PPP: Towards Parallel Protocol Parsing

SHAO Yiyang[1, 2], XUE Yibo[2, 3], LI Jun[2, 3]

[1] Department of Automation, Tsinghua University, Beijing 100084, China
[2] Research Institute of Information Technology, Tsinghua University, Beijing 100084, China
[3] Tsinghua National Lab for Information Science and Technology, Beijing 100084, China

Abstract: Network traffic classification plays an important role and benefits many practical network issues, such as Next-Generation Firewalls (NGFW), Quality of Service (QoS), etc. To face the challenges brought by modern high speed networks, many inspiring solutions have been proposed to enhance traffic classification. However, taking many factual network conditions into consideration, e.g., diversity of network environment, traffic classification methods based on Deep Inspection (DI) technique still occupy the top spot in actual usage. In this paper, we propose a novel classification system employing Deep Inspection technique, aiming to achieve Parallel Protocol Parsing (PPP). We start with an analytical study of the existing popular DI methods, namely, regular expression based methods and protocol parsing based methods. Motivated by their relative merits, we extend traditional protocol parsers to achieve parallel matching, which is the representative merit of regular expression. We build a prototype system, and evaluation results show that significant improvement has been made comparing to existing open-source solutions in terms of both memory usage and throughput.

Keywords: traffic classification; deep inspection; regular expression; protocol parsing

I. INTRODUCTION

The Internet has been dramatically changing ever since it appears, and lots of challenges are emerging in network traffic management [1-2]. Recently, there is a growing demand to identify Internet traffic application types, especially by communication carriers. Internet Service Providers (ISPs) often rely on traffic classification to analyze and optimize their networks, driven by several motivations. First, with the increase of the network bandwidth, more and more organizational users prefer different Quality of Service (QoS) for different applications, so that the delivery of high priority traffic can be guaranteed. To meet this need, service providers must have the capability of identifying various applications in the network. Second, security is a critical issue in the Internet. To protect users from financial loss and information leak, service providers need to identify offensive traffic passing through their Autonomous System (AS). Last but not least, concerning the importance of Big Data, the value of traffic data is being paid more and more attentions, especially by those ISPs who control nearly all entrances of Internet traffic. There is no doubt that with the knowledge of application protocol types, ISPs can mine more information and make better use of their traffic data.

In this paper, we propose a novel classification system employing Deep Inspection technique, aiming to achieve Parallel Protocol Parsing (PPP).

For the reason that the aggregation from packets to flows effectively reduces the processing complexity while guaranteeing enough fine-granularity, many traffic identification works focus on the flow level [3-7]. A flow is commonly referred as packets having several identical header tuples, i.e., the five tuples including the source and destination IP addresses, source and destination port numbers, and the transport layer protocol. Definition of a flow is same in this paper. In the early days of the Internet, traffic classification was easy because the designs of applications were relatively simplex and normalized. Almost all popular network applications obey the TCP and UDP ports specified by Internet Assigned Numbers Authority (IANA) [8]. According to these pre-defined port numbers, a direct inspection over packet headers tells which protocol the flow belongs to. However, with more and more applications using ports beyond the IANA specification, port numbers can hardly be associated with certain protocol. Therefore, inspecting packet headers is no longer sufficient to identify the traffic. Deep Inspection (DI), also known as Payload Inspection, becomes necessary in the scenario of traffic classification.

Generally, Deep Inspection utilizes detail information of packet payload and matches it with exact signatures, which are often described in the style of *Regular Expressions* or *Protocol Parsers*. Compressing packet signatures into pieces of regular expressions, Deterministic Finite Automaton (DFA) is commonly used in the stage of classifying. Compiling multiple regular expressions into a single DFA matching engine can achieve parallel classification effect, however, this will lead that DFA needs too many states as well as transitions and may cause state explosion in practical usage. As a comparison, protocol parser translates raw packet payload into high-level representations. Protocol parser has flexible expression ability, which allows the parsers to describe and identify complex protocols. In case of multiple protocols' classification, parsers still achieve low memory usage, but meanwhile, it is time consuming because of the sequential matching.

Despite many real concerns such as encryption and privacy regulations, DI technique continuously plays the leading role on the stage of traffic classification. In theory, exact knowledge of application protocol syntax and full examination of packet payload according to signatures will lead to a perfect result. This leads DI technique to be more competitive compared with other solutions. However, in most scenarios, we need to examine all possible protocols of traffic, and this will cause either memory explosion or time consuming under the deployment of state-of-the-art solutions. With the increase of network bandwidth, this problem becomes dramatically compelling. In summary, traffic classification is still not out of the woods, and existing DI techniques can not completely meet practical requirements.

In this paper, we propose PPP, a *Parallel Protocol Parsing* system, which achieves practical traffic classification. Inheriting the merit of parallel matching in regular expression based methods, we extend original application protocol parser by pre-filtering traffic into different corresponding candidate subsets, thus make the protocol parsers work efficiently. In this way, traffic identification system based on PPP can organically gain both high classification speed and low memory usage.

Main contributions of this paper include:

- *Inheriting main merits of existing DI techniques*. An in-depth analytical study has been made to motivate the PPP, which organically combines merits of existing popular DI techniques to enhance traffic classification.
- *Fast speed and low memory usage.* Our approach requires a light-weighted finite automata and needs a few times of parsing, which guarantees fast processing speed as well as efficient memory usage.

The design of PPP system makes a novel attempt aiming to achieve parallel protocol parsing in the area of traffic classification. With the trace datasets collected from campus

network, experimental results illustrate that PPP system has great superiority over existing open-source application identification systems, either in classification speed or memory usage.

The rest of this paper is organized as follows. Section II introduces the related work of traffic classification. Section III analyzes respective merits of existing popular DI techniques and section IV proposes our algorithm design. PPP classification system's architecture is presented in section V. Section VI shows evaluation and analysis of experimental results. Finally in section VII, we concludes the paper.

II. Related Work

In the area of traffic classification, research of Deep Inspection commonly involves two aspects. One is the construction of protocol patterns (also known as signatures), the other is the detection algorithm based on these patterns.

Construction of protocol patterns is usually done offline. Patterns of protocol with public specifications are easy to be established under the specifications' guidance. To those protocols without specifications, patterns are usually constructed by analyzing captured traces, which often involve a lot of manual effort and additional complexity during validation. Meanwhile, the diversity of Internet applications and their frequent updates also drive the classifier to regenerate patterns to ensure accuracy. L7-filter community depends on volunteers all around the world to contribute the construction and update of patterns. Many researches study to generate protocol signatures automatically to fundamentally solve this problem [9-12].

With discovered patterns, OpenDPI implements the pattern matching with string comparison in C programs, while L7-filter calls the POSIX regexec() API for regular expression matching (more technique details of L7-filter and OpenDPI are described in Section III). Some semiconductor companies, such as Cavium, build DFA/NFA based coprocessor for pattern matching [13]. Using this kind of hardware acceleration, a DI solution can inspect packets at a very high speed, and thus is more suitable for deployment in backbone routers.

III. Motivation

Our origin idea starts from the analytical study of existing popular DI techniques, which will be shown in this section. We introduce the usage of regular expression in traffic classification first, and then analyze application protocol parser to inspire our design.

3.1 Regular expressions matching engine

Regular expression has been widely used to represent signatures in deep inspection systems, due to its expressive power and flexibility for describing protocol patterns. All protocol signatures in the L7-filter are written in regular expression. In addition, Bro, which is a famous intrusion detection system, uses regular expressions as its pattern language in the protocol identification module [14]. Meanwhile, matching engines used in traffic classification usually have multiple regular expression patterns.

Deterministic Finite Automaton (DFA) and Nondeterministic Finite Automaton (NFA) are commonly used for high-speed regular expression matching. Theoretically, a regular expression of length n can be compiled into an NFA with $O(n)$ states. In the worst case, when an NFA with m-state is converted into a DFA, it may generate $O(2^m)$ states. However, the processing complexity is $O(1)$ to DFA and $O(n^2)$ to NFA for each input character. Considering regular expression matching for high speed packet payload scanning, both DFA and NFA are not feasible. To handle k regular expressions with total length of kn (k is usually in hundreds or even in thousands, n is average length of these regular expressions) using NFA, they can be compiled in one or k automata. In either way, $O(n)$ states may be active concurrently, which results in low performance and large per-flow states to be

maintained. While considering DFA, since the composite DFA may experience exponential growth in state size in most cases, it is usually infeasible to compile a large signature set into a single composite DFA.

Even so, in practice, DFA is widely used for the concerning of time-efficiency. In order to reducing memory usage, many novel solutions have been proposed, which are beyond the topic of this paper. The main merit of DFA we attempt to inherit in this paper is parallel matching effect on the premise of low memory usage. To achieve the goal, we organically combine DFA with protocol parser, which will be introduced next.

3.2 Application protocol parser

In computing, a semantic *parser* refers an *interpreter* or *compiler* that constructs structural representation based on input text, checking for correct syntax in the process. Particularly, in the scenario of traffic classification, based on prior knowledge of each specific protocol, *application protocol parsers* translate raw packet payload into high-level representations of the traffic [15]. As a result, additional information such as behavioral patterns and statistical indicators can be employed to precisely describe application protocols, thus achieving more flexibility and higher accuracy than regular expression based methods. A protocol parser can be built manually, or automatically generated by compilers, and each different application needs to build its own protocol parser.

Derived from the commercial PACE product from IPOQUE, OpenDPI is an open-source protocol parser library for traffic classification. OpenDPI shares a parser library up to 101 different protocols in latest version, including a majority of common application protocols. All parsers in OpenDPI are demonstrated in the style of C program, and a real-time detection rate with near 100% reliability is officially announced, including popular P2P protocols.

To demonstrate the flexibility of parsers clearly, we use SSH protocol parser of OpenDPI as an example. In the process of judging a flow's application-level protocol, SSH protocol parser follows two stages:

- *Packet length judging stage:* if a flow is classified to be SSH traffic, the prerequisite is that it must have a packet payload length between 7 and 100.
- *Packet content judging stage:* the presence of precise signature "SSH-" at the start of packet payload is also a must for SSH traffic.

And as a comparison to protocol parser, classification of SSH protocol based on regular expression follows the pattern `^SSH-[12]\.[0-9]`, which only processes the content of packet payload, ignoring other information such as packet length.

When using protocol parsers to identify network traffic, input packets are serially analyzed by each corresponding parser until it matches one specific protocol parser. Compared with regular expression matching based on finite automata, protocol parser can be compiled as a light-weighted tree structure, and quickly determine whether a flow match a specific application-level protocol. Thus, traffic classification based on protocol parser can achieve high identification speed as well as low memory usage. However, when the number of protocols to be identified increases, classifier has to serially scan each corresponding protocol parser until the result of "match" or "unknown" has been given. If there are N candidate protocols, in the worst case, the classifier gets the final result after N times parsing. On the contrast, regular expression matching based on finite automata can determine different application types through its final state, which attains the effect of parallel matching.

Motivated by the parallel matching effect of regular expression matching engine, carrying on the flexibility and low memory usage of application protocol parser, our design of parallel protocol parser aims to inherit these two merits aiming to achieve practical traffic classification.

IV. Parallel Protocol Parsing

Inspired by the relative merits of regular expression and protocol parser, we extend traditional protocol parsing methods to achieve parallel search via single match. We analyze the characteristics of regular expressions in traffic classification first in this section, and then demonstrate our design in detail.

4.1 Characteristics of regular expressions in traffic classification

Traffic classification and Intrusion Detection System (IDS) are two most widely used forms of deep inspection technique. Although the cores of both DI system leverage similar regular expression matching engine, they have some essential differences. The first is the matching rate. Little traffic finds a match in IDS system when processing real life traffic. For example, over 90 percent of the traffic does not match any signature in the IDS presented in [16]. On the contrary, almost all the flows match certain signature in a traffic classification system.

The second difference is that most of the signatures used for traffic classification are anchored, which means the signatures should be matched only from the beginning of a flow. Around 80% percent of the signatures in L7-filter are anchored [17]. Through a detailed study of popular application-layer protocols, we found that this characteristic in signature-set is reasonable. Generally it has four classes of Internet application-layer protocols: struct-style binary protocols, IETF-style protocols, structured binary protocols, and structured text protocols [18]. The parsers of these kinds of protocols need to parse from the beginning, and most of the Internet application-layer protocols belong to these four classes. That's the reason why the majority of signatures using for traffic classification are anchored.

Knowing the characteristics of regular expression in traffic classification, we extract the partially exact part from anchored patterns, and then compile these partial patterns into a DFA. The final state of DFA can illustrate corresponding partial pattern. Then based on this matched pattern, we can conclude the candidate original patterns and protocol types.

4.2 Parallel parser based on prefiltering

In the scenario that there are many candidate protocol patterns to be classified, a single DFA compiled by all original protocol patterns will cause state explosion and use up memory. Inspired by the characteristics of regular expression patterns using for traffic classification, we simplify original patterns aiming to avoid explosion and filter the traffic into few possible protocols. Then we use corresponding protocol parsers to identify the accurate application of traffic. In this way, we achieve the goal of Parallel Protocol Parser (PPP), ensuring low memory usage as well as fast classification speed. Main aspects of PPP are introduced as follow.

4.2.1 Extracting max-prefix from original patterns

An anchored pattern means it starts with several exact chars, for example the pattern `^abc.*def` has an anchored subpattern `abc`. Thus, we define the *max-prefix* of a pattern as the longest exact anchored subpattern in the original pattern. So the max-prefix of pattern `^abc.*def` is `abc`. Because most original regular expression patterns using for traffic classification are anchored, we can extract max-prefix of these patterns. Using regular expression library of L7-filter as testbed, max-prefix of all anchored patterns are extracted. The max-prefix length are shown in figure 1. Noted that among all 114 original patterns there are 82 anchored patterns, however, the total number of max-prefixes is beyond the number of anchored patterns. This is because some patterns have multiple max-prefixes, for example, pattern `^abc[1-5].*def` have five different max-prefixes.

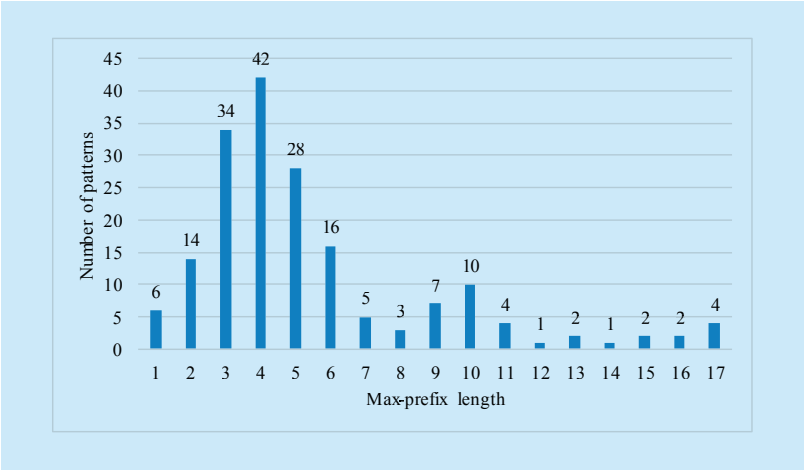

Fig.1 *Max-prefix length of original patterns*

Algorithm 1: Group merging algorithm

Input: Set G, where $G = \{ G_i \mid G_i \text{ is a hash entry}, G_i \neq \emptyset\}$
Begin:
1: ***for*** $G_i \in G$ ***do***
2: ***for*** $G_j \in G$ ***do***
3: ***if*** *similarity*$(G_i, G_j) \geq$ *threshold* ***then***
4: G_i.*hashkey.mix*(G_j.*hashkey*)
5: G_i.*protocols.mix*(G_j.*protocols*)
6: *G.remove*(G_j)
7: ***end if***
8: ***end for***
9: ***end for***

4.2.2 Hashing max-prefixes into different groups

After the extraction of max-prefixes, we hash them into different groups. As figure 1 shows, over 89% of max-prefixes have a length longer than three. So we select the first three chars of each max-prefix as hash key, and for those max-prefixes which length are shorter than three, they will be hashed into all possible hash entries. As an example, a pattern with max-prefix `ab` will be hashed into all entries that begin with `ab`. Noted that some entries in hash table may have more than one items, which means different protocols may have same max-prefix (limited within the first three chars).

4.2.3 Optimizing grouping result

The hash table has 256*256*256 = 16,777,216 entries, which is far more beyond the number of max-prefixes. So in the first step, we remove those empty entries. Then we merge the groups based on their similarity and rewrite corresponding patterns. Algorithm 1 shows the steps of group merging.

The similarity between G_i and G_j is defined as below:

$$similarity(G_i, G_j) = \frac{|G_i \cap G_j|}{\min(|G_i|, |G_j|)}$$

where $|G_i|$ means the number of protocols in this hash entry.

Each non-empty hash entry has two elements: hash key and its corresponding protocols. In the condition that the similarity of two hash entries is greater than the threshold, which is set as 0.8 in our experiment, we merge hash keys and corresponding protocols of the two entries. Mergence of hash keys means to combine two hash keys into a single regular expression. For example, two hash keys `abc` and `abd` will be merged as regular expression `ab[cd]`. Noted that the new regular expressions generated in this way only contains exact characters, character classes, alternation and other necessary metacharacter, without quantifiers like star and plus, which are the crime culprit of state explosion.

After merging all possible entries, we get a new set of regular expressions with its corresponding protocols. The size of this new set reduce a lot compared with original hash table. And the size of DFA generated by all new regular expressions is reasonable. The merging result is shown in figure 2, which illustrates the relationship between each regular expression and its corresponding protocols. From figure 2 we can see that about 65% regular expressions only relate to a single protocol, and the average number of corresponding protocols is 1.44. So after being filtered by these newly generated regular expressions, traffic to be classified only need to be parsed within two times on average.

4.2.4 Parsing candidate protocols

Given the result of DFA matching state based on newly generated regular expressions, we

get all candidate protocols of the traffic. Traffic to be classified will be parsed by each candidate protocol's parser in turn. What's more, the protocol parser library should be rewritten according to each corresponding regular expression, because the regular expression not only filters the traffic into candidate protocol set, but also provides string matching information. Reducing duplicated string matching process in parsers can optimize processing time of protocol parsers.

4.2.5 Processing non-anchored patterns

Above we describe the processing of anchored patterns. For those non-anchored patterns, we remain them original. If the result "not match" is given by DFA matching engine, which means the traffic belongs to non-anchored patterns, original protocol parsers will be used. Though this will lead a poor performance, we argue that it is negligible for the reason that most popular protocols are anchored and leave this possible promotion in our future work.

V. PPP Classification System

According to the design above, we construct our prototype system towards practical traffic classification, and the system architecture is introduced in this section.

5.1 System overview

Classification of a flow in PPP system involves a number of steps. Each network packet is sent to DFA matching engine at first. Given the result of DFA matching state based on newly generated regular expressions and its corresponding protocols, we can get the candidate protocol set of this packet. Then the protocol parsing engine gives the packet's application-level classification result based on the candidate protocol set. Generally, as figure 3 shows, the series of procedures above can be divided as two components:

- *Pre-filtering Component*. In this component, the input packet is processed by DFA engine to gain the matching state that marks the candidate protocols of this packet.

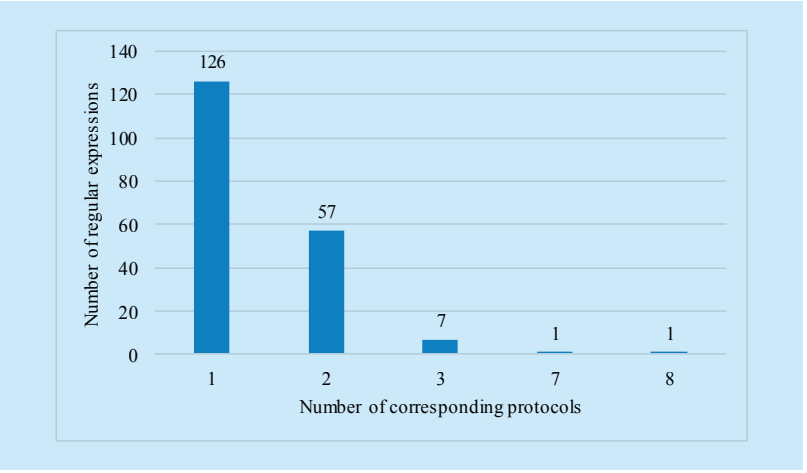

Fig.2 *Number of corresponding protocols of each regular expression*

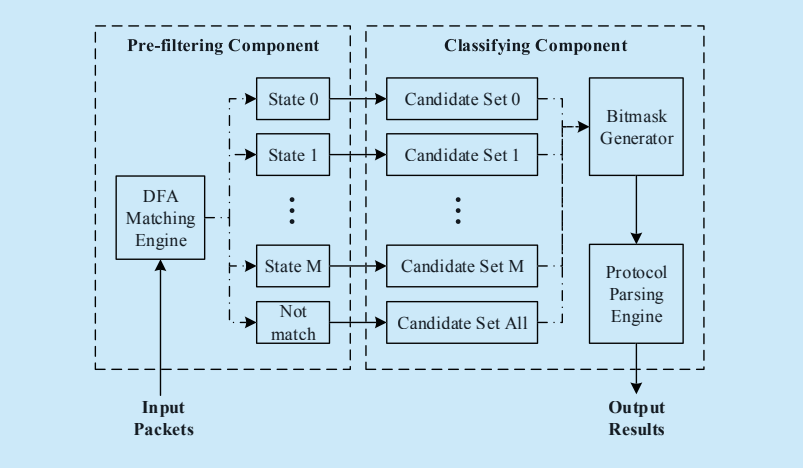

Fig.3 *Framework of PPP system*

- *Classifying Component*. Based on the matching state of pre-filtering component and pre-defined candidate protocols corresponding to this state, a bitmask is compiled by the bitmask generator. Then, this bitmask is used to efficiently decrease the protocols to be scanned in the parsing engine.

5.2 Pre-filtering component

Figure 4 demonstrates the detailed processing procedures of PPP system. Pre-filtering component extracts each input packet's header first, and then involves a query of flow table. If flow record of this packet does not exist in flow table, a new flow will be settled into flow table. Next goes with the judgment that if this

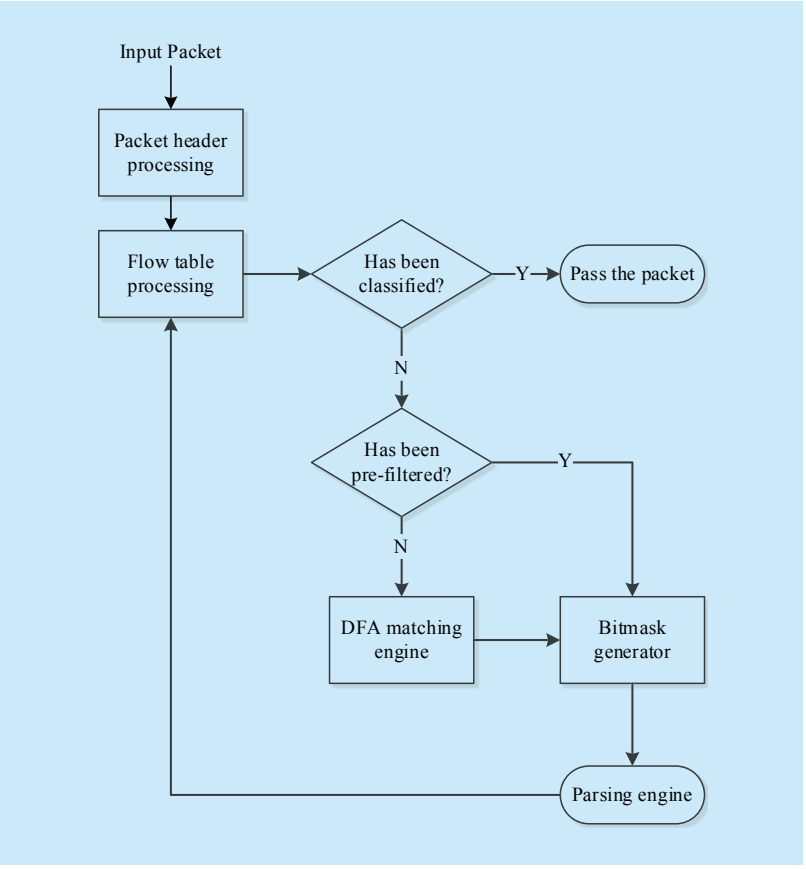

Fig.4 *Processing procedures of PPP system*

flow has been classified before (in the scenario of a new flow, it will directly skip to DFA matching engine). In case of a pre-classified flow, subsequent packets of this flow will be forwarded directly. Otherwise if the flow's application-level information still remains unknown, it needs to determine whether this flow already has corresponding candidate protocol set. For the flow which has been pre-filtered before, a corresponding bitmask will be returned for next component's processing. The flow without candidate protocol set, however, has to be matched in DFA engine. Finally, the matching state of DFA engine represents output result of pre-filtering component, and will be utilized in the classifying component.

5.3 Classifying component

Inheriting the matching state of pre-filtering component, the first step in this part is to generate corresponding bitmask. This step is easy to accomplish because there is a direct mapping relationship among the matching state, candidate protocol set and bitmask. Then, the bitmask is used to activate the engine to parse related protocols. Candidate parsers have to sequentially identify the packet. Once any protocol parser gives the result "match", the classification phase will be terminated and mark the result of this packet with this parser's application protocol type. If all parsers mismatch the packet, result will be "unknown". Finally, the classifying component feedbacks the result to flow table to make later classification efficient.

VI. Evaluation and Analysis

In this section, we evaluate the effectiveness of PPP system. Methodology, testbed setup and evaluation metrics are introduced first to make our experiment clear. Then the performance result of PPP system shows that significant improvement has been made comparing with existing open-source classifier in terms of both memory usage and throughput.

6.1 Methodology

Widely used in many academic research [19-21], L7-filter and OpenDPI are open-source traffic classification system employing DI technique. Protocol patterns in L7-filter are described in regular expressions, while OpenDPI uses protocol parsers. Based on regular expression library of L7-filter we construct the pre-filter component of PPP system, and the parsers in PPP evolve according to OpenDPI. At the same time, the performance of these two systems have been set as benchmark in our experiment.

6.2 Testbed setup

The trace set we use for evaluation is real traffic traces, which were collected at 1) a campus network (includes over ten thousands of servers), 2) a large office building network (includes thousands of servers). The trace sets consist of four traces: *trace_riit*, *trace_seg*, *trace_ap01* and *trace_ap02*, and the global statistics of these traces are shown in Table

I. All these traces were collected in different months during the year 2010. The evaluation platform is a generic PC, with an Intel(R) Core(TM)2 Duo CPU P7450 (2 cores@2.13 GHz) and 2 GB DDR-III memory.

6.3 Evaluation metrics

To characterize the classifier's accuracy, we use common metrics known as *True Positive* (TP), *True Negative* (TN), *False Positive* (FP) and *False Negative* (FN). The correct results consist of two parts: true positive samples and true negative samples. Utilizing these concepts, we use the performance parameters: *Precision*, *Recall* and *Accuracy* [22].

To characterize the classifier's classification speed, we use the concept of *throughput*, which is defined as the size of trace file divided by the execution time. And we use average memory usage during the execution period to evaluate memory occupation.

6.4 Performance result

6.4.1 Accuracy analysis

The accuracy of PPP system is not the most important issue in this paper because the libraries of regular expressions and protocol parsers are employed from open-source achievements, so the accuracy of PPP is not determined by itself. Here we demonstrate the accuracy of PPP system aiming to ensure that our design is reasonable, which means the design of parallel protocol parser based on regular expression is compatible in majority cases. Table II shows the classification accuracy of several popular protocols.

From the metrics shown in table 2 we can see that in nearly all cases the classification results are correct. So we can conclude that protocol parsers are compatible with the filtering by regular expressions. The only outlier in table II shows that regular expression pattern of BitTorrent protocol conflicts with its parser in some cases. After manual examination, we find that this is caused by the difference in patterns between latest version OpenDPI and L7-fiter, which is out of consideration of this paper.

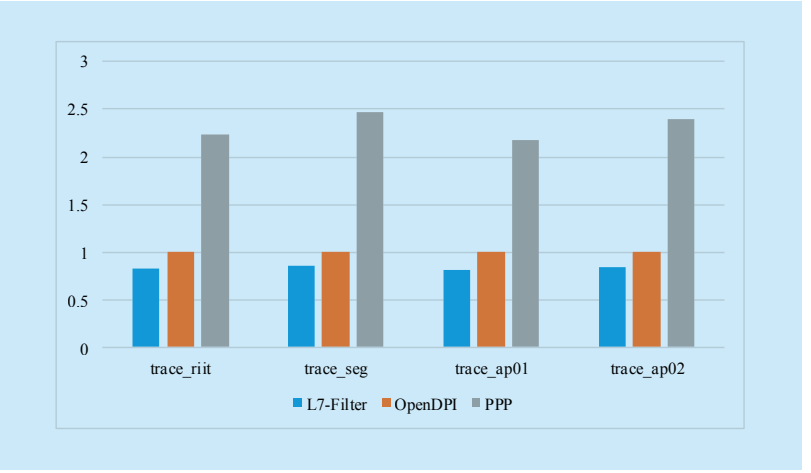

Fig.5 *Comparison of throughput*

Table I *Global statistics of traces*

Trace	Packet #	Flow #	Size (MB)
trace_riit	1,217,285	18802	1901.6
trace_seg	889,967	13890	673.7
*trace_ap*01	1,529,375	19699	1903.5
*trace_ap*02	1,217,285	12738	1900.8

Table II *Classification results*

Protocol	Accuracy	Recall	Precision
HTTP	97.12%	97.19%	99.50%
POP3	100%	100%	100%
SMTP	100%	99.45%	99.99%
SSH	100%	100%	100%
BitTorrent	100%	66.7%	99.97

6.4.2 Throughput analysis

Under the assumption that network traffic protocols are subject to random distribution and each protocol's parsing time is same, theoretically, OpenDPI needs *(N+1)/2* times parsing to get a final result, where *N* stands for the number of active protocols. In the latest version of OpenDPI, there are 101 different protocols, so averagely it needs 51 times parsing per classification in theory. Ignoring the time cost by DFA matching engine, PPP system only needs 1.44 times parsing, about 35 times faster than OpenDPI. However, this theoretical value could hardly to achieve. Our experiment result shows in figure 5. We can see that on

average PPP's throughput is 2.3 times faster than OpenDPI. This is mainly because the following reasons:

- A majority of Internet applications including Web use HTTP/HTTPS protocol to transfer data, and OpenDPI designs to start classification from HTTP/HTTPS protocol parser. So the average parsing times of OpenDPI reduce a lot compared with the theoretical value.
- Matching time of DFA engine cannot be ignored compared with parsing time, and the difference between protocols result in different parsing time among parsers.

Despite the gap between theoretical value and experiment result, as figure 5 shows, the throughput of PPP performs much better than that of OpenDPI as well as L7-filter, illustrating a novel design of PPP.

6.4.3 Memory usage analysis

During multiple times of experiments, our PPP system occupies about 47.2 MB memory on average, while the number of OpenDPI is 41.7 MB. This is reasonable because we filter the traffic with a light-weighted DFA matching engine to achieve the effect of parallel parsing. We leaves the memory usage of L7-filter to be "undeterminable" because during the experiments of L7-filter, memory allocation is almost full, illustrating the appearance of state explosion. However, this does not influence the conclusion that PPP system achieves low memory usage as well as fast classification speed compared with existing open-source traffic classification system.

VII. Conclusions

In this paper, we propose a novel traffic classification system employing Deep Inspection, aiming to achieve Parallel Protocol Parsing (PPP). We start with an analytical study of the existing popular DI methods, namely, regular expression based methods and protocol parsing based methods. Motivated by their relative merits, we extend traditional protocol parsing methods to achieve parallel search via single match, which is the representative merit of regular expression methods. We build a prototype system and evaluation results show that significant improvement has been made comparing to existing open-source classifier in terms of both memory usage and throughput.

In our future work, we will deploy PPP system on multi-core platform to make further promotion. We will also try to optimize the design of protocol parsers, aiming to achieve the parallel effect naturally by the parser itself.

Acknowledgements

This work was supported by the National Key Technology R&D Program of China under Grant No.2012BAH46B04.

References

[1] LABOVITZ C, IEKEL-JOHNSON S, MCPHERSON D, et al. Internet Inter-Domain Traffic[J]. ACM SIGCOMM Computer Communication Review, 2011, 41(4): 75-86.

[2] CALLADO A, KAMIENSKI C, SZABÓ G, et al. A Survey on Internet Traffic Identification[J]. Communications Surveys & Tutorials, IEEE, 2009, 11(3): 37-52.

[3] KARAGIANNIS T, PAPAGIANNAKI K, FALOUTSOS M. BLINC: Multilevel Traffic Classification in the Dark[J]. ACM SIGCOMM Computer Communication Review, 2005, 35(4): 229-240.

[4] Moore A W, Zuev D. Internet traffic classification using bayesian analysis techniques[J]. ACM SIGMETRICS Performance Evaluation Review, 2005, 33(1): 50-60.

[5] YANG B, HOU G, RUAN L, et al. SMILER: Towards Practical Online Traffic Classification[C]// Proceedings of the 7th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. ACM, 2011: 178-188.

[6] L7-Filter[EB/OL]. http://l7-filter.clearfoundation.com/

[7] OpenDPI[EB/OL]. http://www.opendpi.org/

[8] IANA Service Name and Transport Protocol Port Number Registry[EB/OL]. http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml

[9] HAFFNER P, SEN S, SPATSCHECK O, et al. ACAS: Automated Construction of Application Signatures[C]//Proceedings of the 2005 ACM SIGCOMM Workshop on Mining Network Data. ACM, 2005: 197-202.

[10] YE M, XU K, WU J, et al. Autosig-Automatically Generating Signatures for Applications[C]// Proceedings of the 9th IEEE International Con-

ference on Computer and Information Technology. IEEE, 2009, 2: 104-109.
[11] WANG Y, XIANG Y, ZHOU W, et al. Generating Regular Expression Signatures for Network Traffic Classification in Trusted Network Management[J]. Journal of Network and Computer Applications, 2012, 35(3): 992-1000.
[12] YUAN Z, XUE Y, DONG Y. Harvesting Unique Characteristics in Packet Sequences for Effective Application Classification[C]// Proceedings of the 1st IEEE Conference on Communications and Network Security. IEEE, 2013: 341-349.
[13] Cavium[EB/OL]. http://www.cavium.com/
[14] Bro[EB/OL]. http://www.bro.org/
[15] PANG R, PAXSON V, SOMMER R, et al. binpac: a YACC for Writing Application Protocol Parsers[C]//Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement. ACM, 2006: 289-300.
[16] SOURDIS I, DIMOPOULOS V, PNEVMATIKATOS D, et al. Packet Pre-Filtering for Network Intrusion Detection[C]// Proceedings of the 2nd ACM/IEEE Symposium on Architecture for Networking and Communications Systems. ACM, 2006: 183-192.
[17] HE F, XIANG F, SHAO Y, et al. Accelerating Application Identification with Two-Stage Matching and Pre-Classification[J]. Tsinghua Science & Technology, 2011, 16(4): 422-431.
[18] ALBRECHT D R. High Performance Network Intrusion Detection: a New Paradigm is Needed[D]. University of Illinois, 2010.
[19] GUO D, LIAO G, BHUYAN L N, et al. A Scalable Multithreaded L7-Filter Design for Multi-Core Servers[C]//Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. ACM, 2008: 60-68.
[20] GUO D, LIAO G, BHUYAN L N, et al. An Adaptive Hash-Based Multilayer Scheduler for L7-Filter on a Highly Threaded Hierarchical Multi-Core Server[C]//Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. ACM, 2009: 50-59.
[21] MUNOZ A, SEZER S, BURNS D, et al. An Approach for Unifying Rule Based Deep Packet Inspection[C]// Proceedings of the IEEE International Conference on Communications. IEEE, 2011: 1-5.
[22] NGUYEN T T T, ARMITAGE G. A Survey of Techniques for Internet Traffic Classification Using Machine Learning[J]. Communications Surveys & Tutorials, IEEE, 2008, 10(4): 56-76.

Biographies

SHAO Yiyang, is currently a Ph.D. student at Tsinghua University, Beijing, China. He received the B.S. degree in the Department of Automation from Tsinghua University, Beijing, China, in 2011. He has been an IEEE student member since 2013. His research interests focus on security issues of network especially on traffic measurement and classification. Email: shaoyy11@mails.tsinghua.edu.cn

XUE Yibo, is IEEE/ACM member and CCF senior member. He received his B.S. degree and M.S. degree in Computer Science from Harbin Institute of Technology in 1989 and 1992, respectively, Ph.D. degree in Institute of Computer Technology from Chinese Academy of Science in 1995. He is currently a professor in the Research Institute of Information Technology (RIIT) at Tsinghua University. His main research interests are in the areas of network security and computer architecture. *The corresponding author of this paper. Email: yiboxue@tsinghua.edu.cn

LI Jun, is currently Professor and Dean of Research Institute of Information Technology (RIIT), Tsinghua University. He is also Executive Deputy Director of the Tsinghua National Lab for Information Science and Technology. He holds a PhD degree in CS from New Jersey Institute of Technology (NJIT), and MS and BS degrees in Automation from Tsinghua University. He is a member of IEEE since 1996, and his research interest is in network security and Software Defined Network (SDN). Email: junl@tsinghua.edu.cn