POLITECNICO DI TORINO
Repository ISTITUZIONALE

Optimizing Deep Packet Inspection for High-Speed Traffic Analysis

Original
Optimizing Deep Packet Inspection for High-Speed Traffic Analysis / Cascarano, N., Risso, F.G.O., Ciminiera, L.. - In: JOURNAL OF NETWORK AND SYSTEMS MANAGEMENT. - ISSN 1064-7570. - ELETTRONICO. - 19:1(2011), pp. 7-31. [10.1007/s10922-010-9181-x]

Availability:
This version is available at: 11583/2375838 since:

Publisher:
Springer

Published
DOI:10.1007/s10922-010-9181-x

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

12 September 2026

Noname manuscript No.
(will be inserted by the editor)

Optimizing Deep Packet Inspection for High-Speed Traffic Analysis

Niccolò Cascarano · Luigi Ciminiera · Fulvio Risso

the date of receipt and acceptance should be inserted later

Abstract Deep Packet Inspection (DPI) techniques are considered extremely expensive in terms of processing costs and therefore are usually deployed in edge networks, where the amount of data to be processed is limited. This paper demonstrates that, in case the application can tolerate some compromises in terms of accuracy (such as many measurement-based tasks) and in presence of normal traffic, the processing cost can be greatly reduced while even improving the classification precision, making DPI suitable also for high-speed networks.

Keywords Traffic Analysis · Deep Packet Inspection · Network Monitoring · Traffic Classification

Niccolò Cascarano
Politecnico di Torino, C.so Duca degli Abruzzi 24, 10129 Torino, Italy
E-mail: niccolo.cascarano@polito.it

Luigi Ciminiera
Politecnico di Torino, C.so Duca degli Abruzzi 24, 10129 Torino, Italy
E-mail: luigi.ciminiera@polito.it

Fulvio Risso (corresponding author)
Politecnico di Torino, C.so Duca degli Abruzzi 24, 10129 Torino, Italy
Phone: +39-0115647008,
Fax: +39-0115647099,
E-mail: fulvio.risso@polito.it

1 Introduction

The usage of the Internet changed dramatically in the last few years. The Internet new transports traffic generated by many different users and applications including financial transactions, e-business, entertainment and more, which is definitely different from the traffic we had 30 years ago when the network was engineered for email, telnet and FTP. Often, the only way to keep the pace of the new traffic trends is to have an effective infrastructure for real-time measurements in the network, which allow to discover changes in the traffic patten as soon as they appear and adapt to them quickly.

The capability to recognize which application generated the traffic is perhaps one of the most important challenges in network measurements. Several technologies have been proposed so far. First was *port-based* traffic classification [1], which is now considered extremely imprecise [2] due to the large amount of applications using non-standard ports in order to escape network limitations (e.g. bandwidth shaping enforced by network providers). Next step was *Deep Packet Inspection* (DPI), which analyzed also data in the application-layer payload and is usually extremely effective. On the downside, this technology is considered extremely demanding in terms of processing requirements and memory occupancy. Recently, newer techniques [3–13] have been proposed that base on statistical methods, whose main advantage is the capability to detect also encrypted and tunneled traffic, but whose precision is acceptable only if we target a few protocols (often less than a dozen) out of the hundreds available on nowadays networks. Moreover, their processing (and memory) requirements may be comparable with packet-based DPI [14].

Due to the experimental nature of the statistical techniques and their current limitations, the mostly used technology for traffic classification is still DPI. Interesting, the biggest criticism to this technique is not related to its difficulties in classifying encrypted or tunneled traffic, but to its supposed high processing cost. In fact, DPI is extensively used in security applications such as Intrusion Detection Systems (IDS) and firewalls, which have strict requirements in terms of precision. In other words, a single misclassification in such these applications could let an attacker to compromise even an entire network, and is therefore a risk that people do not want to run. In order to minimize the risk of misclassifications, all these DPI implementations tend to privilege the accuracy, without taking the processing cost into much consideration.

However our scenario is different because we focus on applications that use traffic classification mainly for statistics, quality of service (QoS), and monitoring, and can accept some compromises in terms of accuracy. For instance, misclassified traffic may update the byte counter of the wrong protocol but this does not represent a threat such as a misclassification into an IDS. In this environment, the DPI implementation can optimized, for example by getting rid of expensive algorithms (e.g. the ones that reconstruct the entire stream at the application level) that are a must if the precision is a major concern, but that contribute substantially to the processing costs and memory requirements.

This paper proposes (and validates) a set of optimizations that are possible

if the DPI is used for network monitoring applications. Optimizations possible in this particular scenario include both architectural modifications and improvements on the typical processing path of a DPI engine for the average traffic mix present in nowadays networks. For this purpose we used real traffic traces, some of them containing also a relevant portion of peer-to-peer traffic which is often considered unsuitable for DPI classifiers because of its frequent use of encryption and content hiding techniques to avoid the identification. Moreover, two of the three traffic traces used have been collected using the GT suite [15], which provides the ground truth for evaluating the classification accuracy. The result of our optimizations leads to a DPI classifier that can work at very (multi-gigabit) high speed without any hardware speed-up, with an even improved classification accuracy.

2 Related works

Most papers (e.g., [3,7,8,10,12,13]) assume that that DPI is a very expensive traffic classification technique without any further investigation. This misperception is due to the fact that DPI is massively used in security applications (firewalls, IDS, etc.), which are well-known to have scalability problems because of their processing requirements (mainly due to DPI).

However, traffic classification for network measurements has different characteristics than the case of network security. The most evident is the expected degree of precision, since network measurements can accept a limited amount of uncertainty, while an IDS must intercept every single session that may represent a security threat and no compromises can be arranged about this point. A second difference is in the number of protocols that need to be recognized (often in the order of the hundreds) compared to the number of rules in security applications (several thousands[1]), which represents one of the inputs of the pattern matching engine. These observations lead us to the conclusion that the several works that focused on new techniques for fast and scalable regular expression matching [16–20] may not be appropriate in our case, since we can privilege simpler regular expressions (albeit with a limited decrease in terms of accuracy), allowing us to use faster algorithms even if they may suffer from state explosion [16]. In other words, we do not look at new algorithms, but we want to use the fastest ones in their best operating conditions.

In the line of dismantling the myth of the excessive cost of DPI engines, [14] recently demonstrated that the complexity of a well-known traffic classifier based on Support Vector Machine [21] may be comparable to the DPI one. However, this paper assumes a "lightweight" implementation of the DPI engine that operates packet-based instead of message-based, but it does not justify adequately this choice and it does not examine the (supposed) loss in precision in the sake of performance. Along the same line of cost reductions, optimizations based on reducing the amount of data that will be analyzed by the

[1] For instance, the ruleset of the November 2007 release of Snort includes 5549 rules requiring application-level content inspection [16].

pattern matching engine, e.g. using only the first portion of the session data for classification, are present in IDS such as BRO [22] and Snort [23]. Moore et al. [24] had a similar idea based on a cascade of algorithms, each one fed with more data than the one available in the previous step until a positive match is obtained. Another technique consists in stopping the analysis of a session if it is still unclassified after a given number of packets. However those techniques were never validated; the advantages in terms of processing cost and the (supposed) loss in terms of precision are still unknown. Furthermore, an adequate justification of the choice of a packet-based approach (excluding some partial results in [24, 25]) was still missing.

This paper aims at filling this gap by properly justifying of the feasibility of the "lightweight" DPI approach, and by presenting (and evaluating) some more optimizations that are able to decrease the processing cost even further, all based on the assumption that we can tolerate a minor worsening in the classification accuracy. Furthermore, we can leverage some common characteristics of the network traffic present in nowadays networks in order to speed-up the processing of the most frequent processing path in the DPI engine.

3 DPI Improvements

This section describes the basic operation of a DPI traffic classifier and presents the possible improvements with respect to processing speed.

3.1 Deep Packet Inspection basics

A DPI classifier is based on the differences in terms of application-layer protocol headers that are used by each application to initiate and/or control the data exchange. We can associate each TCP/UDP session to the corresponding application by means of a set of regular expressions (commonly called *signatures*), which represent a sort of "fingerprint" of the specific application protocol. The transport-layer payload is compared to the set of available signatures and the matching one identifies the application that generated that traffic. In case several signatures match the same payload, an heuristic is used to select the best one (e.g., the longest signature, priority, etc.). The pattern matching engine inspects only the subset of packets that belong to a session not yet associated to an application-level protocol. In fact, as soon as a session is identified, the 5-tuple SessionID (IP source/destination address, transport-layer protocol, source/destination port) is inserted into a session table and all the following packets that have the same SessionID will no longer be delivered to the pattern matching module of the DPI engine.

Although the previous description is enough to enunciate the principles of DPI traffic classification, in practice several different implementations are possible [25]. The most important are the *packet based, per-flow state* (PBFS) that analyzes data on a packet-by-packet basis, and the *message based, per-flow state* (MBFS) that analyzes application-level payloadd as a unique stream

of data, after applying TCP/IP normalization (Section 3.2.1).

3.2 Architectural improvements

This section presents some architectural choices that can have a huge impact on the performance of the DPI traffic classifier, namely the possibility to operate *per-packet* instead of *per-session* and the algorithm used by the pattern matching engine.

3.2.1 Avoiding TCP/IP normalization

The TCP/IP normalization aims at recovering the problem of IP fragmentation (i.e. in case an IP packet is larger than the MTU of the physical network and hence it must be split in several fragments) and TCP reassembly (e.g. in case an application-layer message cannot fit a single IP packet and hence it is split across different packets). These techniques can be used in the DPI classifier in order to examine a portion of data that exceeds the content of a single IP packet. In fact, the regular expressions used by a DPI engine might require a larger number of bytes than the ones available in the first packet (with a valid payload) of the TCP/UDP session. Actually, this may happen frequently when the regular expression contains the Kleene closure operator (i.e., "$*$"), which means that an arbitrary number of characters might be present in the specified position in the data under analysis.

While security applications tend to use TCP/IP normalization, some earlier works on traffic classification [14, 24, 25] seem to indicate that a packet-based approach is appropriate for all the cases in which a limited worsening of the precision is acceptable, but without an appropriate justification. We tend to agree; in fact we believe that TCP/IP normalization can be avoided in our reference scenario because the number of bytes that need to be examined for a correct classification of TCP/UDP sessions is usually small, i.e. the first packet of the session is usually enough to classify the session. If this assumption is true, the MBFS approach can be replaced with the simpler PBFS with a limited worsening in terms of accuracy; we will provide the numerical evidence supporting this assumption in Section 5.1.

3.2.2 Efficient pattern matching engine

Intuitively, the main cost of a DPI classifier derives from the execution of the pattern matching algorithm; experimental evidence can be found e.g. in [14], which refers to a PBFS traffic classifier. Using the same methodology presented in [14] that splits the DPI engine in its main components, Table 6 shows that the pattern matching can account up to 8900 CPU ticks[2], while the impact of the other two main components of a PBFS classifier (Session ID Extraction, which extracts the source/destination IP address and source/destination

[2] More details on the evaluation methodology will be provided in Section 4.

TCP/UDP port from each incoming packet, and Session Lookup, which determines if the current session has already been classified) is negligible.

Although there are several possible implementations for regular expression matching, we know from the theory that the choice of the proper algorithm for the regular expression engine could make a big difference in terms of performance. In this respect, the best option is the Deterministic Finite Automata (DFA), in which the computational cost depends only on the length of the input sequence, independently from the number of regular expression to be checked. Unfortunately, depending on the characteristics of the signatures, the automaton that represents the set of regular expressions may require a large amount of memory, which is the reason why DFA is barely used in regular expression matching. Despite the common belief, this paper suggests that DFA-based engines can be proficiently used for traffic classification because, under some assumptions (that are verified in our scenario) we do not have state explosion, which is the main reason for the adoption of other regex engines.

Another impact factor on execution cost is the "friendliness" of regular expression used for identifying protocols. A massive use of wildcards can lower the overall performance in terms of processing cost (and impacts on the memory occupancy too). When writing protocol signatures, we can have an additional boost in performance if we pay attention to the form of the regular expression used.

3.3 DPI Optimizations

In this section we focus on the PBFS flavour of the DPI because, as it will be demonstrated in Section 5.1, a packet-based approach is appropriate for our objectives. In this case we can further improve the cost of the DPI classification by implementing two additional techniques that are based on the idea of reducing the amount of data processed by the pattern matching engine.

In fact, referring to the results reported in Table 6, the cost of the SessionID Extraction and Session Lookup is fixed and present in all the packets, while the pattern matching depends on the regular expressions used and on the payload being analyzed (hence of the traffic trace used in the evaluation).

Since the signature matching may be repeated on any new packet until the session is classified, important parameters are the number of times the pattern matching algorithm is invoked (e.g. a trace with many long sessions might require a smaller number of analysis), the presence of unclassifiable sessions such as encrypted communications (that, being encrypted, will never match their signature, although they might trigger some misclassifications), the number of bytes inspected.

For these reasons, the maximum cost reported in Table 6 represents the worst case in case of a PBFS classifier, while the actual cost of the pattern matching is the average value[3] that is derived by calculating the average processing time over all packets submitted to the pattern matching engine for a

[3] Although in case of real-time traffic analysis we should care about the *worst* case in

given trace.

From the above considerations, it is evident that two methods can be used to further improve the execution cost of a PBFS DPI engine: the reduction of the number of bytes of the payload that will be examined (Section 3.3.1) and the reduction of the number of packets submitted to the pattern matching algorithm (Section 3.3.2).

3.3.1 Snapshot-based classification

An inspection of the typical protocol signatures for traffic classification (e.g. the ones in the `l7-filter` [26] or NetPDL [27] databases) shows that most of them require a limited number of bytes for identifying a protocol, and these bytes are usually placed at the beginning of the application data. This makes sense because application signatures usually describe the handshaking phase of application protocols or some kind of data used for synchronization, which is usually placed at the beginning of the application payload; the rest is application-related data that is useless for classification.

We speculate that forcing the DPI classifier to analyze only the first portion of the data in the network packets can provide an advantage in terms of cost, without a significant impact on classification accuracy. The problem consists in determining the amount of bytes required for classifying the most part of the traffic and to evaluate the possible loss in classification accuracy, in order to determine the best trade-off between accuracy and processing cost. While a fixed limit (e.g. inspect only the first n bytes of the payload) seems too crude (e.g. application protocols such as HTTP whose signature include the Kleene closure may require an arbitrarily high number of bytes to match and an hard limit could impede the classification of the session), a more sophisticate approach such as stopping after the shortest (first) positive match shows some problems anyway. For example Gnutella is an HTTP-like protocol whose signature checks for an optional field within the HTTP header. If the classifier stops the pattern search algorithm on the shortest match, all the Gnutella packets will be classified as HTTP and the optional fields (that follow in the payload) will no longer be inspected. For the abovementioned consideration, we decided to follow the first approach, i.e. define an hard limit in the number of bytes examined. We will then determine the best value and evaluate its impact in terms of accuracy in Section 5.4.1.

3.3.2 Limiting the number of classification attempts

This technique aims at reducing the amount of bytes inspected, such as the one presented in the previous section, but looking at a different perspective. In fact, many application-layer protocols repeat their protocol headers at the

order to be sure to sustain any incoming load, this may lead to overprovisioned system, since the probability that all the packets fall in the worst-case scenario is very small. We consider the *average* case a better representation of what we can actually have in a real network scenario.

Table 1 Details on the traffic traces used

Data set	Date and duration	Bytes	Packets
POLITO-GT	December 10, 2008 68 hours	202 GB 76.8% TCP	330M 69.8% TCP
UNIBS-GT	December 17, 2008 56 hours	3.5 GB 99.4% TCP	4.72M 67.6% TCP
POLITO	Dec. 20, 2007 12 hours	419 GB 94.7 % TCP	579M 92.3% TCP

beginning of several packets; this applies particularly to UDP traffic, whose datagrams travel independently on the network and therefore each packet requires its own header (e.g. for sequence numbering). When a PBFS classifier is used, we can imagine that a session that was not classified at the first packets (e.g. because the signature is split across two packets) can be classified later on, when the signature is found on one of its following packets. However, the risk is that the DPI classifier wastes a huge amount of computational resources while performing pattern searches on sessions that cannot be identified anyway. For instance, a protocol cannot be classified when *(i)* the data transported by the session is encrypted, *(ii)* the classifier does not know the signature that describe the protocol transported, or *(iii)* in case of a PBFS classifier, the signature does not match because it is split across two packets.

Assuming that the case (iii) is not relevant (Section 5.1 demonstrates that is very rare at least in the traces used in this work), in the other two cases the DPI classifier should stop its analysis over a given session after a reasonable number of classification attempts and leave the session as "unclassified". This could save processing power since we avoid the analysis of all the packets belonging to the session, while at the same time we decrease the possibility that a random payload matches a signature and brings to a misclassification.

4 Evaluation methodology

4.1 Traffic traces used

Table 1 summarizes the most important characteristics of the three full-payload traffic traces we used to evaluate the proposed optimizations. All of them were collected at the border routers of Politecnico di Torino and University of Brescia campuses and then properly anonymized. The "POLITO-GT" and "UNIBS-GT" traces were obtained using the GT suite [15], which let us know which application generated each session; this information allows to derive the absolute accuracy of the classification process in terms of correctly classified, misclassified and unknown traffic.

POLITO-GT contains mainly peer-to-peer and WebTV traffic, generated by 10 virtual machines running Edonkey, Bittorrent, Skype, PPlive, TVAnts and SopCast on Windows XP, plus four real machines running Linux, Windows Vista and MacOS X with default settings and used by regular users. WebTV applications were executed with an automatic turnover of 1 hour, while P2P

application were downloading and seeding some popular resources for all the duration of the capture. This traffic dataset is known to be very challenging for DPI classifier because of the massive presence of P2P traffic. Some signatures related to WebTV protocols were in fact not available at all, while some have been derived from reverse engineering. Skype traffic is encrypted and P2P clients adopt hiding techniques for avoiding classification. The UNIBS-GT trace was collected in a research laboratory where about 20 PhD students were asked to run the GT daemon while doing their normal activities. This trace is smaller than POLITO-GT in term of volume, but it is interesting because it contains normal users' activity, including some P2P file sharing.

Since POLITO-GT and UNIBS-GT include traffic generated by a limited number of hosts due to the difficulties to deploy the GT suite over many clients, we decided to add also the POLITO trace in our analysis that includes traffic generated by about 6000 hosts during an entire working day in order to extend the evaluation scenario. Although the ground truth is not available on this trace (hence the accuracy of our optimizations cannot be assessed on this trace), it represents the best choice for evaluating the impact of proposed optimizations in terms of processing cost because of the presence of a large variety of traffic that better represents the behavior of a real network.

4.2 Parameters under evaluation

Optimizations are evaluated according to the following parameters: *(i)* the performance speedup of the new classifier compared to the baseline PBFS classifier presented in Section 5.3, and the accuracy in terms of *(ii)* percentage of unknown traffic and *(iii)* misclassified traffic.

The "unknown traffic" is the traffic that does not match any known protocol signature. The "misclassified traffic" is the traffic that matches a protocol signature that does not correspond to the application that generated it. Obviously, the last parameter can be calculated only on traffic traces generated with the GT suite. The traffic correctly classified can be derived by complementing the unclassified and misclassified traffic. When we evaluate an optimization, we tolerate a maximum worsening of 1% in terms of correctly classified traffic compared to the same classifier with the same optimization turned off.

Processing cost has been evaluated by running the classifier on our traces and measuring the average processing cost per packet. The cost was expressed in CPU ticks (using the RDTSC assembly instruction) and include only the time spent in the DPI classifier, excluding all the other portions of the code that were outside the classifier (e.g. loading packet from disk). The measurement platform was an Intel Dual Xeon 5160 at 3GHz, 4GB RAM and Ubuntu 8.04 32bit; the code under examination was compiled with GCC and always executed on the same CPU core.

Table 2 Additional traffic classified with the MBFS approach

Data set	Unknown TCP traffic	Additional classified traffic
POLITO	23.5GB (5.7%) - 405881 sessions	2.6MB - 77 sessions
UNIBS-GT	870MB (29%) - 1310 sessions	0B - 0 sessions

5 Experimental evaluation

5.1 Message vs. packet-based DPI

We analyze here the difference in terms of accuracy between a packet-based and a message-based traffic classifier, whose main difference is in the presence of the TCP/IP normalizer module.

We decided to implement the sole TCP re-assembler algorithm, without the IP de-fragmenter. For instance, our TCP traffic does not have any IP fragmented packet (probably because of the path MTU discovery capability of the TCP protocol), while UDP has many fragments especially in the POLITO-GT trace. However, as it will be demonstrated in Section 5.4.1, UDP traffic classification does not take advantage from IP de-fragmentation because almost all the traffic can be classified within the first 256 bytes of payload (the minimum size for the first fragment is 576 bytes), therefore making this step useless.

We processed our traces with the `tcpflow` tool, which performs TCP session reassembling and returns the resulting application-layer messages. We limited the reassembling to 2MB of data for each session to avoid memory explosion; furthermore, analysis were limited to the POLITO and UNIBS-GT traces because the most part of the unclassified traffic in POLITO-GT is due to the lack of some signatures related to WebTV applications and not because patterns are split across multiple packets.

Table 2 shows the amount of additional traffic that is correctly classified thanks to the TCP reassembly, compared to the traffic correctly classified by a simpler PBFS classifier. Results are discouraging for both datasets, with an additional amount of classified traffic that is definitely negligible. Although these numbers can change with different traffic traces, we believe that our datasets are reasonably representative of the average traffic mix generated by normal users and hence we can conclude that the additional complexity of the TCP/IP normalization (in both processing cost and memory requirements) is not balanced with enough additional accuracy. This, in turns, confirms that the PBFS approach has a better cost opportunity than the MBFS approach, at least when some imprecisions in the classification process can be tolerated.

5.2 Architectural improvements

This section evaluates the impact that different algorithms may have on the cost of the pattern matching, and suggests some methods to improve the "friendlyness" of the regular expressions with respect to DFAs.

5.2.1 DFA as pattern matching algorithm

The algorithm used for the pattern matching has a huge impact on the performance of a DPI classifier. Table 3 analyzes three different algorithms (in the implementation provided by [28]), namely NFA (Non-Deterministic Finite Automata), DFA and compressed DFA (cDFA [19]), when applied to the entire signature database of `l7-filter` [26]. Results include the minimum, average and maximum cost for analyzing a packet (as derived from the the POLITO traffic trace), the number of distinct automata required to compile the pattern set into the corresponding memory structure, and the total memory used.

The first result is that the `l7-filter` database cannot be compiled into a single DFA because of the ambiguities contained in the pattern set, which generates a graph that grows exponentially and cannot be contained in memory. For instance, DFA and cDFA require to partition the signatures in four parts, leading to four different automata that were processed sequentially. The splitting algorithm was very simple, since we created an additional automaton as soon as the number of states exceeded 100K. Vice versa, the NFA does not have any problem of memory explosion and the entire set can be compiled into a single automaton. As expected (results in Table 3) the NFA guarantees very limited memory occupancy but at the expense of the execution cost that is prohibitively high, which may further increase when adding new protocol signatures. In fact, its average cost is about three orders of magnitude higher than the sequential execution of the four DFAs, and two order of magnitude higher when compared to the cDFA case.

As predicted by the theory, a DFA-based algorithm has the best processing performance and its worst-case execution cost is independent from the type of signature used and from the number of regular expressions, as shown in Figure 1. However, the question is whether this approach is applicable in our case, since DFAs are well-known for states explosion. Tests reported before, in fact, used four different DFAs in sequence in order to limit this problem, but in line of principle we cannot guarantee that this approach is feasible, e.g., at we never reach one point in which the number of distinct DFA is so large that it cancels the theoretical advantages of the algorithm. This point will be discussed in the next section.

It is worthy remembering that DFA can have other limitations, e.g., patterns cannot use context-sensitive regular expressions such as `(a*)(b*)(\1)(\2)`. In other words, a DFA implementation guarantees better performance but limits the expressiveness of the signatures. We speculate that these regular expressions are very rare in the real world; for instance, both the `l7-filter` and NetPDL databases do not contain any of those.

5.2.2 Analyzing states explosion in DFAs

Being the memory explosion the most important point against DFAs, we analyzed the memory occupancy in case of the signatures contained in the NetPDL database available online [27]. Results are reported in Figure 1, that measures

Table 3 Cost of different algorithms applied on the `l7-filter` signature database

Algorithm	Cost (CPU ticks) min - avg - max	# automata	Memory
NFA	2.2E4 - 4.1E7 - 8.9E7	1	509KB
DFA	52 - 2.5E4 - 3.6E4	4	230MB
cDFA	268 - 1.2E5 - 1.7E5	4	53MB

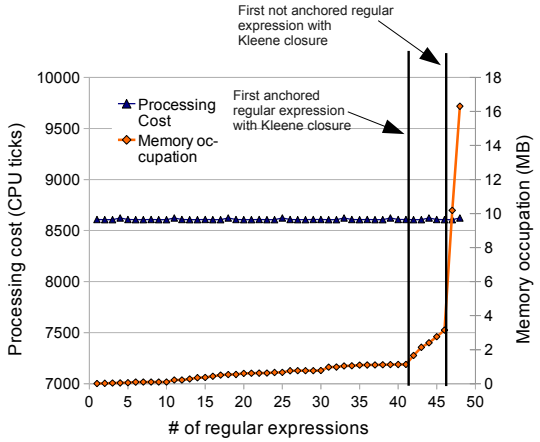

Fig. 1 Cost and memory occupation of DFA implementation of pattern matching algorithm.

the processing cost and the memory occupancy starting with a single regular expression, then repeating the same test with up to $N, N+1, ...$ patterns, till we reach the total number of signatures contained in our protocol database.

It is evident that the memory occupancy does not grow linearly, as it is strongly dependent from the type of signature added. In this respect, we can divide regular expressions in three classes: *(i) anchored regexp* (i.e., begins with the '^' sign), that identifies the regular expressions satisfied only if the pattern is found at the beginning of the payload, *(ii) anchored regexp containing the Kleene closure* (i.e. the '*' wildcard), in which the regular expression can be found in any point of the input data, and *(iii) not anchored regexp containing the Kleene closure*[4].

In fact, Figure 1 shows that the memory occupancy increases linearly when the input patterns contain only expressions of type (i) (first region on the left), but the slope increases when we add also expressions of type (ii) (second region), and it tends to increase exponentially when we add the second expressions of type (iii). This is due to the possible ambiguities in the input pattern that force the addition of a large number of states for matching all the possible cases. It is worthy noting that the number of states explodes when at

[4] The additional category *not anchored regexp not containing the Kleene closure* is omitted since is equivalent to type (ii), where the Kleene closure is at the beginning of the regular expression.

Table 4 Execution cost on different types of regular expressions

Signature type	Match (CPU ticks)	No-match (CPU ticks)
Anchored	832	708
Anchored + Kleene	2811	684
Not anchored + Kleene	2752	1650

least *two* expressions of type (iii) are merged in the same DFA.

In our experience, traffic classification applications are not so prone to state explosion for two reasons. First, the number of patterns in traffic classification is definitely smaller than the ones needed in security applications (hundred against several thousands). Second, we can forge the signatures in order not to trigger state explosion (Section 5.2.3), e.g. avoiding, whenever possible, the Kleene closure and preferring anchored patterns. For instance, the NetPDL signatures used in this paper includes only two patterns of type (iii), one for a protocol encapsulated in TCP (`http`) while the other in UDP (`nt-security-log`), giving us the possibility to split the patterns into two different sets. The two resulting DFAs are reasonably small, each one containing less than 100K states while the total amount of memory used was about 3MBytes, roughly split in half between TCP and UDP. The problem of writing "better" regular expression will be investigated in the next section, which will demonstrate that the form of the regular expression may have an additional impact on the overall performance, particularly in case of a DFA engine.

In case a state explosion may occur (e.g. in case we use the plain `l7-filter` signature database as in Table 3), we can partition the automaton in multiple sub-automaton executed in sequence, and if their cardinality is small (such as we expect to be), we can parallelize their execution on different CPU cores, using an approach investigated in [16, 29].

As the last resort in case a DFA is no longer applicable, we can still use the several solutions (e.g., [17, 18, 20]) available in the literature, with the corresponding worsening in terms of processing cost. However, our tests seem to suggest that the DFA engine is usually applicable in our application scenario.

5.2.3 Writing better signatures for the DFA engine

We evaluated four slightly different signatures that are used to recognize the HTTP protocol by different tools[5]. We measured their execution cost on the

[5] The "anchored" version was derived from the one used in the `tstat` tool: ^((http\/(0\.9|1\.0|1\.1)\ [1-5][0-9][0-9])|(connect|post|get|head|propfind|mSkcol |delete|put|copy|move|lock|unlock)\ ).
The "anchored + Kleene" version was used in the earlier version of `l7-filter`: ^((http\/(0\.9|1\.0|1\.1)\ [1-5][0-9][0-9])|(connect|post|get|head|propfind|mSkcol |delete|put|copy|move|lock|unlock)\ [\x09-\x0d\ -~]*(\ http\/[01]\.[019])).
The "Not anchored + Kleene" was by far the most common one and was derived from the one present in the current version of `l7-filter`: (http\/(0\.9|1\.0|1\.1)\ [1-5][0-9][0-9])|(connect|post|get|head|propfind|mSkcol |delete|put|copy|move|lock|unlock)\ [\x09-\x0d\ -~]*(\ http\/[01]\.[019]).
The "Not anchored, no Kleene" case is omitted since it is equivalent to the second type.

Table 5 Difference in accuracy when using different types of signatures

Signature type	HTTP → unknown (bytes)	unknown → HTTP (bytes)
UNIBS-GT		
Anchored	0%	0%
Anchored + Kleene	0%	0%
POLITO		
Anchored	4.02e-3%	0.38%
Anchored + Kleene	5.03e-3%	0%

POLITO traffic trace in term of clock ticks per packet, differentiating between matching and no-matching cases and averaging the results over the number of HTTP packets inspected.

Table 4 shows that "not anchored" or "with Kleene" signatures often force the algorithm to analyze the entire network packet before being able to conclude if the regular expression matches or not, while the "anchored" usually stops the processing after a few bytes and has therefore the lowest average cost for both the match and no-match cases. The "anchored + Kleene" represents a compromise because of its high cost in case of matching, while the no-matching case is much more favorable because the algorithm usually stops at the beginning of the payload (because of the anchor) and hence the Kleene operator does not have to consume all the payload.

In addition to the processing cost, we measured (Table 5) also the impact of different types of signatures on the classification precision, quantifying the amount of traffic that is (supposedly) incorrectly classified with the simplest signatures. We evaluated the variation in terms of classification accuracy of the different types of signatures against the "not anchored+Kleene" used as baseline. The first result is the amount of traffic (in bytes) that was classified as HTTP with the signature used as reference and that becomes unknown traffic with the signature under testing. The second result refers to the opposite value, i.e. the traffic that went unclassified with the baseline signature and that became HTTP traffic with the other regular expressions.

Due to the limited amount of HTTP traffic present on the POLITO-GT trace, we concentrated our analysis on UNIBS-GT and POLITO traces. Interestingly, the UNIBS-GT traffic trace is classified exactly in the same way, independently from the regular expression used. Very limited variations can be seen on the POLITO trace, mostly referred to some (previously) unknown traffic that becomes HTTP when using the "anchored" signature. Since we do not have the ground truth for that trace, we selected randomly some sessions which changed their classification result and we verified that these are either *(i)* some HTTP-like protocols (apparently generated by our machines on Planetlab) or *(ii)* some non-well formed HTTP requests, possibly generated by other HTTP-like applications.

Summarizing, the "anchored" version of the signature is too permissive and returns some false positives, while the "not anchored+Kleene" is roughly

All these signatures were updated in order to take into account also the new methods defined in HTTP 1.1.

equivalent in term of precision to the "anchored+Kleene" version. In fact, only a few sessions passed from HTTP to Unknown; manual inspection revealed that the most part of these sessions were in fact wrongly classified by the baseline signature. Furthermore, the "anchored+Kleene" signature has a cost in the no-matching case is one forth of the baseline signature, bringing us to the conclusion that a careful selection of the regular expressions can lead to an improvement on execution cost without noticeable loss on classification precision. Furthermore, simplest signatures can contribute to the mitigation of the memory explosion problems highlighted in Section 5.2.2.

Although we know that the definition of simplest signature is not always possible, we demonstrated that this is feasible at least in case of the HTTP protocol and we believe this can be done for many protocols, with a non-noticeable loss in terms of accuracy.

5.3 Performance of the DPI PBFS baseline classifier

Previous section suggested that a PBFS DPI traffic classifier is appropriate for traffic classification when a limited worsening in terms of accuracy is tolerated, and that the DFA algorithm is feasible in our application environment.

This section presents the results that can be achieved by such a classifier, which represents the "baseline classifier" that will be used to evaluate the new optimizations presented in the rest of the paper. This classifier is the same presented in [25] and uses the NetPDL database for the protocol signatures. The current version of the NetPDL database (as of July 2009) includes 72 application-level protocols (39 TCP, 25 UDP and 8 that operate with both TCP and UDP), whose signature are partially derived from the `l7-filter` project. In addition to a pure PBFS classifier, we can analyze "correlated sessions", i.e., the ones created at run-time by some protocols (e.g., FTP, SIP), whose network parameters are negotiated in the control connection, albeit on a per-packet base. These sessions often transport a large amount of data and therefore their impact is noticeable in terms of the amount of bytes classified [24].

Table 6 reports the cost of the main blocks of a PBFS DPI classifiers, calculated per-packet. The cost of the pattern matching can range from 13 to 8900 ticks[6], due to the reasons presented in Section 3.3. The lowest cost is obtained when a packet with 1-byte payload is inspected, while the highest is related to a full-size IP packet (1460 bytes payload) filled with fake data that does no match any pattern. The average cost is referred to the POLITO trace and it is the number of ticks required to inspect a packet submitted to the DPI engine averaged over the total number of packets inspected.

Table 7 reports the classification results (in terms of average processing cost per packet and unknown / misclassified traffic) obtained on the three traffic traces by the baseline DPI classifier. We can note that the average

[6] These numbers have been derived with the DFA algorithm described in [16], which is faster than the one used in [14].

Table 6 Profiling a DPI (PBFS) traffic classifier

Block name	Cost (CPU ticks)
Sess. ID Extraction	78
Sess. Lookup/Update	49
DFA Pattern Matching (min, avg, max)	13 - 4331 - 8900

Table 7 Classification cost and precision of the baseline classifier

Data set	Avgerage cost (CPU ticks/pkt)	Unknown traffic (bytes)	Misclassifications (bytes)
POLITO-GT (tcp)	3209	72.7%	16.9%
UNIBS-GT (tcp)	1240	29.2%	7.73%
POLITO (tcp)	418	5.67%	N/A
POLITO-GT (udp)	1311	0.43%	57.7%
UNIBS-GT (udp)	371	14.7%	1.39%
POLITO (udp)	520	15.3%	N/A

cost per packet is very different from the one presented in Figure 6, which is because here we averaged over all the packets present in the trace (hence taking into account that many packets do not reach the pattern matching engine). Analyzing these results we can see that the traffic mix is very different in our traces; for instance, POLITO-GT has by far the highest percentage of unknown TCP traffic. This is expected because the signatures for the WebTV protocols, which represent the largest part of the traffic captured, are partially unknown and partially derived with reverse engineering (and not very precise). With respect to the UDP portion, the accuracy is even more problematic because of the high percentage of misclassified traffic. Trace UNIBS-GT is less critical than POLITO-GT since the percentage of misclassified traffic is reasonably low; we still have a large portion of unknown traffic due to the use of P2P file sharing applications. For trace POLITO we have only the information of unknown traffic that results to be lower than the other two traces for the TCP case (most hosts on the network use only "standard" applications such as web and email). Vice versa, the unknown UDP traffic is rather high, probably for the presence of Skype traffic that goes mostly undetected.

5.4 DPI optimizations

5.4.1 Snapshot-based classification

We evaluated the impact of the snapshot length optimization by varying the number of bytes analyzed for each packet and by observing the corresponding variation in accuracy and processing cost; then, we compared that number with the same obtained with the baseline classifier. The improvement, shown in Figure 2, is encouraging. For instance, in the best case (obtained with trace UNIBS-GT) the DPI classifier results 7 times faster than the baseline classifier with respect to the TCP traffic.

The POLITO-GT trace is a tough trace for our DPI classifier because of the

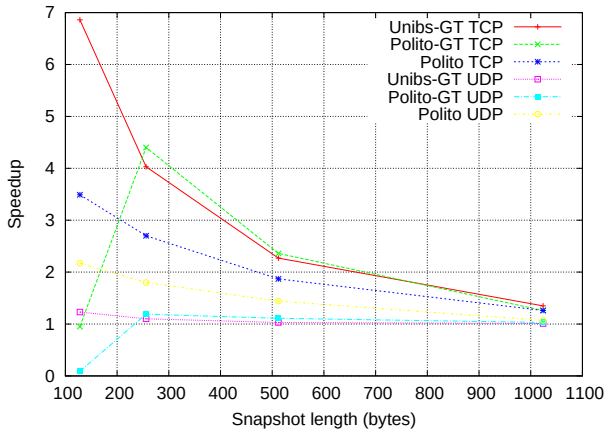

Fig. 2 Performance speedup at different snapshot length.

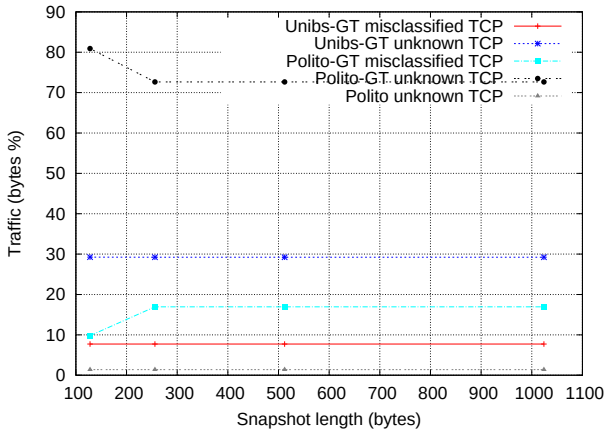

Fig. 3 TCP classification accuracy with different snapshot length.

amount of unclassified traffic present (72.7% of the TCP traffic, as reported in Table 7). For this trace we can observe an interesting phenomenon: processing cost obtained with a 128 bytes snapshot is higher than the one related to the 256 bytes case. This is because the smaller snapshot triggers less false positives on that trace (in fact, misclassifications are lower with 128 bytes), but this implies the growth of the number of classification attempts per session (in fact, the unknown traffic is higher with 128 bytes), lowering the classification speedup consequently. Another observation reveals that UDP traffic shows a limited improvement in terms of processing cost compared to the TCP case; this is mainly due to the average payload size of UDP traffic that is usually smaller than TCP and often below 256 bytes.

Figures 3 and 4 (referred to TCP and UDP) show the corresponding variation of the classification accuracy and demonstrate that in most cases even

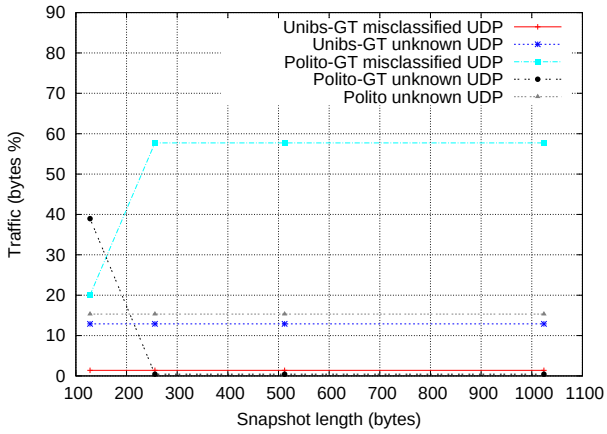

Fig. 4 UDP classification accuracy with different snapshot length.

a snapshot length of 128 bytes does not affect significantly the accuracy; in fact,the result achieved with a snapshot of 256 bytes is almost indistinguishable from the result obtained with full payload. Furthermore, the traffic classified differently includes also some sessions that are misclassified with full payload and that remain unclassified with the snapshot, which represents an improvement since the unknown traffic is usually preferred to misclassifications.

According to these results, an hard limit of 256 bytes seems to be a good tradeoff between the improvements in processing costs (especially for TCP traffic, which improves between 2.8 and 4.4 times the baseline classifier), and the impact on accuracy. For UDP, the improvement is smaller and varies between 1.1 and 1.8 but there are almost no effects in terms of accuracy.

5.4.2 Limiting classification attempts

Table 7 demonstrates that traces with higher percentage of unknown traffic are associated to higher computational costs, suggesting a direct correlation between unclassified traffic and cost. Particularly, Table 8 shows the mean and standard deviation of the number of classification attempts executed over classified sessions; the number of searches is extremely high especially in case of TCP sessions, which last usually longer than UDP ones. Even more interesting is Table 9, which shows the same values but referred to some significant protocols present in our traces. We can observe that some protocols (e.g., Bittorrent, Samba and HTTP) require in average a very low number of classification attempts, suggesting that their signature is able to classify the session within the first packets. Some other protocols (e.g. Telnet and Direct Connect++) require in average an high number of packets to be able to classify the session, suggesting either that the signature is not very accurate or that these are in most cases misclassifications. The analysis over the standard deviation is equally interesting, particularly with respect to protocols that have a low

Table 8 Average number of pattern searches

Data set	$\bar{x}$ (TCP)	$\sigma(x)$ (TCP)	$\bar{x}$ (UDP)	$\sigma(x)$ (UDP)
UNIBS-GT	654	4619	2.62	0.71
POLITO-GT	563	3659	6.05	26.4
POLITO	67.6	1879	9.17	476

Table 9 Average number of pattern searches for some selected protocols

Data set	$\bar{x}$	$\sigma(x)$
bittorrent (tcp)	1	0
samba (tcp)	1.01	0.29
http (tcp)	1.05	15.6
skype (udp)	1.7	437
ssl (tcp)	1.92	267
telnet (tcp)	2599	3276
DC++ (tcp)	30694	60076

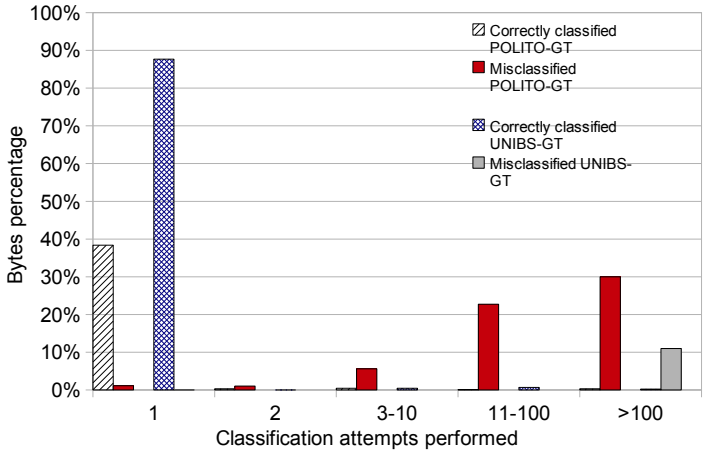

Fig. 5 Distribution of positive matches vs. number of classification attempts performed.

value for the mean and an high value for the standard deviation (e.g. Skype, SSL). These protocols suggest that their signature is reasonably effective (is able to identify the most part of the traffic within the first few packets) but it is not very precise because several sessions are classified after inspecting a large number of packets, which are mostly misclassifications. In all these cases, a reasonable limit on the number of classification attempts per session should not affect the amount of traffic correctly classified, while it should decrease the misclassifications.

Figure 5 shows the percentage of traffic that is either correctly classified or misclassified by the baseline classifier while examining the first N packets of each session. It is evident that the most part of the traffic is correctly classified by examining only the first packet, while the sessions that are classified when examining the N^{th} packet (with $N \geq 2$) is definitely limited. Besides,

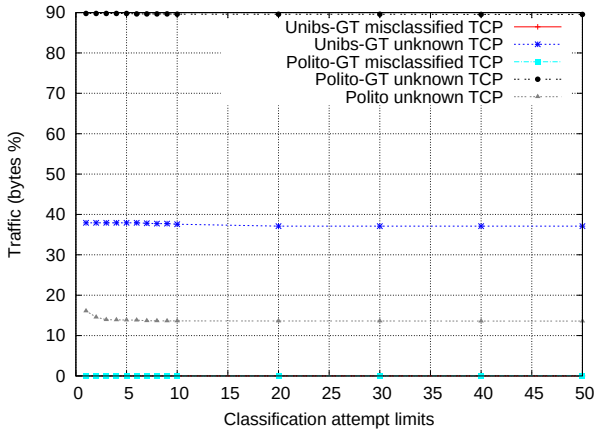

Fig. 6 TCP classification accuracy with different classification attempts limits.

inspecting more packets has the side effect of increasing the percentage of misclassified traffic, because the randomness of application data transported leads to incidentally return a positive match on some "weak" signatures.

Figure 6 confirms that TCP traffic is classified almost entirely at the first packet in both UNIBS-GT and POLITO-GT traces (the curves of unclassified and misclassified traffic does not change sensibly with $N \leq 50$). Considering a limit of $N = 2$, the correctly classified traffic is reduced of 0.95% in the UNIBS-GT trace but misclassifications almost disappear, which vice versa account up to 7.73% in absence of limits. Results on the POLITO-GT are even better, with a loss of 0.20% in terms of correct traffic and almost no misclassifications.

Unfortunately, the analysis of UDP traffic in Figure 7 is less clear. For instance, the amount of correctly classified traffic in the POLITO-GT trace is almost independent from the N limit, because the decrease of the unknown traffic (with higher values of N) is balanced by a corresponding increase of the misclassifications. Vice versa, the UNIBS-GT trace shows an increase in the amount of traffic correctly classified with higher values of N, since the unclassified traffic decreases without a corresponding increase in the misclassifications (which is in some sense expected because the misclassified traffic in this trace is already very low).

For the POLITO trace we can see a decrease in the unknown traffic with higher values of N but we cannot confirm whether this traffic is correctly classified, in both TCP and UDP cases, because we lack ground truth information.

Our analysis suggests that that limiting the number of classification attempts of a DPI classifier is a good strategy for reducing the computational cost introduced by unclassifiable traffic while preserving, and in some cases improving, the classification accuracy and it is particularly effective on traces with high amount of encrypted traffic or P2P applications. In case of TCP traffic, a limit of $N = 2$ seems to be a good tradeoff[7], improving the classifica-

[7] This value is due to the fact that some signatures match the message coming from the

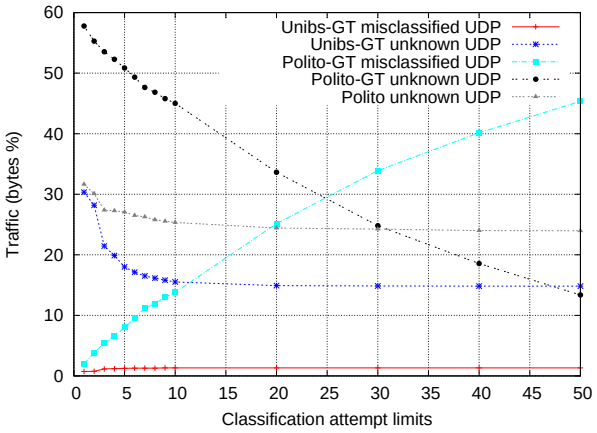

Fig. 7 UDP classification accuracy with different classification attempts limits.

tion accuracy and triggering an improvement of the processing cost that can be up to 50 times in trace POLITO-GT (Figure 8).

With respect to the UDP traffic, this optimization does not seem to guarantee sensible improvements in terms of processing costs even in case of very small values of N (Figure 8 shows 1.2 - 2.9 maximum speedup even with $N = 2$). One reason may be the typical length of a UDP session that is definitely shorter than in the TCP case (e.g. several DNS queries are present in the traces) and hence such a limit on the number of classification attempts may not bring many advantages. Furthermore the impact on the classification accuracy is unclear because the misclassifications are definitely reduced, but the amount of correctly classified traffic may suffer. This may be due to the poor quality of the signatures we use and the fact that some of them require the inspection of several consecutive packets (e.g. RTP, Skype), but this point will surely require further investigations. We suggest a limit of $N = 10$ that has almost no impact on the processing cost (e.g., 1.03 speedup for the UNIBS-GT trace) but it contributes to keep misclassifications at a reasonable value.

5.4.3 Combining the limit on classification attempts and snapshot length

We analyzed the combination of the two previous optimizations in order to determine if they are independent one from the other.

Figure 9, which refers to the TCP traffic of the UNIBS-GT trace, shows the percentage of correctly classified traffic when different snapshot lengths are used, and the number of packets required to classify the session (in fact, the last line of this graph corresponds the data of Figure 5). While for large snapshots almost all the traffic is classified within the first packet, we can observe that the distribution of correctly classified traffic changes when a snapshot < 512

server, which is usually the *second* packet of the session.

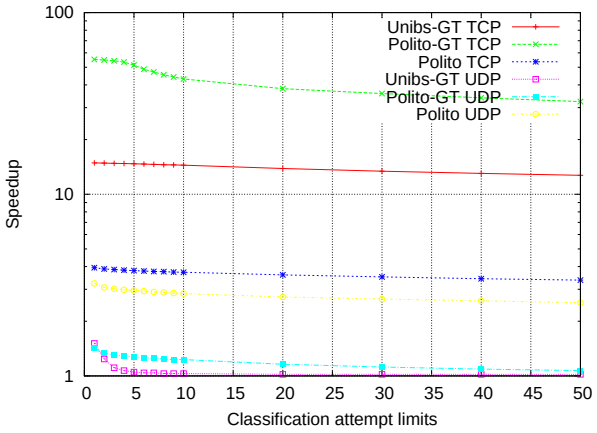

Fig. 8 Performance speedup at different classification attempts limits (Y scale is logarithmic).

Table 10 Effect of snapshot length and classification attempts optimizations on processing speedup in case of TCP traffic

Snap.	**96**	**128**	**256**	**512**	**1514**
Class. att.					
1	19.4	19.6	18.3	16.4	**14.9**
2	19.3	19.1	18.0	16.2	14.8
3	19.2	19.0	17.9	**16.1**	14.8
4	19.1	18.9	17.8	16.1	14.8
5	19.1	18.9	17.8	16.0	14.7
6	19.1	18.8	17.7	16.0	14.7
7	19.0	18.8	**17.7**	15.9	14.6
8	**19.0**	**18.8**	17.6	15.9	14.5
9	19.0	18.7	17.6	15.9	14.5
10	19.0	18.7	17.5	15.8	14.5
No limit	8.49	6.86	4.03	2.27	1

bytes is used; in particular, the number of packets required to correctly classify a session starts growing. This demonstrates that the optimizations are not orthogonal, i.e. the best classification result cannot be derived by using the best snapshot length obtained in Section 5.4.1 and the optimal number of packets derived in Section 5.4.2. As a consequence, we can lose a significant portion of correctly classified traffic if we combine a small snapshot length with a limited number of classification attempts. For instance, the percentage of traffic classified with more than one packet is more than 73% in case of a snapshot length of 96 bytes. In other terms, a limit on the number of classification attempts $N = 1$ would leave more than 73% of the traffic as unclassified, while the same limit operating on full packets would reduce the unknown traffic to less than 2%.

The question is now to determine the best combination of snapshot length and attempts limit, and at which extent this can provide further speedup compared to the two optimizations alone.

Table 10 shows the performance speedup (referred to the UNIBS-GT trace)

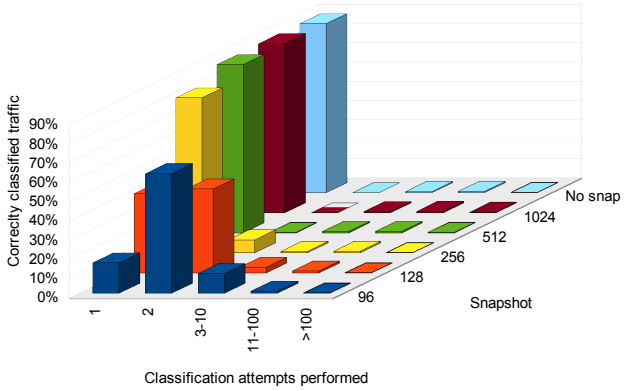

Fig. 9 Distribution of positive matches vs. number of classification attempts performed and vs. snapshot limit.

when combining the two optimizations together, where bold values are in correspondence of the best results, i.e. where the worsening in terms of accuracy remains $< 1\%$. Particularly, the last column represents the results obtained applying only the limit on the number of classification attempts, while the last row reports the results obtained applying only the snapshot length optimization. It evident that even the best combination of the parameters does not provide any noticeable advantage in terms of speedup compared to using the most effective optimization alone. Furthermore, the same analysis repeated on other traces confirm not only the results in terms of speedup, but also that the best combination of parameters is definitely hard to predict.

6 Conclusions

Traffic classification is one of the main components of network monitoring. This paper focuses on applications that may tolerate a limited degree of imprecision (e.g. traffic monitoring, QoS, etc.) and demonstrates that, despite common belief, Deep Packet Inspection is feasible even at very high speed.

This result is achieved thanks to two principles. First, we took into account the typical deployment scenario of DPI technology for traffic classification, which needs to recognize a few hundred protocols. Second, we considered the typical traffic pattern present on nowadays networks, focusing on the average case and avoiding to optimize our solution for corner cases that may be important for security applications, but that are not part of our objectives.

According to the abovementioned principles, we demonstrated that, at least in our traces, a message-based flow-state (MBFS) approach does not provide sufficient advantages that could justify the additional complexity introduced by this technique, and that a packet-based approach is enough for our purposes. Then, we analyzed different pattern matching algorithms and

we concluded that the DFA is feasible in our case, mainly because of the characteristics of our protocol signatures. Furthermore, signatures can lead to very high execution cost if they are not anchored and/or they contain the Kleene closure and we demonstrated that simpler signatures do not necessarily cause a sensible worsening in classification accuracy.

Focusing on a packet-based DPI engine, we exploited the typical structure of protocol signatures and headers and we analyzed the impact of limiting the amount of application-layer data fed to the pattern matching engine. Our results demonstrate that the precision of our DPI classifier does not change when reducing the payload analysis to 256 bytes, with a significant decrease (about 4 times for TCP) in processing complexity. Furthermore, we analyzed the distribution of the classification attempts needed to classify the most part of traffic and we discovered that usually a PBFS DPI classifier requires a few packets to correctly classify a session and that "late" classifications are usually misclassifications. A simple limitation on the number of classification attempts for each session leaded up to a 50x speedup (for TCP traffic) in terms of processing speed compared to the baseline DPI engine, while reducing at the same time the number of misclassications. Finally, we combined the last two techniques together, but we found that the parameters are extremely dependent on the traffic trace considered; our results suggest the adoption of the first technique for TCP traffic (with a limit of 2 packets inspected for each session) and the second for UDP traffic (with a limit at 256 bytes), with an additional limit of maximum 10 packets inspected, which helps to decrease the amount of misclassified traffic.

While the well known problems of the DPI still hold (difficulties in recognizing encrypted/tunneled traffic, extreme sensitiveness to the signature dataset, necessity to manually derive the signatures for new applications, etc.), this paper can change the way DPI is considered at least with respect to processing cost, paving the way for the adoption of packet-based DPI techniques also on very high speed networks. In fact, our results demonstrate that the average cost of the sole pattern matching module (not counting all the other costs present in a real classifier) can account to a few hundred CPU ticks per packets, which means that a 3GHz CPU can potentially handle the impressive number of more than 23M packets per second, i.e. a value comparable to the amount of data transported on a full-duplex 10 Gigabit pipe.

7 Acknowledgement

We would like to thank Luca Salgarelli and Francesco Gringoli at University of Brescia who gave us many suggestions in the earlier part of this work and who contributed to the evaluation of the results presented in this paper.

References

1. D. Moore, K. Keys, R. Koga, E. Lagache, and K. C. Claffy, "The coralreef software suite as a tool for system and network administrators," in *LISA '01: Proceedings of the 15th USENIX conference on System administration*, (Berkeley, CA, USA), pp. 133–144, USENIX Association, 2001.
2. M. Dusi, F. Gringoli, and L. Salgarelli, "Quantifying the accuracy of the ground truth associated with internet traffic traces," *(under submission)*, Jul 2010.
3. M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 1, pp. 5–16, 2007.
4. D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli, "Revealing skype traffic: when randomness plays with you," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 37–48, 2007.
5. A. Este, F. Gringoli, and L. Salgarelli, "Support vector machines for tcp traffic classification," *Elsevier Computer Networks*, vol. 53, pp. 2476–2490, sep 2009.
6. A. Finamore, M. Mellia, M. Meo, and D. Rossi, "Kiss: Stochastic packet inspection," in *TMA '09: Proceedings of the First International Workshop on Traffic Monitoring and Analysis*, (Berlin, Heidelberg), pp. 117–125, Springer-Verlag, 2009.
7. J. Erman, A. Mahanti, M. Arlitt, and C. Williamson, "Identifying and discriminating between web and peer-to-peer traffic in the network core," in *WWW '07: Proceedings of the 16th international conference on World Wide Web*, (New York, NY, USA), pp. 883–892, ACM, 2007.
8. J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms," in *MineNet '06: Proceedings of the 2006 SIGCOMM workshop on Mining network data*, (New York, NY, USA), pp. 281–286, ACM, 2006.
9. T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc: multilevel traffic classification in the dark," in *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, (New York, NY, USA), pp. 229–240, ACM, 2005.
10. S. Zander, T. T. T. Nguyen, and G. J. Armitage, "Self-learning ip traffic classification based on statistical flow characteristics.," in *PAM* (C. Dovrolis, ed.), vol. 3431 of *Lecture Notes in Computer Science*, pp. 325–328, Springer, 2005.
11. A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, (New York, NY, USA), pp. 50–60, ACM, 2005.
12. L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 2, pp. 23–26, 2006.
13. L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *CoNEXT '06: Proceedings of the 2006 ACM CoNEXT conference*, (New York, NY, USA), pp. 1–12, ACM, 2006.
14. N. Cascarano, A. Este, F. Gringoli, F. Risso, and L. Salgarelli, "An experimental evaluation of the computational cost of a dpi traffic classifier," in *Proceedings of IEEE Globecom 2009, Next-Generation Networking and Internet Symposium*, (New York, NY, USA), pp. 1132–1139, IEEE, November 2009.
15. F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso, and K. Claffy, "Gt: picking up the truth from the ground for internet traffic," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 207–218, October 2009.
16. M. Becchi, M. Franklin, and P. Crowley, "A workload for evaluating deep packet inspection architectures.," in *Proceedings of the 2008 IEEE International Symposium on Workload Characterization.*, (Seattle, WA, USA), IEEE, September 2008.
17. R. Smith, C. Estan, S. Jha, and S. Kong, "Deflating the big bang: fast and scalable deep packet inspection with extended finite automata," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 207–218, 2008.
18. M. Becchi and P. Crowley, "Extending finite automata to efficiently match perl-compatible regular expressions.," in *Proceedings of the International Conference on*

emerging Networking EXperiments and Technologies (CoNEXT), (Madrid, Spain), ACM, December 2008.
19. M. Becchi and P. Crowley, "An improved algorithm to accelerate regular expression evaluation.," in *Proceedings of the 2007 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, (Orlando, FL), ACM, December 2007.
20. S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," in *SIGCOMM 2006*, (New York, NY, USA), pp. 339–350, ACM, 2006.
21. V. Vapnik, *Statistical Learning Theory*. John Wiley and Sons, New York, 1998.
22. V. Paxson, "Bro: a system for detecting network intruders in real-time," *Comput. Networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.
23. M. Roesch, "Snort - lightweight intrusion detection for networks," in *LISA '99: Proceedings of the 13th USENIX conference on System administration*, (Berkeley, CA, USA), pp. 229–238, USENIX Association, 1999.
24. A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *In PAM*, pp. 41–54, 2005.
25. F. Risso, M. Baldi, O. Morandi, A. Baldini, and P. Monclus, "Lightweight, payload-based traffic classification: An experimental evaluation," in *IEEE International Conference on, International Conference on Communications (ICC)*, pp. 5869–5875, May 2008.
26. l7-filter, "Application Layer Packet Classifier for Linux," `http://l7-filter.sourceforge.net/`.
27. Computer Networks Group, "NetPDL Protocol Database," `http://www.nbee.org/netpdl`, Politecnico di Torino.
28. M. Becchi, "Regular Expression Processor," `http://regex.wustl.edu`, Washington University, St. Louis, MO, USA, 2008.
29. M. Becchi, C. Wiseman, and P. Crowley, "Evaluating regular expression matching engines on network and general purpose processors," in *Proceedings of the 2009 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'09)*, ACM, 2009.

Author Biographies

Niccolò Cascarano is PhD Student at the Department of Control an Computer Engineering of Politecnico di Torino, Italy. He is author of several papers on traffic classification and high speed packet processing. Recent activities focus on the development of fast regular expression algorithms.

Luigi Ciminiera is full professor at Department of Control and Computer Engineering at Politecnico di Torino, Italy. He graduated in Electronic Engineering at Politecnico di Torino in 1977. His research interests include computer networks and related protocols, distributed systems, he is also involved in research on algorithms and circuits for computer arithmetic.

Fulvio Risso received his Ph.D. in computer and system engineering from Politecnico di Torino in 2000. He is currently Assistant Professor at the Department of Control and Computer Engineering of Politecnico di Torino. His current research activity focuses on efficient packet processing, network analysis, network monitoring, and peer-to-peer overlays.