraphs which will be the correlation groups (partitions). However, this type of Rule Correlation suffers from the drawback that a "leader" rule cannot be selected because there is not sufficient data to distinguish a special one among the others. For that reason, the Quantitative Rule Correlation does not follow exactly the proposed idea of Rule Correlation, which was mentioned at the beginning of this section. However, it can be used in the second stage of processing of a hardware NIDS, which uses Pre-filtering, organizing the rules of every single partition into the memory in that way so that spatial locality is exploited when downloading the firmware of the activated rules onto the Processing Elements reducing possibly the latency due to memory accesses. Finally, another drawback of the Quantitative Rule Correlation is that is limited from the number and the quality of the experiments. It is based more on heuristics and not on the rules' features.

The **Qualitative** Rule Correlation, on the other hand, does not depend on heuristics or experiments because the rules are correlated based on their internal characteristics and more specifically on their header and payload features. It follows exactly the proposed idea of Rule Correlation, which was explained at the beginning of this section and attempts to optimize the first stage of processing of a NIDS that uses Pre-filtering in terms of speed. In this thesis, the proposed Qualitative Rule Correlation technique is used only with the Unique Part Rule extraction technique. In section 3.4, it was mentioned that when it is impossible to find and extract a unique portion of a rule, it is added to the correlation rules. It is reminded that the problem of not finding unique portions for some rules is created due to the fact that the header part and all the possible combinations of the content parts have already been used by other rules that were processed and extracted before. Thus, if a random part of the content was selected, the case of two (or more) same rules during Pre-filtering could appear and be activated together. The drawback is that extra processing is needed in order to match all these same rules during Pre-filtering. Instead of that, these rules are excluded from the set of the unique subrules and are utilized by the Qualitative Rule Correlation technique.

All the rules that are utilized by the Qualitative Rule Correlation technique are assigned with a correlation ID. This is the Signature ID (SID) of the unique subrule, with which the rule is correlated, as it was described at the end of section 3.4. Then, the non-unique rules are partitioned according to this correlation ID. The goal of the Qualitative Rule Correlation technique is that every partition should consist of as few as possible non-unique rules so that if an activated unique rule is a "leader" of a partition, the fewest possible rules will be sent to the Full Match stage. Thus, the number of times that a unique rule was used as a correlated rule is tracked during the rules' extraction by applying the Unique Part Rule technique. If a non-unique rule is matched with many unique ones, the correlated ID is selected to be the SID of the unique rule with the minimum number of times that was used as correlated and its status counter is incremented by one. All the non-unique rules are divided in small groups (partitions) and each group has a "leader" or representative rule which is one specific unique subrule. The maximum number of rules per group can reach the 16 rules but it is rare (in 8 out of 500 groups), while the average number is 3 rules/partition.

In this thesis, both Rule Correlation techniques were studied but the Qualitative is the proposed one. A lot of experiments were performed in order to exploit Quantitative Rule Correlation technique but more experiments should have been done in order to have valid results. In research studies, it has been mentioned that a long time of experiments (around two months) is needed in order to have enough data to correlate alerts, which is probably similar to Quantitative Rule Correlation if the needed procedure (heuristics and experiments) is taken into account. This is another reason to prefer Qualitative Rule Correlation instead of the Quantitative one.

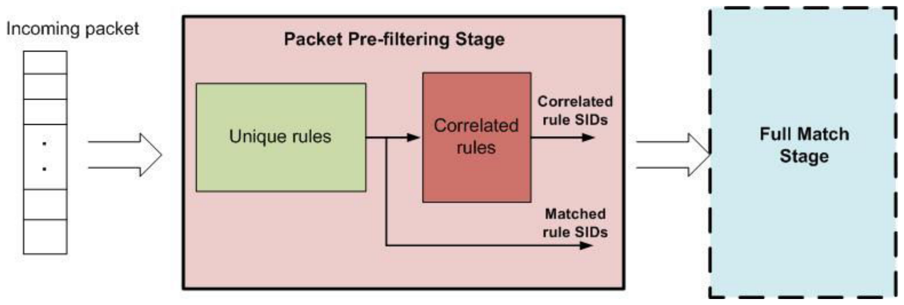


Figure 3.13: How the Qualitative Rule Correlation technique is used in the Pre-filtering stage. The Pre-filtering stage is loaded with the subrules, extracted by the Unique Part Rule technique. The incoming packet activates some unique rules. Some of them may be used to access a small memory which keeps the correlation rules and is indexed based on the "leader" unique rules of each partition. All the activated rule SIDs and the correlated rule SIDs are sent to the second phase for full match along with the incoming packet.

This section is concluded by showing the way in which the proposed Rule Correlation technique can be used in the Pre-filtering phase as shown in Figure 3.13. The activated rule Signature IDs (SIDs) are used to access the memory that keeps the correlated rule SIDs and all of them are sent together to the next stage for full match. To conclude, the Rule Correlation technique can speed up the procedure of finding the activated rules that must be sent to the next phase, improving the Pre-filtering stage's performance. However, the correlated rules must be grouped into small partitions so that as few as possible rules would be sent in the next phase maintaining the goals of the Pre-filtering idea. Among the two Rule Correlated techniques, the Qualitative one is more clever because it is based on the features of the rules and not on heuristics or long time experiments.

3.6 Smart Rule Reuse

Until now, all the proposed Pre-filtering techniques attempt to create a set of subrules, for the Pre-filtering stage, so that after a lightweight processing of it against a single incoming packet, the fewest possible rules are going to be fully matched in the Full Match stage. The Smart Rule Reuse technique approaches a hardware NIDS that uses the Pre-filtering concept from a different point of view. It attempts to improve the second stage of processing based on the observation that *there may be common activated rules between two or more consecutive packets*, hence re-downloading the firmware of them onto the Processing Elements (PEs) is not necessary, as they already exist in the PEs. Smart Rule Reuse attempts to exploit the temporal locality between the rules, similarly as it is exploited in caches in Computer Architecture.

Suppose that we have a hardware NIDS similar to PINE which was proposed by Sourdis in [32]. PINE is depicted in Figure 2.5 of Chapter 2. As it was mentioned in section 2.2.2, the Full Match stage in PINE consists of two Engines: the Guaranteed Throughput Specialized Engine (GTSE) and the Best Effort Processing Engine (BEPE). The GTSE has a number of PEs where the full match takes place by downloading the firmware of the activated rules onto the PEs. Every single PE actually matches one rule against the incoming packet. If the number of activated rules is larger than the number o GTSE's PEs, the Full Match takes place in BEPE. Smart Rule Reuse attempts to optimize the GTSE by monitoring which rule is processed on which PE to avoid re-downloading of the firmware for common rules between two or more consecutive packets. A Smart Rule Reuse block is used for this purpose. Figure 3.14 depicts a GTSE which uses the Smart Rule Reuse technique. The GTSE consists of the PEs, the Configuration Memory which contains the firmware of the original rules and the Smart Rule Reuse block which maps every single PE to a rule. The controller communicates with every unit and transfers the rules' firmware from the memory to the PEs using the mapping of the Smart Rule Reuse block. The Smart Rule Reuse block is provided with the IDs of the activated rules by the controller and provides to the Controller the mapping. In the same figure, the IDs of the activated rules of the previous packet are the $\alpha_1 - \alpha_8$ while $\beta_1 - \beta_8$ are the IDs of the activated rules of the current packet.

It was mentioned that the Smart Rule Reuse block maps every PE to a single rule ID. If two or more consecutive packets have common activated rules, it is not needed to

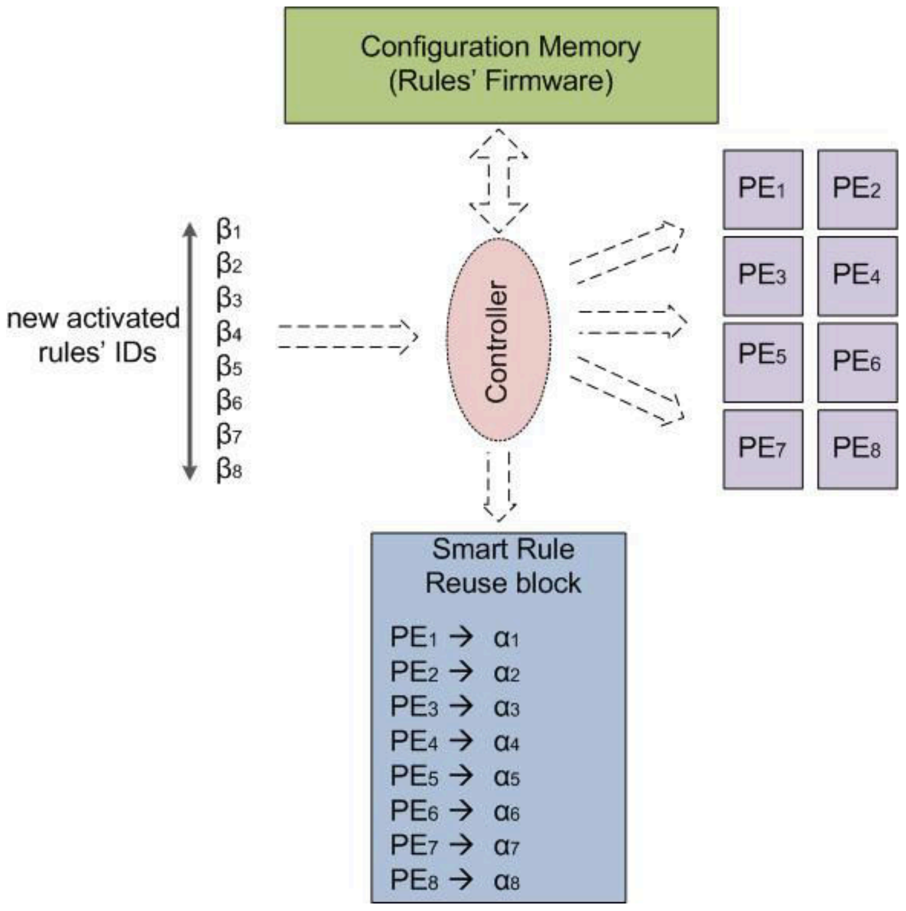
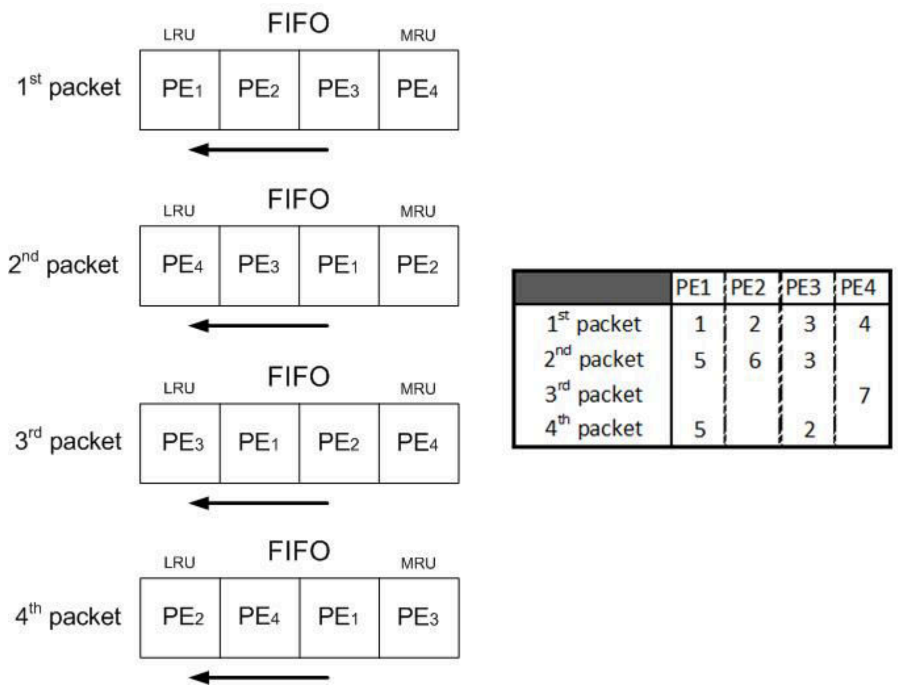


Figure 3.14: Applying the Smart Rule Reuse technique in the Guaranteed Throughput Specialized Engine (GTSE) of PINE. The GTSE consists of the PEs (here 8), the Configuration Memory which contains the firmware of the original rules and the Smart Rule Reuse block which maps every single PE to a rule. The controller communicates with every unit and transfers the rules' firmware from the memory to the PEs using the mapping of Smart Rule Reuse block. Finally, it accepts the newly activated rules' IDs and transfers them to the Smart Rule Reuse block. The $\alpha_1 - \alpha_8$ are the activatd rules' IDs of the previous packet while $\beta_1 - \beta_8$ are the activated rules' IDs of the current packet.

download again the firmware of these rules but only of the others. This means that the PEs which had been previously mapped to these rules will perform again full match of them but against the new incoming packet. In addition, the Smart Rule Reuse block must assign the other rules to the remaining PEs. This is not performed randomly but there is a special FIFO made by registers (sequence of registers), which keeps the PEs' IDs. The most recently used (MRU) PEs are pushed into the FIFO while the least recently used (LRU) ones are pulled out of it. It is special because a PE which is in the middle of the FIFO can be moved from its position to the first and most recently used position and all the other rules before it are shifted one position. Consequently, if there is a common rule between the consecutive sets of activated rules, this is considered as a "hit" and the respective PE is moved from its current position of the FIFO to the most recently used one and all the other rules before it are shifted one position. If a "miss" takes place, or in other words a new rule came and must be assigned, all the PEs are shifted one position and the least recently used PE becomes the most recently used, while the new rule is assigned to it. The FIFO depth equals the number of PEs.



	PE1	PE2	PE3	PE4
1st packet	1	2	3	4
2nd packet	5	6	3	
3rd packet				7
4th packet	5		2	

Figure 3.15: Example of the FIFO mechanism of the Smart Rule Reuse block. The first step is to search the newly coming rules' IDs if there are common rules and if so, the respective PEs are moved from their location to the MRU ("hit") and afterwards, the other rules are assigned to the remaining PEs ("miss"). That is the reason why going from the 3rd packet to the 4th, the sequence of the PEs (going from LRU to MRU) becomes "PE_2, PE_4, PE_1, PE_3" (instead of "PE_2, PE_4, PE_3, PE_1") and rule ID 5 is assigned to PE_1, as in the second packet and rule ID 2 to PE_3.

A simple example of the mechanism is depicted in Figure 3.15. The depth of the FIFO is 4 because the number of the PEs is assumed to be also 4. When new activated rules come, it is firstly searched if there are common rules with the previous packet and if so, the respective PEs change their positions in the way of a FIFO "hit" as it was explained before and afterwards, the completely new rules are assigned to the remaining PEs. In Figure 3.15, at the beginning, rule 1 is assigned to PE_1, rule 2 to PE_2, rule 3 to PE_3 and rule 4 to PE_4. In the second packet, the activated rules are the IDs 5, 3, 6. The first step to check if there is any common rule with the previous packet. It is the rule ID 3. The respective PE, which is the third is moved from its current position to the beginning and PE_4 is shifted one position. The rule IDs 5 and 6 are totally new and assigned to PE_1 and PE_2 respectively by moving the respective PEs from the end to the beginning of the FIFO, giving in total 2 misses and one hit. Similarly, the FIFO sequence in the 4th packet is “PE_2, PE_4, PE_1, PE_3” instead of “PE_2, PE_4, PE_3, PE_1”. In the 3rd packet, there is only one activated rule which will be fully matched against the packet in PE_4. The rules of the other PEs from the previous packet are not unloaded, since they may be used in the future, but it is assumed that the respective PEs are disabled for inspecting this packet.

In conclusion, the Smart Rule Reuse technique can accelerate significantly the second phase of a NIDS that uses Pre-filtering, if it is similar to GTSE. Considering that many attacks spread to many packets, the probability that two or more consecutive packets may have some same activated rules is significant. This improves the performance of the second stage and of the total system because less memory accesses are needed to transfer the firmware of the newly activated rules from the main memory to PEs for the full matching. The mechanism of the Smart Rule Reuse block was also proposed. The assignment of the rules to the PEs is not performed randomly but is based on how recently the PEs have been used. This is known based on their position on a FIFO. It is expected that Smart Rule Reuse technique can optimize the second stage of a NIDS and this will be determined soon in the next chapter where the evaluation of it is taking place.

3.7 Summary

In this chapter, it was discussed about different techniques which attempt to improve a NIDS which uses Packet Pre-filtering, by improving both of its stages. At the beginning of the chapter, it was discussed which features from the rules (header, payload and no-payload) were considered as the most important for the Pre-filtering phase and were selected. Especially from the payload features, the features which require a lot of computations were excluded and the Content and PCRE fields were chosen to be exploited. In addition, section 3.1 provided with a lot of significant details about these specific fields because they are exploited by the three extraction techniques.

The next three sections after 3.1 proposed the three extraction techniques, each one of which attempts to create a set of efficient subrules. Section 3.2 proposed the First Content Prefix extraction technique which is the most trivial one. It extracts the prefix of the content (first content if more than one) in a trivial way. It requires very lightweight preprocessing of rules but many extracted rules may be the same because they may have the same header and content’s prefixes.

In section 3.3, the second technique was proposed. In this case, it was tried to extract a prefix from the PCRE field of the rule. The prefix of a content is extracted only in the cases of the rules that do not contain a PCRE field or it is impossible to extract a prefix from it. Additionally, the extraction rules were presented and explained thoroughly. The PCRE prefix extraction must be performed according to a specific procedure along with the proposed extraction rules to have a correct and efficient subrule-set.

Section 3.4, on the other hand, proposed the Unique Part Rule technique. It attempts to find a unique combination of the header and a part(s) of content(s) in order to create a set of unique subrules for the Pre-filtering stage. The whole procedure, which must be followed, was presented along with two algorithms which perform this task. This proposed technique is the most sophisticated and efficient among the other extraction techniques. However, it requires acceptably more computational power on preprocessing the rule-set than the First Content Prefix and significantly less than the PCRE Prefix, as it is presented in Chapter 4.

The last two sections, sections 3.5 and 3.6 proposed two different techniques which improve the speed of both stages of a NIDS that uses Pre-filtering. The Rule Correlation technique optimizes the Pre-filtering phase while the Smart Rule Reuse technique optimizes the Full Match phase. Section 3.5 proposed the Qualitative Rule Correlation technique which utilizes the rules for which non-unique extracted portions (combinations of header and content(s) part(s)) were found during the extraction, applying the Unique Part Rule extraction technique. Every single one of them is correlated with a particular unique subrule which had been used the least times as a correlated rule. All these correlated rules are divided in small partitions and each partition is led by one representative rule, which is the unique subrule. Thus, if a unique subrule is matched by the basic Engine of the Pre-filtering phase and is indeed a "leader" of a partition, the unique subrules matched by the basic Pre-filtering Engine and the correlated rules (without needing to partially match them too) are sent to the next phase to be fully matched.

Section 3.6, on the other hand, proposed the Smart Rule Reuse technique, which is an efficient way of mapping the PEs to the rules which must be fully matched against the incoming packet in the second stage of a hardware NIDS that uses Pre-filtering. The Smart Rule Reuse technique attempts to optimize the Full Match stage. Its idea is based on the observation that some activated rules between two or more consecutive packets may be the same. As a result, it is unnecessary to download the firmware of these same rules to PEs again, since they have already been there from the previous packet inspection. A FIFO-based mechanism was also proposed on how to assign the rule IDs to the PEs.

To conclude with this chapter, several different techniques for the Packet Pre-filtering were proposed and described. The first three extraction techniques attempt to provide an efficient set of subrules in terms of low processing and high efficiency. On the other hand, the other two techniques approach Pre-filtering from a different point of view and optimize both stages of a NIDS that accommodates the Pre-filtering idea. All these techniques which have been discussed in detail during this chapter are going to be used in practice, in the next chapter, in order to be evaluated.

Evaluation 4

The previous chapter discussed in detail the proposed techniques that were found during this thesis. All the proposed techniques can be applied to a NIDS that uses Packet Pre-filtering. The First Content Prefix, PCRE Prefix and Unique Part Rule extraction techniques attempt to create the set of subrules which is loaded to the Pre-filtering stage. The Rule Correlation attempts to improve the performance or the implementation cost of the Pre-filtering stage by correlating similar rules which are activated without partially matching them and finally, the Smart Rule Reuse optimizes the Full Match stage in terms of speed, taking advantage of the temporal locality of the activated rules that may be the same between two or more consecutive packets, avoiding re-downloading their firmware and hence, reducing the latency due to the reduction of memory's accesses. It is a good point to mention here that the Rule Correlation and the Smart Rule Reuse techniques are orthogonal to the other extraction techniques. In other words, it is possible to create the set of subrules, applying the Unique Part Rule extraction technique and simultaneously use the Smart Rule Reuse technique in the second stage of the NIDS. However, it is not possible to produce the subrules using, for example, the First Content Prefix and the Unique Part Rule extraction techniques simultaneously. This chapter evaluates all the proposed techniques in order to determine whether they accommodate Pre-filtering to meet its requirements and even further to be improved and enhanced.

The evaluation of the proposed techniques is performed using well suited metrics which highlight their advantages and disadvantages in practice. Section 4.1 discusses the experimental setup which is preceded the evaluation and afterwards, in section 4.2, the evaluation of the Pre-filtering techniques takes place. The chapter concludes with section 4.3 which summarizes the most important conclusions of the evaluation.

4.1 Experimental Setup

The first issue which must be addressed before proceeding to the evaluation of the proposed techniques is the experimental setup. It is essential to mention some important implementation issues that were addressed during this thesis because they are also relative to the evaluation. The first implementation issue was actually the way of evaluating the proposed techniques. The second issue is based on the techniques themselves. In chapter 3, three extraction techniques were proposed. In Unique Part Rule, some extraction algorithms were proposed on how to extract a unique part of rule and in PCRE Prefix, specific extraction rules were suggested on how to extract a prefix from a PCRE. Starting from the latter issue and going to the former and most important one, the first issue which should have been solved is the selection of a proper programming language for the implementation of the techniques.

The Perl language was selected because of its simplicity and due to the automatic options it offers in cases of text processing. Another important reason why Perl language was selected, was the very nature of the regular expression (regex) field of the SNORT rules, which is actually a Perl Compatible Regular Expression (PCRE). It would be easier to construct the grammar, which is needed for the PCRE extraction, using the PCRE syntactic rules. Also, in Perl community it is offered the opportunity of using some open-source modules which have been created for specific text processing cases, rather than starting from scratch. It was mentioned in section 3.3 that the RecDescent parser, which was created by Damian Conway, was used to parse the input regex, based on the constructed grammar, and eventually generate the Abstract Syntax Tree (AST). The AST is a nice and flexible form to represent the regex and generally a string in order to extract its prefix.

The next implementation issue which has been addressed is the firstly mentioned implementation issue and also the most important. All the proposed Packet Pre-filtering techniques were discovered and studied with the upper goal of exploiting the Pre-filtering concept and make it more efficient. In order to verify that these techniques serve the Pre-filtering approach in practice, they should be tested using a real NIDS system. SNORT was selected to be this NIDS. SNORT has an interesting module, which is called Performance Profiling, that was used in this thesis. Performance profiling gives statistics about the number of times a rule was matched/alerted/passed, the time that was needed to process a rule, etc. This module was used in order to keep track of the matched rule IDs. However, one important problem that was encountered was that SNORT is not processing all the rules. It processes rules using priorities between them and other details hidden to the external user in order to filter out the alerts, out of all the possible matched rules. The problem was that the purpose of this thesis was to evaluate the Pre-filtering techniques for a generic NIDS, regardless of the special characteristics it has. For that reason, the source code of SNORT was studied in order to find ways of "bypassing" these optimizations and eventually give the whole number of rules that were matched (activated) per input packet using a set of subrules. In order to achieve this goal, the SNORT source code was modified in some parts in order to match the whole rule-set against every incoming packet. The parts that were modified are related especially with the Performance Profiling module. Of course, SNORT speed was decreased after that change but the goal was to know how many rules are activated for every single incoming packet.

In summary, it was discussed about the implementation issues that arose during this thesis and especially regarding the evaluation and the experiment itself and how they were resolved. The next topic is to evaluate the proposed techniques. In order to achieve this, real dangerous input traffic should be used. For that reason, DefCon traces of [38] were utilized as an input scenario traffic.

4.2 Experimental Results

The goal of this section is to evaluate the proposed Pre-filtering techniques to conclude whether they can accommodate or even further improve Pre-filtering to meet the requirements of high processing throughput and low implementation cost. The previous section mentioned the issues regarding the experimental setup for the evaluation. This section discusses about the proposed techniques in practice, presents results and compares them.

The first important issue for the evaluation is to construct the evaluation scenarios. The difference of the scenarios is based on the evaluated technique. The evaluation starts from the extraction techniques that were discussed in sections 3.2, 3.3 and 3.4 which propose different approaches on the problem of extracting portions of the NIDS rules and create the set of subrules for the Pre-filtering stage. There are three different scenarios based on the extraction techniques. From now on, they are referred also as "subrule" scenarios:

1. Extract the prefix of the first Content, applying the First Content Prefix technique.
2. Extract the prefix of the PCRE, applying the PCRE Prefix.
3. Create a unique subrule by finding a unique combination of the header and part(s) of Content(s), applying the Unique Part Rule extraction technique.

The rules which have only header are ported in the Pre-filtering stage *unchanged* at all and used in the same way. The rules, which have both content and pcre fields, are processed by extracting the respective part based on the respective extraction scenario. Cases of rules that have only content and the scenario is the second, are processed normally and the prefix of their first content is extracted. On the other hand, if the rules have only PCRE field and the "subrule" scenario is the first one or the third (both are based on content), the prefix of the PCRE is taken using the second "subrule" scenario. It must be reminded to the reader that if a rule has both content and PCRE fields and the selected "subrule" scenario is the second, but the PCRE(s) contain the "R" modifier, the prefix of the first content is extracted. Table 4.1 summarizes the above issues.

Table 4.1: The "subrule" scenarios and in which "extreme" rules cases, alternative scenarios replace them.

Selected "sub-rule" Scenario	Rule Kinds		
	Content & PCRE	only Content	only PCRE
First Content Prefix (1^{st})	1^{st}	1^{st}	2^{nd}
PCRE Prefix (2^{nd})	2^{nd} (1^{st} [1])	1^{st}	2^{nd}
Unique Part Rule (3^{rd})	3^{rd}	3^{rd}	2^{nd}

In addition, taking into consideration the theoretical analysis of Packet Pre-filtering, another parameter which must be taken into account during evaluation is the size (length) of the extracted payload portion. For the evaluation, 4 different lengths were selected to be evaluated: 4, 8, 12 and 16 bytes. Thus, the number of evaluation scenarios for the three proposed extraction techniques are: 3 "*subrule*" scenarios $\times$ 4 "*prefix/part length*" scenarios = 12 "*final*" scenarios.

Tables 4.2, 4.3 and 4.4 provide the distribution of rules into different subrule categories, after applying each of the proposed techniques or in other words, according to the

[1] If the PCRE(s) contain the "R" modifier, the prefix of the first content is extracted (1^{st} scenario).

"subrule" scenario for the 4 different prefix/part sizes. These rule categories are based on the actual length of the portion or which field (Content or PCRE) has been used. Additionally, Figures 4.1, 4.2 and 4.3 depict this rules' distribution in percentage of the whole rule-set, which consists of exactly 8937 rules. For all the proposed extraction techniques there are 125 rules (∼1.5%) which have only header and were not processed at all, but directly ported to the Pre-filtering stage.

Table 4.2: This table shows how the rules are distributed into different subrule categories according to the applied First Content Prefix extraction technique and internal rules' characteristics (Content or PCRE, smaller length), for all the prefix lengths.

Prefix Length (PL)	equal to PL	> PL	< PL	PCRE prefix instead	only Header
4	4377	0	4427	8	125
8	3353		5451		
12	2456		6348		
16	1797		7007		

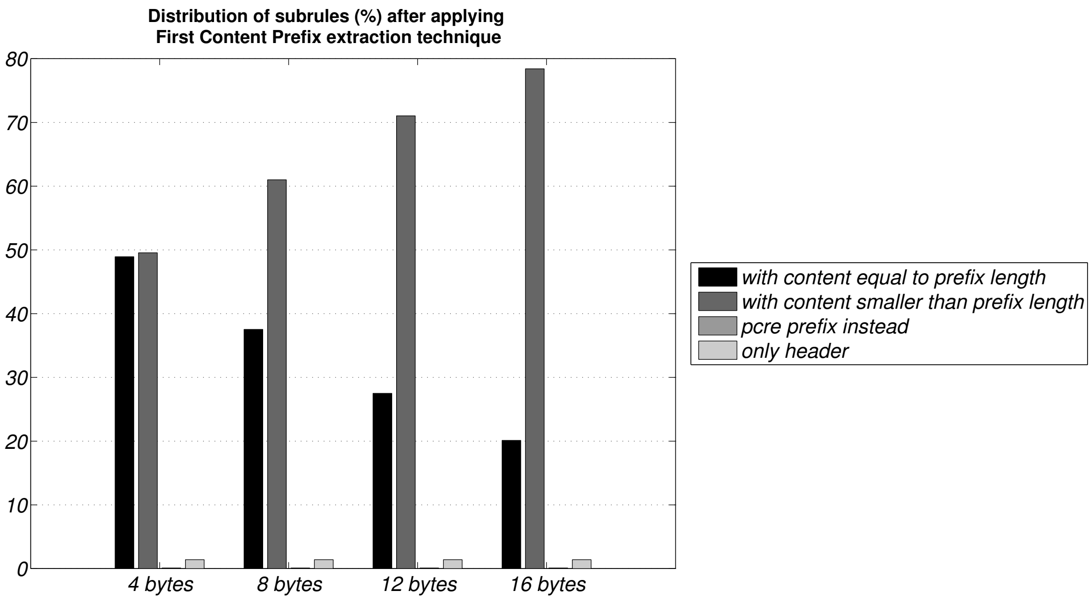


Figure 4.1: This figure depicts the Distribution of Rules (in %) into different subrule categories (due to internal rules' characteristics (Content or PCRE, smaller length)), after applying the First Content Prefix extraction technique, for 4, 8, 12 and 16 bytes of prefix length. The number of extracted rules that have prefix content size equal to the prefix length decreases as the size of prefix increases.

Starting with the First Content Prefix technique (see Table 4.2 and Figure 4.1), it can be observed that the first content's length of almost 50% of the rules is less than 4 bytes. Generally, very small contents (2-3 bytes) increase significantly the probability for the

Table 4.3: This table shows how the rules are distributed into different subrule categories according to the applied Unique Part Rule extraction technique and internal rules' characteristics (Content or PCRE, smaller length), for all the part lengths.

Part Length (PL)	equal to PL	> PL	< PL	Special Unique Rules	Correlated Rules	PCRE Prefix instead	only Header
4	4380	0	647	2092	1685	8	125
8	3416		1628	2440	1320		
12	2597		2449	2417	1341		
16	1897		3149	2370	1388		

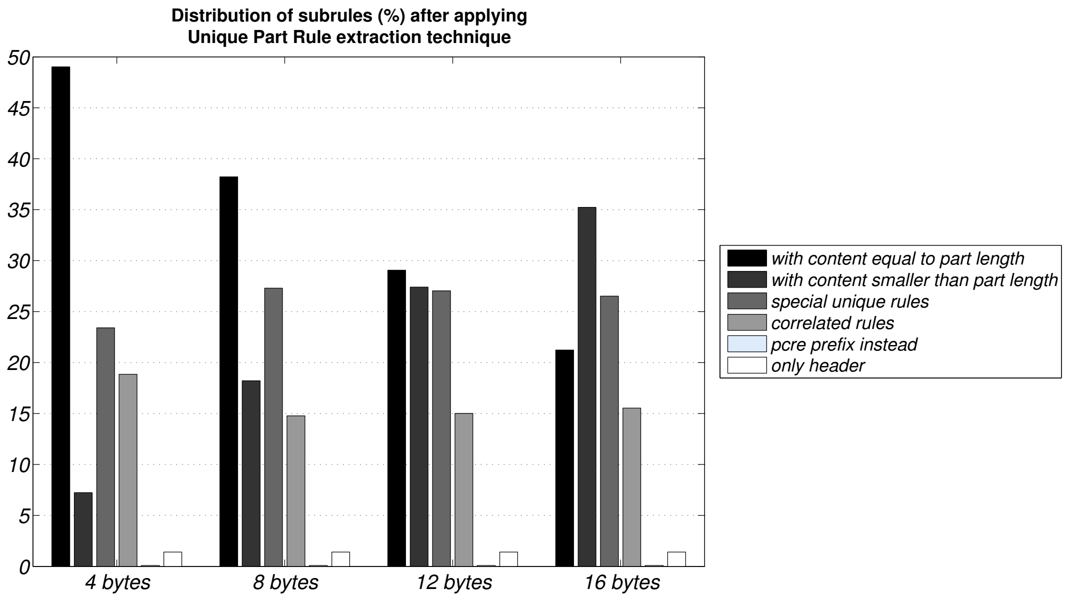


Figure 4.2: This figure depicts the Distribution of Rules (in %) into different subrule categories (due to internal rules' characteristics (Content or PCRE, smaller length)) after applying the Unique Part Rule extraction technique, for 4, 8, 12 and 16 bytes of part length. In this case, around 15-20% of rules will be utilized by the Rule Correlation technique, while around 25% of rules were extracted using Algorithm 2. Again the number of the extracted rules that have content part size equal to the part length decreases as the size of (extracted) part increases.

rules to be activated by many packets. It can also be observed that as the prefix length increases, the number of extracted rules that have prefix length equal to the predefined one decreases. This is the clear disadvantage of this technique: Rules are preprocessed very fast, but the portion of the first content is taken; if it is smaller than the prefix length, the entire content is taken increasing the probability the respective rule to be activated by many packets even though there may exist other bigger contents which would give more efficient prefixes. On the other hand, the sophisticated mechanisms of Unique Part

Rule extraction technique bypass all the previous disadvantages of the First Content Prefix technique (see Table 4.3 and Figure 4.2). Firstly, about 7% of the initial rules have content(s) size less than 4 bytes. Generally, the number of the extracted rules that have content part with length less than the predefined part length is substantially smaller than the case of the First Content Prefix technique. In addition, for all the part sizes, 60% of the whole rule-set can be extracted providing unique subrules, using Algorithm 1 of section 3.4. The rest 40% of rules is processed by Algorithm 2 of the same section. The latter algorithm managed to extract 20-25% Special unique subrules which have a portion of more than one contents, while 15-20% were partitioned in groups and are utilized by the Rule Correlation technique. Finally, using both extraction techniques, there are 8 rules which have only PCRE and are processed using the PCRE Prefix technique.

Table 4.4: This table shows how the rules are distributed into different subrule categories according to the applied PCRE Prefix extraction technique and internal rules' characteristics (Content or PCRE, smaller length), for all the prefix lengths.

Prefix Length (PL)	equal to PL	> PL	< PL	St. Pat. Prefix instead	only Header
4	1582	1760	1540	3930	125
8	1322	2014	1546		
12	1244	2084	1554		
16	1173	2094	1615		

Table 4.4 and Figure 4.3 provide the distribution results of rules into the different subrule categories, for the PCRE Prefix extraction technique. It illustrates a different subrule-set than the previously presented content-based techniques. Firstly, more than 40% of rules have static pattern prefix instead of PCRE. This happens because either the initial rules do not have any PCRE field at all (32% of the whole rule-set) or some of them (8% of the whole rule-set) may be limited by the constraints the 'R' modifier of the PCRE introduces, as it was explained in section 3.3. In the case of the PCRE Prefix extraction technique, there are some rules (∼17-18%) which have PCREs whose length prevents from extracting a portion because they have smaller length than 4 bytes or the predefined length. Moreover, approximately 20% of the rules has prefix more than the predefined prefix length. This happens because PCRE prefix is limited by the constraints that the extraction rules of section 3.3 introduce. For example, if we have the following regex "(abcde)+fgh.*", and the prefix size has been selected to 4 bytes, the prefix will be "(abcde)+" according to the extraction rules which were mentioned in that section. Finally, observing that 13-17% of the whole rule-set have PCRE prefix with length exactly the predefined, someone can say that extracting the prefix from the PCRE is clearly more complex task than extracting prefix/part from the content. Intuitively, the Unique Part Rule technique seems to be more efficient among the three extraction techniques.

The extraction techniques were applied to the SNORT rule-set of July 2008 for the 4 different prefix/part lengths and the set of subrules, for the Pre-filtering stage, was

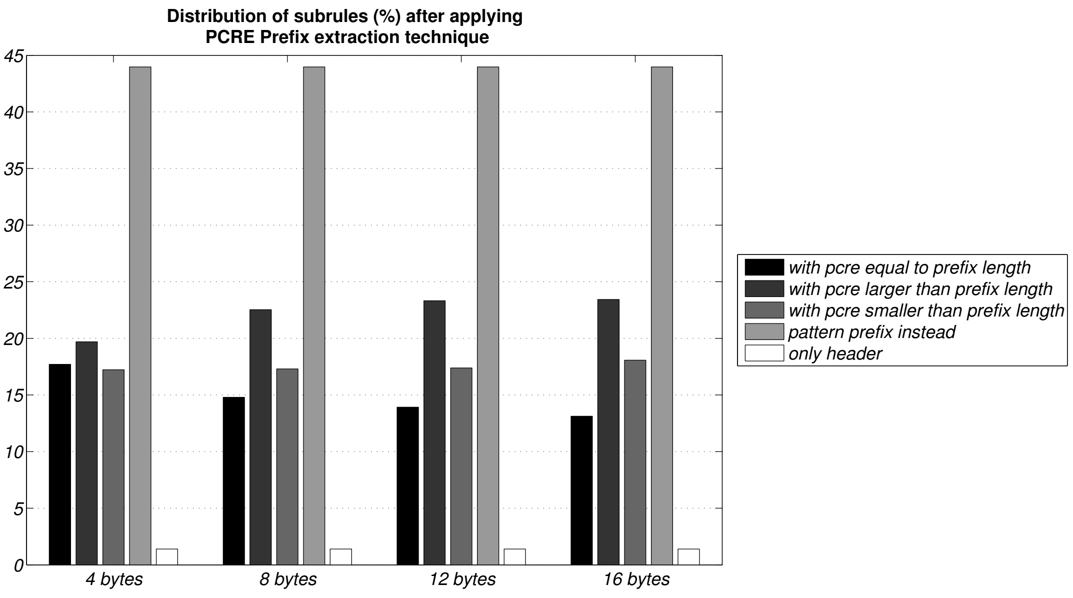


Figure 4.3: This figure depicts the Distribution of Rules (in %) into different subrule categories (due to internal rules' characteristics (Content or PCRE, smaller length)) after applying the PCRE Prefix extraction technique, for 4, 8, 12 and 16 bytes of prefix length. More than 40% of the rules have static pattern prefix instead of PCRE. Most of them (32% of the whole rule-set) do not have PCRE field at all, while about 8% is limited by the constraint the 'R' modifier of the PCRE introduces, as it was explained in section 3.3. Again the number of extracted rules that have PCRE prefix size equal to the prefix length decreases as the size of prefix increases.

created. Some results were presented above about how they are distributed based on the prefix/part size or other parameters (special unique rules, static pattern instead of pcre, etc). To continue with the evaluation of the proposed techniques, some real traffic intrusion scenarios must be input into a real NIDS (SNORT) which is loaded with the set of subrules. Each time, SNORT is loaded with the subrules of one "final" scenario. In addition, two different input traffic intrusion scenarios were used: Defcon10 and Defcon11. Defcon10 consists of approximately 1,5 million packets and Defcon11 of approximately 11 millions. The number of the packets per trace for both Defcon traffic inputs is depicted in Figures 4.4 and 4.5.

A final question that must be answered, before the evaluation, is which metrics are going to be used for it. Some of the metrics which are used for the evaluation part come from the theoretical analysis of Packet Pre-filtering (section 2.2.1). The extraction techniques are going to be evaluated using especially the following metrics:

- AVG_rules: This metric represents the average number of rules which are activated per incoming packet.
- MAX_rules: This metric represents the maximum number of rules which are activated per incoming packet.

However, as it will be seen later, these metrics do not provide always clear conclusions.

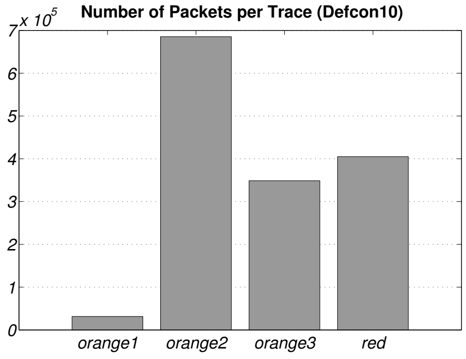


Figure 4.4: Number of packets per trace (Defcon10). The exact total number of packets is 1,470,334.

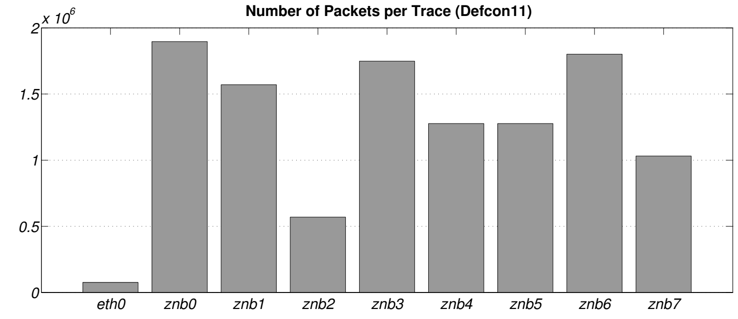


Figure 4.5: Number of packets per trace (Defcon11). The exact total number of packets is 11,246,288.

For that reason, the additional metric of Cumulative Distribution of packets in terms of the activated rules by them will be used as well. It is important to be mentioned here that the Rule Correlation technique is going to be evaluated along with the Unique Part Rule extraction technique. The rules, for which unique rules were not found during extraction, are utilized by the Rule Correlation where they are partitioned into small groups and each group has one unique subrule as a “leader”. These groups are stored in the correlation memory and are utilized in that way during the evaluation of Unique Part Rule extraction technique.

The rest of the evaluation is divided in 5 subsections based on the metric that is going to be used for the evaluation. It must be mentioned that the Smart Rule Reuse is evaluated in the last subsection (4.2.5) using other metrics which are going to be presented in that subsection.

4.2.1 Average Number of Activated Rules

The evaluation starts by presenting results using the AVG_rules metric for the Defcon11 and Defcon10 input traffic traces. The AVG_rules provide us with the average number of activated rules per packet. It shows how many rules are usually activated per packet when a NIDS with Pre-filtering is used. First of all and in all the figures, it is reasonable that the average number of activated rules is the least when the original rule-set is used. This is because the whole rule-set is used unchanged and only few rules are activated per packet, as it is expected. This case is actually the case of a normal NIDS without Pre-filtering.

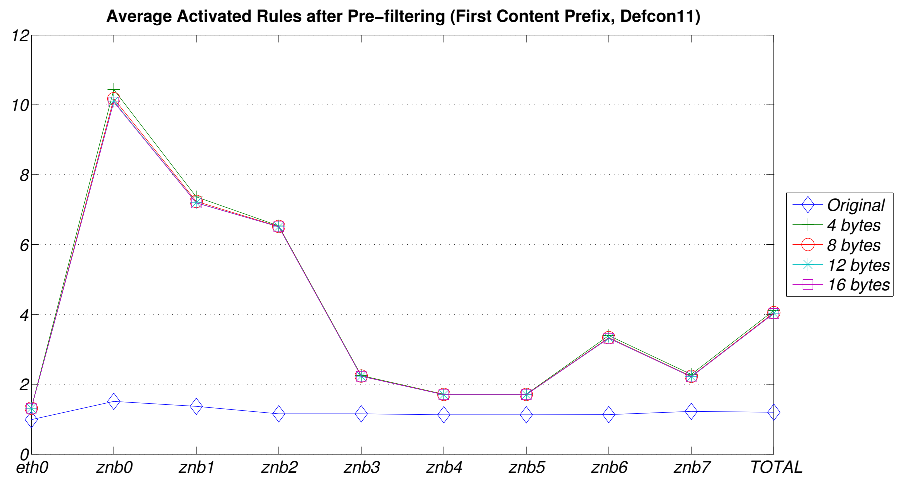


Figure 4.6: Average number of activated rules per packet (Defcon11) after Pre-filtering, as a function of traces. The Pre-filtering stage (actually SNORT) was loaded with rules which were extracted using the First Content Prefix extraction technique. The Average number of activated rules using SNORT with the original rule-set is presented as well.

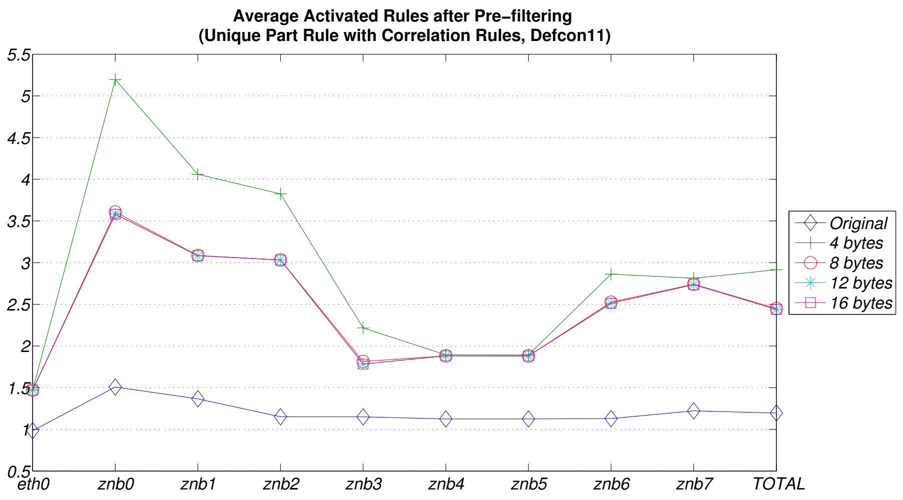


Figure 4.7: Average number of activated rules per packet (Defcon11) after Pre-filtering, as a function of traces. The Pre-filtering stage (actually SNORT) was loaded with rules which were extracted using the Unique Part Rule extraction technique. The Average number of activated rules using SNORT with the original rule-set is presented as well.

Figures 4.6-4.11 show that the average number of activated rules per packet is very small and especially a few tens in the worst case where PCRE Prefix extraction technique was used to create the set of subrules. The Average number of activated rules is even smaller in Figures 4.8, 4.9 and 4.11 where the results for Defcon10 traces are depicted. From the Figures 4.6-4.9, it is clearly shown that the Unique Part Rule extraction technique is better than the First Content Prefix. When the First Content Prefix technique is used (Figure 4.6), it seems that does not really matter what size of prefix is going to be used since for all the prefix sizes the average number of activated rules is the same. On the other hand, when using the Unique Part Rule extraction technique (Figure 4.7) the length of the selected part of rule seems to make sense only when it changes between 4 and 8. These observations are supported also by the respective figures for Defcon10 (Figures 4.8 and 4.9). An interesting observation is that the First Content Prefix extraction technique gives slightly better results than Unique Part Rule technique for Defcon10 traces. But the difference is about 0.2-0.3 rules per packet which is negligible.

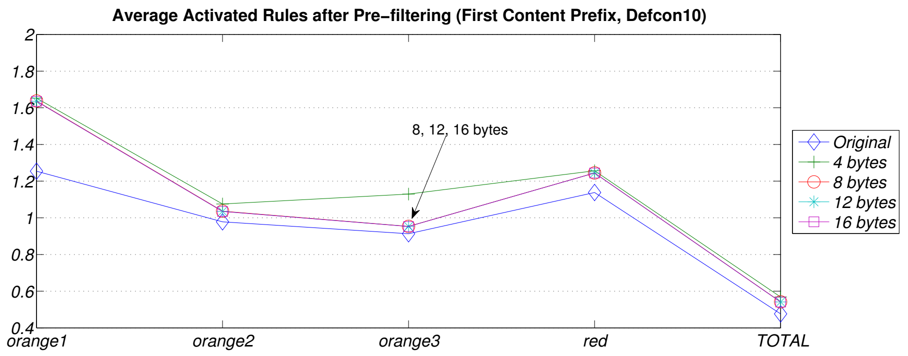


Figure 4.8: Average number of activated rules per packet (Defcon10) after Pre-filtering, as a function of traces. The Pre-filtering stage (actually SNORT) was loaded with rules which were extracted using the First Content Prefix extraction technique. The Average number of activated rules using SNORT with the original rule-set is presented as well.

Finally, Figures 4.10 and 4.11 illustrate that in case of the PCRE Prefix extraction technique, the average number of activated rules is significantly larger than using the previous techniques. One possible explanation for this significant difference is that the First Content Prefix and the Unique Part Rule share the fact of exploiting the content field (content-based) which describes less threats, instead of the PCRE. If the prefix of a PCRE is taken, the number of matched patterns may be increased exponentially than taking just the prefix of a static pattern. For Defcon11 traces, the average number of activated rules is almost an order of magnitude higher than using the previous extraction techniques. Furthermore, it can be easily observed that the prefix size plays more important role here. The difference in the average number of activated rules per packet is significant when the prefix size changes from 8 bytes to 12 bytes (Defcon11). Unfortunately, it is difficult to make clear conclusions for the behavior of the Pre-filtering stage when using the PCRE Prefix extraction technique because almost 50% of extracted rules do not have PCRE field but the prefix of the first content of the rule.

In conclusion, the average number of activated rules per packet for both Defcon traces is just a few tens of rules which is the main idea of Packet Pre-filtering. Even the PCRE

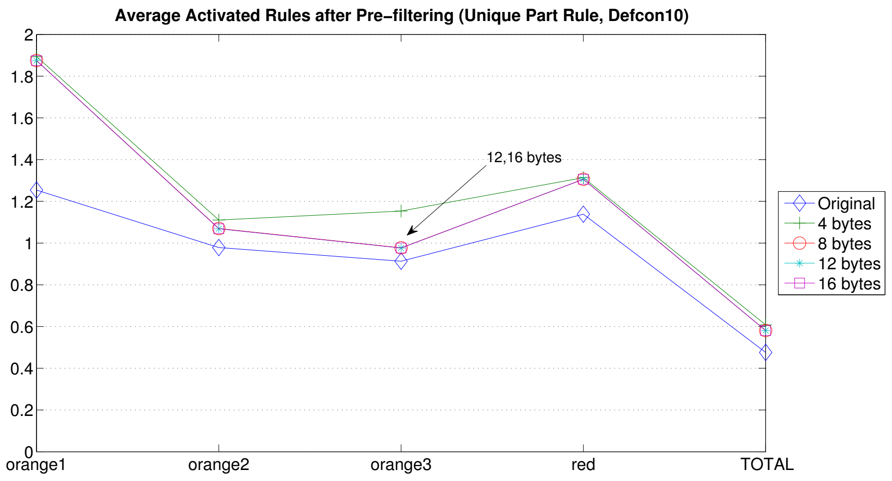


Figure 4.9: Average number of activated rules per packet (Defcon10) after Pre-filtering, as a function of traces. The Pre-filtering stage (actually SNORT) was loaded with rules which were extracted using the Unique Part Rule extraction technique. The Average number of activated rules using SNORT with the original rule-set is presented as well.

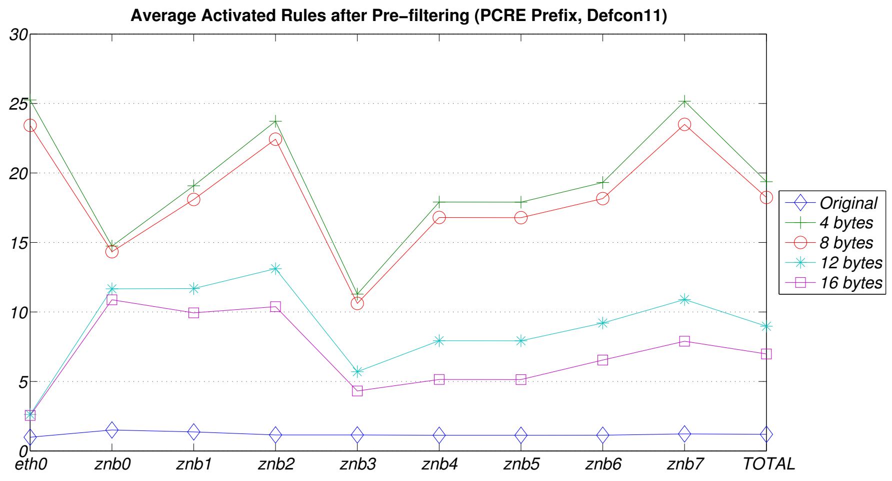


Figure 4.10: Average number of activated rules per packet (Defcon11) after Pre-filtering, as a function of traces. The Pre-filtering stage (actually SNORT) was loaded with rules which were extracted using the PCRE Prefix extraction technique. The Average number of activated rules using SNORT with the original rule-set is presented as well.

Prefix technique supports this idea although the number of activated rules is significantly higher than using the First Content Prefix or the Unique Part Rule. Finally, among the all the extraction techniques, the Unique Part Rule seems to give better results but we are going to make more clear conclusions in the next step of evaluation in section 4.2.2.

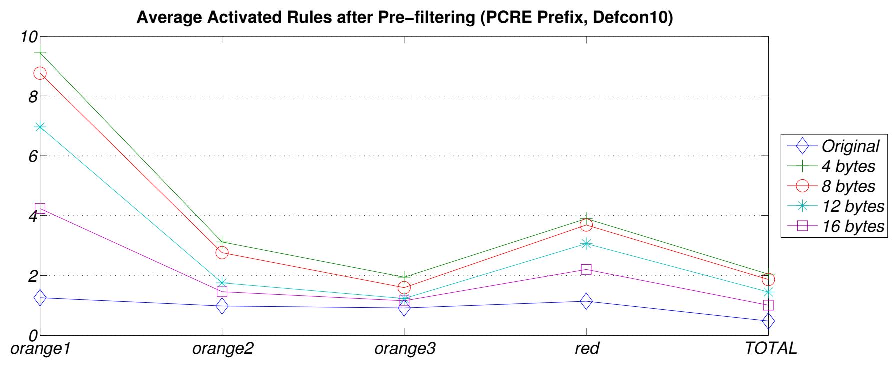


Figure 4.11: Average number of activated rules per packet (Defcon10) after Pre-filtering, as a function of traces. The Pre-filtering stage (actually SNORT) was loaded with rules which were extracted using the PCRE Prefix extraction technique. The Average number of activated rules using SNORT with the original rule-set is presented as well.

4.2.2 Maximum Number of Activated Rules

Section 4.2.1 provided us with the evaluation of the proposed extraction techniques regarding the Average number of activated rules. This metric shows how a NIDS with Pre-filtering works normally. However, it is significant to be aware of the conditions of the worst case scenario. In other words, the Maximum number of activated rules for one single packet in the Pre-filtering stage indicates the maximum number of resources which are needed in the second processing stage (Full Match stage). For that reason, this section is going to evaluate the proposed extraction techniques using the MAX_rules metric.

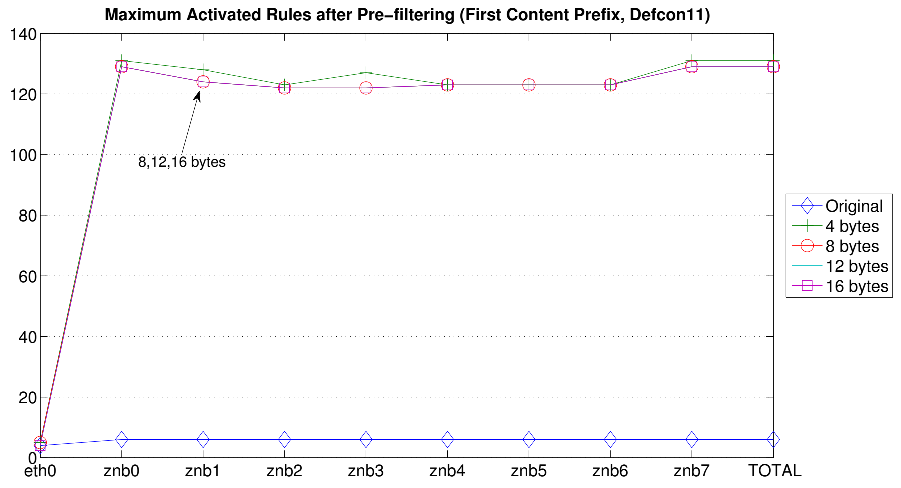


Figure 4.12: Maximum number of activated rules per packet (Defcon11) after Pre-filtering, as a function of traces. The Pre-filtering stage (actually SNORT) was loaded with rules which were extracted using the First Content Prefix extraction technique. The Maximum number of activated rules using SNORT with the original rule-set is presented as well.

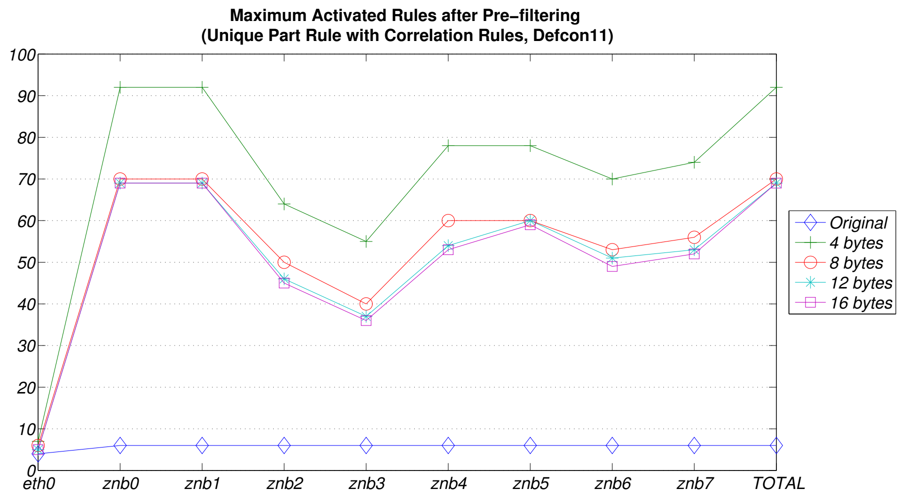


Figure 4.13: Maximum number of activated rules per packet (Defcon11) after Pre-filtering, as a function of traces. The Pre-filtering stage (actually SNORT) was loaded with rules which were extracted using the Unique Part Rule extraction technique. The Maximum number of activated rules using SNORT with the original rule-set is presented as well.

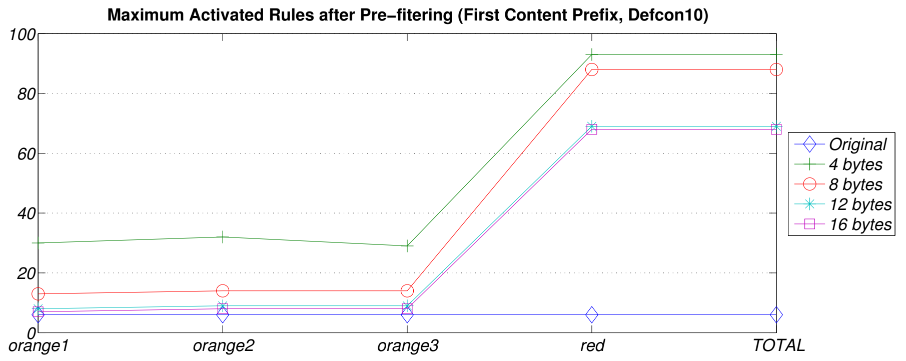


Figure 4.14: Maximum number of activated rules per packet (Defcon10) after Pre-filtering, as a function of traces. The Pre-filtering stage (actually SNORT) was loaded with rules which were extracted using the First Content Prefix extraction technique. The Maximum number of activated rules using SNORT with the original rule-set is presented as well.

Figures 4.12-4.17, where the maximum number of activated rules per packet is presented, verify the observations and conclusions that were made using the metric "average number of activated rules". One difference is that the maximum number of activated rules per packet using the First Content Prefix technique is significantly higher than using the Unique Part Rule technique for both Defcon traces (Figures 4.12-4.15). Also, the selected length of the prefix seems to play an important role only for the PCRE Prefix technique than the other extraction techniques, except the case of the Unique Part Rule technique when the length of the part is selected to 8 bytes, instead of 4. Selecting prefix/part size of 8 bytes for the two content-based techniques, the maximum number of activated rules per packet is approximately 130 for the First Content Prefix

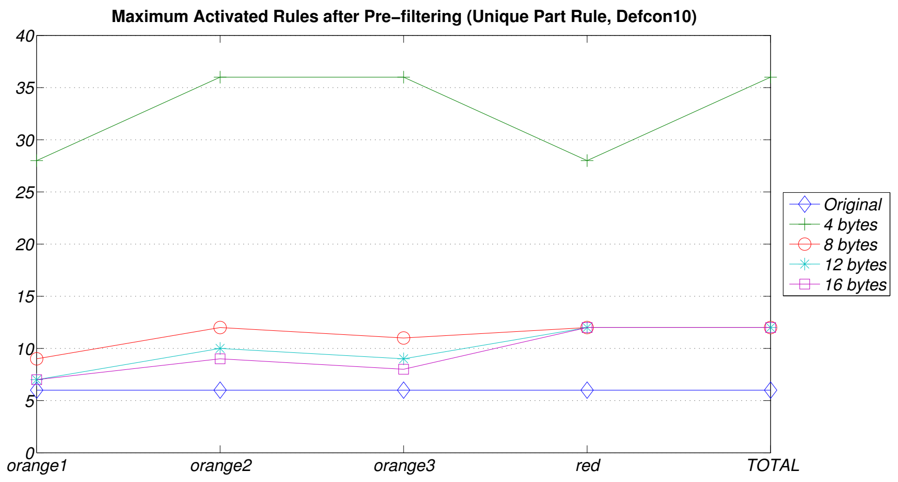


Figure 4.15: Maximum number of activated rules per packet (Defcon10) after Pre-filtering, as a function of traces. The Pre-filtering stage (actually SNORT) was loaded with rules which were extracted using the Unique Part Rule extraction technique. The Maximum number of activated rules using SNORT with the original rule-set is presented as well.

technique and about 70 for the Unique Part Rule (8 bytes part) for the Defcon11 traces. The difference is quite significant. The fact that 50 more Processing Elements (PEs), in a hardware NIDS, would be needed to handle all the possible packets of Defcon11 using the First Content Prefix technique than the Unique Part Rule does make sense. The maximum number of activated rules is smaller for the two content-based techniques for Defcon10 but their difference remains significant and even in that case, the Unique Part Rule extraction technique is more preferable.

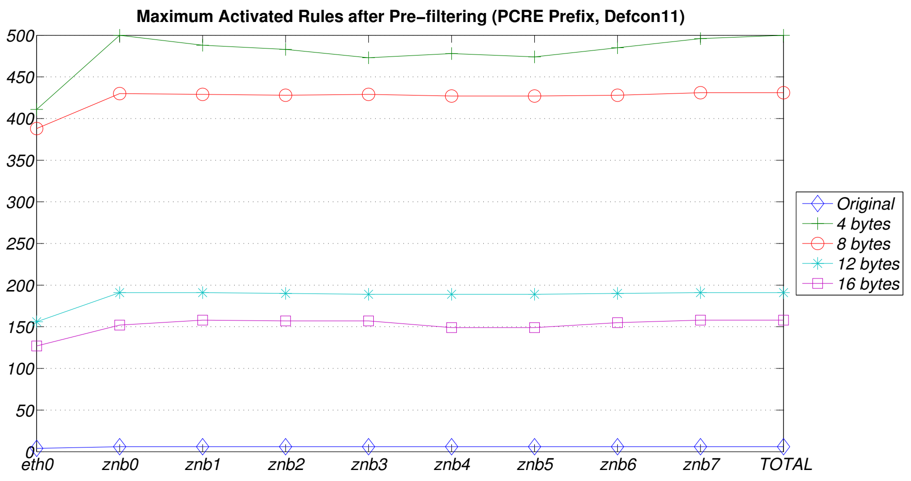


Figure 4.16: Maximum number of activated rules per packet (Defcon11) after Pre-filtering, as a function of traces. The Pre-filtering stage (actually SNORT) was loaded with rules which were extracted using the PCRE Prefix extraction technique. The Maximum number of activated rules using SNORT with the original rule-set is presented as well.

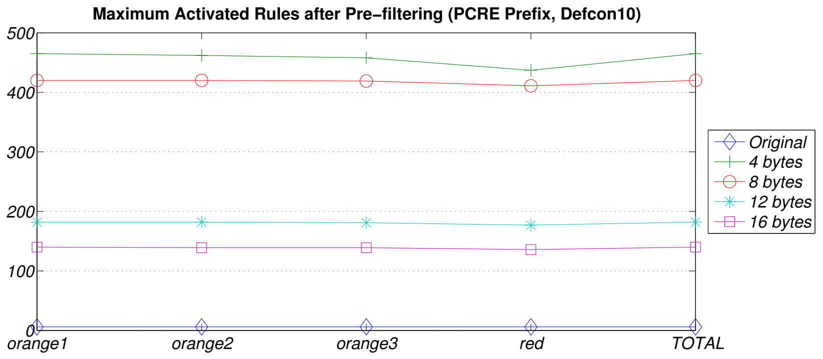


Figure 4.17: Maximum number of activated rules per packet (Defcon10) after Pre-filtering, as a function of traces. The Pre-filtering stage (actually SNORT) was loaded with rules which were extracted using the PCRE Prefix extraction technique. The Maximum number of activated rules using SNORT with the original rule-set is presented as well.

On the other hand, Figures 4.16 and 4.17 show that the maximum number of activated rules per packet increases substantially. The maximum number of rules reaches the 500 when the size of 4 characters has been selected for the PCRE prefix length. However, using larger rule prefixes for the Pre-filtering, 70% fewer maximum rules are activated reaching the number of 120-150 rules which is approximately the maximum number of activated rules using the First Content Prefix technique. Unfortunately, 500 activated rules for a single packet are too many to use PCRE Prefix extraction technique as a Pre-filtering technique because it intends 500 PEs in the Full Match stage. Furthermore, the 150 activated rules/packet are possibly unacceptable considering the substantially higher processing in the Pre-filtering stage, if the prefix length of 16 bytes was selected in combination that PCREs must be matched. Of course, they are not the entire rule-set but the concept of Pre-filtering is to have the maximum processing throughput during partial match and the number of activated rules is the lowest possible. However, it was previously seen that the average number of activated rules, which is the normal behavior, is kept in the magnitude of 10. Consequently, it is possible that the maximum number of rules may be activated only by very few packets processing them using a Best Effort Engine instead of a Guaranteed Throughput Special one, if we used for example the PINE NIDS. The number of packets which are close to the worst case scenario of activating the maximum number of rules will be shown better, in section 4.2.3, which exploits the cumulative distribution of packets.

Before concluding, it is a good point to mention here that all the results for the Maximum and Average (the Average in section 4.2.1) for the Unique Part Rule technique were presented including the correlated rules. About 1700 rules out of 9000 SNORT rules of July 2008 were partitioned into 500 groups and put into the correlation memory of the Pre-filtering stage while the activated unique rules were used in order to access it and send the potentially correlated rules (if memory hit) along with the activated unique rules to the Full Match stage. It would be unfair not to count them (correlated rules) too and include them in our evaluation, although only the “leader” [1] rules have been only partially matched. These rules must be sent also to the second phase to be fully

[1]It is reminded that one specific unique subrule is considered as the “leader” of one partition of correlated rules

matched. However, what if we find a way not to count them and to be "fair" at the same time. It had been mentioned in section 3.5 that the rules that lie in one group of correlation (partition) have the common characteristics of having the same header and a common part of the content. Thus, the idea is: instead of sending more rules to the next phase and requiring as a result more PEs, the common characteristics of these rules can be shared and the remaining unique fields can be encoded into a regular expression that will be matched once in the second phase. Of course, this alternative would be efficient only if there are clues that the number of average and especially the maximum number of activated rules can decrease significantly. The following two figures present the average (Figure 4.18) and the maximum (Figure 4.19) number of activated rules per packet for the Defcon11 traces without taking into account these correlated rules.

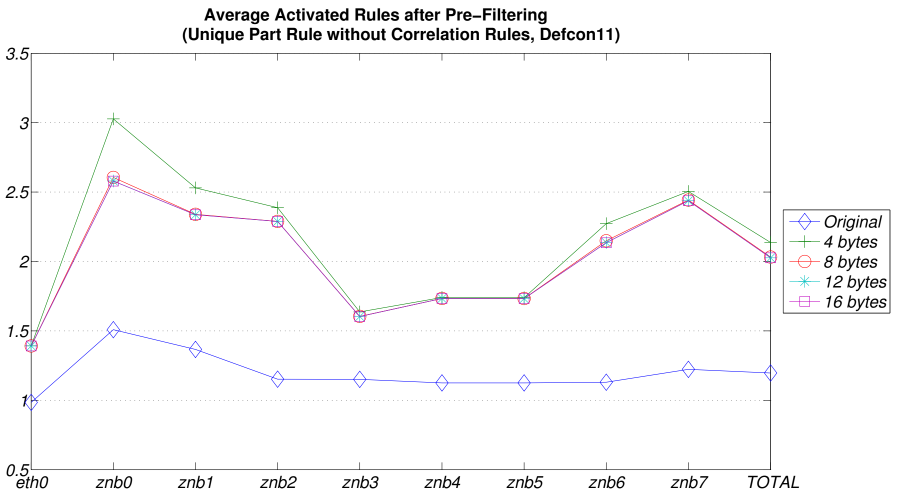


Figure 4.18: Average number of activated rules per packet (Defcon11) after Pre-filtering, as a function of traces. The Pre-filtering stage (actually SNORT) was loaded with subrules which were extracted using the Unique Part Rule extraction technique. The correlated rules are not taken into account in this case.

Comparing Figure 4.18 with Figure 4.7 it is shown that the average number of activated rules per packet has been reduced about 0.5-1 rule. However, comparing Figure 4.19 with Figure 4.13, someone can observe that the maximum number of activated rules has been decreased significantly for all the selected part lengths. The maximum number of rules was 90 rules for part length of 4 bytes and 70 rules for all the other part sizes taking into account the correlated rules, and decreased (substantially) to 50 and 40 respectively by eliminating them. The average number of activated rules has been generally decreased by one, if someone compares Figure 4.18 with 4.7. However, the reduction in the maximum number of rules is very substantial. The average number of correlated rules per group is 3.1 rules and the maximum number of them can reach the 16 rules but the number of those groups is very small. Thus, if the correlated rules can be encoded into one regular expression and all of them are matched as one rule, the Unique Part Rule technique can be proved even more efficient. This idea could be considered as a future work.

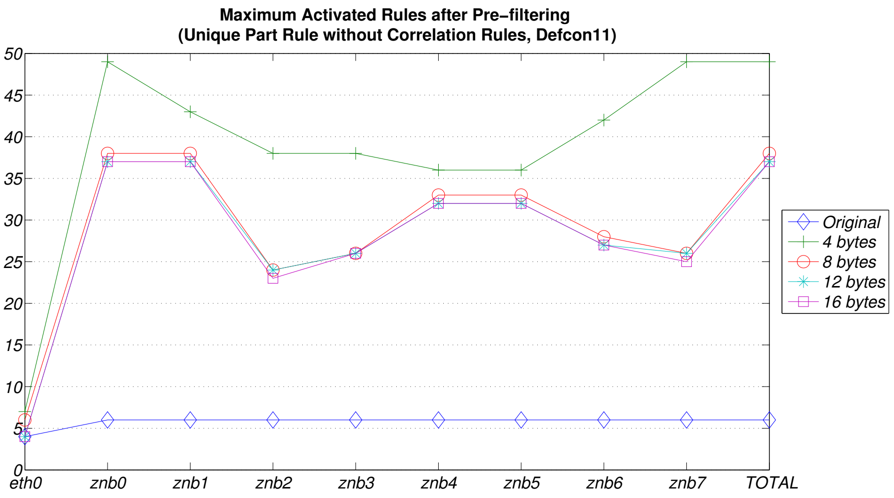


Figure 4.19: Maximum number of activated rules per packet (Defcon11) after Pre-filtering, as a function of traces. The Pre-filtering stage (actually SNORT) was loaded with subrules which were extracted using the Unique Part Rule extraction technique. The correlated rules have been eliminated in this case.

In conclusion, it can be said that it is very clear that the Unique Part Rule extraction technique is significantly better than the other two techniques. In the worst case, approximately 70 PEs will be needed for the second processing stage when the selected part of length is 8 bytes instead of the First Content Prefix which requires 120 PEs with selected prefix length of 4 bytes, or the PCRE Prefix which requires 500 PEs with length of 4 bytes and 420 PEs with 8 bytes (and more processing due to the nature of PCRE). To conclude, it is already obvious that the Unique Part Rule technique is better than the other extraction techniques taking into consideration both Maximum and Average number of activated rules. Additionally, the Unique Part Rule can be substantially improved if the correlated rules are utilized in a different way. Instead of storing them in the correlation memory and fully match all of the activated correlated partitions, we can match their common characteristics once and encode their unique characteristics into one regular expression and fully match only this.

4.2.3 Cumulative Distribution of Packets

This section evaluates the proposed extraction techniques using the Cumulative distribution. This metric shows how all the packets are distributed based on the number of activated rules during inspection of the whole traffic. The results which are summarized by the Figures 4.20-4.27 are the Cumulative distributions of the packets for all the selected prefix/part lengths and for all the extraction techniques and the original rule-set. The results are similar for both input traffics cases (Defcon11 and 10). Actually, the Cumulative distribution shows us the claim that was done in the previous section: “The average number of activated rules per packets is acceptably low and meet the requirements of Pre-filtering; however, the maximum number of rules may be very big in some cases, but only few packets require this significant number of PEs in the second stage”.

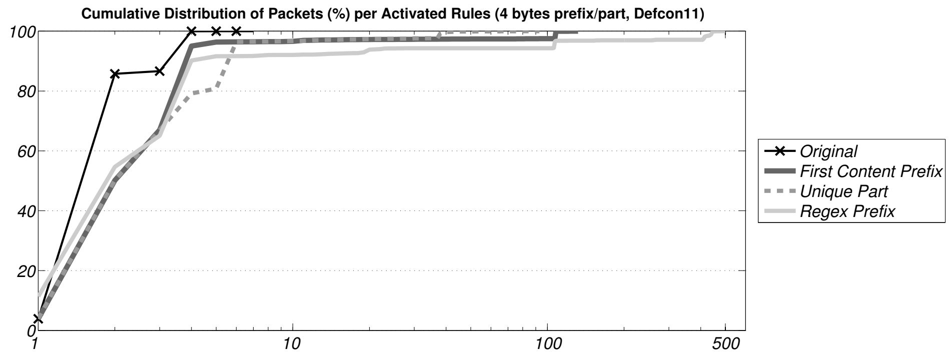


Figure 4.20: The figure depicts how the packets are distributed (%) based on the number of the rules they activate. The Prefix/part size is 4 bytes and the distribution is cumulative.

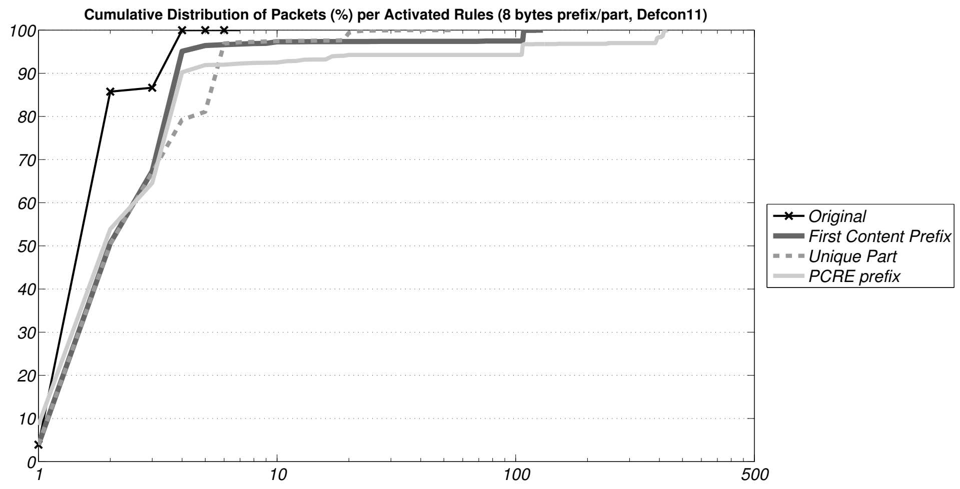


Figure 4.21: The figure depicts how the packets are distributed (%) based on the number of the rules they activate. The Prefix/part size is 8 bytes and the distribution is cumulative.

For the Defcon11 traces (see Figures 4.20-4.23), about the 97% of the packets activates at most 8-13 rules for both content-based techniques for all the prefix/part lengths. From these, most of the packets (96% out of 97%) activate 3-5 rules. After that point, the rest 2% of the packets will activate 20-25 rules for the Unique Part Rule technique and 70-80 rules for the First Content Prefix technique. The last 1% of packets is the amount of packets which activates the maximum number of rules (70 for the Unique Part Rule and 120 for the First Content Prefix). Taking into account that Defcon11 traces consist of approximately 11,000,000 packets, 1% is approximately 110,000 packets.

However, the results are significantly worse for the PCRE Prefix technique. The 90% of packets activates about 2-5 rules. From the rest, 1-4% activates up to 23 rules and after that approximately every 1% activates up to 50 more rules. That means that the 94% of rules activates up to 25 rules and an extra 1% activates about 80. The rest 5% (560000 packets) activates from 100 to the maximum number of rules which depends on

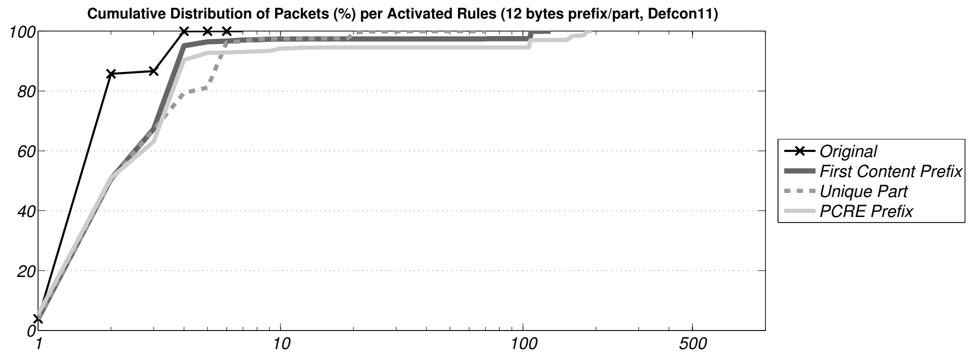


Figure 4.22: The figure depicts how the packets are distributed (%) based on the number of the rules they activate. The Prefix/part size is 12 bytes and the distribution is cumulative.

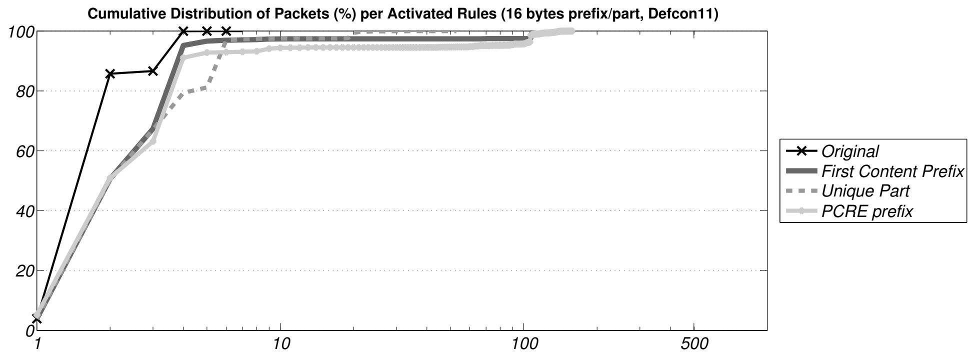


Figure 4.23: The figure depicts how the packets are distributed (%) based on the number of the rules they activate. The Prefix/part size is 16 bytes and the distribution is cumulative.

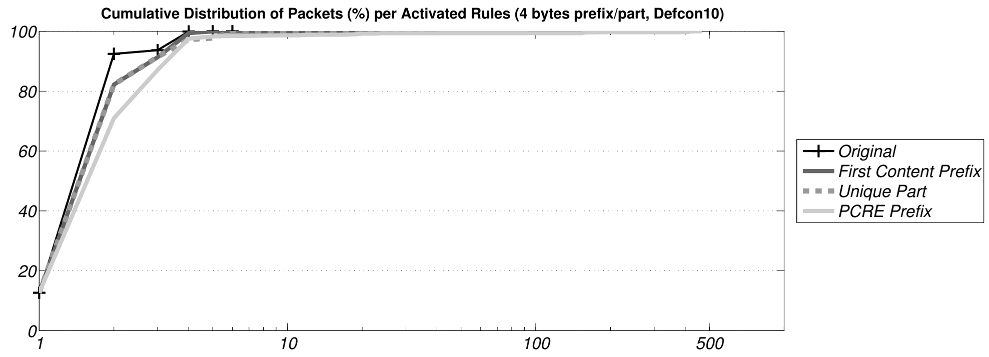


Figure 4.24: The figure depicts how the packets are distributed (%) based on the number of the rules they activate. The Prefix/part size is 4 bytes and the distribution is cumulative.

the prefix length (500 for 4 bytes, 120-150 for 16 bytes) rules. The results are similar and slightly better for Defcon10 input trace. From all the figures, it is also shown that the chart for the First Content Prefix technique is generally saturated faster than the Unique Part Rule. However, the maximum number of rules for the Unique Part Rule

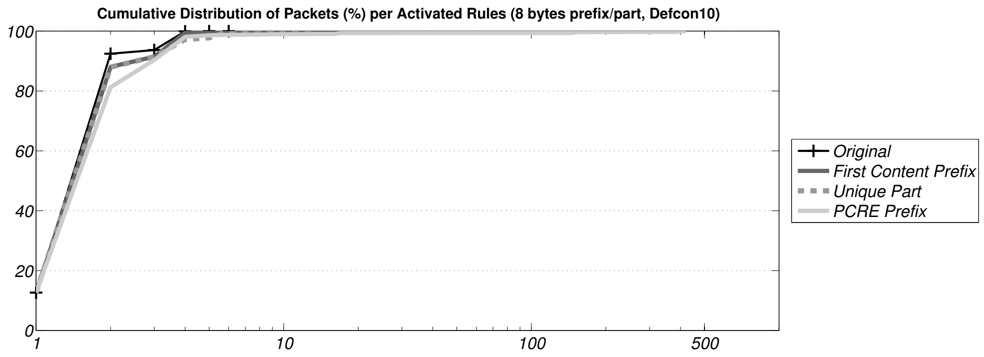


Figure 4.25: The figure depicts how the packets are distributed (%) based on the number of the rules they activate. The Prefix/part size is 8 bytes and the distribution is cumulative.

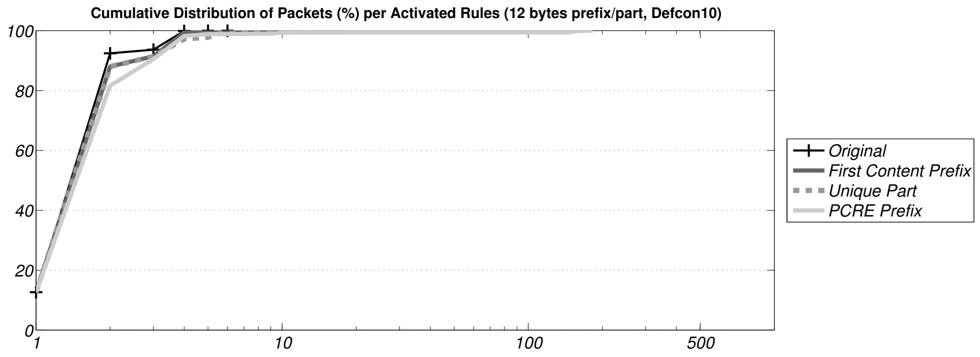


Figure 4.26: The figure depicts how the packets are distributed (%) based on the number of the rules they activate. The Prefix/part size is 12 bytes and the distribution is cumulative.

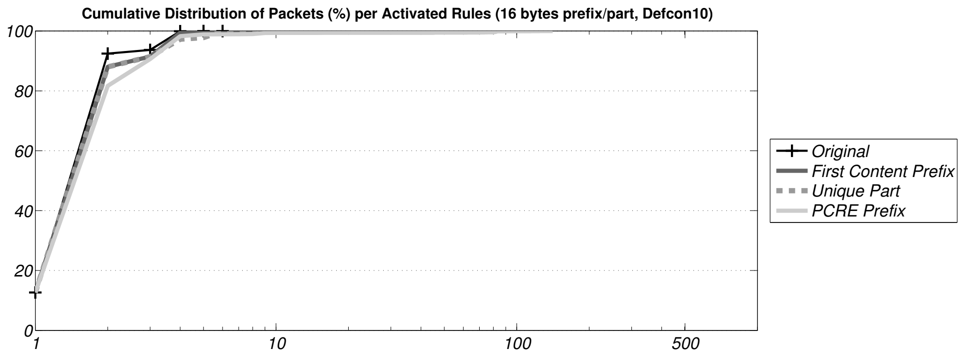


Figure 4.27: The figure depicts how the packets are distributed (%) based on the number of the rules they activate. The Prefix/part size is 16 bytes and the distribution is cumulative.

technique is significantly lower than the First Content Prefix technique and the points of saturation for the two content-based techniques are very close making this variation negligible.

In summary, the cumulative distribution explained clearly why the average number of activated rules per packet is significantly lower compared to the maximum number of activated rules. Generally, most of packets (95%-97% and more) activate very few rules (up to one to two dozens) and the rest of rules activate more and more until reaching the maximum. However, in order to prevent possible bottlenecks leading to performance degradation or denial of service, special precautions should be taken even for this small percentage of remaining packets (3%-5%). This is another reason why Unique Part Rule technique seems to be more promising than the other techniques. It is shown that PCREs should be avoided for the Pre-filtering due to the large number of activated rules per packet and thus the significant amount of required resources or higher latency in the second stage. In addition, in order to achieve a potentially acceptable number of activated rules using PCREs, the processing throughput of the Pre-filtering stage should be significantly reduced, since the biggest prefix length must be selected.

4.2.4 Total Average/Maximum Number of Activated Rules

This section summarizes the already presented results about the Average and Maximum number of activated rules per packet for the entire traffic of Defcon11 and Defcon10.

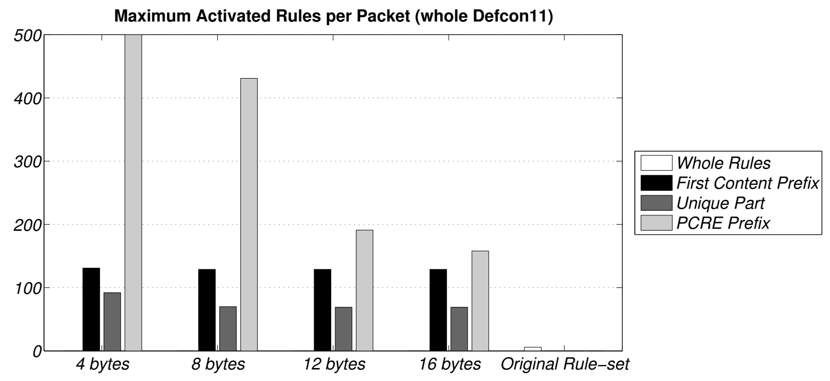


Figure 4.28: Maximum number of activated rules per packet for the entire Defcon11. The results for all the extraction techniques and all the prefix/part sizes are summarized.

Figures 4.28-4.31 present the evaluation of the proposed extraction techniques using the AVG_rules and MAX_rules metrics for all the selected prefix/part lengths and for the entire two versions of Defcon traffic. The Unique Part Rule technique seems to be the better choice among all the extraction techniques. It has lower average number of activated rules per packet and requires significantly smaller number of resources in the second stage if it is a hardware NIDS, or achieves smaller latency if it is a software NIDS. Furthermore, the maximum number of activated rules can be reduced substantially, if the correlated rules are utilized in a different way than the correlation memory, i.e. by encoding their unique characteristics in one regex and match this only once, so that only the unique activated rules are needed to be fully matched. In addition, from these figures, it can be seen more clearly why the selected prefix/part length plays more significant role in the case of PCRE Prefix technique than Unique Part Rule technique or First Content Prefix technique. Of course, the best part length for the Unique Part Rule technique is 8 bytes. For the First Content Prefix the best prefix length is 4 bytes because there is no

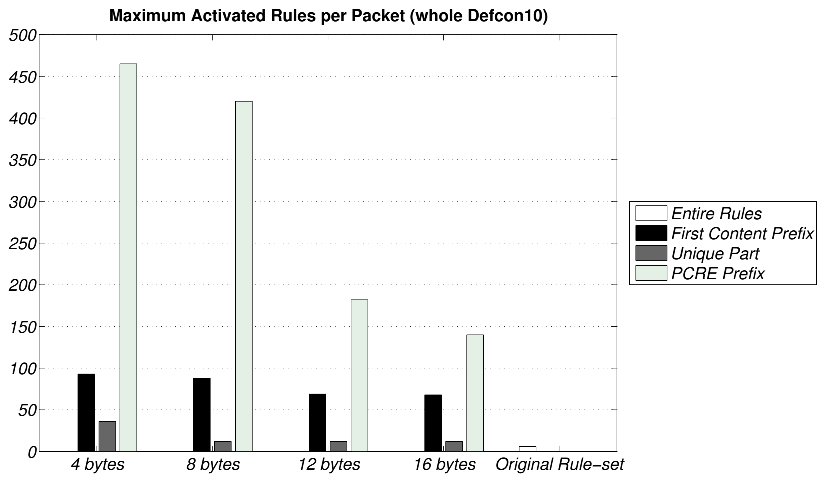


Figure 4.29: Maximum number of activated rules per packet for the entire Defcon10. The results for all the extraction techniques and all the prefix/part sizes are summarized.

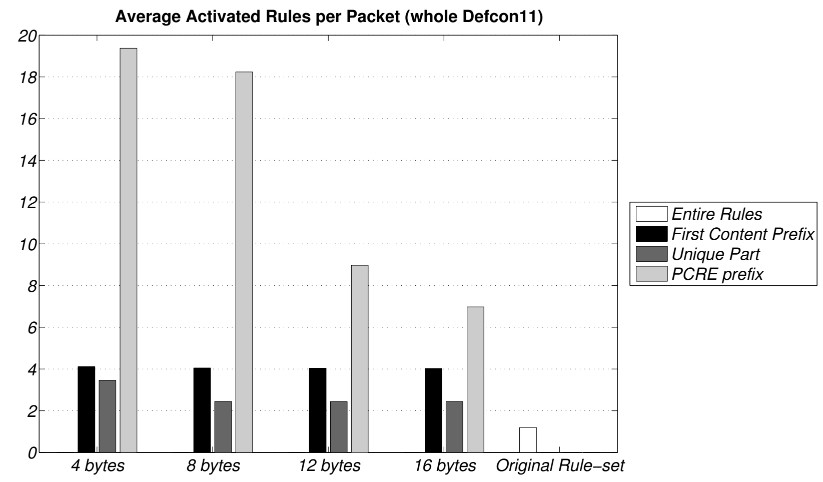


Figure 4.30: Average number of activated rules per packet for the entire Defcon11. The results for all the extraction techniques and all the prefix/part sizes are summarized.

significant difference regarding the number of activated rules using larger lengths but it is important for the processing throughput. In the case of the PCRE Prefix, 12 or even 16 bytes would be the best choice, despite the potentially high processing throughput in the Pre-filtering stage.

Another metric that can be used in order to compare the three extraction techniques is the time that is required by each technique to preprocess the entire rule-set and create the set of subrules for the Pre-filtering stage. Table 4.5 summarizes the results for the preprocessing times of all the extraction techniques using version 2.8 of SNORT rules. These measurements were taken using the “time” command in Linux OS, created

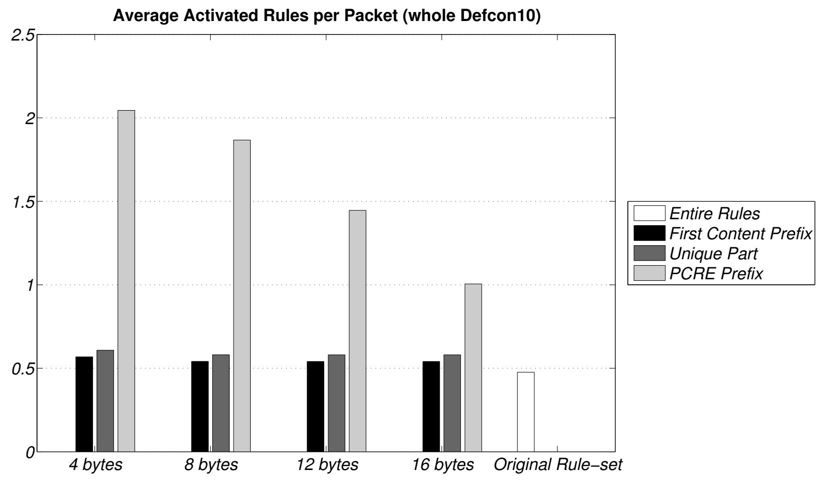


Figure 4.31: Average number of activated rules per packet for the entire Defcon10. The results for all the extraction techniques and all the prefix/part sizes are summarized.

by David MacKenzie, using a dual core processor (1.66GHz) and 2 GBs of memory. The First Content Prefix needs the least required time to preprocess the rules due to its simplicity, as it was expected. The preprocessing time for the Unique Part Rule includes also the time for correlating the rules and creating the partitions (∼0.3 seconds). The preprocessing time for the Unique Part Rule and can be considered acceptable, taking into account the fact that it achieves the lowest average and maximum number of activated rules/packet. Finally, the PCRE Prefix needs a lot of processing in order to preprocess the rules and this is another reason why the PCRE Prefix should be avoided to be used for the Pre-filtering.

Table 4.5: Required time for preprocessing the original set of rules of SNORT version 2.8 of July 2008, by every extraction technique in order to produce the set of subrules for the Pre-filtering stage.

Extraction Technique	Preprocessing time (in seconds)
First Content Prefix	2.69
PCRE Prefix	4537.64
Unique Part Rule	65.87

In general, the Unique Part Rule extraction technique seems to be the most promising among the three extraction techniques. It has the lowest maximum number of activated rules per packet and the lowest average with the correlation memory in use. In addition, the cumulative distribution figures support the claim that very few packets activate the maximum number of rules so that very rarely more than few tens of rules must be fully matched in the second stage of the NIDS. The qualitative correlation technique was evaluated by using the correlation memory in the evaluation of Unique Part Rule extraction technique. On the other hand, it should not be fair to attempt to evaluate the quantitative correlation technique due to the lack of a reliable amount of results. The next thing is to evaluate the Smart Rule Reuse technique.

4.2.5 Smart Rule Reuse Evaluation

The Smart Rule Reuse technique has a totally different idea than the previous techniques and is used in the second stage of a *hardware*, only, NIDS that uses Pre-filtering. It is orthogonal to the extraction techniques since it can be used with anyone of them. The second stage of a hardware NIDS that uses Packet Pre-filtering consists of the Processing Elements (PEs), each one of which fully matches a single activated rule against the packet. In order to do it, the firmware of the activated rule must be downloaded onto the respective PE. The idea of Smart Rule Reuse exploits the *temporal locality* of the activated rules between two or more consecutive packets based on the observation that a potential attack may span into multiple packets. Thus, the Smart Rule Reuse block tracks the PEs to be aware of which rules have been assigned to which PEs and hence, when a new group of rules is activated, it searches them for common rules and if so, it requests to download only the newly activated ones. The newly activated rules are assigned to the "empty" PEs using a FIFO as it was explained in section 3.6. This FIFO of the Smart Rule Reuse block is evaluated in order to evaluate the technique itself due to the lack of a hardware NIDS.

The evaluation metrics, which were used for the evaluation of the Smart Rule Reuse, are well-known from the computer architecture theory and are the number of Hits/Misses and the Hit/Miss Rate (%). In section 3.6, it was mentioned that when there is a common activated rule, the respective PE pointer is removed from its current position in the FIFO and is pushed into the Most Recently Used (MRU) position (FIFO's beginning), after all the previous pointers have been shifted one position, giving a "hit". On the other hand, when a new activated rule is needed to be assigned to a PE, the Least Recently Used (LRU) PE is pulled out of the FIFO and pushed into it, in the MRU, giving a "miss". Another important issue apart from the evaluation metrics is to find the best size for the Smart Rule Reuse FIFO. It was mentioned in section 3.6 that the depth of the FIFO is actually the number of PEs of the second stage of the NIDS. Four different numbers of FIFO depths were selected: 8, 16, 32 and 64. Of course, 8 and 16 are very small numbers of PEs considering also the fact that the lowest maximum number of activated rules per packet, which has been achieved, is 70 using the Unique Part Rule extraction technique with selected part length of 8 bytes. However, this section attempts to evaluate Smart Rule Reuse FIFO. It is assumed that the second stage of processing consists of a specific number of PEs instead of having two processing engines like in PINE. If a packet activates more rules than the number of PEs, the procedure of fully matching it is broken in more cycles.

The experiment was performed using a script that simulates the FIFO of the Smart Rule Reuse block which is used in the second stage of a NIDS with Pre-filtering. The second stage is like the one described in the previous paragraph and the FIFO contains the currently activated rules' SIDs instead of the PEs IDs. This script is input with an empty FIFO and the output log files with all the activated rules per packet (of both Defcon 11 and 10) by SNORT loaded with the subrules that were produced using every possible extraction technique and for all the possible prefix/part lengths. The script checks whether every single activated rule of these log files (actually the potential threat) exists in the FIFO. If yes, then it is a "hit" and the specific rule ID is moved from its location to the first and most recently used position and all the other rules before it are shifted one position. Otherwise, it is a "miss" and all the rules are shifted one position to the least recently used position and the least recently used rule is pulled out of the FIFO and the new rule is inserted in the MRU.

The experiment was performed for all the extraction techniques and the selected prefix/part lengths, for the four different FIFO depths (number of PEs) and for both Defcon traffic inputs. The number of Hits and Misses was counted and is presented in Tables 4.6-4.9, while the Hit/Miss rate (%) is presented in Figures 4.32-4.39. On the x-axis there are 16 different cases which are actually the proposed extraction techniques for all the selected prefix/part lengths. FCP is the First Content Prefix; UP is the Unique Part Rule and there are two kinds of it: UPwC (UP taking into account the correlation rules) and UPnC (UP eliminating the correlation rules); PP is the PCRE Prefix technique.

Table 4.6: This table shows the number of Hits, Misses and Total Accesses to the FIFO using the Defcon11 input traffic trace for 8 and 16 PEs (FIFO depth=8 and 16). FCP is the First Content Prefix technique. There are two cases for the Unique Part Rule technique. One is taking into account the correlation rules (UPwC) and the other without taking into account them (UPnC). PP is PCRE Prefix technique. l represents the predefined length (size) of the prefix/part.

Extraction Technique	Prefix/Part length	FIFO depth = 8			FIFO depth = 16		
		Hits	Misses	Total Accesses	Hits	Misses	Total Accesses
FCP	4	20,148,953	33,415,770	53,564,723	20,616,723	32,948,000	53,564,723
UPwC		29,097,535	7,410,381	36,507,916	36,156,047	351,869	36,507,916
UPnC		22,718,686	3,660,347	26,379,033	24,685,843	1,693,190	26,379,033
PP		17,585,248	217,558,903	235,144,151	19,276,821	215,867,330	235,144,151
FCP	8	21,113,682	32,332,628	53,446,310	21,611,750	31,834,560	53,446,310
UPwC		23,307,588	6,032,535	29,340,123	24,357,723	4,982,400	29,340,123
UPnC		23,119,818	1,033,718	24,153,536	23,860,945	292,591	24,153,536
PP		17,822,219	198,795,942	216,618,161	20,266,404	196,351,757	216,618,161
FCP	12	20,589,373	31,916,260	52,505,633	21,049,807	31,455,826	52,505,633
UPwC		23,327,818	5,903,469	29,231,287	24,338,592	4,892,695	29,231,287
UPnC		23,119,784	925,408	24,045,192	23,806,643	238,549	24,045,192
PP		17,922,524	104,362,129	122,284,653	21,052,167	101,232,486	122,284,653
FCP	16	20,811,594	31,846,183	52,657,777	21,224,443	31,433,334	52,657,777
UPwC		23,369,337	5,898,365	29,267,702	24,379,699	4,888,003	29,267,702
UP		23,161,446	920,161	24,081,607	23,849,010	232,597	24,081,607
PP		18,077,136	78,107,644	96,184,780	19,742,833	76,441,947	96,184,780

Figures 4.32-4.35 present the results of using a FIFO with depths 8, 16, 32 and 64 and as an input traffic the Defcon11 traces. Using the First Content Prefix technique the FIFO has a Hit Rate of about 38% for a FIFO depth of 8 or 16. A small improvement of 1-2% has been achieved using a FIFO depth of 32 and approximately 42% is achieved with depth equal to 64. Changing the prefix sizes for FCP does not affect the Hit Rate which is quite reasonable taking also into account the fact that changing these sizes, the maximum and the average number of activated rules have not been improved

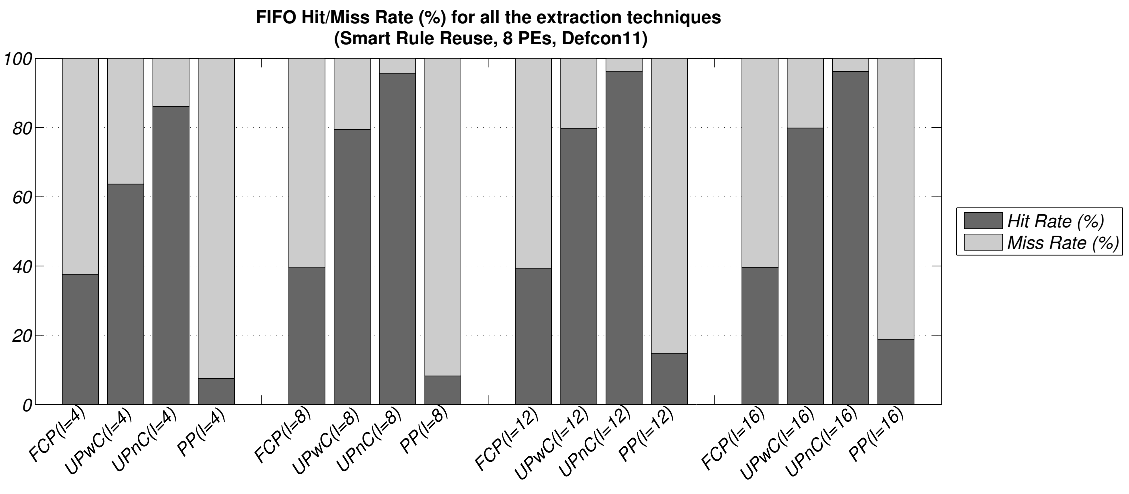


Figure 4.32: Smart Rule Reuse technique evaluation (Defcon11). FIFO depth is 8. FCP is the First Content Prefix technique. UP is the Unique Part Rule technique and PP is PCRE Prefix technique. l represents the predefined length (size) of the prefix/part. There are two cases for the Unique Part Rule technique: 1) UPwC is taking into account the correlation rules and 2) UPnC is not taking into account them.

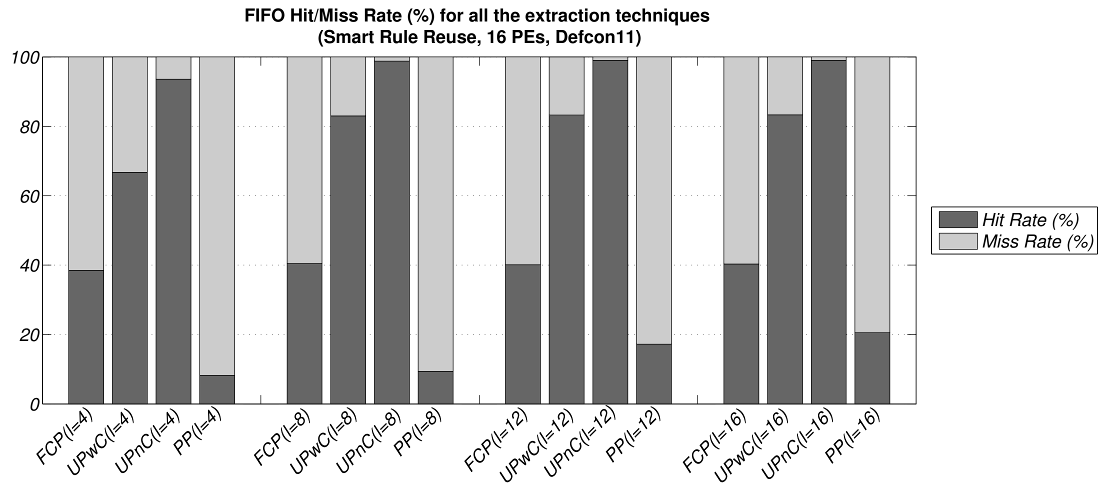


Figure 4.33: Smart Rule Reuse technique evaluation (Defcon11). FIFO depth is 16. FCP is the First Content Prefix technique. UP is the Unique Part Rule technique and PP is PCRE Prefix technique. l represents the predefined length (size) of the prefix/part. There are two cases for the Unique Part Rule technique: 1) UPwC is taking into account the correlation rules and 2) UPnC is not taking into account them.

significantly, as it was presented in the previous sections (sections 4.2.1 and 4.2.2). Using the PCRE Prefix extraction technique, the performance of the Smart Rule Reuse does not seem to be affected significantly when the FIFO depth changes from 8 to 16 (Figures

4.32 and 4.33) or from 32 to 64 (Figures 4.34 and 4.35), apart from the case when it changes from 16 to 32. However, increasing the prefix size for the PP technique, keeping constant the FIFO depth, the FIFO Hit Rate is improved substantially (from 10% when the prefix length is 8 bytes to 20% when the prefix length is 16 bytes) as it was exactly happened with the average and maximum number of activated rules (for example, see Figures 4.10 and 4.16).

Table 4.7: This table shows the number of Hits, Misses and Total Accesses to the FIFO using the Defcon11 input traffic trace for 32 and 64 PEs (FIFO depth=32 and 64). FCP is the First Content Prefix technique. There are two cases for the Unique Part Rule technique. One is taking into account the correlation rules (UPwC) and the other without taking into account them (UPnC). PP is PCRE Prefix technique. l represents the predefined length (size) of the prefix/part.

Extraction Technique	Prefix/Part length	FIFO depth = 32			FIFO depth = 64		
		Hits	Misses	Total Accesses	Hits	Misses	Total Accesses
FCP	4	21,236,253	32,328,470	53,564,723	22,206,287	31,358,436	53,564,723
UPwC		29,097,535	7,410,381	36,507,916	36,156,047	351,869	36,507,916
UPnC		25,645,042	733,991	26,379,033	26,571,618	13,889	26,585,507
PP		31,335,764	203,808,387	235,144,151	32,084,012	203,060,139	235,144,151
FCP	8	21,955,116	31,491,194	53,446,310	22,295,165	31,151,145	53,446,310
UPwC		28,979,640	360,483	29,340,123	29,261,319	78,804	29,340,123
UPnC		240,73,647	79,889	24,153,536	24,139,997	13,539	24,153,536
PP		25,380,008	191,238,153	216,618,161	25,629,597	190,988,564	216,618,161
FCP	12	21,218,793	31,286,840	52,505,633	21,421,446	31,084,187	52,505,633
UPwC		28,894,761	336,526	29,231,287	29,159,816	71,471	29,231,287
UPnC		23,981,931	63,261	24,045,192	24,035,852	9,340	24,045,192
PP		25,909,804	96,374,849	122,284,653	26,118,734	96,165,919	122,284,653
FCP	16	21,343,499	31,314,278	52,657,777	21,588,270	31,069,507	52,657,777
UPwC		28,933,314	334,388	29,267,702	29,196,481	71,221	29,267,702
UPnC		24,019,180	62,427	24,081,607	24,072,287	9,320	24,081,607
PP		26,289,331	69,895,449	96,184,780	27,798,906	68,385,874	96,184,780

Generally, the FIFO Miss Rate is very high for the PP extraction technique even for the FIFO with depth of 64. A possible explanation on why this is happening is that the average number of activated rules per packet is one order of magnitude larger than FCP or UP (UPwC and UPnC). Considering that the number of activated rules per packet is 18-19 in general for PP and that almost 10% of the packets activates more than 23 rules (Figure 4.23 of section 4.2.3), it is obvious that the performance of the Smart Rule Reuse and the second stage with 8 or 16 PEs will degrade substantially. For 32 or 64 PEs the performance may be slighty better but the FIFO has still a lot of misses also because the number of common rules between many consecutive packets is not expected to be significantly high. However, in the case of FCP or PP with prefix size of 16 bytes, where the average number of activated rules is 6-8, the number of Hits and Misses can be more balanced. Despite this, the FIFO will still have a lot of misses because even in the case of PP with prefix length of 16 bytes, more than 5% of packets (almost half million of packets) activates at least 80 rules and reaches the maximum of 160.

On the contrary to FCP and PP, using the Unique Part Rule technique the performance of Smart Rule Reuse in the second stage is substantially improved. In addition, the improvement becomes even more significant increasing the number of PEs when the selected technique is the UPwC. Regarding the two different cases of Unique Part Rule (with or without counting the correlation rules), there is significant difference between

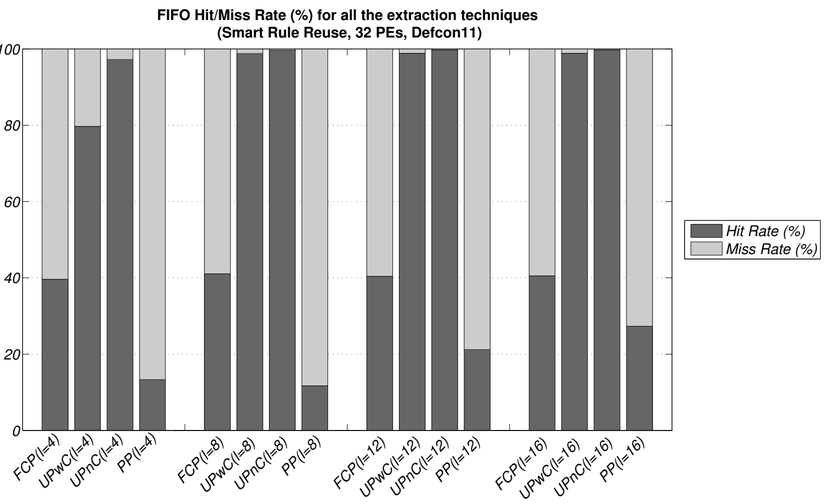


Figure 4.34: Smart Rule Reuse technique evaluation (Defcon11). FIFO depth is 32. FCP is the First Content Prefix technique. UP is the Unique Part Rule technique and PP is PCRE Prefix technique. l represents the predefined length (size) of the prefix/part. There are two cases for the Unique Part Rule technique: 1) UPwC is taking into account the correlation rules and 2) UPnC is not taking into account them.

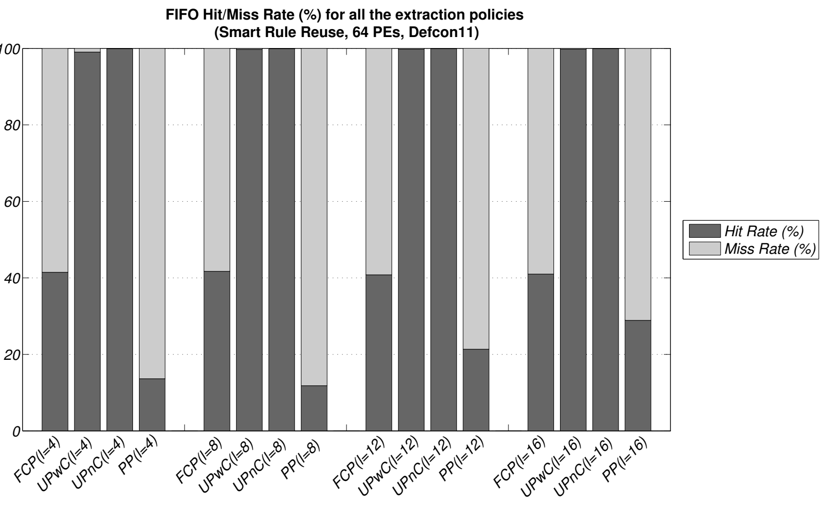


Figure 4.35: Smart Rule Reuse technique evaluation (Defcon11). FIFO depth is 64. FCP is the First Content Prefix technique. UP is the Unique Part Rule technique and PP is PCRE Prefix technique. l represents the predefined length (size) of the prefix/part. There are two cases for the Unique Part Rule technique: 1) UPwC is taking into account the correlation rules and 2) UPnC is not taking into account them.

UPwC and UPnC, as it was expected. UPnC achieves about 20-25% better performance in terms of Hit/Miss Rate for 8 and 16 PEs. For depth of 32, the variation between UPwC and UPnC is significant (18%) only for prefix length of 4 bytes, while for all the other prefix lengths and for 64 PEs, it is only 1-2%.

In general, the correlated rules deteriorate the performance of the Smart Rule Reuse especially for a small number of Processing Elements because the probability for two packets to activate the same unique "leader" rule is very small. In other words, if a packet activates a unique rule which is a "leader" of a partition of correlated rules the activated rules which must be fully matched in the second stage will be $x + n$, where "x" is the number of activated unique rules and "n" the number of activated correlated rules. This has two potential drawbacks: 1) the number of activated rules may be significantly higher than the actual number of resources and 2) if the next packets do not activate this specific unique rule[2] again, the "n" PEs will be useless and new rules will be assigned to them. These drawbacks lead to higher Miss Rates using UPwC than using UPnC.

From Figures 4.32-4.35, it can be easily determined that Smart Rule Reuse has substantially better results when using one of the two cases of Unique Part Rule technique instead of any other extraction technique, for all the selected amounts of PEs. For 8 or 16 PEs, the Hit Ratio is significantly lower of the UPwC than using 32 or 64 PEs because the number of available resources is very close to the average number of activated rules. On the other hand, when using more resources (32 or 64 PEs) the performance of Smart Rule Reuse is significantly better. Comparing all the selected amount of resources, it can be concluded that 32 PEs is the best number of available resources using the Unique Part Rule extraction technique. In this case, the performance of Smart Rule Reuse is perfect taking into account that the best part length for the Unique Part Rule is 8 bytes, as it was mentioned in section 4.2.4. For the other extraction techniques, there are not significant improvements in the performance of Smart Rule Reuse when 64 PEs is selected. Perhaps, using more resources (80 or 90), the Smart Rule Reuse may have better performance using FCP but it was derived that the Smart Rule Reuse achieves almost the best with the Unique Part Rule extraction technique and with significantly fewer PEs.

Figures 4.36-4.39 present the results of Smart Rule Reuse, as before, for the same FIFO depths but for Defcon10. The results are substantially better for FCP and both cases of Unique Part Rule even for very small amount of resources. A first explanation is that Defcon10 has almost 10% of Defcon11's packets. Despite that fact, the average number of activated rules per packet is just 0.5 for both content-based techniques. In addition, the maximum number of activated rules is very low for UP and in the case of FCP which is acceptably high very few packets (0.5-1% of total Defcon10 trace) activates them. Furthermore, UPwC and UPnC provide to the Smart Rule Reuse the same performance because probably significantly few Defcon10 packets activate the unique rules which are "leaders" of correlated partitions. However, the FIFO performance remains low using PP but is improved when the selected prefix size increases, as in Defcon11 results.

To conclude with the evaluation of the Smart Rule Reuse technique, it was shown why the Smart Rule Reuse technique is orthogonal to the extraction techniques or in other words, it can be used along with every single one of them. The most preferable FIFO depth is 32 because after that there is no significant benefit. The best extraction technique which improves the performance of Smart Rule Reuse is the Unique Part

[2]temporal locality cannot be efficiently exploited with correlated rules

Table 4.8: This table shows the number of Hits, Misses and Total Accesses to the FIFO using the Defcon10 input traffic trace for 8 and 16 PEs (FIFO depth=8 and 16). FCP is the First Content Prefix technique.There are two cases for the Unique Part Rule technique. One is taking into account the correlation rules (UPwC) and the other without taking into account them (UPnC). PP is PCRE Prefix technique. l represents the predefined length (size) of the prefix/part. There are two cases for the Unique Part Rule technique.

Extraction Technique	Prefix/Part length	FIFO depth = 8			FIFO depth = 16		
		Hits	Misses	Total Accesses	Hits	Misses	Total Accesses
FCP	4	1,861,385	22,230	1,883,615	1,869,253	14,362	1,883,615
UPwC		1,914,267	33,489	1,947,756	1,931,563	16,193	1,947,756
UPnC		1,888,275	27,243	1,915,518	1,900,495	15,023	1,915,518
PP		2,008,830	3,030,627	5,039,457	2,068,028	2,971,429	5,039,457
FCP	8	1,777,475	11,969	1,789,444	1,783,160	6,284	1,789,444
UPwC		1,835,330	16,733	1,852,063	1,847,796	4,267	1,852,063
UPnC		1,807,174	12,817	1,819,991	1,816,345	3,646	1,819,991
PP		1,814,782	2,707,461	4,522,243	1,852,574	2,669,669	4,522,243
FCP	12	1,778,110	9,544	1,787,654	1,783,292	4,362	1,787,654
UPwC		1,836,106	14,803	1,850,909	1,847,802	3,107	1,850,909
UPnC		1,807,647	11,190	1,818,837	1,816,358	2,479	1,818,837
PP		1,817,983	1,509,771	3,327,754	1,840,232	1,487,522	3,327,754
FCP	16	1,778,382	8,551	1,786,933	1,783,053	3,880	1,786,933
UPwC		1,839,251	14,749	1,854,000	1,850,892	3,108	1,854,000
UPnC		1,810,794	11,134	1,821,928	1,819,449	2,479	1,821,928
PP		1,822,237	825,855	2,648,092	1,841,197	806,895	2,648,092

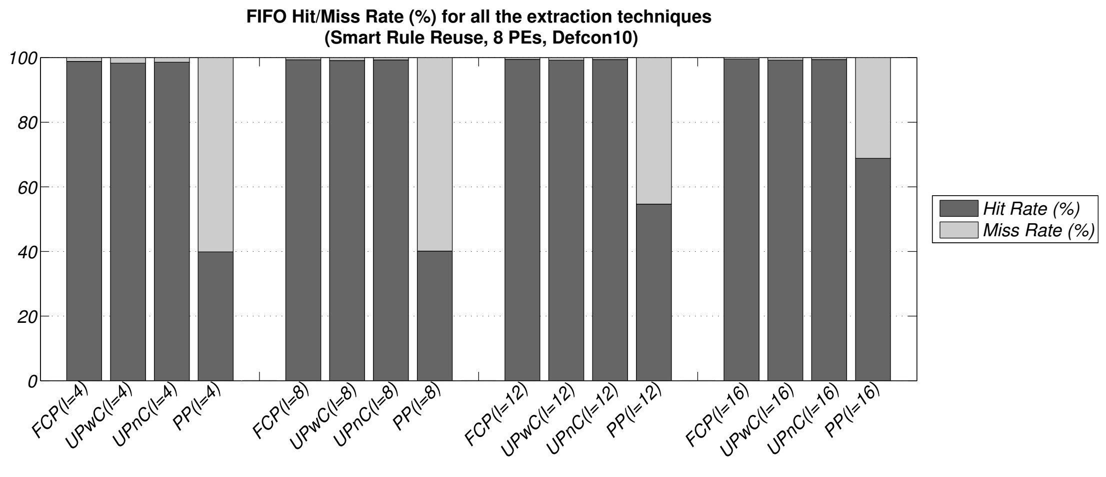


Figure 4.36: Smart Rule Reuse technique evaluation (Defcon10). FIFO depth is 8. FCP is the First Content Prefix technique. UP is the Unique Part Rule techniques and PP is PCRE Prefix technique. l represents the predefined length (size) of the prefix/part. There are two cases for the Unique Part Rule technique: 1) UPwC is taking into account the correlation rules and 2) UPnC is not taking into account them.

Rule. Even if it is used with the correlation memory, for part length of 8 bytes, it makes Smart Rule Reuse achieve the same Performance as with UPnC. Keeping the number of available resources to 32, the Smart Rule Reuse using the FCP or PP requires about

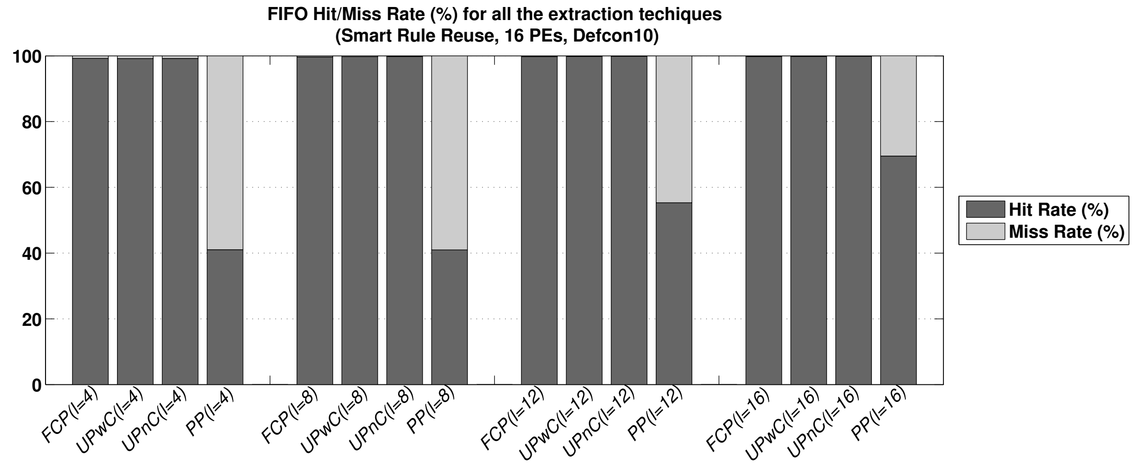


Figure 4.37: Smart Rule Reuse technique evaluation (Defcon10). FIFO depth is 16. FCP is the First Content Prefix technique. UP is the Unique Part Rule technique and PP is PCRE Prefix technique. l represents the predefined length (size) of the prefix/part. There are two cases for the Unique Part Rule technique: 1) UPwC is taking into account the correlation rules and 2) UPnC is not taking into account them.

Table 4.9: This table shows the number of Hits, Misses and Total Accesses to the FIFO using the Defcon10 input traffic trace for 32 and 64 PEs (FIFO depth=32 and 64). FCP is the First Content Prefix technique.There are two cases for the Unique Part Rule technique. One is taking into account the correlation rules (UPwC) and the other without taking into account them (UPnC). PP is PCRE Prefix technique. l represents the predefined length (size) of the prefix/part. There are two cases for the Unique Part Rule technique.

Extraction Technique	Prefix/Part length	FIFO depth = 32			FIFO depth = 64		
		Hits	Misses	Total Accesses	Hits	Misses	Total Accesses
FCP	4	1,875,168	8,447	1,883,615	1,880,427	3,188	1,883,615
UPwC		1,938,019	9,737	1,947,756	1,945,814	1,942	1,947,756
UPnC		1,906,222	9,296	1,915,518	1,913,746	1,772	1,915,518
PP		2,137,438	2,902,019	5,039,457	2,162,840	2,876,617	5,039,457
FCP	8	1,786,475	2,969	1,789,444	1,787,414	2,030	1,789,444
UPwC		1,850,786	1,277	1,852,063	1,851,770	293	1,852,063
UPnC		1,818,911	1,080	1,819,991	1,819,753	238	1,819,991
PP		1,870,335	2,651,908	4,522,243	1,875,426	2,646,817	4,522,243
FCP	12	1,785,108	2,546	1,787,654	1,785,901	1,753	1,787,654
UPwC		1,849,981	928	1,850,909	1,850,704	205	1,850,909
UPnC		1,818,068	769	1,818,837	1,818,672	165	1,818,837
PP		1,854,430	1,473,324	3,327,754	1,856,732	1,471,022	3,327,754
FCP	16	1,784,630	2,303	1,786,933	1,785,231	1,702	1,786,933
UPwC		1,853,074	926	1,854,000	1,853,800	200	1,854,000
UPnC		1,821,167	761	1,821,928	1,821,769	159	1,821,928
PP		1851191	796,901	2,648,092	1,919,160	728,932	2,648,092

40% or 15%-30% (based on the prefix length) respectively more memory accesses than using the Unique Part Rule.

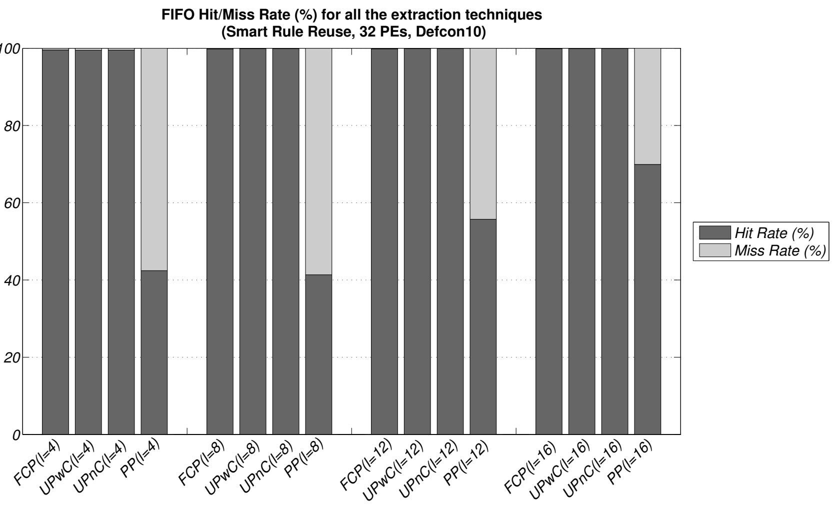


Figure 4.38: Smart Rule Reuse technique evaluation (Defcon10). FIFO depth is 32. FCP is the First Content Prefix technique. UP is the Unique Part Rule techniques and PP is PCRE Prefix technique. l represents the predefined length (size) of the prefix/part. There are two cases for the Unique Part Rule technique: 1) UPwC is taking into account the correlation rules and 2) UPnC is not taking into account them.

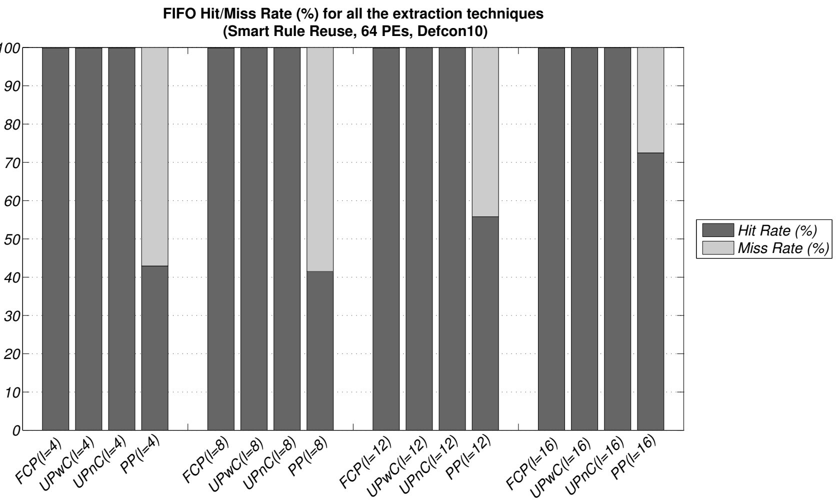


Figure 4.39: Smart Rule Reuse technique evaluation (Defcon10). FIFO depth is 64. FCP is the First Content Prefix technique. UP is the Unique Part Rule technique and PP is PCRE Prefix technique. l represents the predefined length (size) of the prefix/part. There are two cases for the Unique Part Rule technique: 1) UPwC is taking into account the correlation rules and 2) UPnC is not taking into account them.

4.3 Conclusions

This chapter presented the evaluation of the proposed Pre-filtering techniques of this thesis. All the techniques were designed in that way so that they can accommodate and improve the NIDS that uses Pre-filtering to meet the requirements of lightweight processing in the Pre-filtering stage and low implementation cost/high processing throughput in the Full Match stage.

The chapter started by presenting the implementation issues that were encountered during this thesis and the experimental setup. Perl language was selected to implement the techniques while SNORT NIDS was selected for the evaluation purposes. Section 4.2 presented the evaluation of the proposed techniques. At the beginning, the experiment's scenarios were constructed based on the three proposed extraction techniques and four different prefix/part lengths that were selected for the evaluation. The total number of scenarios is 12. Using each one of them, the 12 sets of subrules were created using the SNORT rule-set of July 2008 (version 2.8). The evaluation of the proposed techniques was performed by loading SNORT with one (each time) of the Pre-filtering sets and run it using real intrusion traffic inputs (Defcon11 and 10). The section was then divided in 5 subsections. The first three evaluate the extraction techniques based on the evaluation metric that is used among the Average number of activated rules (AVG), Maximum (MAX) number of them, Cumulative distribution of packets per activated rules. AVG and MAX metrics had been also used by Sourdis in [32] for the Pre-filtering evaluation. The Cumulative distribution was used because it provides with important information about which percentage of packets activate 2, 3, 5, maximum, etc rules/packet. This piece of information is actually "hidden" by the previous metrics and the Cumulative distribution results helped significantly to make final conclusions. Section 4.2.4 present the final results and conclusions for them while the last subsection evaluates the Smart Rule Reuse technique.

Among the three Pre-filtering techniques, the Unique Part Rule extraction technique is the best and the most promising after comparing the results with the ones of other techniques. Both First Content Prefix (FCP) and Unique Part Rule (UP) techniques achieved very low average number of activated rules per packet (3 to 4 rules) but especially, UP technique achieves significantly the lowest number of maximum activated rules (about 50 less than the FCP). The prefix/part size does not affect significantly the efficiency of FCP and UP except the case of selecting the part length of 8 bytes instead of 4 for UP. Additionally, it was shown that the UP technique can be proved even more efficient (45% less maximum activated rules/packet for part length of 4 bytes and 42% for all the other lengths) if the correlated rules could *fairly* eliminated somehow. One suggestion is to find the common characteristics of these rules and match them once, and encode all the rest of their characteristics into one regular expression which will be matched in the next phase once, instead of fully matching all of them.

On the other hand, using the PCRE Prefix technique the average number of activated rules per packet is still a few tens as Pre-filtering concept desires but the number of maximum activated rules is enormously high. Increasing the prefix length can improve it (70% fewer maximum activated rules/packet for prefix length of 16 bytes and an order of magnitude fewer average activated rules) with the drawback of higher latency in the Pre-filtering stage. Of course, as it is also shown by the Cumulative distribution figures, few packets reach the maximum number of activated rules. It is 1-2% of the total number of packets for all the trace cases for the content-based techniques and 5-6% for the PCRE Prefix technique. However, in order to avoid cases of performance degradation and denial of service even the maximum number of activated rules should be handled efficiently by the Full Match engine of the respective NIDS.

After the extraction techniques, the Smart Rule Reuse technique was evaluated. The idea is totally different than the previous ones and attempts to exploit the temporal locality of the activated rules between two or more consecutive packets in the second stage of processing. The technique was evaluated by measuring the number of Hit/Misses on the special FIFO of the Smart Rule Reuse block for all the extraction scenarios. Four different number of resources for the second stage and hence four FIFO depths were selected: 8, 16, 32 and 64. The best FIFO depth was determined to 32 because UP with part length of 8 bytes achieved almost 99% Hit Rate for Defcon11 traffic trace. That means that with 32 PEs, few re-downloadings of the firmware of the rules may have to be done for a potentially large amount of packets.

To conclude, Unique Part Rule extraction technique seems to be the most promising extraction technique for Pre-filtering and it is also more scalable because if a unique rule portion is not found, it can be utilized by the Rule Correlation. However, it requires acceptably more computational power in order to preprocess the rules and produce the set of subrules than the First Content Prefix technique. On the other hand, if the system's requirements are very strict for very fast updating of the rules inside the system, the First Content Prefix technique is better to be used but this will require more PEs (in HW NIDS) or more computational power (in both SW and HW NIDS) in the second stage. Finally, PCRE Prefix should be avoided because it requires substantial higher computational power in order to preprocess the rules and requires also a lot of resources in the second stage. However, it should be mentioned that SNORT uses PCREs especially to verify the correctness of previous content matchings. However, the concept of Pre-filtering targets all the NIDS machines and not just SNORT and perhaps, PCRE Prefix can be more efficient in NIDS that exploit PCREs in a more efficient way. Finally, it was shown that the Smart Rule Reuse technique can substantially improve the second stage of a NIDS that uses Pre-filtering.

5 Conclusions

The attacks have been evolved in the last years. Deep Packet Inspection (DPI) is the most efficient method to clearly determine if an attack takes place by scanning deeply the content of the incoming packets. Packet Pre-filtering technique attempts to optimize DPI by dividing it into two stages to improve it in terms of processing throughput, implementation cost and scalability. These issues were addressed by this thesis by proposing techniques which optimize the Pre-filtering approach. Three extraction techniques (First Content Prefix, PCRE Prefix, Unique Part Rule) have been proposed which create the set of subrules which is used in the first stage of processing. The set of subrules must be trivial to require lightweight processing and also properly selected to activate only a few rules to be processed in the second stage. The most efficient among the extraction techniques is the Unique Part Rule which extracts the most efficient set of subrules according to the previous requirements. Two more techniques were proposed: the Rule Correlation, which optimizes the first stage of processing and the Smart Rule Reuse which optimizes the second stage of processing, both in terms of speed.

This chapter is organized as follows: Section 5.1 summarizes the proposed techniques and the issues that were addressed by this thesis. Section 5.2 presents the contributions while section 5.3 concludes this thesis with the future suggestions.

5.1 Summary

Network Security is a significant issue nowadays, due to the enormous flow of information. Many types of security systems exist but the attacks have been significantly evolved, using more sophisticated methods to penetrate them. The key characteristic of the current attacks is that the threat is "hidden" inside the packet. For that reason, Deep Packet Inspection, which is the most efficient method, must be performed in every single packet of the flow, in order to determine whether it is malicious or not. The Deep Packet Inspection (DPI) classifies the packet based on its header, scans in deep its payload and searches for known attack-types (intrusion detection rules) that are stored in a database to eventually determine whether it is hostile. The task of DPI is performed especially by Network Intrusion Detection Systems (NIDS). The disadvantage of DPI and therefore NIDS' disadvantage is that a significant amount of processing is needed to determine whether one of the thousands of intrusion rules describes a single specific packet. The Multi-stage packet inspection and more precisely Packet Pre-filtering is the solution to this.

The idea of Packet Pre-filtering, as it is mentioned in Chapter 2, is to divide the DPI into two stages. The first stage, which is called Pre-filtering stage is loaded with the preprocessed rule-set that consists of smaller portions of rules. A single incoming packet will be scanned using them and the activated rules will be sent to the second stage. The second stage is referred as Full Match stage and matches fully only the activated rules against the incoming packet. The Full Match stage may be implemented in hardware and consists of Processing Elements or may be implemented in software. The question that arises is how to select a portion of each rule so that the Pre-filtering stage needs lightweight processing and only few rules are activated to be processed in the second stage. Chapter 2 describes also an open-source NIDS, SNORT, which was used during this thesis. The rules' characteristics are explained to be aware which characteristics make the computational power increase, in order to be avoided and which should be used. The entire header and the Content and the PCRE from the payload options of the rule were finally selected to be exploited by the proposed techniques.

Efficient techniques were proposed that accommodate Pre-filtering to meet the above requirements and to improve and enhance it. Chapter 3 presents and discusses them in detail. There are three approaches on how to select the proper portion of the rules in order to extract an efficient, according to the requirements, set of subrules: 1) First Content Prefix (FCP), 2) PCRE Prefix (PP) and 3) Unique Part Rule (UP). The First Content Prefix is the brute-force technique since it extracts the prefix of the first (if more than one) content. It suffers from the drawback that many rules may have the same header and the same first content or prefix of it, as a result the same rules are activated together. However, it requires the least processing time to preprocess the rules (2.69 seconds for the SNORT rule-set of July 2008). On the other hand, the PCRE Prefix attempts to extract the prefix of the PCRE, which is a specific type of regular expressions. The portion selection of a PCRE is a difficult task due to the nature of PCREs because even a slight change of it may change its usage (i.e. match packets that should not match and vice versa). An amount of extraction rules was presented in order to correctly and efficiently create the set of subrules based on the PCREs. Preprocessing the rules of the SNORT rule-set of July 2008 using the PCRE Prefix requires a significant latency of about 75 minutes. Finally, the Unique Part Rule extraction technique searches for a part/parts of content(s) in order to find unique combinations of header and content parts so that the set will consist of unique subrules. It is the most sophisticated extraction technique and the most efficient but preprocessing the SNORT rule-set of July 2008 and produce the set of subrules requires an acceptably significant latency ($\sim$65 seconds). The processing parts of UP are: (a) extract the unique subrules using the header and any part of one content, (b) extract the unique subrules using the header and any parts of two or more contents (if more than one), for the failed rules of (a), and (c) correlate the total failed rules using Rule Correlation.

Apart from the extraction techniques, the Rule Correlation and the Smart Rule Reuse techniques were also proposed. The first one improves the Pre-filtering stage since it correlates rules which have similar characteristics. The proposed idea is the Qualitative Rule Correlation and in this thesis is used in combination with the UP technique. Actually, the failed rules of the previously mentioned processing parts (a) and (b) during Unique Part Rule extraction technique are utilized by Rule Correlation by assigning to them the ID of one unique subrule which matched with. Then, the correlation partitions are formed based on the requirement that the fewest rules are contained in each partition. The partitions are stored into a memory called "Correlation

Memory". The "leader" of one partition is one specific unique subrule. Consequently, if some of the activated rules are "leaders" then the unique and the correlated rules are sent to the next stage without partially matching the correlated ones. The idea of the Rule Correlation is generic and can be also used, for example, with the FCP technique to correlate the rules that have the same prefix of the first content in a similar way as with UP. As opposed to the Rule Correlation, the Smart Rule Reuse takes advantage of the potential temporal locality that may exist between the activated rules of consecutive packets. Several attacks may span through multiple packets and thus, there may be common activated rules between consecutive packets. The Smart Rule Reuse can be applied only to hardware NIDS and is aware of which rules have been assigned to which Processing Elements of the second stage so that if there are common activated rules, only the firmware of the new ones is needed to be downloaded. A special FIFO is used for the assignment of the rules to the PEs.

The proposed Pre-filtering techniques were implemented and evaluated as presented in Chapter 4. The experimental setup was presented at the beginning of this chapter. SNORT was used in order to evaluate the proposed techniques and for that reason, SNORT source code was modified bypassing some of its optimizations in order to provide with all the matched rules and not only with alerts. Then, the experiment scenarios were constructed. There are 12 different scenarios (3 extraction techniques $\times$ 4 selected prefix/part lengths(4, 8, 12, 16 bytes)). Defcon (versions 11 and 10) attack traffic traces were used as input to SNORT. The metrics Average and Maximum number of activated rules and Cumulative distribution of packets per activated rules were used for the evaluation of the extraction techniques while the FIFO Hit/Miss Rate was used for the Smart Rule Reuse technique's evaluation.

The Unique Part Rule with part length of 8 bytes is the most efficient Pre-filtering technique among the three extraction techniques. 2.5 rules are activated per packet on average while the maximum number of them is approximately 64 counting also the correlated rules. This number indicates also the number of required resources in the second stage of processing and is 50 and 80 less than using the FCP and the PP (for prefix length equal to 16 bytes) respectively. In addition, the Cumulative distribution showed that 1-2% of the total packets activates an amount of rules that is close or equal to the maximum for FCP and UP, and 5-6% for the PP. It was also concluded that PP should be avoided because it needs a lot of processing to create the set of subrules and does not assist Pre-filtering to achieve the desired performance. Rule Correlation was evaluated along with UP. It achieved to correlate 1700 rules out of the 9000 of the entire set providing to the Pre-filtering stage higher processing throughput. If the correlated rules are not taken into account in the evaluation of UP using a "fair" way, the achieved number of activated rules would be 2 on average while the maximum only 38, meaning a substantial improvement in UP. One suggestion was to share the common characteristics of the correlated rules of each partition and match them as one and encode all their unique characteristics in a regular expression to match it once in the second stage. Finally, Smart Rule Reuse optimizes significantly the second stage of processing. It is orthogonal to the extraction techniques since it can be used along with any of them. It was evaluated by measuring the Hit/Miss Rate of the FIFO of the Smart Rule Reuse block. Using 32 Processing Elements instead of 8, 16 or 64 the Hit Rate is almost 99%

using the UP extraction technique which means that re-downloading of the firmware of the activated rules is not needed for a significant amount of packets.

All the proposed techniques have been implemented using Perl programming language. The set of subrules is produced automatically using any of the proposed extraction techniques. The correlated rules are also partitioned in an automatic way.

5.2 Contributions

This thesis proposed several techniques which accommodate Pre-filtering to meet its requirements and some of which also improve it substantially. In a NIDS that uses Pre-filtering, the Deep Packet Inspection is divided into two stages, where the first stage must have high processing throughput while the second must have small implementation cost. Additionally, the NIDS should be also scalable as the number of rules increases to support more threats. Each one of the proposed techniques managed to address some of the above issues. The Unique Part Rule is the most efficient, while the PCRE Prefix should be avoided. In addition, all the proposed techniques except Smart Rule Reuse can be used in both software and hardware NIDS (sequential and parallel processing) that uses Pre-filtering. The contributions are divided into the contributions for the extraction techniques and general contributions.

The contributions regarding the extraction techniques are:

- **First Content Prefix**: This technique extracts the prefix of the first content of each rule. It is the most trivial and can be used in NIDS that have strict limits on fast processing of the first stage and fast rules' updating and have a lot of available resources (approximately 120 Processing Elements (PEs) are needed if it is implemented in hardware) in the second stage.

- **PCRE Prefix**: This extraction technique utilized the regular expressions of the rules in order to create the set of subrules. From the evaluation in SNORT, it was concluded that it should be avoided since it makes Pre-filtering stage need significant processing; rules' updating is slow and the number of required resources in the second stage is enormously high (about 400 PEs for 8 bytes prefix length and 150 for 16 bytes). However, it may better perform in NIDS that exploits regular expressions more efficiently than SNORT.

- **Unique Part Rule**: It is one of the main contributions of this thesis. Unique Part Rule is the most sophisticated and efficient extraction technique of the proposed ones. It creates a set of unique subrules for the Pre-filtering stage. It supports the Pre-filtering stage to achieve high processing throughput since every single incoming packet is checked with static patterns of only 8 bytes. The number of the required resources in the Full Match stage can be significantly small (approximately 64) if it is implemented in hardware, or the processing throughput can be significantly high if it is implemented in software. With this number of available resources, Unique Part Rule can handle efficiently the Denial of Service attacks. It is also the most scalable technique since the updating of the subrules is very scalable (also using Rule Correlation). It needs though acceptably significant processing ($\sim$65 seconds) to perform the rules' preprocessing.

The general contributions include the remaining two techniques, which can be used with any of the extraction techniques, and the way of evaluation and are:

- **Rule Correlation technique**: The proposed Rule correlation correlates similar or same subrules in an efficient way in order a packet to activate them without partially matching them in the Pre-filtering stage. It is considered a significant contribution since this technique may accelerate the processing in the Pre-filtering stage because less rules are needed to be matched. It is orthogonal to the extraction techniques since it can be used in combination with any of them. Furthermore, it assists the Unique Part Rule technique significantly in terms of scalability. Finally, when it was used in combination with the Unique Part Rule extraction technique, it correlated 1700 rules out of the 9000 of the SNORT rule-set of July 2008, substantially improving the speed or the implementation cost of the first stage.

- **Smart Rule Reuse technique**: The proposed technique optimizes the second stage of processing of a hardware NIDS that uses Pre-filtering reducing the memory accesses, downloading only the newly activated rules based on the fact that consecutive packets may activate the same rules. It is also orthogonal to the extraction techniques. An amount of 32 PEs can handle efficiently the processing of packets of several attacks in the second stage, downloading the newly activated rules only few times.

- A final contribution is that all the above techniques were evaluated using a real NIDS (SNORT), a real rule-set and real traffic inputs.

5.3 Future Suggestions

This thesis has proposed several efficient techniques some of which optimize substantially the Pre-filtering which is one of the best proposed solutions that efficiently addresses the high processing requirements of Deep Packet Inspection. However, there are some topics that have been left to be studied and addressed in order to have more complete conclusions on some topics. Some of the future works can be:

- Regular expressions and more specifically PCREs are matched complementally to content in SNORT. An interesting work would be to use the proposed PCRE Prefix technique on other rule sets and other NIDS that exploit regular expressions better than SNORT.

- Another future topic related to regular expressions is based on the observation that PCREs contain some static patterns inside them to more efficiently distinct if a previous content matching is a false positive or not. Thus, it would be an interesting technique to efficiently extract this part of the regular expression and use it in the Pre-filtering.

- In addition, an interesting work would be to write intrusion detection rules including a field for Pre-filtering.
- A significant subject, which should be studied, is an efficient way to "eliminate" the correlated rules from the Pre-filtering stage. An interesting work would be to discover efficient techniques which will gather and encode the unique characteristics of the correlation rules into one so that the number of activated rules/packet which must be fully matched in the second stage is substantially smaller.
- Finally, a work which could give more clear conclusions on how significant is the role of the proposed techniques would be to divide SNORT or other NIDS in two stages and explicitly simulate them as NIDS with Pre-filtering, measuring the processing throughput and the implementation cost.

Bibliography

[1] Alfred V. Aho and Margaret J. Corasick, *Efficient string matching: an aid to bibliographic search*, Commun. ACM **18** (1975), no. 6, 333–340.

[2] Anonymous, *Maximum security*, SAMS, 2001.

[3] S. Antonatos, M. Polychronakis, P. Akritidis, K. G. Anagnostakis, and E.P Markatos, *Piranha: Fast and memory-efficient pattern matching for intrusion detection*, In Proceedings 20th IFIP International Information Security Conference (SEC 2005, 2005.

[4] Michael Attig, Sarang Dharmapurikar, and John Lockwood, *Implementation results of bloom filters for string matching*, FCCM '04: Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (Washington, DC, USA), IEEE Computer Society, 2004, pp. 322–323.

[5] Ricardo Baeza-Yates and Gaston H. Gonnet, *A new approach to text searching*, Commun. ACM **35** (1992), no. 10, 74–82.

[6] Zachary K. Baker and Viktor K. Prasanna, *A methodology for synthesis of efficient intrusion detection systems on fpgas*, FCCM '04: Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (Washington, DC, USA), IEEE Computer Society, 2004, pp. 135–144.

[7] Z.K. Baker, Hong-Jip Jung, and V.K. Prasanna, *Regular expression software deceleration for intrusion detection systems*, Field Programmable Logic and Applications, 2006. FPL '06. International Conference on (2006), 1–8.

[8] Ken Baylor, *http://www.securitypronews.com/news/securitynews/spn-45-20060911evolutionofthehackerthreat.html*.

[9] Robert S. Boyer and J. Strother Moore, *A fast string searching algorithm*, Commun. ACM **20** (1977), no. 10, 762–772.

[10] Damian Conway, *Recdescent parser. http://search.cpan.org/dist/parse-recdescent/*.

[11] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, *Deep packet inspection using parallel bloom filters*, High Performance Interconnects, 2003. Proceedings. 11th Symposium on (2003), 44–51.

[12] Vassilis Dimopoulos, Giorgos Papadopoulos, and Dionisios N. Pnevmatikatos, *On the importance of header classification in hw/sw network intrusion detection systems.*, Panhellenic Conference on Informatics, Lecture Notes in Computer Science, vol. 3746, Springer, 2005, pp. 661–671.

[13] Maya Gokhale, Dave Dubois, Andy Dubois, Mike Boorman, Steve Poole, and Vic Hogsett, *Granidt: Towards gigabit rate network intrusion detection technology*, FPL

'02: Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications (London, UK), Springer-Verlag, 2002, pp. 404–413.

[14] P. Gupta and N. McKeown, *Algorithms for packet classification*, Network, IEEE **15** (2001), no. 2, 24–32.

[15] Dan Gusfield, *Algorithms on strings, trees, and sequences: computer science and computational biology*, Cambridge University Press, New York, NY, USA, 1997.

[16] Philip Hazel, *http://www.pcre.org/*, Version 7.8.

[17] SecureWorks Corporate Headquarters, *http://www.secureworks.com/research/newsletter/2008/01/*.

[18] D. Knuth, J. Morris, and V. Pratt, *Fast pattern matching in strings*, SIAM Journal on Computing **6** (1977), no. 2, 323–350.

[19] Evangelos P. Markatos, Spyros Antonatos, Michalis Polychronakis, and Kostas G. Anagnostakis, *Exclusion-based signature matching for intrusion detection*, In Proceedings of the IASTED International Conference on Communications and Computer Networks (CCN, 2002, pp. 146–152.

[20] James Moscola, Young H. Cho, and John W. Lockwood, *A reconfigurable architecture for multi-gigabit speed content-based routing*, HOTI '06: Proceedings of the 14th IEEE Symposium on High-Performance Interconnects (Washington, DC, USA), IEEE Computer Society, 2006, pp. 61–66.

[21] James Moscola, John Lockwood, Ronald P. Loui, and Michael Pachos, *Implementation of a content-scanning module for an internet firewall*, FCCM '03: Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (Washington, DC, USA), IEEE Computer Society, 2003, p. 31.

[22] G. Papadopoulos and D. Pnevmatikatos, *Hashing + memory = low cost, exact pattern matching*, Field Programmable Logic and Applications, 2005. International Conference on (2005), 39–44.

[23] D. Pnevmatikatos and A. Arelakis, *Variable-length hashing for exact pattern matching*, Field Programmable Logic and Applications, 2006. FPL '06. International Conference on (2006), 1–6.

[24] M. Rabin, *Fingerprinting by random polynomials*, Tech. Report TR-15-81, 1981.

[25] Ramaswamy Ramaswamy, Lukas Kencl, and Gianluca Iannaccone, *Approximate fingerprinting to accelerate pattern matching*, IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (New York, NY, USA), ACM, 2006, pp. 301–306.

[26] Martin Roesch, *http://www.snort.org/*.

[27] Reetinder Sidhu and Viktor K. Prasanna, *Fast regular expression matching using fpgas*, FCCM '01: Proceedings of the the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (Washington, DC, USA), IEEE Computer Society, 2001, pp. 227–238.

[28] Haoyu Song and John W. Lockwood, *Efficient packet classification for network intrusion detection using fpga*, FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays (New York, NY, USA), ACM, 2005, pp. 238–245.

[29] Haoyu Song, Jonathan Turner, and John Lockwood, *Shape shifting tries for faster ip route lookup*, ICNP '05: Proceedings of the 13TH IEEE International Conference on Network Protocols (Washington, DC, USA), IEEE Computer Society, 2005, pp. 358–367.

[30] I. Sourdis, V. Dimopoulos, D.N. Pnevmatikatos, and S. Vassiliadis, *Packet prefiltering for network intrusion detection*, in 2nd ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), December 2006, pp. 183–192.

[31] I. Sourdis, D. Pnevmatikatos, S. Wong, and S. Vassiliadis, *A reconfigurable perfect-hashing scheme for packet inspection*, Field Programmable Logic and Applications, 2005. International Conference on (2005), 644–647.

[32] Ioannis Sourdis, *Design and algorithms for packet and content inspection*, Ph.D. thesis, Computer Enginnering lab, Delft University of Technology, 2007.

[33] Ioannis Sourdis, Joao Bispo, Joao M. Cardoso, and Stamatis Vassiliadis, *Regular expression matching in reconfigurable hardware*, J. Signal Process. Syst. **51** (2008), no. 1, 99–121.

[34] Ioannis Sourdis and Dionisios Pnevmatikatos, *Pre-decoded cams for efficient and high-speed nids pattern matching*, FCCM '04: Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (Washington, DC, USA), IEEE Computer Society, 2004, pp. 258–267.

[35] David E. Taylor, *Survey and taxonomy of packet classification techniques*, ACM Comput. Surv. **37** (2005), no. 3, 238–275.

[36] Gianni Tedesco, *http://www.scaramanga.co.uk/firestorm/*.

[37] Gianni Tedesco and Uwe Aickelin, *Data reduction in intrusion alert correlation*, CoRR **abs/0804.1281** (2008).

[38] Crypto. The shmoo group. Hacking, Moose, *http://www.shmoo.com/*.

[39] Untangle, *http://www.untangle.com/*.

[40] Alfonso Valdes and Keith Skinner, *Probabilistic alert correlation*, Recent Advances in Intrusion Detection, 2001, pp. 54–68.

[41] J. van Lunteren and T. Engbersen, *Fast and scalable packet classification*, Selected Areas in Communications, IEEE Journal on **21** (2003), no. 4, 560–571.

[42] Trevor Warren, *http://www.freeos.com/articles/3496/*.

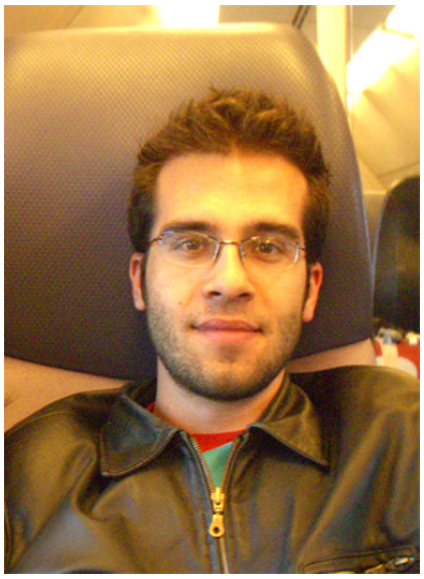
[43] Bleeding Edge Threats website, *http://www.bleedingthreats.net/*.

[44] Bro NIDS website, *http://www.bro-ids.org/*.

[45] S. Wu and U. Manber, *A fast algorithm for multi-pattern searching*, Tech. Report TR-94-17, 1994.

[46] Bin Zhu and Ali A. Ghorbani, *Abstract alert correlation for extracting attack strategies*, 2005.

Curriculum Vitae



Angelos Arelakis was born in Thessaloniki, Greece, on 16th of March in 1984. In 2001, he graduated from the High School of Irakleia - Serres with GPA 18.7/20.0 (Excellent). The same year and through the Greek national examinations, he enrolled into the Electronic and Computer Engineering (E.C.E.) department of Technical University of Crete, Greece.

The duration of studies in E.C.E. was 5 years and he received his diploma in July 2006 with GPA 7.93/10.0 (Very Good). His thesis title was "Extension of the HashMem Architecture into the Variable-Length HashMem Architecture for Exact Pattern Matching". The thesis work was elaborated in the Microprocessor and Hardware Laboratory under the supervision of Associate Professor Dionysios Pnevmatikatos. A publication was also resulted from his thesis in the FPL conference, 2006 with title "Variable-Length Hashing for Exact Pattern Matching".

In September of 2006, he was accepted in the MSc program of Computer Engineering laboratory in Delft University of Technology where he is an MSc student until now. His research interests include Network Intrusions Detection Systems and generally network security, fault tolerant computing, computer architecture, networks-on-chip and whatever includes HW/SW co-design and reconfigurable computing.