Federal University of Pernambuco

Graduation in Computer Science

Informatics Center
2008.2

THIAGO DE BARROS LACERDA

ON THE OPTIMIZATION OF DEEP PACKET INSPECTION

Diploma Thesis

Advisor: Djamel Sadok (jamel@cin.ufpe.br)
Co-advisor: Stênio Fernandes (stenio@gprt.ufpe.br)

Recife, PE
2008

Federal University of Pernambuco

Graduation in Computer Science

Informatics Center
2008.2

ON THE OPTIMIZATION OF DEEP PACKET INSPECTION

Diploma Thesis

Diploma Thesis presented to the Informatics Center of the Federal University of Pernambuco by Thiago de Barros Lacerda, advised by Prof. PhD. Djamel Sadok. As requirement to obtain the Bachelor of Science degree in Computer Science.

Advisor: Djamel Sadok (jamel@cin.ufpe.br)
Co-advisor: Stênio Fernandes (stenio@gprt.ufpe.br)

Recife, PE
2008

APPROVAL SHEET

THIAGO DE BARROS LACERDA

ON THE OPTIMIZATION OF DEEP PACKET INSPECTION

Thesis approved on December 4th, 2008.

Thesis Committee

_____

Prof. Djamel Fawzi Hadj Sadok, PhD – UFPE

(Advisor)

_____

Prof. Nelson Souto Rosa, PhD – UFPE

(Examiner)

To everyone that has supported me, in any aspect: family, girlfriend, professors, classmates, friends and work colleagues. These four years and a half were very tough, but worthwhile and I would do it all again, if necessary.

“The significant problems we have cannot be solved at the same level of thinking with which we created them.”

Albert Einstein

ACKNOWLEDGEMENTS

First of all, I would like to remark that the following acknowledgments are not only for those who have helped me on this thesis, but for all that have been with me during my academic years. It is also important to remark that those acknowledgements are not in order of importance.

I'd like to thank my father Cândido Lacerda and my mother Catari Lacerda, for those years of support, love and provided education.

I also want to thank my Project Manager and Co-Advisor Stênio Fernandes, who always helped me when I needed, for all the support provided within this work even with his "'to do' buffer" full of important (and Canadian) things (tending to overflow) to do. I cannot forget to sincerely thank my advisor Djamel Sadok, for all support within this thesis and at GPRT. Not less important, I would like to thank Judith Kelner, for all those almost 3 years of support at GPRT.

To all those who have been with me on this undergraduate journey at the Informatics Center (especially those that worked together with me in "Projetão"), for all nights working together, parties, meetings and unforgettable times of fun: Guedes (also for all valuable and productive conversations about this work), Pigmeu (for all rides and Friday's beer time), Japa, David, Jesus (no, not the son of God) (even considering me only a colleague), Digão (wheelbarrow), Janga Boy (for showing to me that only modeling is enough), Gravatá, Seabra, Rilter Seabra (Seabra's brother), Rebeka (Rosalvão), Moxinho, Vivi, Xeifer, Chico (remainder of the name omitted, due to its extensiveness) and Josias. I cannot forget to also thank Cavanhaque, Bigode and Kelly, for all provided moments of happiness and fun.

My sincere thanks to Thaís, for all those years of support, advises, love and patience when I had to spend nights and weekends working/studying at the University.

Additionally, I would like to thank all my colleagues at GPRT, for all friendship, advises, support and apprenticeship provided: Arthur Callado (the oracle), Rafael Antonello (Patrão), Ana Cristina, Alysson (rubro-negro), Fabrício, Nadia, Manu, Feitosa (the key's lender), Duda, Rover, Rodrigo Germano, Hachid, Curupa, Nacha and Kalil.

I would like to thank God.

Please forgive me, if I forgot someone.

ABSTRACT

Among the current mechanisms to perform traffic measurement and classification, Deep Packet Inspection (DPI) is the most used among Internet Service Providers (ISP) and network managers due to its accuracy. However, the present speed of the current Internet links is forcing such motoring task to rely on specialized hardware and software, instead of commodity ones present on common PCs. This is caused by the heavy processing performed by comparisons made between the packet's payload, from a given flow, against networked applications' signatures, commonly expressed as regular expressions.

This thesis presents some techniques aiming to optimize DPI systems, without appealing to dedicated hardware and software solutions. At first, it is shown how a precise architecture software design can leave an open path to further optimizations. After that, some techniques intending to optimize the time spent with regular expression matching operations are evaluated: analysis of only the first 7 packets of a given flow, payload truncating e regular expressions rewriting. Then, with the mentioned techniques and optimizations, it was possible to obtain a reduction of almost 100% on the packet loss rate of a DPI system, running on a Linux box with an incoming rate of 900Mbit/s, and increase the volume classification completeness on 250%. It is worth telling that all evaluations were performed with real trace files, collected at a backbone of one of the most important Brazilian ISPs.

Keywords: Computer Networks Performance, Deep Packet Inspection, Traffic Analysis.

RESUMO

Dentre os atuais mecanismos de monitoração, medição e classificação de tráfego, *Deep Packet Inspection* (DPI) é o mais utilizado entre Provedores de Acesso a Internet e administradores de redes, devido a sua precisão. Porém, por causa das atuais velocidades de tráfego que os enlaces vêm alcançando, esse tipo de monitoração torna-se inviável com dispositivos de *hardware* e *software* encontrados em PCs comuns, com isso, tal monitoração, comumente utiliza-se de dispositivos de *hardware* e *software* especializados. Essa inviabilidade é causada pelo pesado processamento, decorrente das comparações feitas na carga útil (*payload*) de cada pacote, de um determinado fluxo, com assinaturas aplicações de rede, geralmente representadas por expressões regulares.

O presente trabalho propõe técnicas visando a otimização de sistemas baseados em DPI, utilizando *hardware* e *software* presentes em PCs comuns. Primeiro será mostrado como uma boa arquitetura de tais sistemas pode abrir caminho para diversas otimizações. Após isso, técnicas visando diminuir o tempo gasto em operações de *matching* com expressões regulares serão aplicadas: análise dos 7 primeiros pacotes de um determinado fluxo, diminuição do *payload* a ser analisado e reescritas de expressões regulares. Com tais técnicas e otimizações, foi possível diminuir em quase 100% a taxa de perda de pacotes de um sistema DPI, operando a 900Mbit/s numa máquina com Linux, assim como aumentar em 250% o volume de tráfego classificado pelo mesmo. Vale salientar que todos os experimentos foram realizados com traces de pacotes reais, coletados no *backbone* de um dos maiores provedores de acesso à Internet do Brasil.

Palavras-Chave: Análise de Tráfego, *Deep Packet Inspection*, Desempenho de Redes de Computadores.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Internet Service Providers (ISP) and network administrators always had deep interest about knowing what type of traffic is going through their backbone. Therefore, network administrators and managers need to constantly perform network monitoring and traffic analysis. Such tasks are very important to provide overall information about the network status, such as: network problems, protocols and applications that are being used by users and other information about their network infrastructure. Additionally, this management must be very precise, since erroneous assumptions about the network, provided by monitoring mechanisms, can lead to undesirable expenses.

One of the most used types of network monitoring technique is the Traffic Classification, which can be online and offline. In order to know about network traffic characteristics instantly, ISPs and network managers are commonly relying on Online Traffic Classification (OTC). OTC is a great aid to traffic management, since the ISPs can take decisions and some management actions in real time, according to the traffic that is flowing through their backbone. For example, they can decrease, or increase, the Quality of Service (QoS) of some determined application, or block anomalous flows. In the past, OTC was based on the well known port approach, where Internet applications had well-known port numbers to send and receive their packets. However, with the current dynamic behavior of the Internet, mainly caused by peer-to-peer (P2P) applications [11], this type of classification is not sufficient and accurate anymore, since there are applications that use tunneling, e.g. through HTTP, to bypass firewalls and thousands of new applications, using different port numbers. Additionally, the classification based on the port-application tuple is not efficient to detect malicious flows and attacks over the Internet.

Nowadays, since the traditional Traffic Classification based on port numbers is not efficient anymore, OTC is relying on Deep Packet Inspection (DPI), which is the monitoring approach that provides the most detailed information about the packets that are traversing the network. DPI systems capture the whole packet and try to perform some kind of matching, between the packet's payload and an application signature, commonly represented by Regular Expressions (RegEx). Therefore, DPI systems are considered the most precise and maintainable, since

RegExes can fully describe various protocols' messages and expressions can be modified or added to the system, in order to fix some broken pattern, or recognize new applications.

In order to deploy systems with good classification completeness, those DPI systems are equipped with a huge set of RegEx signatures, representing a great variety of Internet applications. However, the RegEx matching process performed by DPI systems is the most expensive task on such applications, consuming over 90% of the CPU time in the whole system [26]. Therefore, the high processing time consumed by the RegEx matching forces DPI systems to use specialized hardware and software solutions, striving to obtain good performance, especially on high speed links. Additionally, in their majority, such hardware and software solutions used by DPI systems, and the DPI system itself, are quite expensive, since these use dedicated high end technology components to achieve the highest performance and effectiveness possible. Consequently those systems are not financially feasible to some end users and companies.

Deploying DPI systems that are able to deal with high speed links (e.g. 1Gbit/s) in commodity hardware and software is still an open challenge, among the industry and academic community. Such challenges are mainly caused by the huge and rapid growth of the speed of the Internet links, therefore that type of classification is becoming even more difficult to accomplish, without packet loss and while maintaining the desirable performance. Therefore, the scientific community has been working on how to achieve classification accuracy and performance, in order to combine good accuracy and acceptable, or zero, loss rate.

Traditionally, DPI systems that use commodity software commonly rely on Linux-based Operating Systems (OS), which are the most used systems for network applications. But Linux suffers from degradation, in packet capture throughput, when handling high speed links. Such degradation is attributed to the additional packet copy performed from the kernel space to the user space, where the application, which the packet is destined for, is operating. Therefore, to a Traffic Classification Application, that copy, represents an overhead that can lead to performance decrease. Also, traditional OSs, such as Linux, are deployed to be general purpose systems, sharing all resources with other process and applications, instead of dedicated OSs that are deployed, with improved capabilities, serving dedicated applications, such as network monitoring tools. Additionally, commodity PCs and

NICs are not prepared to deal with high loads of traffic. They do not have dedicated components, such as network processors, buses and memory, and therefore they have a poor performance when submitted to extreme loads.

To improve DPI systems, the research community has been working on several approaches such as multicore architectures [14], algorithms to enhance the performance of RegEx matching [26], Linux kernel modifications that lead to no packet copies [5][6], Graphic Processing Units (GPU) usage [8], etc. Additionally, some optimizations in DPI systems are done by means of dedicated and expensive hardware, which incredibly outperforms software solutions.

As seen in the previous paragraphs, obtaining good performance in DPI systems, using commodity hardware and software, such as on a common Linux box, is still an open problem within the research and industry community. Therefore, this thesis aims at showing how some architectural and software layer modifications can lead to considerable performance gains, using commodity hardware. With those gains, a system that was not able to deal with 100Mbit/s without losing packets, will reach 900Mbit/s with almost no losses and no decrease in classification completeness. To show such gains, a traditional DPI system, running on a Linux box, relying on commodity hardware and on common DPI features, is taken as a baseline. Those common features are sequential processing, libpcap library for packet capture, general purpose Hash Table and a set of RegExes signatures. Then some optimizations are performed, within this DPI system, in several levels, in order to obtain the best performance possible without appealing to specialized hardware solutions. First, an architectural level optimization is made, in order to take advantage and fully utilize all available resources of the measurement machine. After that, a slight modification on how packets are forwarded to user space is implemented and a data structure level optimization is performed. Additionally, this thesis shows how other two techniques can lead to a considerable enhancement to this DPI system, such techniques are Packet Counting and Payload Truncating. At last, but not at least, this thesis will also show how subtle modifications on some RegExes signatures can increase its processing speed, reducing the size of the generated state machine.

The remainder of this thesis is organized as follows:

- Section 2: this section presents the essential background needed for the thesis understanding, covering topics such as Regular Expressions, Deep Packet Inspection and packet capture mechanism on Linux-based systems. Additionally the DPI system that is restructured by this work is presented;
- Section 3: it presents the related work that has been done in the research community;
- Section 4: the proposed project for the thesis is presented in this section. It also describes the proposed architecture that was implemented, in order to enhance the DPI system presented at section 2. Finally it presents the methodology that is followed by the evaluations performed;
- Section 5: this section shows the evaluation results obtained with the new architecture and proposed approaches;
- Section 6: this section has an open space to discuss some important points that were observed;
- Section 7: at this section, the main concluding remarks of the thesis are going to be presented. Additionally, this section presents some possible future works.

2 BACKGROUND

A brief description of Regular Expressions, Deep Packet Inspection Systems and the mechanism of packet capture in Linux based systems is given, in order to provide the essential background needed for the remainder of this thesis.

2.1 Regular Expressions

RegEx is a set of strings, which represent a pattern used to match a certain string of characters. They provide an enormous expressiveness without necessity to express the desired patterns one by one, so a RegEx can fully represents a complete protocol communication. Additionally they are widely used in computer science, from compilers and programming languages to text editors.

Also, RegExes use a formal language to describe it, having two widely known writing patterns, namely POSIX [19] and Perl [17]. Its formal language has a set of metacharacters, used to build a RegEx, which are described at Table 2.1.

RegExes are commonly represented as state machines, which can be Deterministic Finite Automata (DFA), Non-Deterministic Finite Automata (NFA) and Extended Finite Automata (XFA). These automata can consume lots of memory if the RegEx is too complex and also take a huge amount of time to report a successful match. Additionally, one of the things that is directly interconnected with the matching time is the number of quantifiers used in the RegEx (* and +), which are greedy operators, matching as much as they can. For instance, if a pattern like **r.*e** is going to search for patterns in the string "**regular expressions**", the reported match would be "**regular expre**" when "**re**" would suffice. This additional searching time does not play a big difference in applications that do not have to give results in real time or deal with low traffic rate. But when those matching are performed in real time systems, or those that deal with high traffic rates, this additional time can be crucial. Therefore, RegExes must be carefully written, in order to perform acceptable searching times.

The DPI system described on this thesis relies on RegExes to perform the packet classification. Therefore, for instance, it contains signatures like **http/(0\.9|1\.0|1\.1) [1-5][0-9][0-9] [\x09-\x0d -~]*** and **^ssh-[12]\.[0-9]**, which respectively represent the HTTP and SSH protocols. Later, this thesis shows that the

matching time can be reduced when those patterns are rewritten, eliminating some unnecessary quantifiers.

Table 2.1 – RegEx metacharacters

Metacharacter	Meaning	Example
.	Matches any character, excepts new line	A.B can match any string that has an A, another character after it and a B
|	OR operator, a pattern can match any of its side	AB|CD represents a RegEx that can match any substring AB or CD
()	Used to group a set of patterns inside a unique pattern	A(C|D)E can match ACE or ADE substrings
?	Denotes a matching of zero or one occurrence of the preceding pattern	ABC?D can match ABCD or ABD substrings
*****	Matches the preceding pattern zero or more times	AB*C represents a string that has an A, an undefined number of Bs and a C
+	Similar to *, but instead of denoting zero or more repetitions it denotes one or more repetitions	AB+C matches a substring with, at least, one B between A and C
{M,N}	Repetition of patterns	A{2,8} says that A will appear at least 2 times and at most 8 times
[ ]	Represents a class of characters	[ABC]D can match a substring that has an A, B or C before a D
^	Anchor. Says that the RegEx must start with the pattern that immediately follows it	^ABC denotes a string that must start with an A
[^]	Matches any characters, except those inside the bracket	[^A]BC matches any substring that has a character different than A before BC
$	End of string	B$ matches any string that ends with the character B

2.2 Deep Packet Inspection

Deep Packet Inspection (DPI) systems aim at analyzing, indentifying and classifying the traffic that is going through the network. Such systems accomplish these tasks by comparing messages within the packet's payload against strings, which represent signatures of applications, web attacks or some desired content to be scanned. Commonly, the used signatures rely their description on the use of

RegEx. This has improved their expressiveness and can represent a set of signatures in a single string. This type of classification, with comparisons performed using RegEx, is often seen as the most reliable of currently available techniques for traffic classification. One strong reason for that is that currently, traffic classification cannot rely on information given by the port-application tuple, because of a variety of applications that use well-known ports (e.g. port 80) to pass through firewalls, e.g. P2P applications. Hence, port classification can lead to misleading results.

Since DPI systems have good classification accuracy, ISPs have been recently relying on such products, because of its effective way of knowing what is going through their backbone. Throughout the global Internet, ISPs use DPI systems to perform different kinds of services, like:

1. Content based filtering and charging: ISPs can block, filter or even charge customers that are using certain kind of applications (maybe undesirable ones) within their network. Examples of these include Voice over IP (VoIP) and P2P applications.
2. Lawful Intercept: ISPs are commonly required by various governments to perform some kind of lawful intercept when needed by law enforcement agencies.
3. Quality of Service (QoS): Some applications like VoIP and Video over Demand (VoD) require low latency and higher priority against common traffic, such as web browsing. Knowing the traffic is the first step towards giving it higher QoS when desirable.

Also, commonly, Intrusion Detection Systems (IDS), like Snort [24] and Bro [25], rely on DPI to detect attacks from the Web. For that purpose they have a large database of signatures used to identify Buffer Overflow attacks, Malware, worms and viruses within the packet's payload.

With regard to the signature matching process, RegEx libraries are used to perform such task, by taking the signature, expressed by some RegEx, and building their state machines representations. After the state machine's creation, the packet's payload is analyzed by those machines searching for a successful match with the represented RegEx. Additionally, there are many commercial and non-commercial tools that use RegEx for traffic identification and classification, like Snort, Bro, L7-filter [10] and PacketLogic [20] (from Procera Networks), which are widely

used. The L7-filter has a wide and open RegEx signature database, which is used by the scientific community and identifies over a hundred of well-known applications.

DPI systems perform very heavy processing operations, having RegEx matching as the most expensive. Therefore, in order to achieve a high packet capture rate over high speed Internet links (e.g. 1Gbps), some optimizations must be made to avoid high packet loss rates. Such loss can be described as a classical Producer Consumer problem: The packet incoming rate is higher than the packet consumption rate (i.e. packet inspection), due to the heavy and resource consuming operations performed (e.g. RegEx matching). Consequently, the Operating System buffers rapidly become overwhelmed by packets and subsequent packets are then dropped.

2.2.1 TAM (Traffic Analyzer Module)

The DPI system described on this thesis, called TAM (acronym for Traffic Analyzer Module), uses RegExes signatures provided by L7-filter DPI system, non RegEx signatures from IPP2P project [9] and other signatures that were created by packet's payload inspection. Those signatures represent more than 60 applications falling into 9 different application's classes, which are listed at Table 2.2. Additionally, TAM was developed using the C language relying on the traditional libpcap library, for packet capture, and the C library *regex* for RegEx matching.

TAM has four modules, 1) the capture module, which is responsible for the online packet capture from an Ethernet device (Figure 2.1), 2) the aggregation module, which is responsible for aggregating the packets in flows (Figure 2.2), 3) the classification module, which compares the packet's payload (only packets that belongs to unclassified flows) with the RegExes signatures, aiming to classify that packet's flow (Figure 2.3) and 4) the cleanup module, which is responsible to go through every entry in the Hash Table and verify whether the flow has expired (Figure 2.4).

The Capture Module, presented in Figure 2.1, dequeues the IP packets from the capture buffer, checks the layer 4 protocol and forwards them to the Aggregation Module. Additionally, it waits until the other modules finish their operations to resume his activities, dequeuing the next packet.

Table 2.2 – Recognized Classes and Applications

Class	Application
P2P	eDonkey BitTorrent KaZaA Gnutella SoulSeek Ares Mute EarthStation Xdcc Direct Connect Waste GoBoogy Soribada WinMX Napster MP2P
Web	HTTP
Chat	AIM Yahoo Messenger MSN Messenger IRC
Network Management	DNS NetBios NBDS NBNS BootStrap
Streaming	Video Over HTTP RTSP Video Over HTTP (QuickTime) Audio Over HTTP
E-mail	IMAP POP3 SMTP
Data	FTP MySQL
VoIP	Skype
Secure	SSH SSL

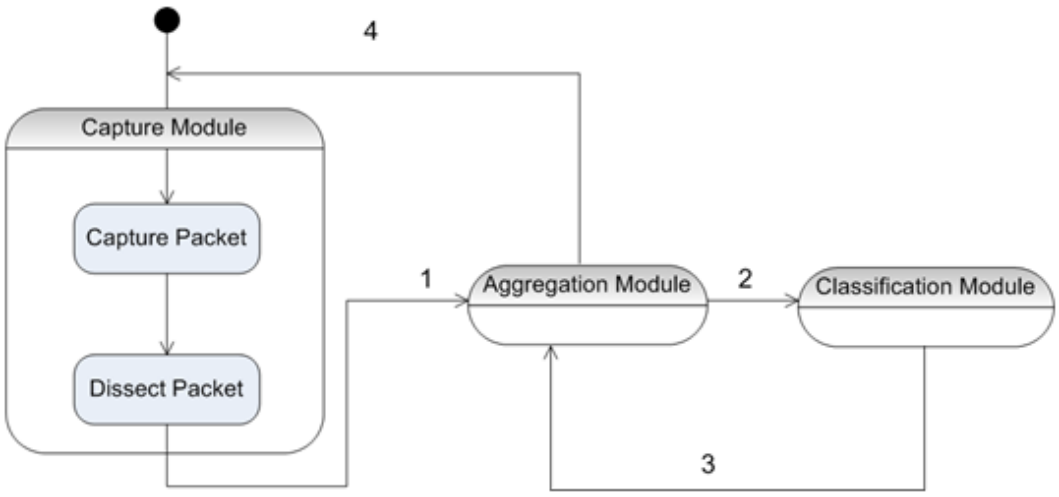

Figure 2.1 – TAM's Capture Module

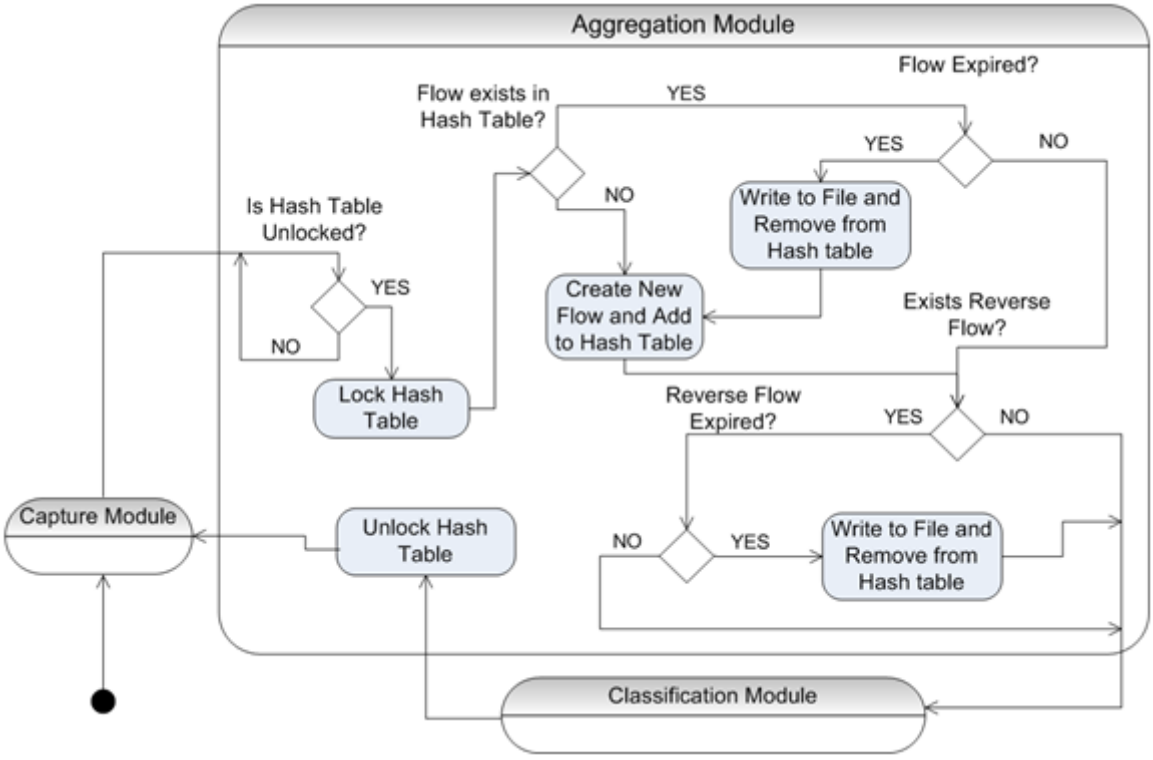

Figure 2.2 – TAM's Aggregation Module

As seen at Figure 2.2, the Aggregation module synchronizes the Hash Table with the analysis thread, by the locking operation. Then it checks whether the flow that represents the received packet exists in the Hash Table, creating a new one if it does not exist. If the flow exists and has expired, the module writes it to an output file and creates a new flow with the incoming packet. After that, the flow's reverse flow is sought in the Hash Table and the same expiration check is performed. Finally, those information, flow and reverse flow, are forwarded to the Classification Module.

The Classification Module, depicted at Figure 2.3, checks whether the flow is already classified. If its reverse flow exists and it was previously classified, then the flow that is being analyzed receives that classification, if it is unclassified. However, if the flow already has a classification, it only updates its information (byte volume, timestamp and number of frames). When the Classification Module finishes its execution, it returns to the Aggregation Module to unlock the Hash Table.

Finally, the Cleanup Module, depicted at Figure 4.2, synchronizes the Hash Table by the locking operation and checks each flow. If the flow has expired, it is removed from the Hash Table and written to an output file. Otherwise, the Cleanup Module retrieves the next flow from the Hash Table.

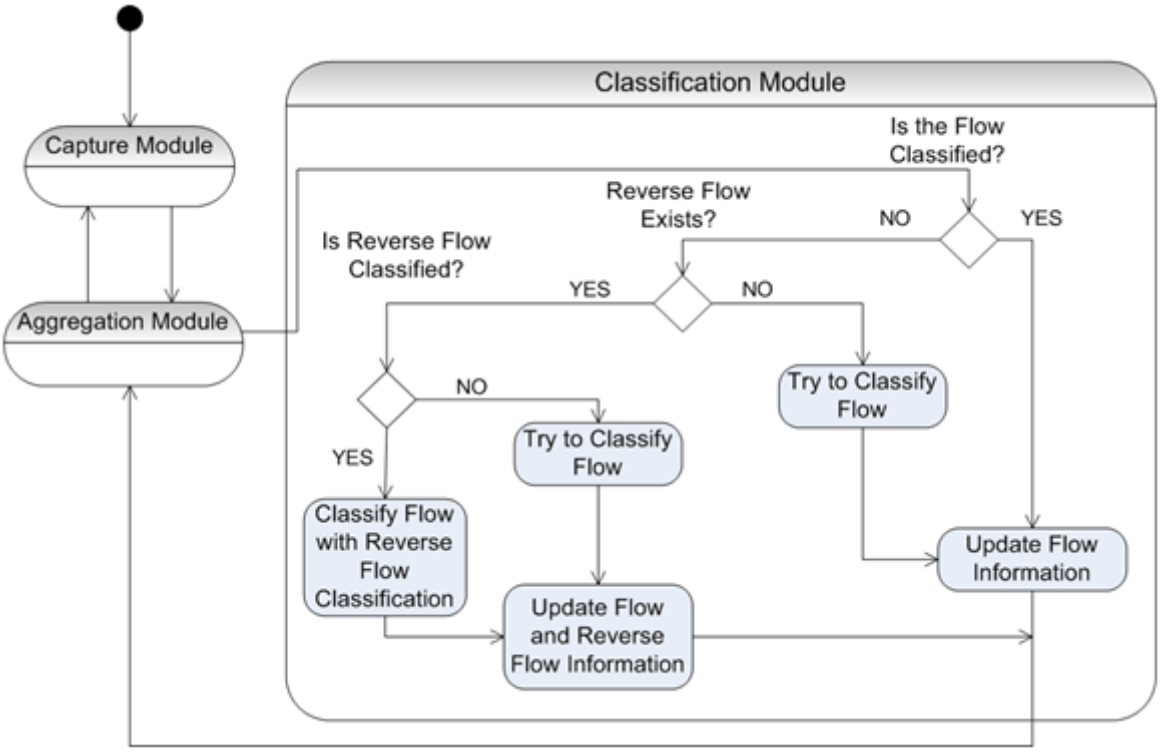

Figure 2.3 – TAM's Classification Module

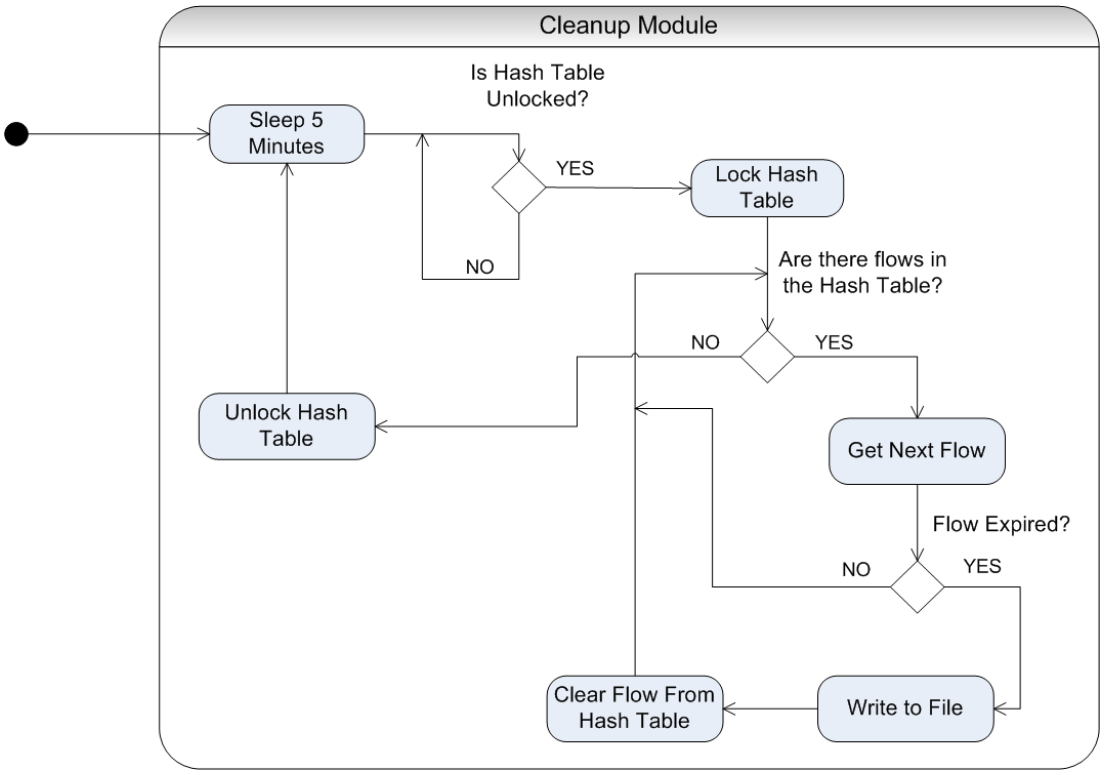

Figure 2.4 – TAM's Cleanup Module

The main objective of TAM is to capture all the packets that are traversing the network, classify those packets on the fly, aggregating them in flows with the lowest packet loss rate. Initially, TAM was developed to run in a 34Mbit/s backbone, at the Point of Presence of Pernambuco (PoP-PE), linked with the Point of Presence

of Rio de Janeiro (PoP-RJ), working with no loss of packets at this speed. But TAM's requirements have changed; now it has to run at high performance networks, reaching 1Gbit/s of speed. Now, TAM has to capture, aggregate and classify packets losing the less number of packets possible without decreases in classification completeness.

Those Flows, constantly mentioned in this thesis, consist of a unidirectional series of IP packets with the same source and destination addresses, port numbers and layer 3 protocol (this set of information is commonly called the flow's 5-tuple). Additionally, the aggregation in the format of flows can group all relevant information about a set of packets in a unique registry and, consequently, reduces the storage space occupied by the traffic related information.

In order to reduce memory consumption by the identified flows, TAM establishes a timeout of 64 seconds to write out the expired flows. So if a flow does not receive any packets in an interval of 64 seconds, its entry will be deleted from the hash table and it will be written to an output file. Therefore, there is a thread in TAM that is triggered in intervals of 5 minutes and it scans the whole hash table, searching for expired flows. This thread must be synchronized with the analysis thread, to handle some problems that are going to be described later on this section.

The present system faces some problems, which cause poor packet capture performance. One of the main problems is the packet capture model performed by libpcap. This poor performance can be explained by the time spent with unnecessary packet copies, performed by Linux-based systems, from kernel space to user space. Another problem is the general purpose Hash Table used by this version, which resolves collisions by chaining, i.e. each Hash Table bucket has a single linked list, so objects that have the same key are inserted on the list. Such Hash Table implementation has some bottlenecks, which were identified:

- With a single linked list, every removal operation, of an element X, has to perform a search operation in the table, in order to get the element that is placed before X in the list and makes it "next" pointer point to the element after X.
- Particularly in this Hash Table, when an insertion is going to be performed, it first checks whether the element, which is going to be inserted, exists in the table. Such check is unnecessary, since the element is searched at the

Hash Table before its insertion, so the information about its existence in the Hash Table is known beforehand.

Another problem, which really compromises the performance of TAM, is the time that the system will stay without analyzing any packets, because of the Hash Table cleanup thread. This occur because the cleanup operation has to be synchronized with the main analysis thread, since the cleanup module can be removing an expired flow from the hash table at the same time that the analysis thread is accessing such flow. Even this module resulting on some packet losses, this cleanup operation is really necessary because of some reasons:

- Memory consumption monitoring is an essential task in real time systems (so is in TAM) and therefore flushing out expired flows to an output file is very important, in order to avoid memory leaks.
- Performing a periodically Hash Table cleaning can lead to a shorter search time, since there will be less flows in the linked lists, leading to reduced flow searching time.

2.3 Packet Capture Mechanism

An explanation about the packet capture process and its details within a Linux-based system is important for a better understanding of the problems that are faced by a typical capture model. In the section that follows the packet journey inside the Linux Kernel is dissected. Additionally, this section describes the core structure for packet capture, namely Socket.

2.3.1 Packet Reception

Here, in order to understand how packet capture, using libpcap, and kernel sockets work on Linux-based systems, a brief description about the frame reception and processing inside the Kernel is given. The Figure 2.5 describes the frame reception flow inside the kernel and the pointed numbers during the text are referring to the steps depicted by the figure.

Whenever a packet arrives at the NIC, it is first copied into memory. It then triggers an interrupt, which will be handled by the NIC driver (1). Therefore, the driver disables future interrupts and allocates a `sk_buff` (short for socket buffer) structure, which is the packet representation inside the kernel, fetches the data from

the NIC buffer into the new `sk_buff`, in most cases via Direct Memory Access (DMA), and invokes `netif_rx`, which is the generic network reception handler.

The `netif_rx` (2) function puts a pointer to the new `sk_buff` inside the CPU's queue for incoming packets (there is one queue for each CPU), raises a software interrupt (`softirq`) and returns the congestion level of the queue to the caller. Please note that in the event that this queue is full, the incoming packet is then discarded. The function that processes the `softirqs` is known as `do_softirq` (3). It checks a bit mask and calls the appropriate handling routine, which represents the set bit. In this thesis, the interrupt of our interest is the `NET_RX_SOFTIRQ` (which is `softirq` raised when a packet has arrived) and the handle routine is the `net_rx_action` (4), which basically dequeues the first packet from the current CPU queue and calls the processing functions, present on two protocol handler lists.

Protocol handlers are functions which are registered into the kernel to handle specifics types of packets, such functions are registered in two lists: `ptype_all` and `ptype_base` (containing respectively protocol handlers for all types of packets and for specific protocols). The functions registered inside `ptype_all` list are those with the `ETH_P_ALL` flag (for generic packets), the ones at `ptype_base` are the others with `ETH_P_*` flags (e.g. `ETH_P_IP`, for IP packets). So, `net_rx_action` loops, until there are no packets to be dequeued or a threshold of maximum number of packets has been reached, calling each protocol handler registered, to take care of the incoming packet (`NET_RX_SOFTIRQ` is enabled again, if `net_rx_action` leaves a non empty queue of packets).

The protocol handler function responsible for handling IP packets is the `ip_rcv` (5) function. It performs all the integrity checks on the IP packet (checksum, header fields, etc.) and then, if the packet is considered correct, passes the packet to any Netfilter hook registered and finally calls `ip_rcv_finish`.

The `ip_rcv_finish` (6) function deals with packet routing. Within it, the `ip_route_input` function, decides whether the packet will be forwarded or locally delivered. Finally, the function `ip_local_deliver` is called.

`ip_local_deliver` (7) deals with IP defragmentation (reassembling the fragmented packets) and then, similarly to `ip_rcv`, it calls `ip_local_deliver_finish` (8), but before that it also calls Netfilter hooks. Next,

the ip_local_deliver_finish takes care of the tasks that are still pending for the packet to be passed to upper layers, additionally it also checks if there are any RAW Sockets listening, in order to forward the packets to them, by calling the raw_v4_input function.

Therefore, after ip_local_deliver_finish checks whether there are any raw socket opened, it calculates a hash function based on the protocol number present at the IP header, and gets back the layer 4 function that will handle the packet. Such functions are registered in a Hash Table, called inet_protos, which is filled when the Linux kernel is initialized, with the layer 4 protocol handlers. These handlers are called tcp_v4_rcv, udp_rcv, icmp_rcv and igmp_rcv, representing each of the layer 4 protocols. Those layer 4 functions should return a value indicating if an ICMP Destination Unreachable message has to be returned to the packet sender, which must occur if the layer 4 protocols do not find a socket opened that matches the incoming packet.

If the packet contains a TCP segment, then the tcp_v4_rcv (9) handler is the function that will be called. It performs some header integrity checks, searches whether there are any sockets listening for TCP packets and performs the TCP Finite State Machine processing. If a TCP socket was registered inside the kernel, then the filter expressions, registered with each socket, are evaluated, in order to decide whether the packet should be forwarded to a current socket or not. If the packet passes the filter then the tcp_v4_do_rcv function is called, otherwise a value indicating that an ICMP Destination Unreachable packet should be sent back.

After the filtering process, the corresponding function, which matches the current state of the TCP connection, is called. If the connection has been already established, the function tcp_rcv_established (11) is called, where acknowledgements mechanisms and header processing are going to be performed. Additionally, at tcp_rcv_established, the function data_ready (12) will be called, which will alert the current process, owning the socket, that a packet has arrived and is ready to be consumed.

On the other hand, if the packet contains a UDP segment, the process is much simpler. At udp_rcv (10), some header and integrity checks are performed and then the same tasks as in those for TCP handlers are performed including:

search for a UDP socket matching the current packet (if no socket is found the packet is discarded), filtering (`sk_filter` function) and finally calls `data_ready`.

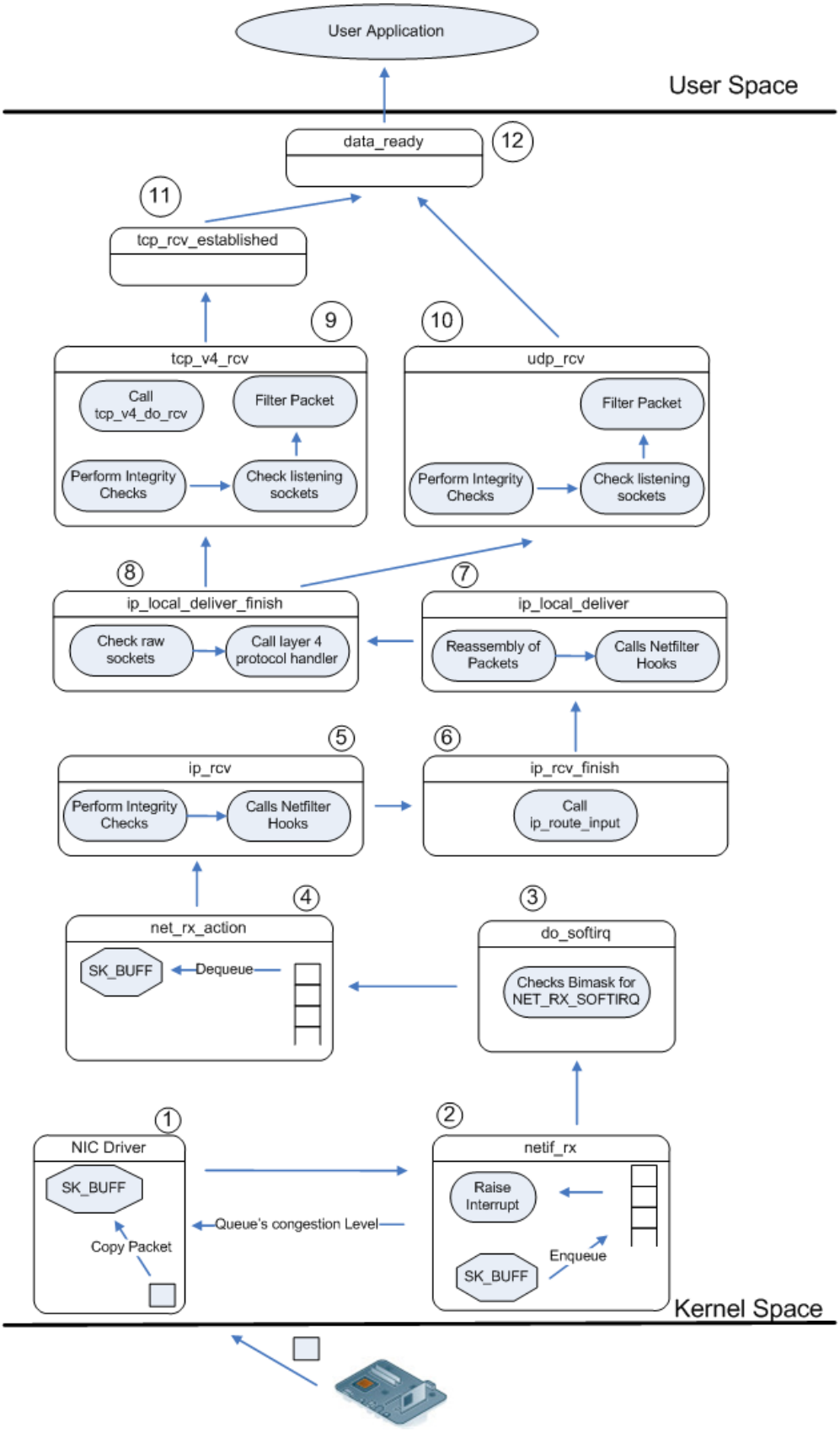

Figure 2.5 – Frame Reception on Linux

2.3.2 Sockets

When a programmer opens a socket (by invoking the `socket` system call) for packet reception, internally the kernel calls `__sock_create`, which will create a new a sock structure (the kernel representation of sockets), and registers a protocol handler for the specified protocol.

The socket function has the following interface: `socket(int family, int type, int protocol)`, where:

- `family`: Refers to the protocol family that the user will handle with the socket (e.g. `PF_INET` for sockets based on IPv4 protocol, `PF_PACKET` for sockets which want to communicate directly with the link layer)
- `type`: The type of the socket communication semantics, common types are:
    - `SOCK_STREAM`: Provides sequenced, reliable, two-way, connection-based byte streams (commonly used with TCP connections)
    - `SOCK_DGRAM`: Datagrams support (commonly used by UDP connections)
    - `SOCK_RAW`: Allow users to have access to lower-level communication protocols. The protocol stack processing stops at layer 3, the packet is not forwarded to layer 4 processing.
- `protocol`: The protocol of the packets that the socket will receive (e.g. `ETH_P_IP` for IP, `ETH_P_ALL` for all types of packets, etc.)

These sockets are basically represented as a queue which stores pointers to packets inside the kernel and each of these sockets is responsible for a class of packets. The `PF_PACKET`, for instance, handles all the packets that arrive at the Network Interface Card (NIC) and skips all the processing that is usually performed by the TCP/IP stack inside the kernel (depicted at Figure 2.5). On the other hand, the `PF_INET` socket only deals with IP packets, but there are other types of sockets that only deal with TCP, UDP packets, etc., at the Linux protocol stack.

When an application is capturing packets using libpcap library, it first registers a `PF_PACKET` kernel socket, inside the kernel to handle all packets that are arriving in the NIC. Therefore, whenever a packet arrives at the NIC, an interruption is triggered to call the corresponding kernel driver for that NIC. Next, the packet is

copied to the driver and a pointer to the packet is stored inside every socket's queue that matches the packet type, therefore the Linux Socket Filter (LSF) registered with such socket will filter the packet and decide whether it will be copied to user space or not. In summary, the capture process performs the following operations: packet copy from the NIC to the driver, pointer storage at PF_PACKET socket queue and another packet copy from kernel to user space.

3 RELATED WORK

A great deal of work has been done among the scientific community in order to achieve higher performance and accuracy in traffic analysis and classification. In [22] Schneider performs a deep analysis on how packet capture works on both BSD and Linux based systems. Additionally he points out some problems that exist in the present model of capture, such as packet copying operations.

Deri, in [5], proposes a new type of kernel socket, called PF_RING, which is able to improve dramatically the packet capture rate on Linux-based systems. He has patched the Linux kernel, in order to make available to users that new kind of socket. Using PF_RING sockets, the packet does not have to travel through all the network protocol stack processing, like it usually does, but instead a straight path from kernel to user space is taken. This new socket is based on a circular buffer, a memory region to handle incoming packets, which is shared between kernel and user spaces. Hence, user applications can directly access the packet that is queued in the ring, thus avoiding the expensive packet copy operation. He has evaluated his technique and compared it to other approaches for packet capture, like libpcap and Mmap libpcap [15] and showed that his solution outperforms them. Another important point in this work is that it also provides the user with a SDK, to build plugins for packet capture applications based on PF_RING.

After the PF_RING release, Deri in [6] extended his work and proposed nCap. This is seen as yet a further improvement to the packet capture rate in Linux-based systems. The author created a solution for packet capture and transmission at wire speed (bellow 1Gbps), creating a straight path from the NIC driver to user applications. When a network adapter is opened for packet capture it becomes a dedicated device for such task. Then the kernel no longer has control of it anymore letting only the application to transmit or receive packets from the adapter. Instead of creating one circular buffer, shared between kernel and user space (like PF_RING), nCap creates two: one buffer for incoming packets and another one for outgoing packets. These are directly managed by the network adapter, without any intervention from the kernel. So whenever a packet arrives at the network adapter, it copies the packet directly to the incoming buffer and alerts the kernel that a new

packet has arrived. His evaluation showed that this approach has better performance than PF_RING, and this was the case regardless of packet size. But, as nCap is at driver layer, it is bound to a specific kind of NIC, the Intel GE 32-bit.

Schneider et al. on [21] argue that with the current PC commodity architectures is impossible to reach 10Gbit/s capture rates, due to serious bottlenecks faced by the bus bandwidth and disk throughput. Nowadays the fastest bus bandwidth transfer that can be achieved with PCI-Express busses and it is around 8000 Mbytes/s, although the actual boards only use those busses for graphic cards. Also the actual disk throughput, for writing operations, is not enough to support a throughput from a 10Gbit links. Because of these problems, the authors, in order to handle 10Gbit/s traffic load, proposed to split the traffic among 10 machines, each one receiving traffic at most of 1Gbit. They built a testbed with a switch, to split all incoming traffic (at 10Gbit) among the machines and then tested it with different Operating Systems and processors. They have measured the packet capture rate among FreeBSD and Linux Operating Systems and AMD Opteron and Intel Xeon processors, having the combination of FreeBSD/AMD outperforming the others. Additionally they pointed out the time spent on copies made from kernel space to user space and emphasized the use of device polling for better performance.

Bernaille et al. [3] performed a study on how to classify a TCP flow with low CPU consumption. His approach consisted of analyzing the size of the first five packets of a TCP flow. He applied some techniques of machine learning, in order to build a training database of flow's behavior (based on the size of the first five packets), extracted from a packet trace file collected at a university's backbone. Therefore, he showed that by analyzing only the packet's size, a high packet capture rate can be achieved, since no expensive operations, such as RegEx matching. Although that approach has several drawbacks including: 1) the fact that it does not perform well for very short flows (with less than five packets), 2) it needs to track the reverse flow (sometimes the reverse flow does not follow the same path), so running this technique at an ISP router, for instance, over a link that sees only a single side of the conversation makes it ineffective, 3) packet's size is a not very precise classification criterion as different applications can have the same packet size distribution and 4) finally it is not applicable to UDP flows.

Sen et al. in [23], made a study on peer-to-peer (P2P) application classification, using signature matching and performing analysis on five widely known

P2P protocols: KaZaA, eDonkey, DirectConnect, Gnutella and BitTorrent. They used fixed string signatures of those protocols, mentioned previously, to evaluate the accuracy of those signatures. They were able to show that the packet examination is only needed on the first packets of a given flow (less than 10 packets), which led to less than 5% of false positive rate, in some protocols.

Regarding RegEx matching, Yu et al. [26] proposed a fast and memory-efficient solution to RegEx matching for payload scanning. The authors consider only DFA-based approaches because NFA-based approaches are inefficient on serial processors or processors with limited parallelism (e.g., multi-core CPUs in comparison to FPGAs). They focused on general-purpose processor-based architectures. In order to achieve good performance, the authors analyzed the computational and storage cost of building individual DFAs for matching RegEx, and identified the structural characteristics of the RegEx in network applications. Based on such analysis, they proposed two rewriting rules that can dramatically reduce the size of the resulting DFAs. Next, techniques to combine intelligently multiple DFAs into a small number of groups to improve the matching speed were developed. The rewritten rules reduced some DFAs by as much as 98% and 99% of their original size. On the other hand, the grouping algorithms generated DFAs 2 to 3 orders of magnitude faster than widely used NFA implementation and 1 to 3 orders of magnitude faster than a commonly used DFA-based parser.

In [12], Kumar et al. introduced a new representation for RegEx, called Delayed Input DFA (D^2FA). This modification in the original DFA formalism substantially reduces space requirements as compared to a DFA. The D^2FA is based on a technique used in the Aho-Corasick string matching algorithm [1]. The authors observed that, in the case of practical rule-sets from commonly used in network intrusion detection system, many groups of states share sets of outgoing transitions. Therefore, to explore the redundancy present in these DFAs, they introduced a special kind of transitions called default transitions. With these modifications when matching an input string, a default transition is used to determine the next state, whenever the current state has no outgoing edge labeled with the current input character. The main idea is to reduce the number of edges of a given automaton, but recognizing the same set of patterns. That approach can generally reduce the number of edges by more than 95% for the more complex DFAs that arise in network

applications, dramatically reducing the space needed to represent the DFA. They also created some heuristics to construct an efficient D^2FA.

In [14], Villa et al. proposed a string searching algorithm, based on the Aho-Corasick algorithm, to process separate chunks of the input text. They also used the multicore approach, IBM Cell Broadband Engine, in order to achieve a good throughput in string matching processing. Their approach consisted of breaking the input text in chunks, which would then be processed by different DFAs, achieving high byte throughput.

The authors in [7] have shown some ways to obtain considerable performance gains, when analyzing packet's trace files. Their techniques consisted in only analyze the first packets of a given flow and truncate the packet's payload, in order to reduce the size of the message which would be inspected by the DPI system. With the packet counting technique, they reported that by establishing 7 packets as a threshold would be sufficient to successfully classify a flow. Also it is worth telling that performing analysis only within the first 7 packets of a flow can dramatically reduce the processing time (almost 80%). Additionally, they have shown that analyzing only first 750 bytes of a packet's payload can lead to a good balance between performance and accuracy. In spite of the obtained results, they did not combine the two techniques, in order to evaluate how the classification behaves, and only performed offline analysis. This thesis combines the two techniques and checks their impact on classification. Also all experiments were done with online traffic, in different transmission rates.

4 PROJECT DESCRIPTION

The present thesis aims to improve the packet capture rate and speed of TAM (described at section 2.2.1). For such purpose, this thesis presents a new architecture for it, in order to achieve higher packet capture throughput. As a starting point, the original version of TAM relies on the libpcap library, for packet capture, a general purpose Hash Table (in order to aggregate the incoming packets in flow structures) and a set of RegEx, representing the applications previously presented at Table 2.2. Therefore its performance is going to be evaluated when it receives different loads of traffic. After that, comparisons are going to be made with the new version, in order to note how much performance was gained with the proposed architecture.

Both versions have the following modules:

- *Packet Capture Module*: Responsible for grabbing the packet that has arrived at the NIC and forward it to the Aggregation Module.
- *Aggregation Module*: This module will be responsible to search at the Hash Table for the packet's flow. The module will create the flow, if it does not exist yet. Additionally it will perform flow's timeout checks and search the reverse flow of the packet.
- *Classification Module*: This module will perform the application identification task, comparing the packet's payload against TAM's signature database.
- *Cleanup Module*: This module will be triggered for execution, on intervals of 5 minutes. It will go through every Hash Table entry, checking if there are expired flows, if so, such flows will be removed from the Hash Table and written to an output file.

Later, this thesis will show that Original TAM version has problems even when dealing with low loads of traffic, losing a great amount of packets. These problems are attributed to the packet capture mode, with libpcap, the general purpose Hash Table that it uses and the RegEx matching operations. Additionally this thesis will show how these problems were put aside by the proposed architecture and techniques.

4.1 Proposed Architecture

This section aims at explaining the New Architecture proposed for TAM. Since its requirements have changed, it has to deal with packet incoming rate, in high speed links.

4.1.1 Multicore Architecture

Nowadays, multicore architectures are widely deployed and used, among industry and end users. On the other hand, the previous version of TAM was relying on a sequential execution flow, not fully exploiting the available resources of the measurement machine. Additionally, when the cleanup module starts it execution, the capture, aggregation and classification modules are blocked, in order to handle synchronization problems. This is necessary to avoid scenarios where the cleanup module is deleting a flow from the Hash Table, at the same time that the classification module is trying to classify it. Therefore, when this module was executed, the capture module had to stop to dequeue packets from the capture buffer, dropping the packets that were arriving, due to overflow on such buffer.

In order to solve the presented problem, a new multi-threaded architecture was deployed which has got several threads effectively working in parallel. Each one of these threads will act as an independent TAM, having the first three modules described at section 4, namely Packet Capture, Aggregation and Classification modules. Each worker thread will have its own capture buffer using PF_RING and an ID number (starting from 0 for the first thread). Now, all the heavy work is distributed among several threads and there are several packet capture buffers.

Additionally several tests were performed, in order to discover the best thread's number threshold. After that, it was observed that this model performs better with 4 worker threads, which fits better with testbed's measurement machine processor (described at section 4.2).

4.1.2 Load Balance Module (LBM)

Even with a multicore approach, this new architecture still has some problems and it has to deal with two different levels of synchronization:

- Each thread must be locked when the cleanup module starts its operation.

- Only one thread would be working a time, since they must be synchronized with each other, in order to avoid different threads modifying the same flow in Hash Table.

Additionally, by opening one buffer for packet capture per worker thread, all those buffers would contain the same packets, since there is nothing responsible to distribute the incoming packets among threads. Therefore, a mechanism to detect that a packet is already being analyzed by another worker thread would have to be implemented. Also, even with the mentioned mechanism, the buffers would be full of packets that could be later dropped by the worker thread, because of other thread already analyzing it.

Since different threads can analyze different packets of the same flow, the Hash Table has to be shared among them. Consequently, the worker threads have to be synchronized, having only one thread accessing the Hash Table a time, leading to a sequential execution.

All those problems could be eliminated if the threads were independent, without necessity to share information among each other. This could be achieved if the flows were divided by threads, namely if different threads were not analyzing different packets of the same flow. Therefore, no thread synchronization would be necessary, since each thread would have their own set of flows, leading to full parallel execution. Also, each thread could have a private Hash Table, decreasing the time spent to search a flow.

This problem could be eliminated if each worker thread performed the following operations:

1) The worker thread dequeues a packet from its capture buffer
2) Checks whether it belongs to its set of flows or not, by applying some hash function
3) If does not belong, drop the packet, otherwise analyzes it

But this implementation would overload the threads' capture buffers with packets that would be dropped in the future, also dropping packets that belong to the thread's set of flows due to lack of space inside the its capture buffer.

For such purpose a Load Balance Module (LBM) was integrated in this new architecture, which was implemented as a Linux Kernel Module. This module is responsible to distribute the packet load among the worker threads, in a way that

different threads will not have packets of the same flow, avoiding many synchronization problems. This module will intercept all packets that are arriving at the NIC, before their entrance on the capture buffers, filling the threads' buffers only with packets that belong to it. With LBM this new TAM can have a private Hash Table for each thread, since each thread will always have its own set of flows, consequently it does not have to share flow's information among the other threads.

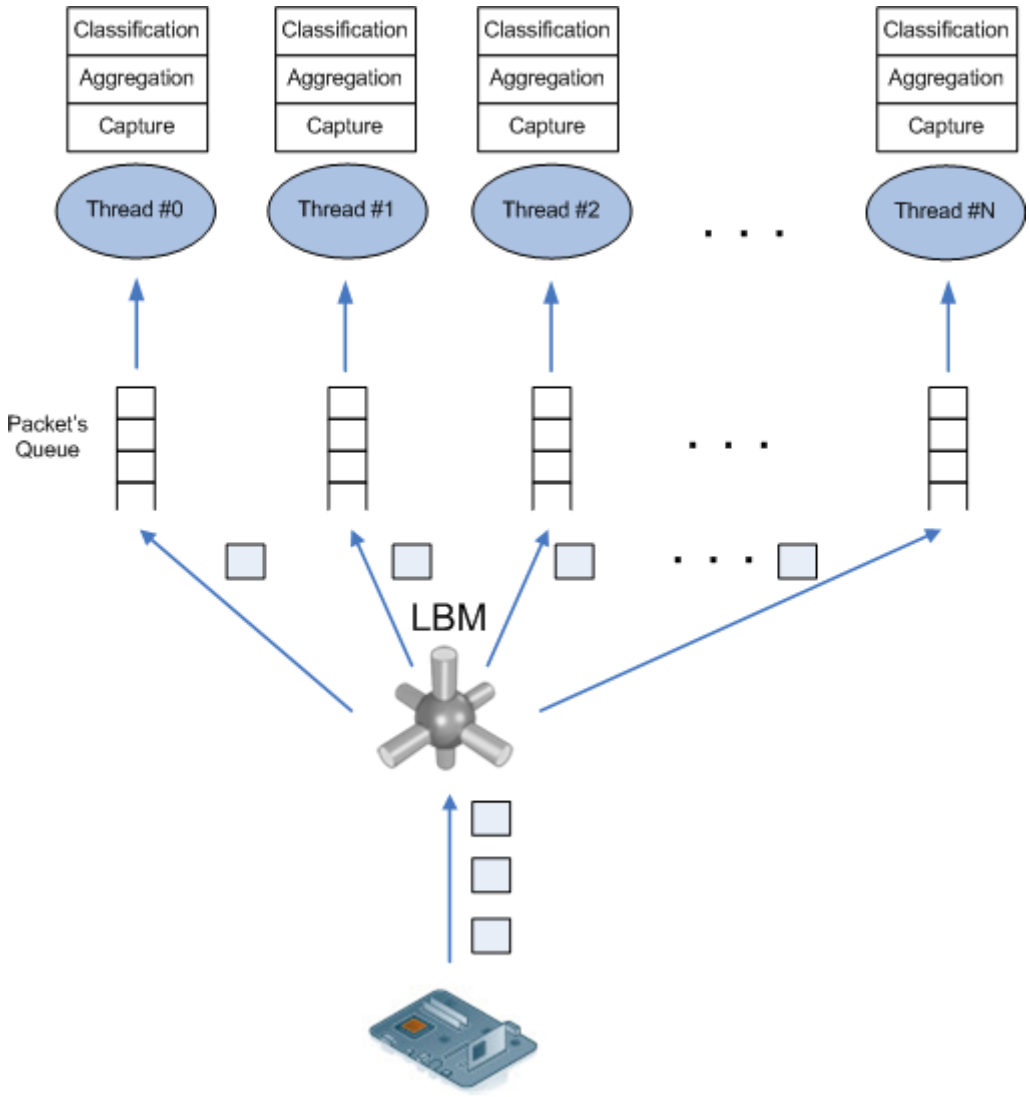

Figure 4.1 – New Architecture

The key point of the LBM is a simple and effective hash function that is calculated whenever a packet arrives at NIC. Such function is described at Equation 1.

Equation 1 - Hash Function

$$threadID = (IPSrc + IPDst + layer4SrcPort + layer4DstPort + layer3Protocol) \ \boldsymbol{mod} \ numberOfThreads$$

The presented function sums up the following fields, present in the IP packet: *IPSrc* and *IPDst*, representing, respectively source and destination IP addresses; *layer4SrcPort* and *layer4DstPort* representing, respectively source and destination ports, present on the layer 4 protocol (TAM only deals with TCP, UDP and ICMP protocols) and *layer3Protocol*, which represents the protocol number that is encapsulated by IP protocol. Therefore, the equation gets that result and makes a module operation with the number of worker threads, resulting in the thread ID representing the thread that will receive the packet. With this simple hashing function LBM can guarantee that a given flow, and its reverse, will be always forwarded to the same thread. Also, now it is possible for each thread to have its own Hash Table, thus reducing the time spent with flow search. This New Architecture is depicted in Figure 4.1.

4.1.3 Unique Cleanup Module

As seen previously at section 2.2.1, the original TAM version has undesirable packet losses, because of the analysis thread being blocked by the cleanup module. Since this New Architecture now has a private Hash Table per thread, it can have an important gain with a unique cleanup module, which is depicted at Figure 4.2.

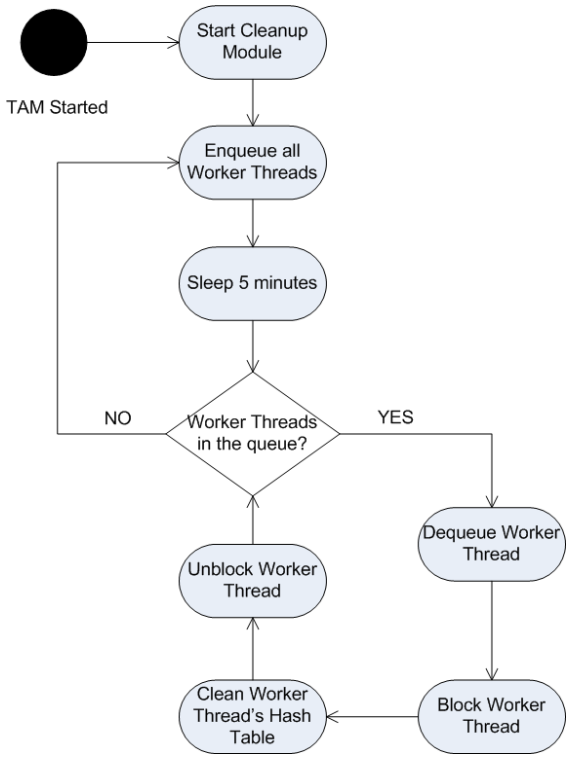

Figure 4.2 – Unique Cleanup Module

Since the cleanup module will only clean one Hash Table a time, the New Architecture still have N-1 threads fully working in parallel. Therefore the whole system would not be blocked, as it occurred in the previous version, thus increasing the packet capture rate. However, if a cleanup module was implemented for each thread, there would a probability that the system could have all cleanup modules running at the same time, thus having the same problem faced by the Original TAM version.

4.1.4 Additional Optimizations

It is widely known that the most expensive operation in DPI systems is the RegEx matching process, so the modifications proposed will not suffice when the system is dealing with very high speed links (e.g. 1Gbit/s). Therefore some other optimizations are proposed, which are going to be integrated to this New Architecture.

RegEx matching performance is directly interconnected with the size of the generated state machine, representing the expression. For instance, if a RegEx is full of wildcards (*) its state machine's size can become exponential, which dramatically decreases RegEx matching performance. In order to reduce such latency, some RegEx signatures are going to be rewritten, aiming at the reduction of the size of the generated state machine, but without loss of precision (i.e. the rewritten patterns continue matching the strings that were matched by the original patterns).

The authors in [7] have shown that DPI systems can successfully classify a flow by only analyzing its first packets, having 7 as the best threshold. They have also have shown that only a fraction of the payload is needed, in order to successfully classify a flow. Additionally, with those modifications, they have achieved a considerable gain in the time spent to analyze the packets. Therefore, adding such techniques to the New Architecture is believed to dramatically reduce the packet loss rate, when dealing with high throughputs. At first, performance gains obtained when applying the packet counting technique, using 7 packets as the threshold, are analyzed. Next, the New Architecture truncates the packet's payload, using only its first 750 bytes. Finally the New Architecture mixes those techniques and the obtained gains are going to be evaluated.

4.2 Methodology

In order to evaluate the packet capture rate in a scenario, in which different loads of traffic are being generated to a measurement machine, a testbed described in Figure 4.3 was built.

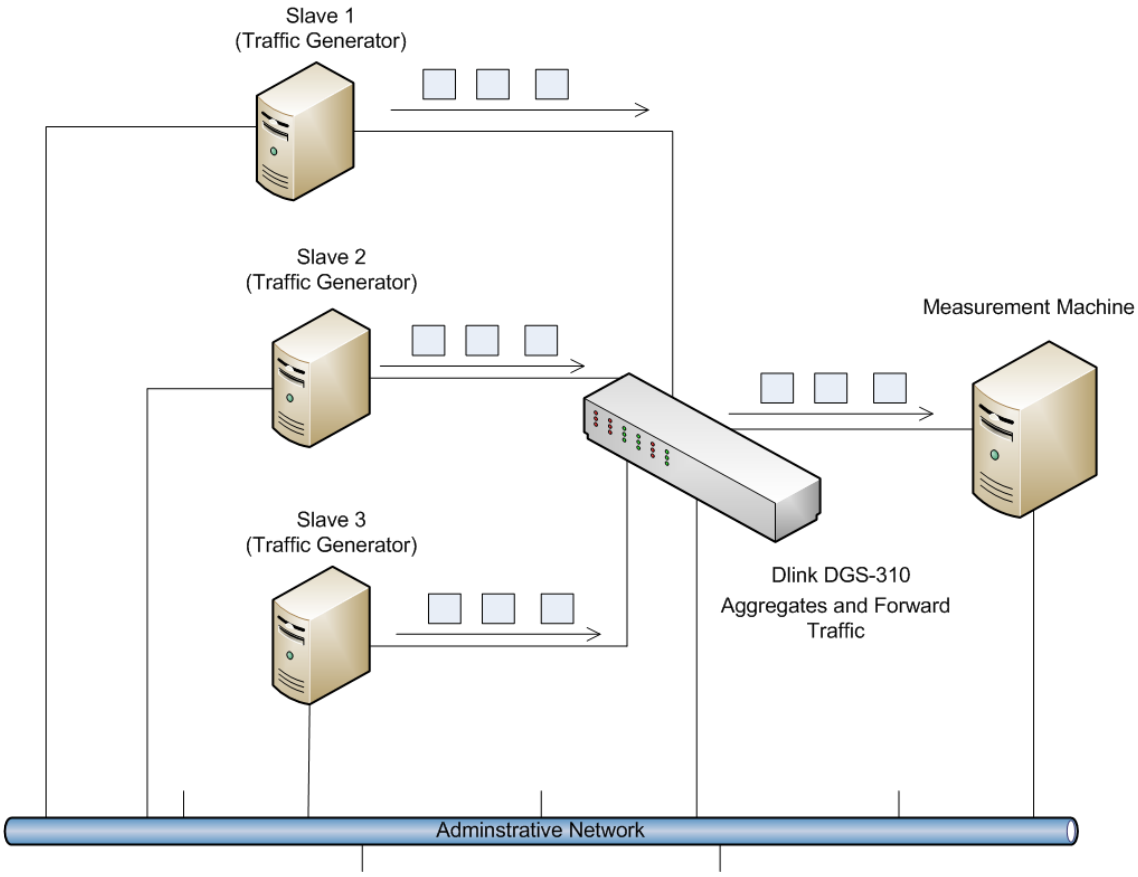

Figure 4.3 – Testbed Architecture

The testbed is composed with the following machinery:

- **Measurement Machine (M)**: This machine receives packets from the switch, which aggregates traffic generated by all slave machines. Is at this machine that the DPI software runs and analyzes the traffic.
- **Slaves Machines (S1, S2, and S3)**: These machines will generate traffic, at different rates, to the measurement machine.
- **Dlink Switch (SW)**: In order to aggregate the traffic generated from the three slave machines, a Gigabit switch was placed in the path between the slaves and the measurement machine. The switch will receive traffic from the slaves, on a separate port for each and then aggregate that traffic and forward it to a Gigabit port, which is connected to the measurement machine**.**

Table 4.1 – Testbed Configuration

Machine	Processor	RAM Memory	Administrative NIC	Traffic Generator/Receiver NIC	HD	Operating System
M	Intel Xeon X3210 Quad-core	4GB DDR	Onboard Gigabit	Offboard, 3Com Gigabit	3x 500GB Sata HDs	Linux, 2.6
S1	Intel Xeon 5110 Dual-core	2GB DDR	Onboard Gigabit	Offboard, 3Com Gigabit	1x 250GB Sata HD	Linux, 2.6
S2	AMD Athlon 64x2 Dual-core	1GB DDR	Offboard 10/100Mbit	Onboard, nVidia Gigabit	1x 300GB Sata HD	Linux, 2.6
S3	AMD Athlon 64x2 Dual-core	1GB DDR	Onboard Gigabit	Offboard , Intel Gigabit	1x 300GB Sata HD	Linux, 2.6

The rate of the traffic generated by the slave machines was varied, in order to evaluate how much traffic each TAM version can handle, evaluating the packet loss rate. The starting point was 100Mbit/s, after that the traffic rate generation was increased by a factor of 100, up to 900Mbit/s. Therefore, the performance of the New Architecture is evaluated and compared against the old one, when dealing with different rates. Summarizing, this work aims at showing how subtle modifications can lead to important improvements without appealing to dedicated hardware solutions, using only commodity ones.

It is worth telling that all experiments were performed using real packet-level traces collected at one of the largest ISPs in Brazil. A router port was mirrored, in order to not interfere in the normal transit traffic, which consists of traffic from/to around 50.000 ADSL subscribers. To make the collected data more representative of the traffic diversity, the network was sniffed for several days, accumulating almost 6TB of real Internet traffic in different periods of the day. Therefore, a representative sample from this collection was selected, to be replayed to the measurement machine, such replay was done by the tcpreplay[1] tool running in all slave machines.

The following metrics are considered:

1) *Packet Loss Rate*

   This is the most important metric, since packet losses are the main problem faced by DPI systems and they directly impact the classification completeness. Therefore, the New Architecture is evaluated when receiving different traffic rates and compared with the Original TAM version.

2) *Classification Completeness*

[1] http://tcpreplay.synfin.net/trac/

There is no sense in making a DPI system that loses no packets, by applying different techniques, but does not have good classification completeness. Therefore, the techniques that can impact on the classification completeness, namely packet counting, payload truncating and pattern rewriting, are going to be evaluated in other to measure their impact in classification.

5 EVALUATION AND RESULTS

In this section, the results of the evaluations performed on each version, are shown. The factor that is going to be considered in the evaluation is the packet incoming rate. This factor will assume different levels, starting at 100Mbit/s up to 900Mbit/s, varying by 100Mbit/s.

5.1 Original TAM

As mentioned previously, this version is the baseline, i.e. the performance gains obtained are compared against it.

At first, it is shown how TAM performs on the lowest incoming rate (100Mbit/s), at the graph in Figure 5.1. To build this graph, time bins of one minute were established, then the packets that have arrived at this time bin and also how much of those packets were dropped during the bin, were computed.

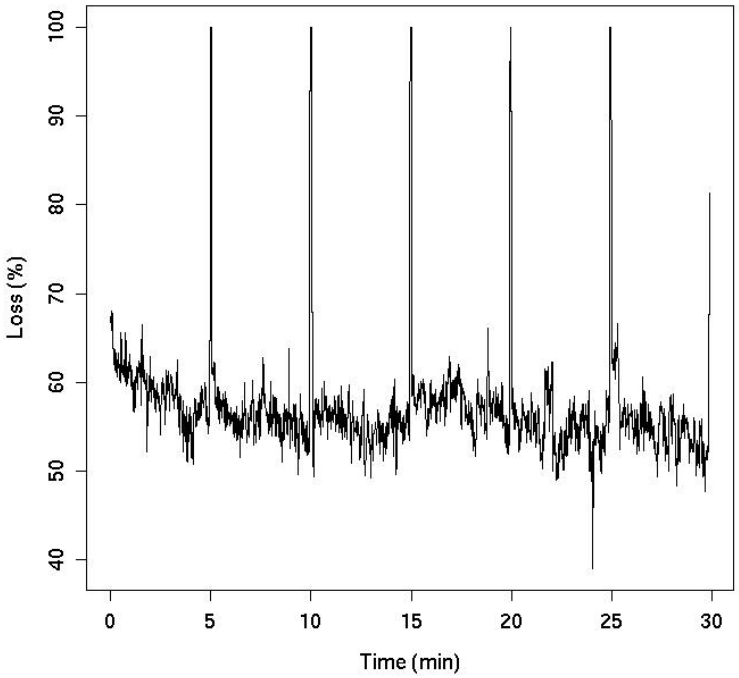

Figure 5.1 – Packet Loss, Original TAM 100Mbit/s

As presented at the graph, even with the lowest packet incoming rate, the original TAM has serious losses of packets, reaching almost 60%. The graph also shows the expected behavior when the cleanup thread is triggered.

The cleanup operation is very expensive, since it blocks the analysis module in order to remove expired flows from the Hash Table. This block is caused by the synchronization necessary between the two modules, assuring that each

thread will access its critical region (the Hash Table) alone. Therefore there will be a short period of time that no packets are going to be dequeued from the packet's queue, leading to packet losses, since that the packet transmission will not stop. This behavior is depicted at Figure 5.1, which is represented by the loss peaks every 5 minutes, reaching 100% of packet loss at the cleanup period.

Additionally, as expected, the packet loss keeps growing as the incoming rate grows, losing almost all packets when it reaches 900Mbit/s. Such poor performance is caused by 4 key points, namely RegEx matching, the cleanup operation, Hash Table operations and libpcap.

Another problem of this version is that it uses the traditional libpcap library for packet capture, so it has got the additional packet copy operation (from kernel space to user space), which can lead to some degradation in the packet capture mechanism and consequently increase the packet loss rate.

Finally, RegEx matching operation is the main bottleneck of TAM, being responsible for about 90% of all processing time. Additionally there are some RegExes that generate huge automata, impacting directly in the matching time spent by the TAM's classification module.

Aiming to enhance the performance of TAM, each problem described above was attacked, in order to have gains in all levels.

5.2 New Architecture Results

5.2.1 Hash Table Optimization

The results, of the Original TAM version, confirmed that its architecture has a high packet loss rate, getting even higher as the packet speed grows. Such losses are mainly generated by the RegEx matching, which is very expensive and the packet copies operations performed by libpcap, but the general purpose Hash Table has its contribution. Previously, at section 2.2.1, the unnecessary table lookup operations, in the insertion and removal tasks performed by the general purpose Hash Table, were pointed. Therefore, as the number of flows grows these additional table lookups become more and more expensive.

For that reason, aiming at confirming the expensiveness of these operations, a code profiling in the Original TAM was done, and the result is described in Table 5.1. Additionally, in order to analyze only the Hash Table operations, the

classification module was removed in this experiment. This table just proves that those operations are contributing to packet losses, since they consume additional CPU cycles, which could be wasted with packet handling.

Table 5.1 – Code Profiling result, Old Hash Table

Function	Duration per call (milliseconds)	% total time spent in TAM
Insertion	0.00044	1.9
Removal	0.00030	1.1
Cleanup Module	1826	3

At a glance, the time spent in insertion and removal operations do not seem to be expensive, within TAM. But they are responsible for 1.9% and 1.1% of all execution time of TAM. Nevertheless, the function that deserves more attention is the Cleanup Module, spending almost 2 seconds in each execution, namely almost 2 seconds with no packets withdraw from the capture buffer, which is unacceptable to a system that must handle a 1Gbit/s incoming rate.

To reduce such overhead a new Hash Table was built from scratch, also resolving collision by chaining, but having a double linked list instead single linked one. Additionally, there is no more verification of whether the element which is going to be inserted exists in the table. Therefore, with this new structure the time complexity of *O(1)* (constant) is now guaranteed, for the removal and insertion tasks, since insertion and removal will only consist of moving pointers operations.

In order to only evaluate and compare the performance of both Hash Tables, the classification module was removed from TAM and then two versions, one using the old Hash Table and the other using the new one, were evaluated with the highest incoming rate, i.e. 900Mbit/s, in order to perform a stress test. The packet loss rate was evaluated, when there are only operations of capture and flow aggregation in TAM. The best Hash Table would be the one that has less packet loss. The results are presented in Figure 5.2.

As seen in Figure 5.2, the loss rate of the New Hash Table is slightly smaller than the losses of the old one, reaching around 8% with the new Hash Table, and 10% with the old one. However, this difference is more representative when the results present the number of packets received and lost, by each version.

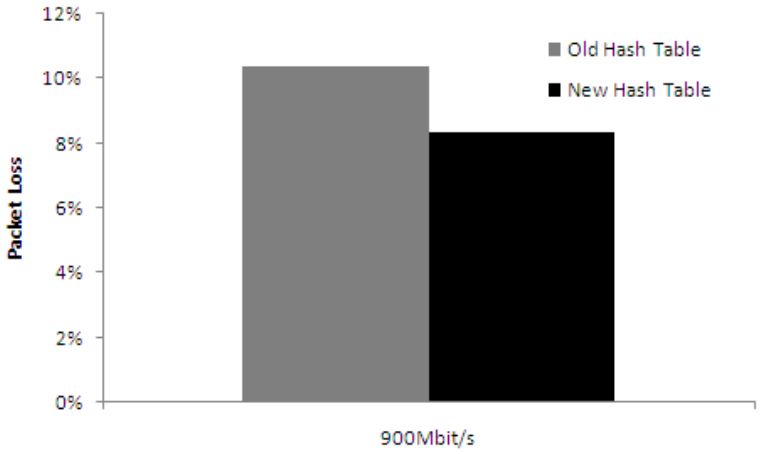

Figure 5.2 – Hash Tables' Packet Losses

As seen on Table 5.2, the new Hash Table drops much lesser packets than the old one, having over 7.5 million packets of difference.

Table 5.2 – Received/Dropped Packets by each version

Hash Table	Received Packets	Dropped Packets
New	395336588	33704370
Old	397342393	41273020

These gains in the packet capture rate were obtained due to the time reduction suffered in the Hash Table operations, mainly because of the Cleanup Module. This critical module has suffered a reduction in its execution time of more than 70%. Now, none of the functions is responsible for more than 1.8% of all TAM's execution time. These results are shown in Table 5.3.

Table 5.3 – Code Profiling result, New Hash Table

Function	Duration per call (milliseconds)	% total time spent in TAM
Insertion	0.00010	0.3
Removal	0.00012	0.4
Cleanup Module	506	1.8

5.2.2 Optimized TAM Architecture

In this section, the results obtained with the New Architecture are going to be compared and some comments will be made about some performance gains obtained. It is worth telling that all the following results, in the next sections, were achieved with the classification module activated.

This version has changed the packet capture mechanism, from libpcap to PF_RING (described on section 3). Now, with the shared circular buffer between

kernel and user space, unnecessary packet copies are avoided, increasing the packet throughput. Additionally, the user applications can access the packet directly from the buffer, avoiding expensive system calls (which are performed in the classical packet capture model with sockets). Therefore, with these new features provided by PF_RING, the packet reception speed was improved.

Also, the New Architecture is using the new deployed Hash Table, which has shown better performance, when compared with the old one at section 5.2.1. With this new Hash Table, a considerable processing time gain was obtained, since that, now, the insertion and removal tasks operate in constant time (*O(1)*).

Figure 5.3 shows that the New Architecture absolutely outperforms the original one in a incoming rate of 100Mbit/s, having only few peaks at the beginning of the analysis. Since that, at the application startup almost every packet that arrives represents a new flow, therefore it tries to classify almost every packet, leading to such loss peaks. However, when those flows are becoming classified by the application and the rate of new flows starts to reduce, it does not lose packets anymore. However its final loss rate is of 0.54%, what is almost imperceptible.

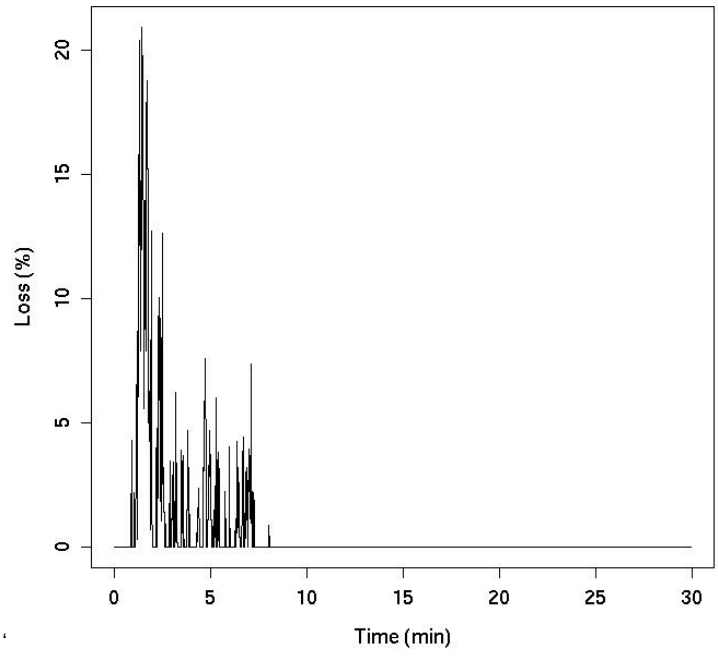

Figure 5.3 – Packet Loss, New Architecture 100Mbit/s

When the packet incoming rate is increased, the new version starts to lose a considerable amount of packets, but it stills losing much less than the Original TAM. Nevertheless, at the comparison made between the two versions, at Figure 5.4, is noted that the New Architecture dramatically improves the packet capture rate up to 500Mbit/s. Even though, with incoming rates greater than 500Mbit/s, the gain is

not too considerable, reducing from 93% to 86.48%, 94.71% to 90%, 94.90% to 91.60%, 96.29% to 92.09% and from 96.57% to 93.03, representing respectively 500, 600, 700, 800 and 900Mbit/s.

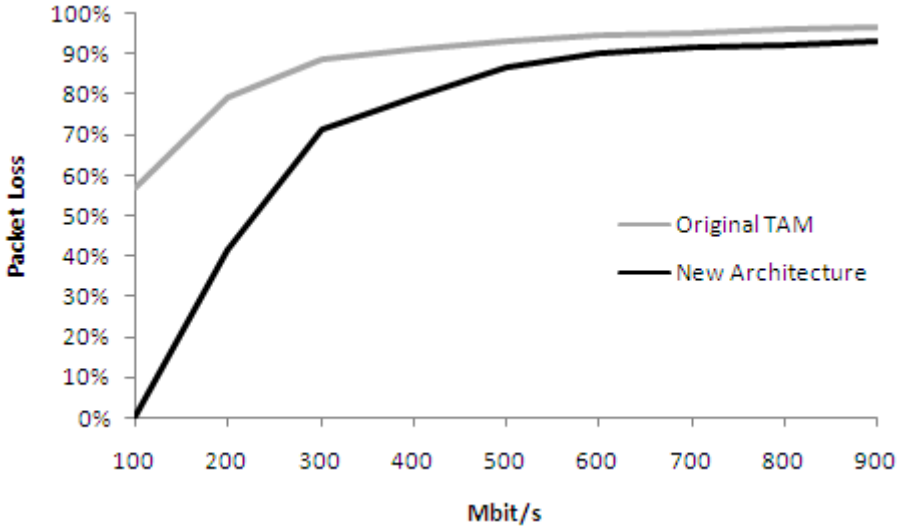

Figure 5.4 – Packet Loss, Original TAM vs. New Architecture

All those performance gains are attributed to the optimizations that were made in different levels of TAM: the Hash Table, packet capture and synchronization optimizations.

One of the key optimizations was the modification of the capture and analysis module. Therefore, the new multicore architecture, distributing the capture and analysis work among several threads has mitigated this problem. But only this would not suffice, since all worker threads would have to be synchronized, in order to not modify the same flow at the same time, since there can be different threads dealing with the same flow. Therefore LBM was built, eliminating such problem, since it will guarantee that there will exist a N:1 relation between flows and threads. Thus, the worker threads will effectively work in parallel. Additionally, the single cleanup module can guarantee that, even when it starts its operations, there will be, at least, N-1 threads working.

Also, it is worth telling that some gains in the packet capture rate were obtained due to the optimizations performed within the Hash Table, eliminating the additional operations that were performed with the insertion and removal tasks (section 5.2.1).

As a final evaluation, the hash function, used to distribute the load between threads, was tested in order to prove that it equally distributes packets between threads. If this function did not make an equal load balance, there would be

an overloaded thread, causing high losses of packets. To evaluate this possibility, some tests were performed in order to guarantee that the LBM almost equally distribute packets among the worker threads. The results of such tests are presented at Figure 5.5.

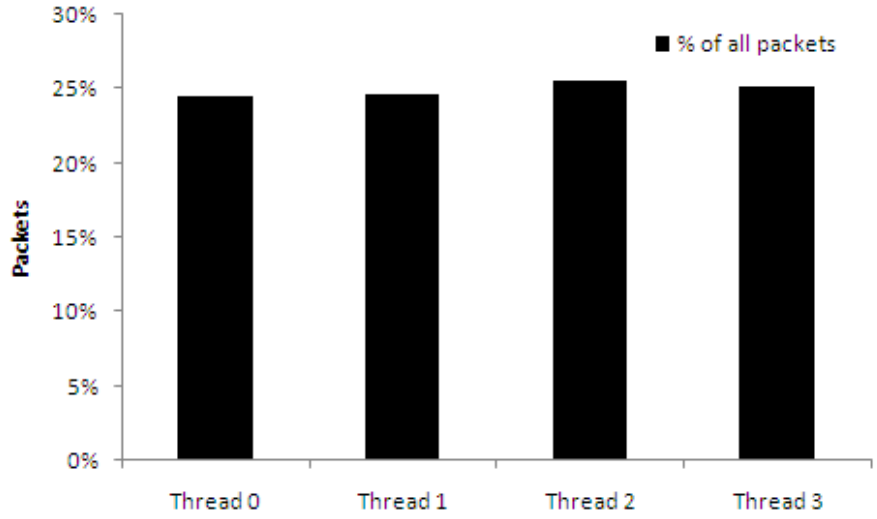

Figure 5.5 – Packet Distribution among Threads

5.2.3 Packet Counting Evaluation

Additionally, as said on previous sections, the New Architecture will be evaluated with the packet counting technique, inspecting only the first 7 packets of a given flow, analyzing its behavior. Therefore the New Architecture was modified, in order to inspect only the first 7 packets of a given flow. The evaluation results are presented at Figure 5.6.

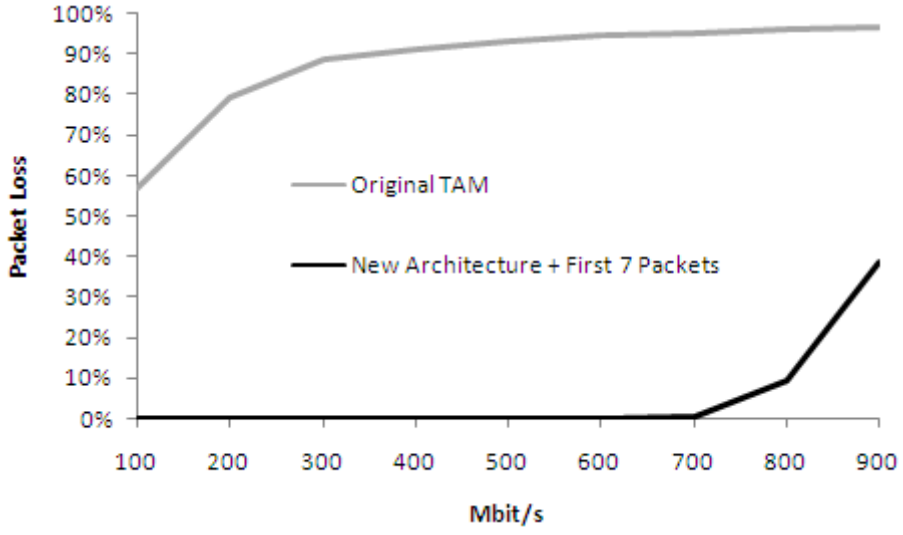

Figure 5.6 – Packet Loss, Original TAM vs. New Architecture (first 7 packets)

The results show that the packet loss rate can be dramatically reduced when the RegEx matching is performed in, at most, the first 7 packets of a given flow,

having only 38.58% as the highest packet loss rate, against 96.57% of the version that analyzes all packets, until classify the flow. Even not analyzing all packets of a flow, the classification completeness does not suffer decreases. Instead, the graph shown at Figure 5.7 shows that this approach was able to classify much more flows then the version that analyzes all packets.

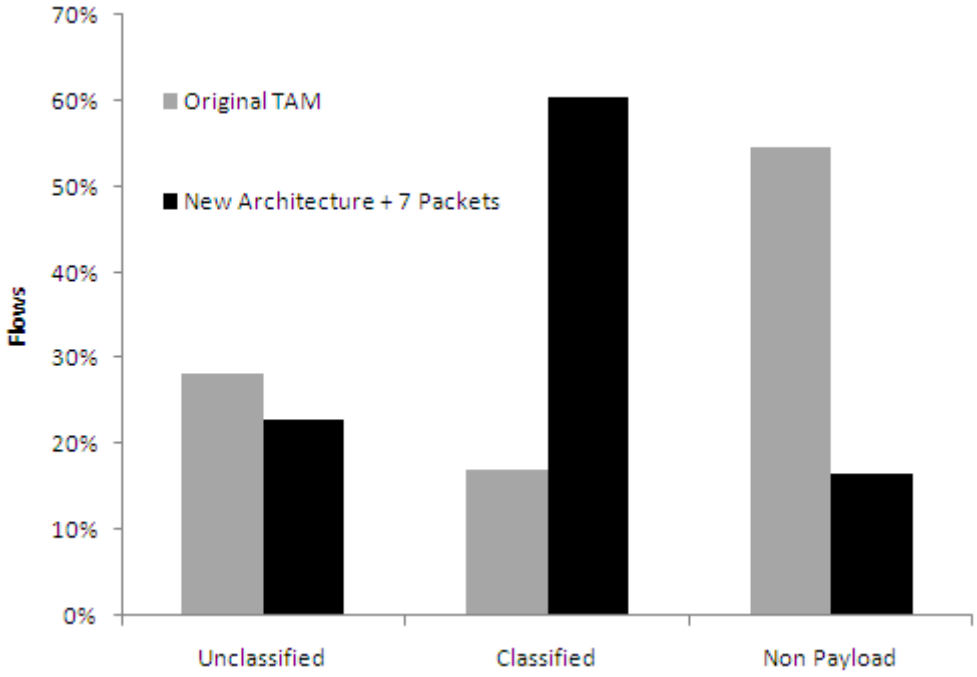

Figure 5.7 – Classification Completeness, first 7 packets analysis – 700Mbit/s (Flows)

As depicted at Figure 5.7, the version that analyzes all packets from a given flow, was not able to classify only around 30% of all flows, while the other version, analyzing only the first 7 packets, is not able to classify around 23%. This difference is not too significant, but it is composed of flows with huge traffic volume (shown at this section on the volume analysis). Another point that must be noticed is the number of Non Payload flows, which is around 54% on the version that analyzes all packets, against 16% on the other version. The high number of Unknown flows occurs because the version that analyzes all packets has a high packet loss rate, reaching 94.9% at 700Mbit/s, so there is a high probability that those lost packets could contain payload the signature of an application recognized by TAM, leading to high number of Unknown flows. The same behavior is applied to the Non Payload flows, since the lost packets can be the ones that were carrying payload, leading to a high number of Non Payload flows. Therefore, since the traffic that is replayed to both versions are the same, these Non Payload flows are misclassified.

On the other hand, the version that analyzes only the first 7 packets of a given flow has a lower number of Unknown and Non Payload flows, since it captures more packets. So there is a higher probability for this version to catch the flow's packets that have payload and some recognized signature. Additionally, those gains obtained, in the classification completeness, are more significant than it appears to be, since only 1% represents more than 130000 flows. The Table 5.4 shows that, due to less packet loss, the number of flows and byte volume captured by the New Architecture, analyzing only the first 7 packets, is incredibly higher than the flows and byte volume captured by the Original TAM.

Table 5.4 – Analyzed traffic statistics (700 Mbit/s)

Version	Number of Flows Captured	Volume Captured (GB)
Analyzing all packets	3222939	6.56
Analyzing first 7 packets	13149126	136.86

This difference in classification gets even higher when the classified volume is considered, instead of flows, as seen in Figure 5.8. The New Architecture, analyzing only the first 7 packets, is able to classify almost 77% of all analyzed traffic volume, against 25% on the Original TAM.

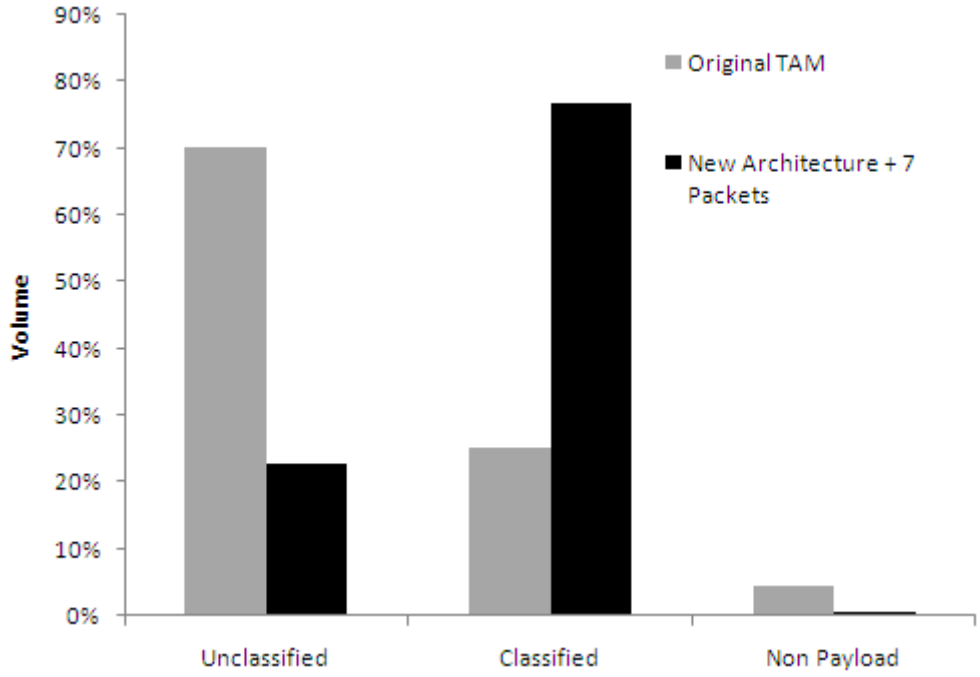

Figure 5.8 – Classification Completeness first 7 packets analysis – 700Mbit/s (Bytes)

The volume of unclassified traffic volume is incredibly high in the Original TAM, reaching 70.34% against 22.69%, in the New Architecture analyzing only the first 7 packets of a flow. Also, the volume of Non Payload traffic represents 4.47%, in

the Original TAM version, against 0.65%, in the new version. These results are caused because of elephant flows (with huge byte volume) that could not be classified due to several packet losses in Original TAM (94.90% in 700Mbit/s) and now can be classified with the New Architecture. Additionally these elephant flows are basically composed by huge data transfers (P2P, WEB and Streaming Flows), which have their protocol signature present in few packets, most at the beginning of the flow, probably lost by the Original TAM.

5.2.4 Payload Truncating Evaluation

Now the results obtained with the New Architecture analyzing only the first 750 payload bytes are going to be shown at Figure 5.9.

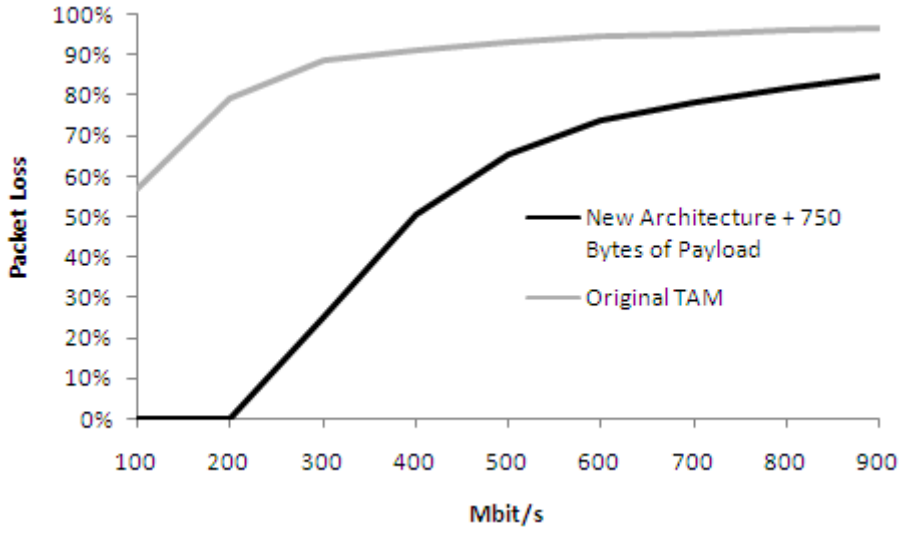

Figure 5.9 – Packet Loss, Original TAM vs. New Architecture (750 bytes of payload)

As shown in Figure 5.9, the packet loss rate had a considerable reduction when the payload truncating technique was applied, but not too noticeable as the results shown with the packet counting technique, presented at Figure 5.6. It happens because with the payload truncating technique it is still losing a great amount of packets when the incoming rate gets over 300Mbit/s. However, it starts losing no packets at the 100Mbit/s and 200Mbit/s rates, against 56.94% and 79.11%, respectively, at the Original TAM. Additionally the packet losses at the highest rate (900Mbit/s) have suffered a considerable reduction, from 96.57% to 84.72%.

In addition, a new evaluation was proposed, in order to reduce even more the packet loss rate. Such evaluation consisted on mixing the two techniques, namely payload truncating and packet counting. The results of this mixing are very impressive, shown in Figure 5.10.

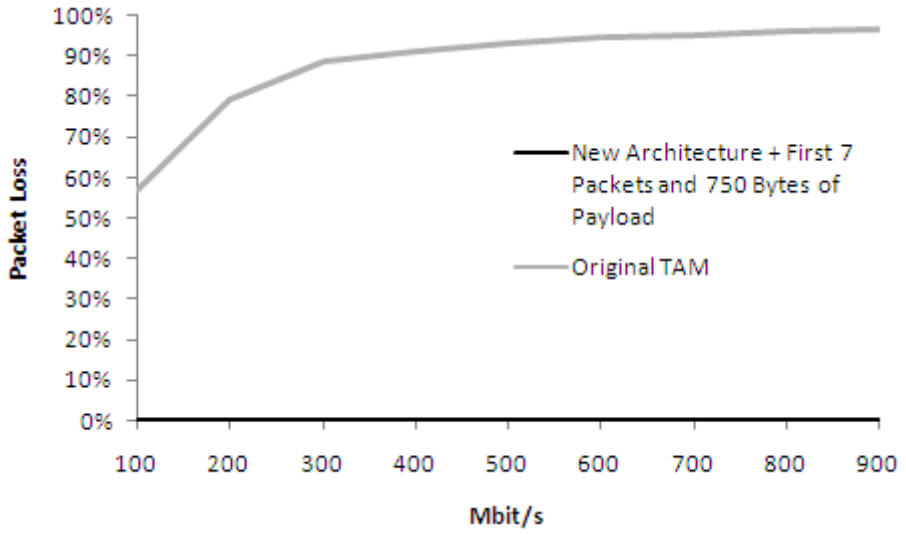

Figure 5.10 – Packet Loss, Original TAM vs. New Architecture (7 packets + 750 bytes)

This experiment was useful to show how the bottleneck identification is important in real time systems, which was the RegEx matching process. Therefore, optimizing its execution could lead to an incredibly decrease in packet loss rate, having 0.17% as the highest rate in 900Mbit/s.

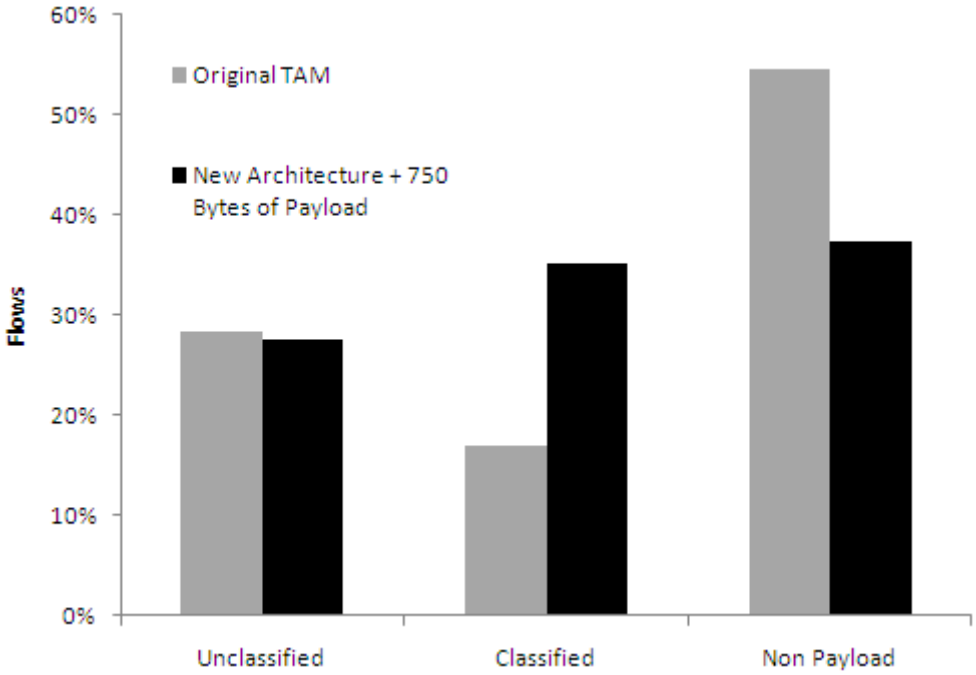

Figure 5.11 – Classification Completeness, 750 bytes analysis – 700Mbit/s (Flows)

Additionally, these experiments had the same behavior of the other performed at the previous section, regarding the impact on classification. As seen of Figure 5.11. The number of unclassified flows, when inspecting only the first 750 bytes of payload is around 27.58%, whereas the original TAM analyzing full payload is around 29%. But, the number of Non Payload flows in the original TAM is around

54%, against 37% when analyzing the first 750 Bytes of payload. The justification for such behavior is the same that occurred with the packet counting evaluation: Greater number of packets captured. Figure 5.12 shows the classification completeness, regarding the byte volume, between the Original TAM and the New Architecture inspecting only the first 750 bytes of payload. As expected the classified volume has increased and the unclassified byte volume has decreased.

Figure 5.12 – Classification Completeness, 750 bytes analysis – 700Mbit/s (Bytes)

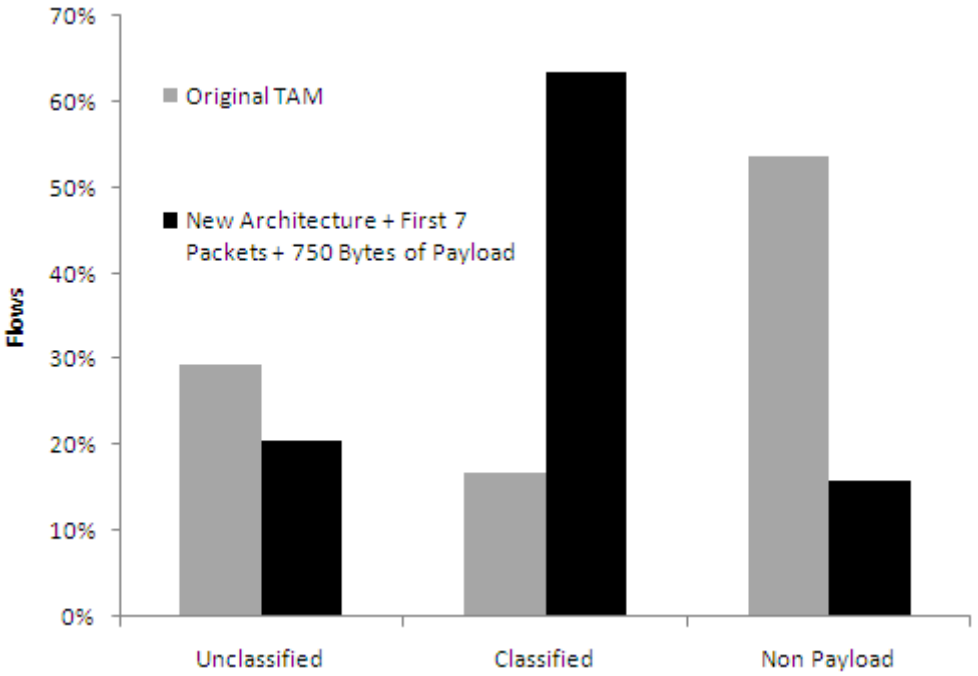

Figure 5.13 – Classification Completeness, 7 Packets and 750 bytes analysis – 700Mbit/s (Flows)

Finally, Figure 5.13 and Figure 5.14 show the classification completeness, regarding volume amount and number of flows that were classified on the comparison made between the Original TAM and the New Architecture inspecting only the first 7 packets and 750 bytes of payload. The New Architecture was able to classify more than 60% of all flows and almost 80% of all byte volume.

Figure 5.14 – Classification Completeness, 7 Packets and 750 bytes analysis – 700Mbit/s (Bytes)

5.2.5 Evaluation with Rewritten Patterns

It has been seen that the main bottleneck of TAM is the RegEx matching operation. So, in order to obtain some performance gains, some patterns, that were using unnecessary greedy quantifiers like * and +, were rewritten. Such modification on those patterns have reduced its generated automata size and consequently reduced its searching time.

For instance, TAM has the following pattern to recognize the HTTP protocol, described at Figure 5.15.

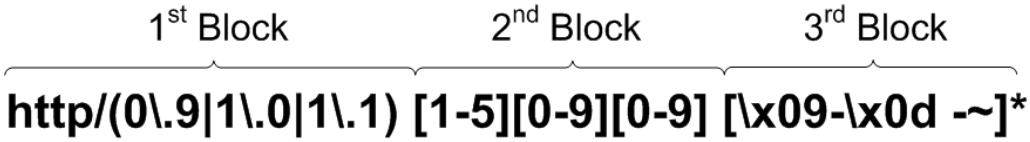

Figure 5.15 – HTTP RegEx

This pattern has a greedy quantifier at its end, which denotes zero or more repetitions of any characters in the range specified inside the brackets (3rd

Block). However, this quantifier is completely unnecessary, since that a payload, to successfully match this RegEx, only needs to have the first two blocks of the expression, making no difference whether there are more characters or not after these blocks. This indifference is caused by the quantifier *****, denoting zero or more repetitions. Thus, removing the final block of this expression can lead to a considerable gain in the matching speed, since the greedy quantifier will not be present spending unnecessary time, looking for more characters. Additionally, if the state machine has reach the 2nd block of the expression, the RegEx has already performed a successfully match. Therefore, the resulting expression, after the rewriting operation is depicted at Figure 5.16.

```
http/(0\.9|1\.0|1\.1) [1-5][0-9][0-9]
```

Figure 5.16 – Rewritten HTTP RegEx

Other RegExes have suffered subtle modifications like the one performed on the HTTP RegEx. For instance, TAM has got the following signature to classify the IRC chat application, depicted at Figure 5.17.

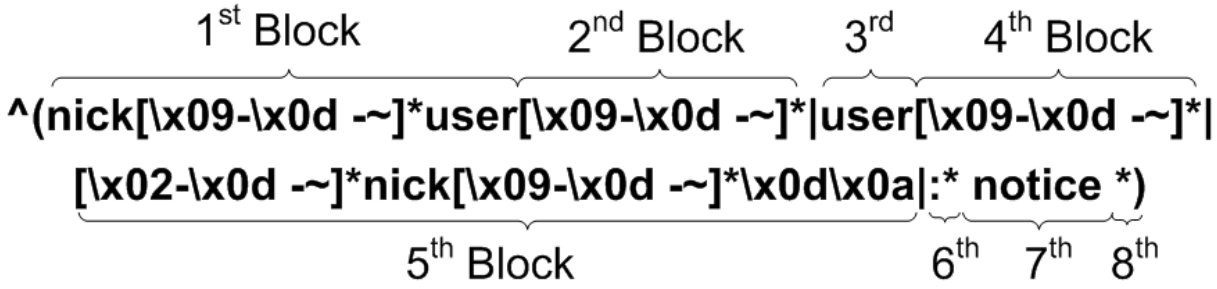

Figure 5.17 – IRC RegEx

This RegEx has several greedy quantifiers, which can be removed without loss of correctness in classification. Therefore, the quantifiers present at the 2nd, 4th, 6th and 8th can be removed of the RegEx, by the same justification given with the HTTP RegEx. After that, TAM has a new RegEx for IRC identification depicted at Figure 5.18

```
^(nick[\x09-\x0d -~]*user|user|[\x02-\x0d -~]*nick[\x09-\x0d -~]*\x0d\x0a| notice)
```

Figure 5.18 – IRC RegEx Rewritten

The presented RegExes at this section were not the only expressions that were rewritten, there were other rewritten RegExes following the same idea of HTTP and IRC. These subtle modifications in the RegExes used by TAM have lead to

considerable performance and, consequently, packet capture rate gains, which are present on the graph at Figure 5.19.

As seen on the graph presented at Figure 5.19, the packet loss rate was reduced from 56.94% to 0.0%, in 100Mbit/s, and from 96.57% to 90.32%, in 900Mbit/s. Also, as expected, the classification completeness has increased, due to more packets captured, as shown in Figure 5.20.

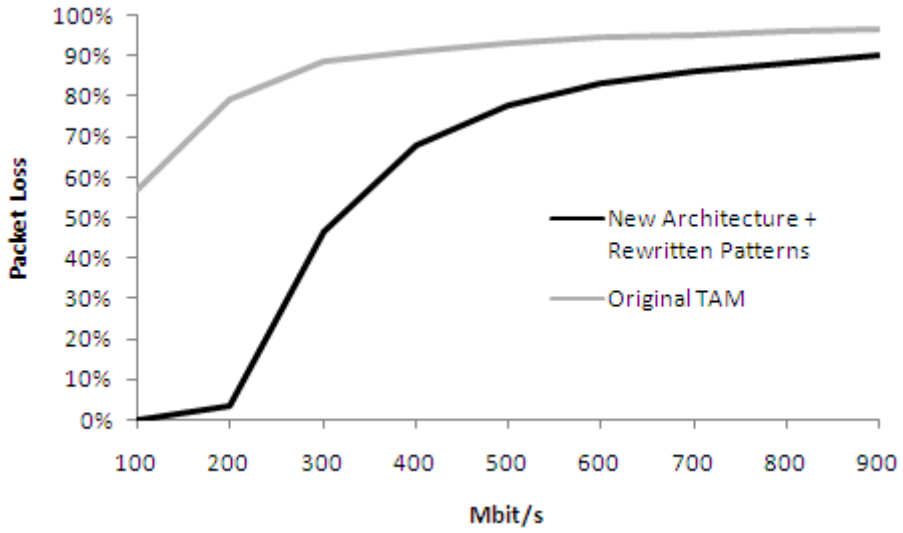

Figure 5.19 – Packet Loss, Original TAM vs. New Architecture (Rewritten Patterns)

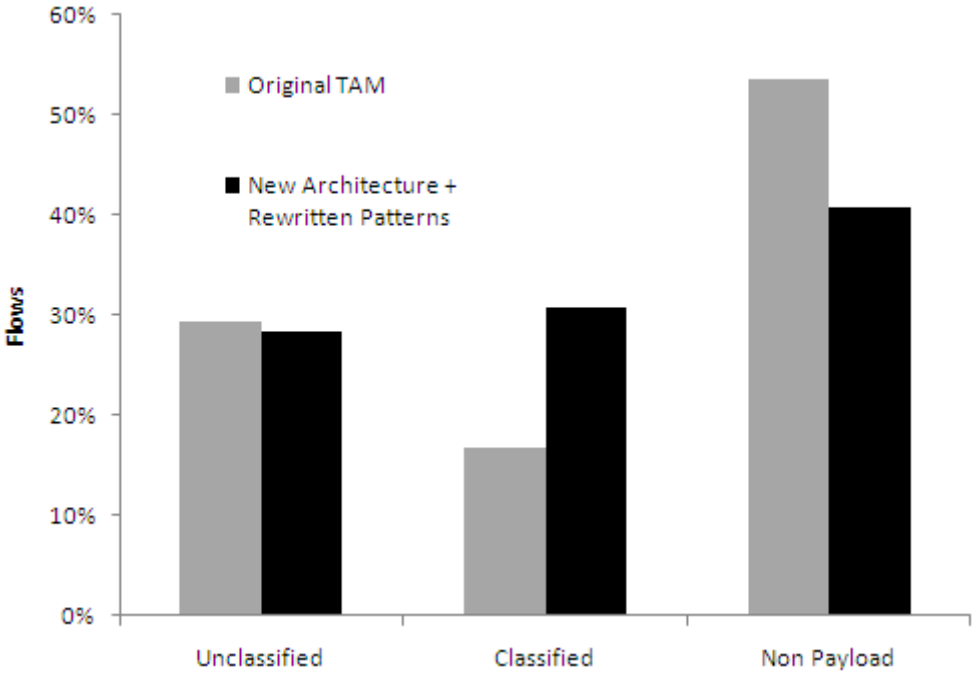

Figure 5.20 – Classification Completeness, Rewritten Patterns – 700Mbit/s (Flows)

The number of unclassified flows has reduced from 29.48% to 28.40%, whereas the number of Non Payload flows has reduced from 53.67% to 40.84% and the number of Classified has almost double from 16.85% to 30.77%, which represent

a considerable gain. As said on previous sections, this slight reduction in the number of unclassified flows, from the Original TAM to the New Architecture, is very important, since they represent flows with huge volume of traffic. Therefore, Figure 5.21 proves it. Since, the classified volume has increased from 23.27% to 51.78%.

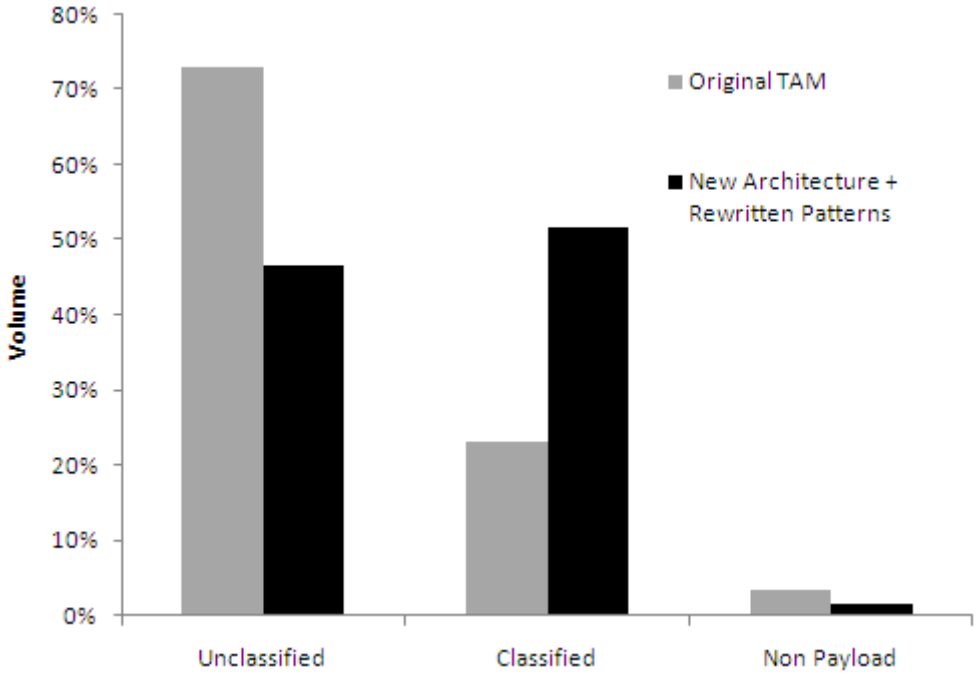

Figure 5.21 – Classification Completeness, Rewritten Patterns – 700Mbit/s (Bytes)

5.2.6 All Techniques Together

Now, in order to obtain the highest packet capture rate, the techniques mentioned on this thesis were combined with the New Architecture, leading to an incredible improvement in both packet capture rate and classification completeness. The results obtained in the packet loss metric analysis are shown at Figure 5.22.

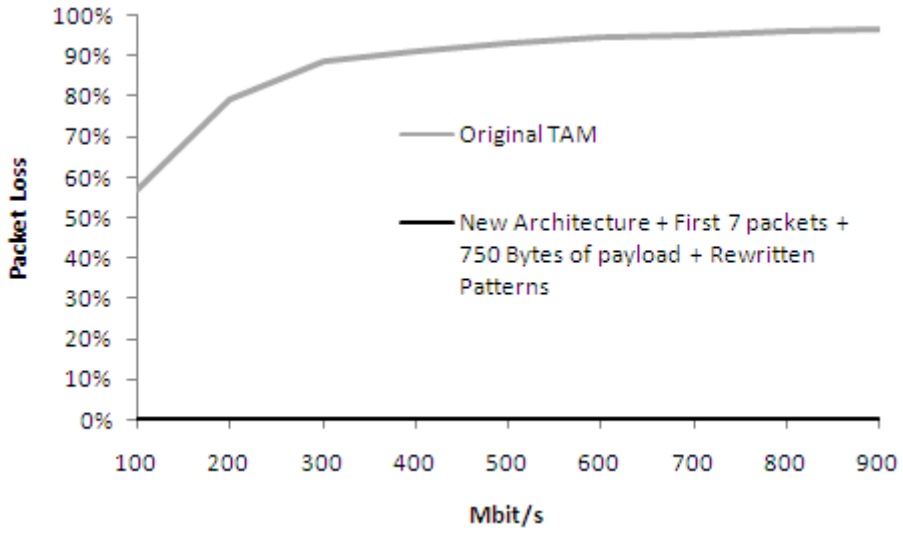

Figure 5.22 – Packet Loss, Original TAM vs. New Architecture (all techniques)

Also, the New Architecture mixed with the other techniques has started to lose packets only at 800Mbit/s. Thus reducing from 96.57%, with the Original TAM performing at 900Mbit/s, to 0.15%, with the New Architecture and the techniques mixed together.

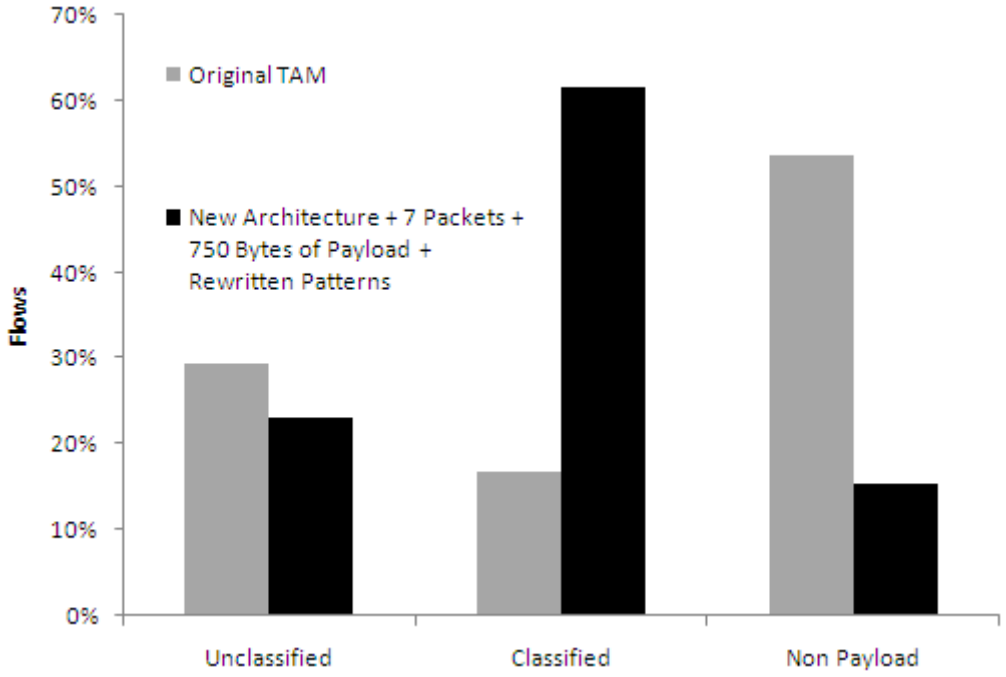

Figure 5.23 – Classification Completeness, Mixed Techniques – 700Mbit/s (Flows)

Additionally, the classification completeness reached the greater number of classified flows, when compared with the previous results. The Figure 5.23 shows the achieved results. The number of unclassified flows has reduced from 29.48%, with Original TAM, to 23.02%, with the New Architecture and mixed techniques and the number of classified flows has changed from only 17% to 61.51%. Also, the number of Non Payload flows has suffered greater reduction, from 53.67% to 15.47%.

When analyzing the classification completeness, regarding the byte volume that was classified, the obtained results are even better, as seen in Figure 5.24. The Unknown traffic volume, have reduced from 73.15% to 21.45% and the Non Payload traffic volume from 3.58% to 0.63%, thus representing a considerable gain. Hence, the New Architecture mixed with all presented techniques has bumped the number of classified volume, from 23% to almost 80%, a gain of 250%.

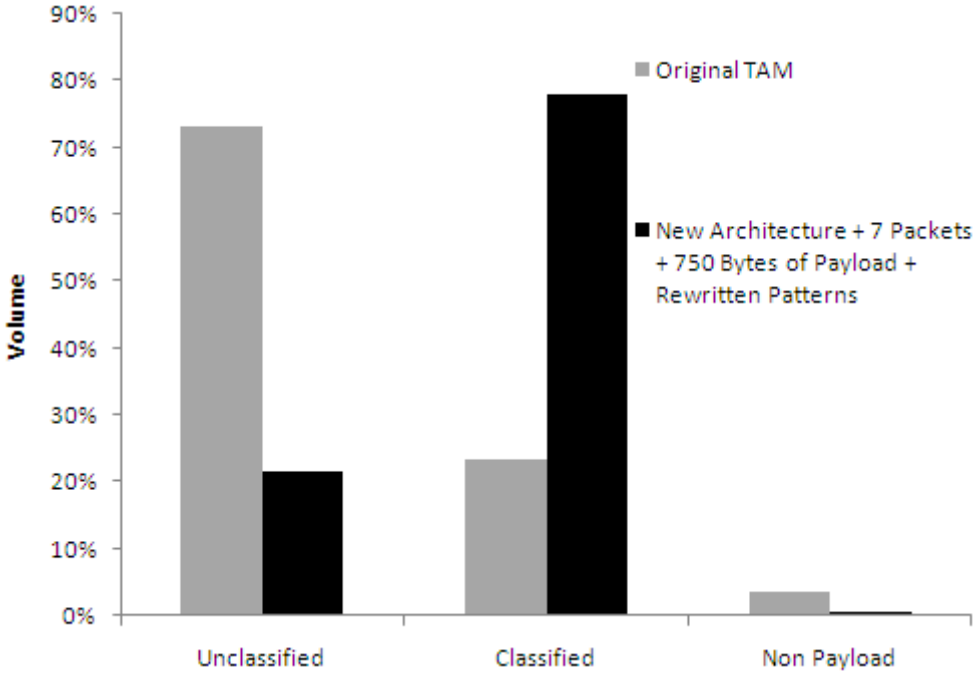

Figure 5.24 – Classification Completeness, Mixed Techniques – 700Mbit/s (Bytes)

6 DISCUSSION

In the previous sections, some optimizations were discussed, implemented and evaluated. One of the key points was to confirm that problem solving always start with a carefully architecture and design planning, combined with correct choice of the used data structure. Those points were reflected on the combination of the Multi-threaded approach and LBM, in architecture and design, and the Hash Table optimization, in data structure choice. Additionally, without those preliminary optimizations, none of the presented gains would be possible to achieve, since they were the core of the new system.

However, the presented optimizations, namely packet counting and payload truncating, have to be used with care. The payload truncating technique, for instance, can change the traffic profile of the analyzed network, since other signatures could perform a successful match, instead of the correct one that would match at the truncated payload's block. Hence, choosing a low level of truncating can lead to optimization gains, with no considerable changes in classification. Additionally, if the packet counting is performed with a small number of packet's threshold, a great amount of traffic can migrate to the Unknown Class, since some applications' protocol do not put their signatures in the first packets of a flow.

7 CONCLUDING REMARKS

This thesis has shown how DPI system developers can take advantage of subtle modifications that lead to considerable performance gains. Now, DPI systems that were not able to deal with high packet incoming rate (e.g. 1Gbps) can handle such load of traffic with increase in the classification completeness. Additionally, such modifications empower DPI systems, in order to provide ISPs with a wider view of what is passing through their network, aggregating a great amount of traffic information.

Those mentioned performance gains are summarized in the steps made in each optimization phase, thus resulting in a gain of almost 100%, reducing from 96.57% of packet losses, in the Original TAM, to 0.15%, in the New Architecture with all optimizations together. Additionally, this thesis has shown how DPI systems can take a great advantage when analyzing the first packets of a given flow, and truncated payload, with considerable gain in classification completeness.

Finally, there are some future works that are worth performing, in order to obtain even better results.

In section 5.2.5 the results have shown that RegEx rewriting is a valuable technique that can lead to considerable performance, without huge effort. However, only few RegExes were rewritten, those that presented the most critical problems, in set of about 40 RegExes. Hence, inspecting the other RegExes, searching for problematic signatures, and rewriting them can lead to interesting results. Additionally, there are other RegEx libraries that can be evaluated and compared with the one used in TAM, such libraries are for instance: libpcre [16] (used in Snort) and Boost Xpressive [4].

Also, those used thresholds in the Packet Counting and Packet Truncated techniques, namely 7 packets and 750 bytes, could be evaluated in order to find if they can be reduced, without decrease in classification completeness.

Last, but not least, if the task performed by LBM could be performed in hardware level some gains could be obtained. For instance, a commodity Intel Network Card (Intel PRO/1000 PT) distributes the packet load between different processors, guaranteeing that different processors will not receive packets belonging to the same flow. Thus, with the interruption rate for incoming packets divided among

processors, each worker thread could be “pinned” to only work in a given core, increasing the cache hit rate when dealing with packets from their set of flows.

REFERENCES

[1] Aho, A. V. and Corasick, M. J., “Efficient string matching: an aid to bibliographic search,” *Communications of the ACM* 18, 6 (Jun. 1975), 333-340.

[2] Benvenuti, Christian, “Understanding Linux Network Internals”, U.S.A.: O’Reilly, December 2005.

[3] Bernaille, L., Teixeira, R., Akodkenou, I., Soule, A., and Salamatian, K. 2006. “Traffic classification on the fly,” *SIGCOMM Comput. Commun. Rev.* 36, 2 (Apr. 2006), 23-26.

[4] Boost Xpressive. http://www.boost.org/doc/libs/1_37_0/doc/html/xpressive.html, visited in November 19, 2008.

[5] Deri, L., “Improving Passive Packet Capture: Beyond Device Polling,” *Proceedings of SANE 2004*, 2004.

[6] Deri, L., “nCap: wire-speed packet capture and transmission,” *Proceedings of the End-to-End Monitoring Techniques and Services*. Workshop, p.47-55, May 15-April 30, 2005.

[7] Fernandes, S., Westholm, T., Antonello, R., Lacerda, T., Santos, A. and Sadok, D., “Performance Optimization for Deep Packet Inspection Systems,” UFPE Technical Report, September 2008.

[8] Goyal, N., Ormont, J., Smith, R., Sankaralingam, K. and Estan, C., “Signature Matching in Network Processing using SIMD/GPU Architectures,” *Technical Report TR1628*, Department of Computer Sciences, The University of Wisconsin-Madison, January 2008.

[9] IPP2P Project - “A NetFilter extension to identify P2P filesharing traffic.” http://www.ipp2p.org, visited in November 15, 2008.

[10] J. Levandoski, E. Sommer, and M. Strait, “Application Layer Packet Classifier for Linux.” http://l7-filter.sourceforge.net/, visited in November 14, 2008.

[11] Karagiannis, T., Broido, A., Brownlee, N., claffy, kc, and Faloutsos, M., “Is P2P dying or just hiding?,” *IEEE Globecom 2004 - Global Internet and Next Generation Networks*, 2004.

[12] Kumar, S., Dharmapurikar, S., Yu, F., Crowley, P., and Turner, J. 2006. “Algorithms to accelerate multiple regular expressions matching for deep packet inspection,” *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications* (Pisa, Italy, September 11 - 15, 2006). *SIGCOMM* '06. ACM, New York, NY, 339-350.

[13] Mogul, J. C. and Ramakrishnan, K. K. 1997. “Eliminating receive livelock in an interrupt-driven kernel,” *ACM Transactions on Computer Systems.* August 1997, 217-252.

[14] Oreste Villa, Daniele Paolo Scarpazza and Fabrizio Petrini, “Accelerating Real-Time String Searching with Multicore Processors,” *Computer*, vol. 41, no. 4, pp. 42-50, Apr., 2008.

[15] P. Wood, *libpcap-mmap*, Los Alamos National Labs, http://public.lanl.gov/cpw/, visited in November 15, 2008.

[16] PCRE - Perl Compatible Regular Expressions. http://www.pcre.org/, visited in November 19, 2008.

[17] Perl Regular Expressions. http://perldoc.perl.org/perlre.html, visited in November 20, 2008.

[18] Po-Ching Lin, Ying-Dar Lin, Yuan-Cheng Lai and Tsern-Huei Lee, "Using String Matching for Deep Packet Inspection," *Computer*, vol. 41, no. 4, pp. 23-28, Apr., 2008.

[19] POSIX Regular Expressions. http://www.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap09.html, visited in November 20, 2008.

[20] Procera Networks. http://www.proceranetworks.com/, visited in November 15, 2008.

[21] Schneider, F., Wallerich, J. and Feldmann, A., “Packet Capture in 10-Gigabit Ethernet Environments Using Contemporary Commodity Hardware,” *Proc. Passive and Active Measurement Conference*, 2007.

[22] Schneider, Fabian, “Performance evaluation of packet capturing systems for high-speed networks,” Master Thesis, 2005, Technical University of Munich.

[23] Sen, S., Spatscheck, O. and Wang, D. 2004. “Accurate, scalable in-network identification of p2p traffic using application signatures,” *Proceedings of the 13th international Conference on World Wide Web* (New York, NY, USA, May 17 - 20, 2004). WWW '04. ACM, New York, NY, 512-521.

[24] Snort – The Open Source Network Intrusion Detection System. http://www.snort.org, visited in November 15, 2008.

[25] Vern Paxson, “Bro: A system for detecting networks intruders in real time,” *Computer Networks* 31, 23-24 (Dec. 1999), 2435-2463.

[26] Yu, F., Chen, Z., Diao, Y., Lakshman, T. V. and Katz, R. H., “Fast and memory-efficient regular expression matching for deep packet inspection,” *Proceedings of the 2006 ACM/IEEE Symposium on Architecture for Networking and Communications Systems*. ANCS '06. ACM, New York, NY, 93-102, 2006.