MP-ROOM: Optimal Matching on Multiple PDUs for Fine-Grained Traffic Identification

Hao Li, *Student Member, IEEE*, and Chengchen Hu, *Member, IEEE*

***Abstract*—This paper studies the fine-grained traffic identification (FGTI) for better understanding and managing networks. Instead of only indicating which application/protocol that a packet is related to, FGTI maps the traffic packet to a meaningful user behavior or application context. In this paper, we first propose rule organized optimal matching (ROOM), which splits the identification rules into several fields and elaborately organizes the matching order of the fields. As a result, ROOM can only activate the matching operations on a (small) part of the rules that could be possibly hit. We formulate the optimal rule organization problem of ROOM mathematically and demonstrate it to be NP-hard, and then we propose a heuristic algorithm to solve the problem with the time complexity of $O(N^2)$ (N is the number of fields in the rule set). Based on ROOM, we further propose MP-ROOM, which is extended to well support the rules cross multiple protocol data units (PDUs) for traffic identification. In addition, we implement a prototype system including MP-ROOM and related work for evaluations. The evaluations show very promising results: 1.5 $\sim$ 71.3 times throughput improvement is obtained by MP-ROOM in the real system with less than 300-MB memory consumption. With multiple-thread parallel programming, we successfully achieve the throughput over 40 Gb/s for real traces.**

***Index Terms*—Deep packet inspection, semantic-based rules, traffic identification.**

I. Introduction

THE emerging cloud computing and mobile Internet are quite successful in business, which have generated huge amount of network traffic from hundreds of thousands of new applications [1]. As a result, fine-grained traffic identification becomes critical for service providers and network operators to measure and monitor the network in a deeper level and empowers them to better manage and control the cloud and mobile services [2].

Generally speaking, four kinds of identification solutions have been proposed in the literature to detect the specific applications or protocols, i.e., header/port based method, content based method, machine learning method, and Deep Packet Inspection (DPI) based method [3]–[8]. The traditional traffic identification reports at the protocol or application granularity, which can only tell what application generates the certain packet, e.g., QQ or Skype. We call such identification as Coarse-Grained Traffic Identification (CGTI), which fails to provide sufficient information for nowadays' complicated use. The following scenarios have motivated us in this paper to investigate Fine-Grained Traffic Identification (FGTI).

- Profile users' preference [9]. It is of great interest for advertising and competition analysis if users' preferences can be sketched. However, the native applications become more and more comprehensive, that one application often provides several functions for users. For example, a user who likes tweeting and another one who likes reading others' tweets should belong to different user groups, though both of these two functions are provided by Twitter. We should tag the first one as an "active user" and a "passive user" for the later one, instead of both "twitter user." CGTI only reports at the application granularity and cannot provide such kind of ability.
- Charge on contents rather than bits. It has been advocated for years to charge differentially according to contents, which will better motivate the service provider to sharp the Service Level Agreements (SLAs) and better balance the benefit/profit among end users, service providers and content providers [10]. A premise for sharping the SLA is to differential the content from the traffic and makes an authoritative report, which CGTI cannot provide.
- Finer network measurement [11]. Quality of Experience (QoE) is a measure on user's experiences with a function of the application service. It is worth noting that the objective measure of QoE must bind with a specific function of an application. The traffic awareness therefore becomes an important component for QoE. However, different types of content can be transferred over same application, such as text messaging and file transferring by QQ, or plain text and video over HTTP protocol. CGTI cannot differentiate the traffic from others precisely.

FGTI provides finer identification ability than CGTI [12], which tells not only what application generates the certain packet, but also what user behavior/semantic the packet performs, such as "tweeting using Twitter application with an iPhone" and "transferring files using QQ in a PC." FGTI therefore leads to a better understanding of the network as well as the user. An intuitive stab of FGTI is using transitional DPI-based method to match the packet header as well as the packet payload with the pre-defined rules with simple strings or regular expressions (regex). However, string or regex cannot accurately describe the finer-grained behavior of applications,

Manuscript received December 31, 2013; revised May 12, 2014; accepted June 3, 2014. Date of publication September 18, 2014; date of current version November 26, 2014. This work was supported in part by the 863 Plan under Grant 2013AA013501, by the National Science and Technology Major Project under Grant 2013ZX03002003-004, by the NSFC under Grants 61272459, 61221063, and 61170245, by the Research Plan in Shaanxi Province of China under Grant 2013K06-38, and by the Program for New Century Excellent Talents in University.

The authors are with MOE KLINNS Lab and the Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: huc@ieee.org).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2014.2358835

and as a result, FGTI has to evolve the rules into a semantic-based form, which describes the signatures in layer-7 after parsing the protocol. The size and the matching complexity of the semantic-based rule set have been significantly increased, which pose great challenges to the FGTI system design. The necessity and the challenges of semantic-based rules will be discussed in Section II. This paper studies efficient method to support semantic rules for FGTI so as to provide high identification throughput with controlled memory consumption.

We observe and leverage the feature that segments (a.k.a., fields) in different matching rules of FGTI would share the same signature. Based on this, in the preliminary version of this paper [12], we proposed the idea of Rule Organized Optimal Matching (ROOM) to eliminate the matching on redundant fields. ROOM splits the rules into fields, determines the matching order of each field, and selects only a (small) part of the rules that could be possibly hit to activate the matching process. In some cases, the identification rules needs to check multiple Protocol Data Units (PDUs),[1] which is not well supported by the original ROOM since it does not organize multiple matching structures and its straightforward extension may lower the efficiency of throughput and memory. In this paper, we further propose MP-ROOM to enhance ROOM, which supports FGTI over multiple PDUs with high efficiency. Additionally, we have performed more experiments to demonstrate the accuracy, the optimality and the performance of the proposed method in this paper. Specifically, we evaluate the accuracy by sampling real traces with ground truth, compare the propose heuristic algorithm with the brute-force method to demonstrate the optimality, and compare MP-ROOM with a famous open-source approach in the system level.

In this paper, we make the following contributions.

1) We first propose ROOM to construct hierarchical Layered Matching Tree (LMT), which reduces the space complexity and improves throughput for matching and we further propose MP-ROOM to support the identification over multiple PDUs based on ROOM.
2) We have formally defined the problem of constructing an optimal LMT and demonstrated it to be NP-hard. We have further proposed a heuristic algorithm to solve the problem.
3) We have implemented a real system of MP-ROOM and have made comprehensive comparisons with related works. Compared with previous work, MP-ROOM improves the performance-cost ratio by 1.6 $\sim$ 63 times. Further with parallel acceleration, MP-ROOM achieves over 40 Gb/s throughput with 8 cores on the real trace.

The remainder of the paper is organized as follows: Section II indicates the technical challenges of FGTI and our basic idea to solve the bottlenecks for FGTI. Section III describes the detailed design and the comprehensive analysis of constructing the optimal layered matching tree for ROOM. Section IV proposes MP-ROOM to support multiple PDUs matching and presents the matching processing after the construction. Section V introduces the implementation of MP-ROOM with parallel acceleration. Section VI conducts extensive experiments with a prototype of MP-ROOM under real traces. Section VII reviews the previous work and discusses some related issues. Section VIII summarizes the paper.

II. Challenges and Basic Idea

A. Technical Challenges

The wire speed of the Internet keeps increasing towards 40 $\sim$ 100 Gb/s in the core. Even we can design a parallel system and split the traffic into several small FGTI systems, very fast processing in a single FGTI system is expected. To provide high throughput, fast memory should be employed, which is however small in size usually [13], [14]. Although, CGTI or Intrusion Detection system (IDS) also has similar challenges on efficient memory usage, there are new challenges for FGTI to meet the requirements of system throughput and memory consumption caused by several reasons.

First, FGTI system has to employ semantic-based rules instead of regex-based rules in CGTI. A regex-based rule will be hit no matter which part of a packet matches with it, while a semantic-based rule is considered matched only when the values in all the specified fields of the packet matches with the rule.[2] A typical regex-based rule describing a behavior on a certain HTTP header would be like:

```
Signature timeline-weibo
   http-request-header
     /.*GET.*gettimeline.php HTTP/1.1\r\n
      Host: weibo.cn\r\n\r\n/
```

However, a non-HTTP protocol packet can also carry payloads with such content in either occasional or intentional cases. A semantic-based rule with same purpose specifies a set of requirements in "Method" field, "Uniform Resource Identifier" ("URI") field, and "Host" field of HTTP protocol, which would be like this:

```
Signature timeline-weibo
   proto HTTP
     Method/GET/
     URI/gettimeline.php/
     Host/weibo.cn/
```

Obviously, FGTI has to match such kind of structure instead of a simple string/regex-based rule, and therefore the whole processing of FGTI is much more complicated than CGTI.

Moreover, FGTI employs more rules than CGTI. For example, one rule previously in CGTI detecting QQ now evolves into a number of rules to identify login message of QQ, text message of QQ, audio conference of QQ, file transfer of QQ, heartbeat message keeping the connection of QQ, etc. Table I compares simple rule sets of CGTI and FGTI. More rules lead to heavier overhead. Even only considering the simpler

[1]PDU, in this paper, is the atomic data unit that are sent between two application endpoints.

[2]The rule is matched only when all the fields with AND logic among them are satisfied. Single rule requiring values of fields with OR logic can be rewritten into separated rules.

TABLE I
DIFFERENCES OF RULE SETS BETWEEN CGTI AND FGTI

App. Catalog	# of Rules in CGTI	# of Rules in FGTI
Browser	4	67
QQ	1	41
Online Video	1	18
App Market	3	80
Mail	1	13

Note: Most rules in CGTI are written according to the applications, which means the number of rules is determined by the number of applications. On the contrary, the rules in FGTI are written according to the application behaviors, which leads to a much larger size of the rule set.

regex-based rules, it is demonstrated that the combination of a selected rule set with 794 regular expressions consumes 5.29 GB memory statically [15].

IDS is previously used in the cases of traffic identification in layer-7, which however fails to support FGTI completely for several reasons. First, traditional IDSes such as Snort [16] and Bro [17] apply regex-based rules as the identification rules, which is insufficient for FGTI as discussed above. Second, the matching targets of IDS and FGTI are very different. IDS focuses on only few packets related to intrusions, while FGTI system is interested in almost every packet in the network. In addition, after identifying a packet, IDS and FGTI process the following packets with different policies. The following packets in an IDS can be simply verified and processed (pass or drop) by the 5-tuple (source and destination IP address, source and destination port and protocol) in the header, while each packet needs a full check on the rule set with FGTI even their flow have already been identified, since they may be mapped to different behaviors. Therefore, the previous studies cannot directly solve the problems.

Finally, the rules may cross PDUs. The applications or protocols of the packets can be identified on a single PDU, but the user behaviors would be described cross multiple PDUs. For instance, the behavior of buying an electronic accessory on Amazon.com cannot be identified by only checking the request PDU of HTTP protocol, since the catalog of the accessory can only be found in the reassembled response PDU. To illustrate such behaviors, the semantics of the rules have to be further evolved to indicate the precedence and/or cyclic relationship between PDUs. Such evolution aggravates the challenge of organizing and processing rules in FGTI system.

B. Basic Idea to Respond to the Challenges

In the literature, there are two ways to match a large number of semantic-based rules in CGFI and IDS systems [15], [18]. As shown in Fig. 1(a), the first one constructs one matcher for each rule, by checking each field of the rule sequentially [18]. Matchers are checked one by one as well. The system moves to the next matcher if no match in current matcher, and stops when one of the matchers (rule) is hit. Obviously, the matching speed is the bottleneck since time scales with the number of rules, matchers and fields in them.

To improve the processing speed, following the idea of regex-based rules, all the rules can be divided according to their fields,

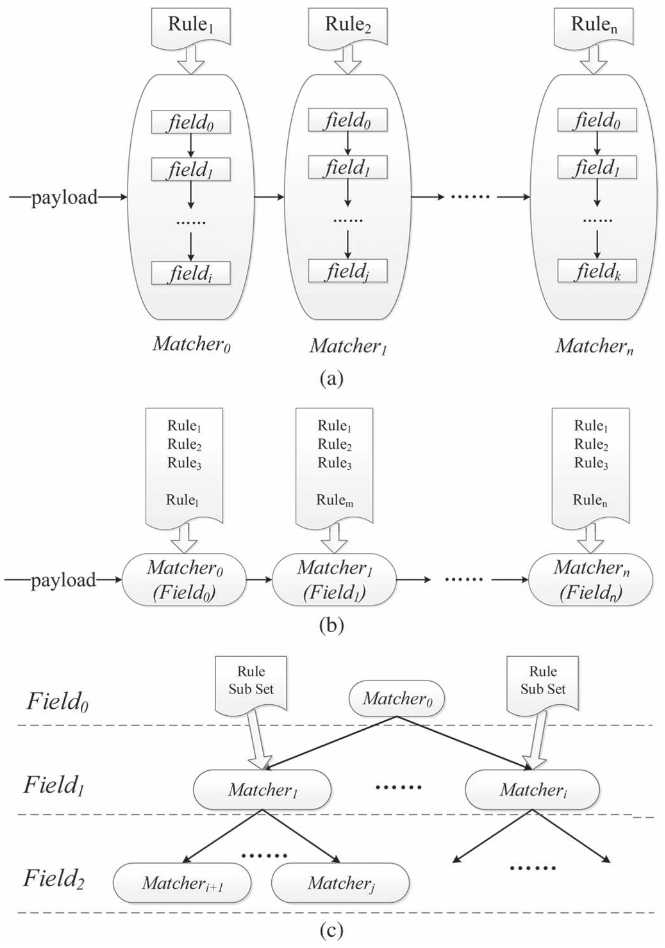

Fig. 1. Indicative comparisons of different matching systems. (a) One matcher is constructed for each rule. Matchers are checked sequentially as well as the fields inside them. The system moves to the next matcher if nomatch returns for any of the fields of current matcher, and stops when one of the matchers (rule) is hit. The speed is the major concern. (b) All the rules can be divided to their fields, and RegExs in same field can be merged to construct one DFA-based matcher. By checking the matcher sequentially, the system stops when a matcher failed and moves to next matcher if any rule is hit. The memory explosion is the main constrain. Also, it usually cost extra time and space for intermediate results. (c) The whole rule set is divided into sub-sets according to fields. Each matcher is constructed on a rule sub-set. The matching result in each matcher determines the next matcher. At most m matchers will be activated to identify a packet, where m is the max. number of fields in the rules.

and regular expressions in the same field can be merged to construct one DFA-based matcher, as shown in Fig. 1(b). The system moves to next matcher if any rule is hit [15], and stops when any of the matchers fails. Although DFA matcher saves the matching time, there are two main concerns: the combination of values in rules brings risks of memory explosion, besides, it costs extra space to save the intermediate result of each matcher, and extra time to merge them to get the final hit rules. The problem becomes severer with the growing number of both rules and fields.

We first present our idea as depicted in Fig. 1(c). With ROOM, the whole rule set is divided into sub-sets according to the fields, which is similar with the second method, to maintain the matching speed of DFA-based matcher. The matchers in the lower layers are constructed with rule sub-sets, which are the matching results of the parent matchers. Therefore, the transitions between matchers remove the extra overhead for

TABLE II
SIMPLIFIED RULE SET SAMPLE

RuleID	Host	URI	User-Agent
1	weibo.cn	gettimeline.php	Chrome
2	weibo.cn	getnews.php	Chrome
3	3g.qq.com	*	Opera
4	*	.jpg	IE
5	weibo.cn	*	*

Note: A set of semantic-based rules can be sort into a table structure where a column represents a protocol field, and a row is a matching rule defined on several fields. "$*$" in the table is a wild card which means the rule does not depend on that field.

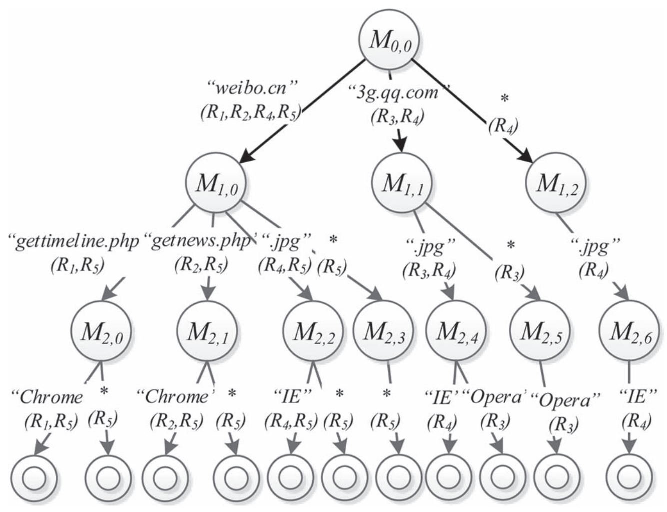

Fig. 2. Matchers constructed by ROOM on the rule set in Table II. Each matcher is constructed with a rule sub-set according to the possible matching results of its parent matcher. When matching, the matchers are activated according to the matching status one by one in different layers, and the results can be indicated directly from the rule sub-set of the last activated matchers.

intermediate result in second method, which may lead to a higher performance. Besides, this kind of construction splits huge DFAs into several small DFAs, which leads to much lower risk on memory explosion.

To give an intuitive explanation on how ROOM works, let us suppose that we have a simple rule set in Table II, where a column represents a protocol field and a row is a matching rule defined on several fields. Based on the rules, we can first construct a tree structure called "layered matching tree" (LMT) with the above idea as shown in Fig. 2 (details of the construction reasoning will be explained later). When a PDU with "$Host ==$ "$weibo.cn$"$\&\&URI ==$"$/ttt/gettimeline.php$" $\&\&User-Agent =$ "$GoogleChrome$" comes into ROOM system, in the first step, ROOM searches "Host" field of the PDU in $M_{0,0}$ and selects the left branch to activate the $M_{1,0}$, then ROOM matches "URI" field of packet with this matcher and gets a hit on the left branch which activates $M_{2,0}$. Since "Chrome" matches with the input payload "Google Chrome," we get $\{R_1, R_5\}$ as the final hit rule set.

The matching speed for each packet in each matcher is only determined by the length of the input packet field, and the total matching time is determined by the number of matchers to be performed. Given 869 rules and a real trace, it consumes 1479.0 seconds, 31.5 seconds, 20.7 seconds, for the matching time using the structure in Fig. 1(a)–(c) respectively. And the memory costs are 121.2 MB, 254.3 MB, 201.8 MB, respectively. If we define normalized matching throughput divided by memory cost as the performance-to-cost gain, ROOM strikes the best trade-off between processing speed and memory cost. The gain becomes larger when the number of rules increases, which will be shown in our evaluations later.

MP-ROOM is based on ROOM to enhance the matching ability on multiple PDUs. The basic idea of MP-ROOM is to organize multiple LMTs by organizing multiple fields in ROOM. Instead of building LMTs with all the rules cross multiple PDUs, MP-ROOM builds multiple LMTs according to the "leaf" matchers of the parent LMT, which indicate the possible candidate rules for next PDU matching. The large LMTs are split into smaller ones, and no intermediate results are produced or processed through matching, which therefore save the memory and accelerate the matching.

We would like mention that Firewall Decision Diagram (FDD) [19] designed to check and optimize the firewall rules is orthogonal with this work in different contexts, although FDD's layered structure is similar with LMT in this paper. There are two major differences between FDD and MP-ROOM. First, FDD is supposed to be deployed off-line, which targets on the consistency and completeness of the firewall rules. The rules would be matched by their original way (sequentially in firewall devices mostly) after being checked with FDD. On the other hand, LMT as a matching structure is designed to be deployed on-line. The rules in FGTI system would be directly matched on LMT. Thus, LMT optimizes the matching speed, which FDD actually does not care. Second, FDD provides a reduction algorithm to simplify the layered structure, which however focuses on the semantic assurance of the rules to further ensure the completeness. For MP-ROOM, we need further employ optimization method to contract a memory efficient LMT, which is also not the focus of FDD.

With this idea in mind, the key issues of MP-ROOM are: 1) how to construct the optimal layered matchers (LMT), 2) how to do matching under the constructed matchers, especially when the matching process may cross multiple PDUs? These two problems will be articulated in Sections III and IV.

III. ROOM ON SINGLE PDU

In this section, we first introduce the matcher construction of ROOM on a single PDU. Next, we propose an optimal construction by properly organizing the rules. Two further optimizations on matcher constructions are also presented.

A. Primary Matcher Construction

The matchers are sorted into an LMT as shown in the example of Fig. 2. In each layer, the matchers are constructed according to a field. In ith field, there can be several matchers and we use $M_{i,k}$ to denote the kth matcher for the ith field.[3] We define a matcher as a two-tuple $(Field, InitRuleSet)$, where

[3]The primary method does not rely on specific order of field. The order can be changed to obtain better performance as discussed later in Section III-B

$InitRuleSet$ is the whole rule set for constructing the matcher, and $Field$ demonstrates which part of the rules should be used according to the layer-7 protocols. The matcher and its descendants are completely determined by these two arguments. If the maximum number of the fields are N, then the maximum depth of the LMT is also N.

The primary method constructs matchers by recursively selecting one unmatched field. In the initial, the whole rule set is denoted by $H_{0,0}$ and the first field is selected to construct the first layer of the LMT. The number of branches equals to the number of unique values in $H_{0,0}$. The unique values of the first field in the rule set $H_{0,0}$ form a set $\{F_{1,j}\}, j = 1, 2, \ldots, |F_1|$, where $|F_1|$ is the element number in $\{F_{1,j}\}$. The rules whose values of the first field equals $F_{1,j}$ are the possible hitting rules in $M_{1,j}$, which is denoted as $H_{1,j}$. If we remove the rules with a wildcard "$*$" in the first field (denoted by $O_{0,0}$), the rest rules are denoted by $S_{0,0}$. Also, we use $V_{1,j}$ denotes the field value set that excludes "$*$" from $F_{1,j}$.

When building matcher $M_{1,j}$, set $F_{1,j}$ and $V_{1,j}$ are first outlined from $H_{0,0}$. Then, the rule sub-set $H_{1,j}$ and $S_{1,j}$ are associated with the matchers. Next, the lower layer matchers are constructed by its parent layer recursively: $H0,0 \to \{F_{1,j}, V_{1,j}\} \to \{H_{1,j}, S_{1,j}\} \to \cdots \to \{H_{i,k}, S_{i,k}\} \to \{F_{i+1,l}, V_{i+1,l}\} \to \{H_{i+1,l}, S_{i+1,l}\} \to \ldots$.

The matcher is the node of the LMT, and the rule sub-set on the transition is H for the next matcher. LMT clearly indicates the status of the matching, which will always active a correct next matcher according to the current matching result (values on the transitions). LMT is pre-constructed and keeps steady if the rule set is fixed. There is an exception if S is an empty set while O is not. In this case, all rules in this matcher do not depend on this field. To handle this exception, there should be a flag to indicate whether S is empty. When it turns true, ROOM simply jumps to the next matcher with all rules in current matcher without any construction.

Still take the rule set in Table II as an example to explain the construction of LMT in Fig. 2. As $H_{0,0}$ is defined as the whole rule set, $M_{0,0}$ is the only matcher in the first layer. We use the "Host" field for the first layer. $M_{0,0}$ =("Host," $\{R_1, R_2, R_3, R_4, R_5\}$). $S_{0,0} = \{R_1, R_2, R_4, R_5\}$, $O_{0,0} = \{R_3\}$. For $F_{1,0} = \{$"weibo.cn", "$*$"$\}$, $V_{1,0} = \{$'weibo.cn"$\}$, and $H_{1,0} = \{R_1, R_2, R_3, R_5\}$. Next, we can build $M_{1,0}$ by using $H_{1,0}$. Continue this process and finally we can recursively build an LMT depicted in Fig. 2.

B. Optimal Rule Organization

Different field orders in the aforementioned construction algorithm lead to different LMTs. In the example of Fig. 2, the construction takes "Host" as the first field, "URI" as the second and "User-Agent" as the last field. If we change the order the last two fields, LMT will be the form in Fig. 3, which has 6 matchers in the third layer. The sum of the states of LMT matchers in Figs. 2 and 3 are 85 and 68, respectively.

This example brings a question: how to determine the optimal order of the fields used to construct the matching tree? We define it as a Rule Organization Problem (ROP). Given a rule set, ROOM determines an optimal field order to construct the LMT, so the memory cost is minimized. Suppose that the field i has a new order $\Psi(i), i = 1, 2, \cdots, N$, after the organization, and the memory consumption of the matchers in field i is $D_{\Psi(i)}$. So ROP seeks $\Psi(i), i = 1, 2, \cdots, N$, to minimize the total cost as shown below.

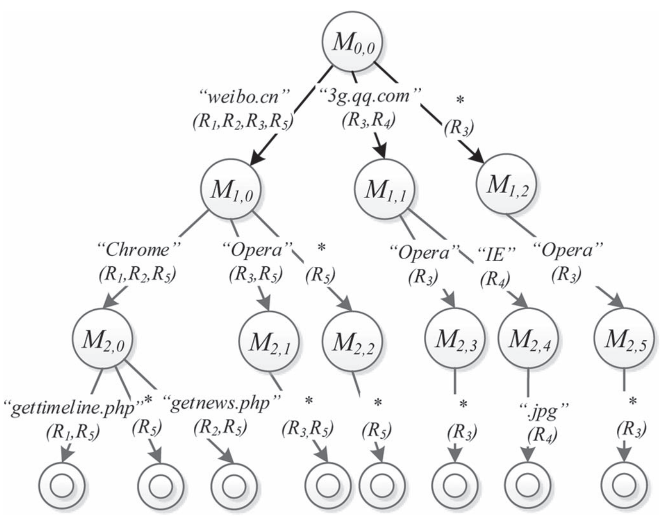

Fig. 3. A different LMT with 6 matchers in the third layer consumes less memory than the LMT in Fig. 2.

$$\min_{\psi \in T_N} \sum_{i=1}^{N} D_{i\psi(i)}, \tag{1}$$

where T_N is the set of all the permutation: $\Psi : \{1, 2, \cdots, N\} \to \{1, 2, \cdots, N\}$.

In fact, ROP can be reduced from Quadratic Assignment Problem (QAP) [20] in polynomial time, denoted as $QAP \leq_p ROP$. QAP considers allocating n facilities to n locations. There are three costs in the problem: the cost function of distance between locations, the function of flow between facilities and the cost placing a facility in a location. The objective of QAP is to minimize the total cost related to the assignment of facilities to locations. If we remain the first two costs in QAP and set the cost of a facility-location placement to be zero, the problem can be formulated as,

$$\min_{\phi \in S_n} \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} d_{\phi(i)\phi(j)}, \tag{2}$$

where $\phi(i), i = 1, 2, \cdots, n$ is the location used to placing facility i, f_{ij} and $d_{\phi(i)\phi(j)}$ are the cost function related to flow between facilities and distance between locations.

To reduce QAP to ROP, we first rewrite the ROP formulation in (1) as

$$\min_{\psi \in T_N} \sum_{i=1}^{N} \sum_{j=1}^{N} D_{ij} I_{\psi(i)\psi(j)}, \tag{3}$$

$$I_{\psi(i)\psi(j)} = \begin{cases} 1, & \text{if } j = \psi(i) \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

If we map the original fields order (of ROP) to facility (of QAP) and the organized order (of ROP) to the location (of QAP), (2) can be transferred to (3) by mapping the variables and functions: $n \to N, f \to D, d \to I, S_n \to T_N, \phi \to \Psi$. Since QAP is known as an NP-hard problem [20], ROP is also NP-hard with a polynomial reduction from QAP.

It is worthy of noting that the number of fields in a rule can be as large as tens or even hundreds in FGTI. Even without considering the payload, OpenFlow 1.4.0 has already advocated to check 41 fields in protocol headers only [21]. The brute-force method to calculate the ROP leads to unacceptable calculation time even if it is pre-computed, while building LMT with random order of fields (for example, the native order in the segments) significantly increases the memory consumptions (experiment results will be shown in Section VI). As a result, we propose a heuristic algorithm to organize the field order so as to get the optimal memory consumption.

Notice that the values of rules in a particular field are fixed. All values have to be constructed into a DFA structure at least once. Therefore, although not very accurately, the complexity of the field depends on the number of matcher a lot that more matchers in this field, more equally values will distribute. And it is easy to imagine that N values distributing in N matchers is much better than in one matcher in space complexity. Furthermore, we can see that the number of matchers in current layer depends on the number of unique values in the previous layer. So there is a trend that if we choose a field which brings the most unique values for higher layers, values in lower layers will distribute more equally. However, there is a complexity tradeoff between the lower layer and the current layer, for it is possible the field bringing more "branches" (unique values) for the next layer also brings huge overhead in the current layer.

Another factor that should be considered is the redundancy of LMT. We can see that values of R_3, R_4, and R_5 are constructed more than once in Fig. 2, which results from the "$*$" cases in the given rule set. Since the rules contributing "$*$" in a certain layer would be copied to all other branches, the effect of the redundancy is defined as the product of these two factors: a) the complexity of the redundant values brought by "$*$" cases, and b) the number of branches, to which all "$*$" rules would be copied.

In general, we want to choose a field, which brings more branches for next layer, has low complexity itself and brings less redundancy to LMT. Here we first define a notation C, where $C(ValueSet)$ represents the number of nodes of the matcher constructed by the values in $ValueSet$ and $C_{field,i}$ represents the number of the nodes in the matchers on ith layer in total if $field$ is placed in this layer. Now we try to describe the feature of the field which will affect the choice.

$$W_{field,i} = \sum_{k=1}^{P_i} \left(C(E_{i,k}) \times |V_{i,k}|\right). \tag{5}$$

This notation represents the redundancy effect of this field where $E_{i,k}$ describes the set of values brought by "$*$" cases, and $|V_{i,k}|$ describes the branches the matcher generated. For instance, if we choose "Host" as the first field, $P_i = 1$ and $E_{0,0}$ is {".jpg," "IE"}.

Now, we define a complexity factor for each field when we need to choose one.

$$G_{field,i} = \frac{C_{field,i} \times W_{field,i}}{|V_{field,i}|}. \tag{6}$$

The algorithm simply selects the field holding $G_{\min}$. As it will be performed every time choosing the next field, the time complexity is $O(N^2)$ where N is the number of the fields.

Still, we take the rules in Table II as the example. For the first layer, we compute $C_{\text{“}Host\text{”},0} = C(\{\text{“}weibo.cn\text{”}, \text{“}weibo.cn\text{”}, \text{“}3g.qq.com\text{”}, \text{“}*\text{”}, \text{“}weibo.cn\text{”}\}) = 18$, $C_{\text{“}URI\text{”},0} = C(\{\text{“}get\text{-}timeline.php\text{”}, \text{“}getnews.php\text{”}, \text{“}*\text{”}, \text{“}*\text{”}, \text{“}.jpg\text{”}\}) = 30$ and $C_{\text{“}User-Agent\text{”},0} = 15$. Moving to the calculation for W, $W_{\text{“}Host\text{”},0} = (C(\{\text{“}.jpg\text{”}, \text{“}IE\text{”}\}) \times 3 = 18$, and $W_{\text{“}URI\text{”},0} = (C(\{\{\text{“}3g.qq.com\text{”}, \text{“}Opera\text{”}\}, \{\text{“}weibo.cn\text{”}\}\}) \times 4 = 88$. We can get $G_{field,0}$ as,

$$\begin{aligned} G_{\text{“}Host\text{”},0} &= \frac{C_{\text{“}Host\text{”},0} \times W_{\text{“}Host\text{”},0}}{|V_{\text{“}Host\text{”},0}|} \\ &= \frac{18 \times 18}{3} \\ &= 108. \end{aligned} \tag{7}$$

We obtain $G_{\text{“}URI\text{”},0} = 616$ and $G_{\text{“}User-Agent\text{”},0} = 112$ by the same way. For $G_{\min}$ is held by the field "Host", we choose "Host" for the first layer. Then we choose a field for the next layer from the rest. The calculation in the second layer depends on the choice for the first layer. By looping the choosing process, finally we get the optimal fields order: "Host," "User-Agent" and "URI," which is the example we depict in Fig. 3. In this case, the rules are simple and constructing their DFA does not bring exponential growth of space as the worst case. Therefore, the positive factor $|V_{field,i}|$ for $G_{field,i}$ seems not very significant here, but it will show its importance with the growth of rule complexity.

C. Further Optimizations

We present two more optimizations to further save the memory consumption. First, "$*$" cases are critical to the space complexity. We observe that some rules have already been matched completely in the upper layer, and therefore contribute "$*$" to all of the lower layers. For instance, R_5 in $M_{1,0}$ in Fig. 3 can be tagged as a matched rule, for it does not depend on this layer or any of the lower layers. We call such kind of rules "afore-matched" rules. And we further discover some matchers and their descendants actually do nothing for the matching, since all rules in them are afore-matched rules. For instance, $M_{2,5}$ in Fig. 3 contains only one rule R_3, which is an afore-matched rule. Such matchers are called afore-matched matchers. The afore-matched rules/matchers don't contribute any values to the DFA structures to consume more memory, but increase the number of matchers and transitions between them which affect the memory consumption. Based on the above observations, LMT could be further optimized by cutting the "$*$" cost brought by the afore-matched rules/matchers. The afore-matched rules would not be involved in the construction process and the afore-matched matchers would be tagged as a "leaf" matcher to indicate that the matching process should stop in these matchers.

On the other hand, a matcher can be defined as a two-tuple $(Field, InitRuleSet)$ as discussed in Section III-A. Two

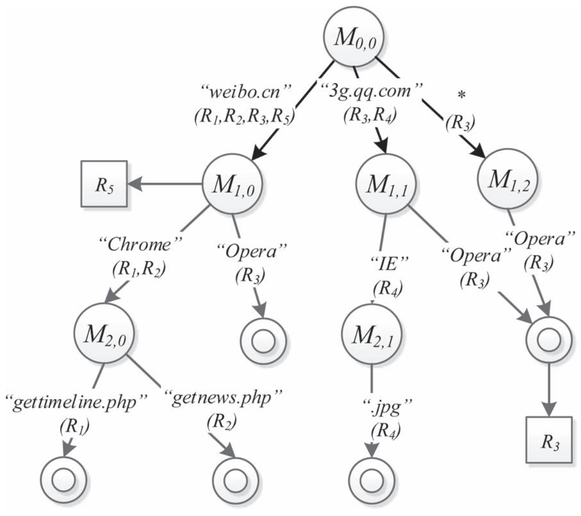

Fig. 4. Optimal rule set organization. The order of the fields are computed by the aforementioned heuristic algorithm. The afore-matched rules and matchers are cut off and the same matchers are merged to further save the memory.

or more matchers with the same $Field$ can share the same $InitRuleSet$ too, which can be merged for efficient memory. In the case depicted in Fig. 3, $M_{2,3}$ and $M_{2,5}$ share the two-tuple ("User-Agent," $\{R_3\}$), and can be merged as well as their descendants.

The final optimal organization for rules in Table II is shown in Fig. 4. Compared with LMT in Fig. 2, it now contains five non-leaf matchers while the original contains eleven. Notice that though the leaf matcher in the right branch of $M_{1,0}$ seems identical with the leaf matcher under $M_{1,1}$ and $M_{1,2}$, these two leaf matchers cannot be merged, because the first leaf matcher stores the implicit information of R_5 afore-matched in $M_{1,0}$. This information is useful when it comes to multiple PDUs matching, which will be introduced in the next section.

IV. MP-ROOM ON MULTIPLE PDUs

A. Organize Multiple LMTs

Section III designs the algorithm of ROOM for the single PDU case, which needs to be enhanced to support the rules cross multiple PDUs. In straightforward way, assume the maximum number of the required PDUs in the rule set are N, there would be N LMTs as well. The kth LMT is constructed by the requirements of the kth PDU in the whole rule set. If the previous LMT get a hit on some rules requiring more PDUs, it jumps to the next LMT and starts matching for the following PDUs until the last LMT get a hit. The intersection of the matched rules in each LMT is the final matched rule set. The method suffers from the low efficiency of storing and calculating the intermediate results. MP-ROOM is proposed to better organize the multiple LMTs, whose basic idea is to construct LMTs according to the matching results of the previous LMT instead of the whole rule set. Since the intermediate costs are removed, the performance on multiple PDUs could be improved by MP-ROOM.

Specifically, each "leaf" matcher in an LMT could contain several rules requiring more PDUs, and a much smaller LMT

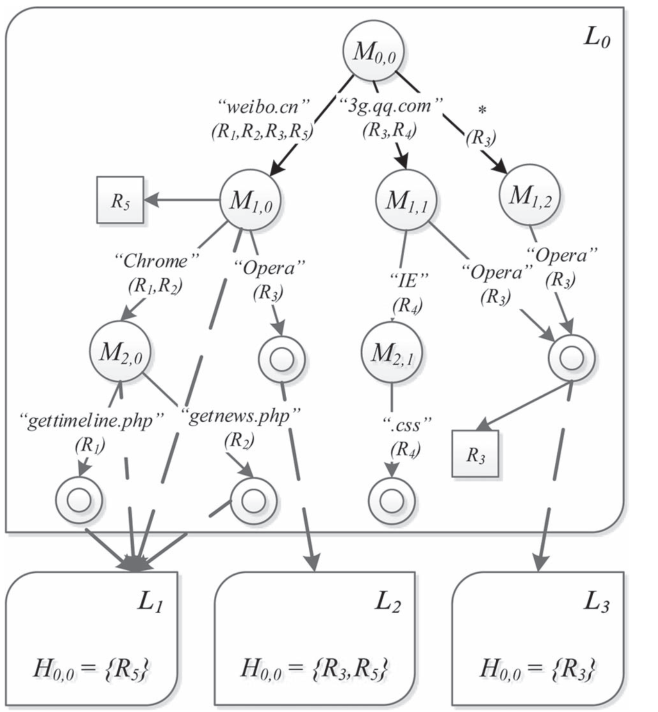

Fig. 5. Extended multiple LMTs based on LMT shown in Fig. 4 by assuming R_3 and R_5 require second PDUs.

can be constructed according to this sub rule set. Assume that R_3 and R_5 in Table II require a second PDU for matching, consider the LMT shown in Fig. 3, every "leaf" matcher containing R_3 and/or R_5 could link to another LMT constructed by R_3 and/or R_5. The structure removes the intermediate cost between multiple LMTs and saves the matching time and memory cost. However, this basic structure cannot serve the optimal LMT shown in Fig. 4, since R_5 is an afore-matched rule and does not appear in any of the leaf matchers. Actually, for R_5 is considered matched in $M_{1,0}$ and its descendants, all these matchers should link to an LMT constructed by the rule subset containing R_5 at least. For instance, $M_{2,0}$ should link to an LMT constructed by $\{R_5\}$, and the leaf matcher in its right branch should link to an LMT constructed by $\{R_3, R_5\}$. Fig. 5 depicts the final organization of the multiple LMTs.

Please note that the leaf matcher in the right branch of $M_{1,0}$ cannot be merged with the rightmost leaf matcher. The first leaf matcher stores implicit information of the afore-matched rule set, $\{R_5\}$ in this case. The rule set should be involved when descendant matchers need to construct multiple LMTs. Therefore, the second optimization in Section III-C can be re-stated as follows: matchers sharing the same 3-tuple ($Field$, $InitRuleSet, AforeMatchedRuleSet$) can be merged to reduce the memory consumption.

B. Matching on MP-ROOM

We first present the matching process on a single constructed LMT. The matching starts from matcher $M_{0,0}$. Inside a matcher, it tries to match the given payload with SRD-DFA structure [22] to distinguish the result with different rules. The matching result points to the next matcher on the next field. If no match is hit in any of the matchers, the matching process stops instantly and reports no rule matches with the given payload. The matching traced from $M_{0,0}$ to $M_{N,k}$, and $M_{N,k}$ is a specific matcher on Nth (also the last) field. If the next matcher is a leaf matcher, it means all of the upper matchers in this path are matched, and rules in $F_{N+1,l}$ are the final matched rules, which should be reported. It is possible that the leaf node includes two or more rules, which usually results from the inclusions between multiple rules. ROOM/MP-ROOM would not choose a "proper" rule but just reports the matched rule set, since the inclusion cases should be considered when providing the rules beforehand.

When MP-ROOM is triggered for matching multiple PDUs, each flow maintains a status variable to indicate the activated LMT, where the matching starts for the current PDU. When the last LMT is hit by the current PDU, the status variable is reset to be the first LMT. The complete pseudo-code of matching is depicted in Algorithm 1.

Algorithm 1 MatchingProcess(PDU)

```
 1: $L \leftarrow PDU.Flow.L$ and $M \leftarrow L.M_{0,0}$ and $R \leftarrow NULL$
 2: while $M$ is not a leaf matcher do
 3:   for $r$ in $M.AforeMatchedRuleSet$ do
 4:     if $r$ does not require a next PDU then
 5:       Add $r$ into $R$
 6:   if value of $PDU$ in $M.Field$ is accepted by $M$ then
 7:     $M \leftarrow$ the next matcher according to the result
 8:   else
 9:     if there is a "$*$" branch then
10:       $M \leftarrow$ the "$*$" branch
11:     else
12:       if there is a next LMT then
13:         $PDU.Flow.L \leftarrow$ next LMT
14:       Stop and Wait for next PDU
15: for $r$ in $M.InitRules$ do
16:   if $r$ does not require the next PDU then
17:     Add $r$ into $R$
18: if there is a next LMT then
19:   $PDU.Flow.L \leftarrow$ next LMT
20: else
21:   $PDU.Flow.L \leftarrow$ the first LMT
22: return $R$
```

V. Implementation

Basic Implementation: A complete FGTI system would contain three major components: a packet capturer, a protocol identifier/parser, and a matching engine. The packet capturer is to capture and reassemble packets in lower layers, and the protocol parser is to resolve packets into structural form according to the protocol specification. PDU is assembled in the protocol parser. Finally the PDU is processed in the matching engine, which reports the final results. In this paper, MP-ROOM focuses on improving the last and most critical one. We implement MP-ROOM with about 3,000 lines C/C++ code. We also

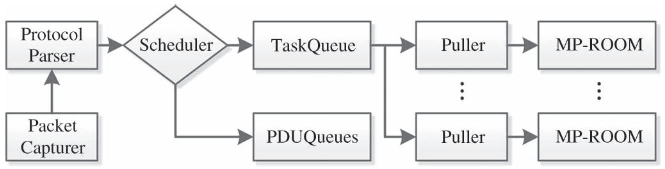

Fig. 6. The high level architecture of the implemented system. PDUs with same hash values are mapped to the same PDUQueue, and TaskQueue stores the entries of PDUQueues, which can be pulled by the puller.

implement the packet capturer and the protocol identifier/parser which are shared by other prototypes for a fair comparison in the evaluation section. To remove the bottleneck lying in the network interface and hard disk I/O, the packet capturer and protocol parser work off-line to pre-load the assembled PDUs into memory. Note that MP-ROOM is independent from the protocols of the traffic, since it deals with the (*field,value*) pairs instead of the raw payload. These pairs are generated by various protocol parsers, which is not the focus of this paper. As illustration, we only implement two protocol parsers for HTTP and QQ protocol in our prototype. These two protocols hold 63.1% packets and 71.4% volume in our real traces under study.

Parallel Acceleration: We implement the "Last Flow Bundle" (LFB) model proposed in [23] to balance the load in processing units. LFB is a flow burst scheduling algorithm designed with the goal of maximizing cache affinity. The idea is to process all of the PDUs in the system that map to the same flow on a single thread and not interleave the processing of PDUs that map to other flows. LFB is composed of two components: a packet scheduler and a packet puller. The scheduler is deployed in a single processing unit to dispatch PDUs into the sharing queues. The puller is to pull PDUs from the sharing queues and pipeline them to the post stages for reassembling and matching in application layer.

The high level architecture of the implemented system is depicted in Fig. 6.

VI. Evaluation

A. Experimental Settings

In this section, we compare the performance of MP-ROOM with related work in algorithm level and system level. In algorithm level, to the best of our knowledge, NetShield [15] achieves the best performance with semantic-based rules, which has the similar structure as Fig. 1(b), thus we first involve it in the comparison. We also implement sequential matching (SM) method for comparison whose architecture is depicted in Fig. 1(a) In system level, we choose a widely accepted open-source approach nDPI, which partially supports FGTI for a comprehensive comparison with our prototype. All the evaluations are performed on a platform with Intel Xeon E5606 (8-core, 2.13 Ghz), 32 GB memory and Linux 2.6 kernel.

In the experiment, we involve 869 rules with HTTP and QQ protocols, which contained 6 independent fields. The rule set can identify both wired and mobile Internet application behaviors. The rules for wired Internet traffic are selected from Snort [16] and are rewritten to be semantic-based rules. We further investigate the mobile Internet traffic, and give an

TABLE III
APPLICATION AND BEHAVIORS IN THE RULE SET

App Catalog	**Typical Behaviors**	**# of Rules**
Weibo	Login, Logout, Tweeting, Re-tweet, Following, Blocking, *etc*.	292
SNS	Login, Logout, Blogging, Following, Blocking, Avatar, *etc*.	183
Online Video	Searching, Watching, Comment, *etc*.	18
Browser	Page viewing, Downloading, Picture viewing, *etc*.	67
App Market	Searching, Downloading, Rating, Comment, Watching, *etc*.	80
Online Shopping	Searching, Adding Star, Rating, Buying, *etc*.	150
QQ	Login, Logout, Text messaging, Video conference, Status changing, *etc*.	41
Thunder	Login, Logout, Searching, Downloading, P2PDownloading, *etc*.	25
Mail	Sending, Receiving *etc*.	13

Note: One "App Catalog" in the list may contain multiple applications, such as it contains six popular browsers' rules in "Browser" catalog. And many applications' protocols and behaviors are carried by HTTP protocol, such as Weibo and Online Video *etc*. We sort them into different classes for that more and more applications tend to provide comprehensive functionalities to users based on same protocol.

TABLE IV
SPECIFICATIONS OF THE REAL TRACES USED IN EXPERIMENTS

	Univ. trace	ISP trace
Capture time	15/12/2012	11/17/2013
Duration	58 min.	70 min.
Trace size	22GB	7.9GB
# of packets	27M	14M
Mean packet length	818B	566B

additional rule set to cover the mobile applications' behaviors. Table III illustrates the details of the combined rule set.

Two real traces are used as the experiment input. One real trace is captured from the gateway of the wired network of a university in China, named as "Univ. trace." The other is collected in Nanjing, Jiangsu province of China, from a radio network controller of a leading ISP in China, which is denoted as "ISP trace." Table IV illustrates the features of the two traces. Each experiment is performed 1000 times to avoid randomness. SM consumes too much time on some conditions, so we just use the average value over 10 experiments.

Notice that the traces are different from our preliminary paper [12], which have larger mean packet length. Besides, the rule set is larger (869 vs. 262) and contains more multiple-PDU rules. Thus, the throughput and memory cost in this paper would not be same with them in the original ROOM.

B. Experimental Results in Algorithm Level

During the evaluations in algorithm level, we check four metrics for the three matching methods: identification throughput, memory cost and the accuracy of the identification results. Also, we perform experiments to exam the efficiency of **LMT** construction algorithms.

1) Throughput: When testing throughput, MP-ROOM is firstly evaluated with NetShield and SM on single thread for the comparison. And the aforementioned parallel optimization will be further applied on MP-ROOM to tap potentials of the prototype.

Single Thread: As depicted in Fig. 7(a) MP-ROOM performs 8.49 Gb/s and 4.85 Gb/s on the two real traces, respectively, which is $1.52 \sim 1.56$ times faster than NetShield and $19.62 \sim 71.34$ times faster than SM. The different throughputs on the two traces result from the different behaviors' densities. Since ISP trace has smaller mean packet length and longer duration time, it tends to have greater density of behaviors than Univ. trace. MP-ROOM and NetShield are slowed down in this case for they have to match with more packets. On the contrary, SM benefits from the higher behavior density, for a packet not matching with any rules is the worst case of this method. Therefore, with the growth of the behaviors' density, the throughput of MP-ROOM or NetShield decreases, while SM's throughput is increased.

Parallel Acceleration: MP-ROOM can be accelerated by parallel programming as we have discussed in Section V. It is shown in Fig. 7(b) that MP-ROOM achieves over 40 Gb/s throughput on Univ. trace when eight cores are used for matching. Since the bandwidth in the core network has reached over 40/100 Gb/s, it is important for FGTI system to reach the increasing wire-speed. MP-ROOM shows the potential ability to meet such requirement.

2) Memory Consumption: We evaluate two kinds of memory consumption: the static memory usage which is used to store the pre-constructed matchers, and the dynamic memory usage which further involves the memory of caching the intermediate results. Table V shows the memory usage of the three methods. MP-ROOM costs less static memory than NetShield, for it optimizes the organization of rule set by splitting the large matchers into small ones, which lowers the memory explosion risk. In the dynamic case, MP-ROOM also performs more efficient memory usage than NetShield, since it does not produce or save any intermediate results. To SM which does not organize the rule set, its static memory is only used for storing the original rule set. In addition, no intermediate results are produced when matching with SM either. Therefore, SM costs the least memory in both static and dynamic cases.

We further introduce the "performance-cost ratio" to measure the prototypes in both terms of throughput and memory cost for a comprehensive comparison of the three prototypes, which is defined as the normalized throughput divided by the normalized memory cost. The ratios of MP-ROOM, NetShield and SM are 1.26, 0.66, and 0.02 on Univ. trace, respectively. MP-ROOM is the most cost-effective approach which is $1.9 \sim 63$ times better than NetShield and SM.

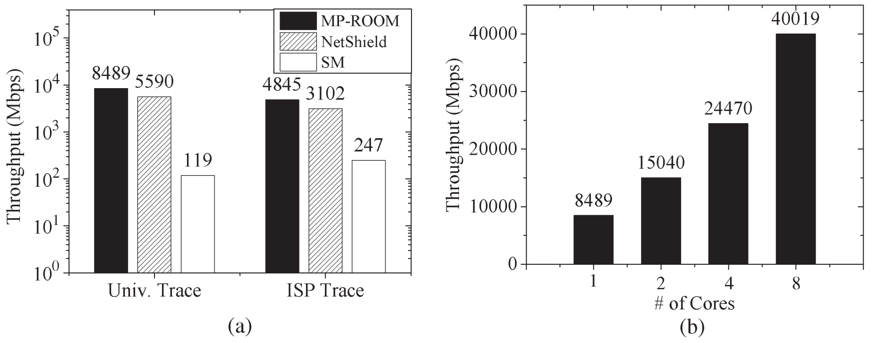

Fig. 7. The throughput of MP-ROOM and related work. MP-ROOM performs the best throughput when comparing with NetShield and SM on single thread, and can achieve over 40 Gb/s with parallel acceleration. (a) Matching throughput for three prototypes on the two real traces with single thread in logarithmic coordinates. (b) Matching throughput of MPROOM on Univ. trace with parallel acceleration.

TABLE V
MEMORY CONSUMPTION

		Univ. trace	ISP trace
MP-ROOM	Static	34.25MB	34.25MB
	Dynamic	201.85MB	243.87MB
NetShield	Static	43.28MB	43.28MB
	Dynamic	254.34MB	287.09MB
SM	Static	11.0MB	11.0MB
	Dynamic	121.24MB	183.30MB

3) Accuracy: It has been already demonstrated that semantic-based rules are much more accurate than regex-based approaches [24]–[26]. In this section, we check the correctness of the semantic-based rules to ensure the right identification results. To confirm the accuracy, we sample three pieces of the real traces, and manually check the applications/behaviors in them. Table VI illustrates the ground truth of the application distribution of the three samples. These sampled pieces are used as the input of MP-ROOM to measure the accuracy. The results show that MP-ROOM accurately detects all the behaviors/ applications in the three pieces by comparing the ground truth with the matching results. Furthermore, we evaluate a "clean" trace without any behaviors and MP-ROOM does not generate any results.

4) Performance of the Heuristic Algorithm for ROP: In this section, the proposed heuristic algorithm of determining the optimal order of the fields is evaluated. The experiment uses the aforementioned real rule set (869 rules, 6 fields) and a set of artificial rule sets which contain 1000 randomly generated rules with different number of fields. To demonstrate the efficiency of the proposed algorithm, we also introduce a brute-force method, which finds the global optimal order by checking all the permutations; and a native-order method, which sorts the fields by the order in the PDU. The consumptions to generate the rules are averaged by 100 times independent experiments. Table VII shows the performance gaps between the three methods. The results demonstrate that the heuristic algorithm provides near-optimal memory, which is 96.6% to the optimal value and only 6.6% to the native-order method in average. In addition, Table VIII shows the cost of the heuristic algorithm with the same real and generated rule set. We observe that the calculation time is reduced by 6.7 $\sim$ 310.5 times compared with the brute-force method. The time cost of brute-force method on 11 fields is unacceptable. Next, we check the effect introduced by the number of rules and rule fields under real traces and generated traces in Fig. 8(a) and (b) depict the relationship. The number of fields are all six in these two experiments. Both the brute-force method and heuristic algorithm show a near linear

TABLE VI
APP. DISTRIBUTIONS OF SAMPLES

Application	Behavior	P_1	P_2	P_3
WebQQ	Login	0.35%	0.60%	1.67%
	Chatting	18.81%	6.68%	9.87%
	ReceiveMsg	18.99%	23.35%	13.08%
	ExtraReq	5.39%	3.12%	4.63%
	GetLevel	1.43%	2.55%	5.15%
	Total	**44.97%**	**36.30%**	**34.4%**
Youku	TV	5.47%	8.59%	0.12%
	Movie	6.08%	10.34%	0.10%
	News	3.67%	0.16%	0.13%
	Play	8.29%	17.54%	0.27%
	Comment	2.34%	10.45%	0.16%
	Total	**25.85%**	**47.08%**	**0.78%**
Weibo	Login	1.41%	0.79%	5.63%
	Posting	3.77%	2.13%	12.61%
	Reply	11.96%	4.36%	9.20%
	Forward	5.92%	5.90%	21.58%
	Star	1.71%	1.12%	6.93%
	Search	4.41%	2.32%	8.87%
	Total	**29.18%**	**16.62%**	**64.82%**
Total		**100%**	**100%**	**100%**

Note: P_1 and P_2 are sampled from the Univ. trace, and P_3 is sampled from the ISP trace.

TABLE VII
MEMORY CONSUMPTION OF HEURISTIC ALGORITHM

	Real Rules 869 rules 6 fields	Gen. Rules 1000 rules 6 fields	Gen. Rules 1000 rules 11 fields
Brute-Force Method	34MB	187MB	164MB
Native-Order Method	557MB	2861MB	2291MB
Heuristic Algorithm	34MB	199MB	171MB

TABLE VIII
TIME COST OF HEURISTIC ALGORITHM

	Real Rules 869 rules 6 fields	Gen. Rules 1000 rules 6 fields	Gen. Rules 1000 rules 11 fields
Heuristic Algorithm	14.4s	18.9s	80.8s
Brute-Force Method	100.5s	126.6s	25085.2s
Speed Up Ratio	**7.0**	**6.7**	**310.5**

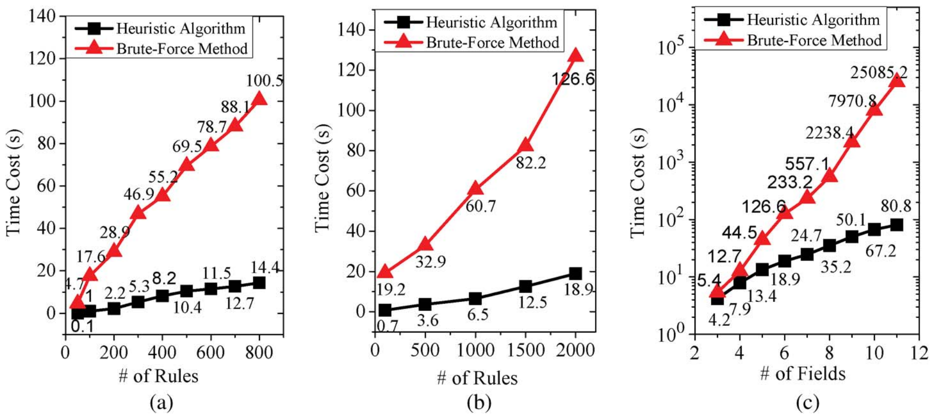

Fig. 8. Time cost trends with number of real and generated rules and fields. (a) Time cost vs. # of real rules. (b) time cost vs. # of generated rules. (c) time cost vs. # of fields (1000 rules).

TABLE IX
MP-ROOM PERFORMANCE IN MULTIPLE PDUS CASES

	Throughput	Memory (dynamic)
ROOM	4951Mbps	167MB
MP-ROOM	5632Mbps	134MB
Increase Ratio	**1.138**	**0.802**

increase to the time cost, but the line slope of the heuristic algorithm is much smaller. Fig. 8(c) shows the punishment on computation time when the rule fields increases. The brute-force method here shows an exponential increase when the number of rule fields grows and it costs about 7 hours for 11 fields (for the huge computation time, we did not run the experiment multiple times for the average consumption), while the heuristic algorithm remains the small increase trend.

5) Performance of MP-ROOM in Multiple-PDU Cases: We further evaluate the throughput and memory consumptions of MP-ROOM and ROOM in the context of multiple-PDU cases to demonstrate the improvement of MP-ROOM. The experiment takes an extra rule set containing 153 multiple-PDU rules as the input. These rules requires at least two PDUs for matching, and the average length of the required PDU is 3.4. Table IX shows that MP-ROOM is 14% faster and saves 20% memory than original ROOM. It mostly results from the organization of the multiple LMTs as we discussed in Section IV-A.

C. Experimental Results in System Level

nDPI [18] is a superset of the popular OpenDPI [27] library by extending the original rule set. The rules in nDPI are written in native code (C language) to enhance the throughput performance, which partially supports FGTI, e.g., 8 behaviors in QQ, 3 behaviors in Skype, etc. nDPI now provides more than 300 rules for 125 applications/protocols. We build the latest version of nDPI, and compare it with MP-ROOM from throughput and memory consumption respectively. The two aforementioned real traces are used for the comparison, and all evaluations are performed on the same platform mentioned in Section VI-A.

Throughput: Fig. 9(a) shows the experimental result for the throughput comparison. MP-ROOM achieves 8.49 Gb/s on the real traces at most with a much larger rule set (869), while nDPI performs 1.44 Gb/s at most on the same traces with only about 300 rules. The relative low performance of nDPI mainly results from the unorganized rule set. For nDPI does not support standard semantic-based rules as mentioned in Section III it uses native code to describe the fine-grained behaviors, which obviously lowers the performance, for each packet has to be checked bit by bit with each rule. On the other hand, native code naturally enhances the performance for it can be pre-compiled and optimized by the compiler. Therefore, nDPI performs better than sequential matching method but worse than NetShield and ROOM.

Memory Consumption: For nDPI does not pre-construct rules, we only evaluate the dynamic memory consumptions. Fig. 9(b) shows that MP-ROOM and nDPI cost 243.9 MB and 314.2 MB memory at most with the same two traces respectively. The major factor of the gap is that nDPI caches packets to make the complete identification, while MP-ROOM only tags the identification conditions and abandons the packets right after a PDU is assembled. Therefore, nDPI requires more memory with the volume growth of traffic while MP-ROOM remains low and stable memory cost. Combine throughput and memory cost, the performance-cost ratio of MP-ROOM and nDPI are 1.26 and 0.18, respectively.

VII. RELATED WORK AND DISCUSSIONS

There are several Intrusion Detection/Prevention Systems (IDS/IPS) inspecting the packet payload. Snort [16] uses PCRE [28] to match regular expressions with a huge number of reliable regex-based rules. The throughput of Snort is slow since it matches rules sequentially. Bro [17] is another popular IDS with good flexibility. Bro partially supports FGTI with an expressive rule language. Still, the matching speed of Bro is relatively slow [15].

In the context of traffic identification, previous literatures have explored header/port based method, content based method, machine learning method, and DPI-based method [3], [4], [7]. nDPI system [18] and its previous work OpenDPI [27] have paid a lot of efforts on detecting the applications, which can recognize over 100 popular applications including IM,

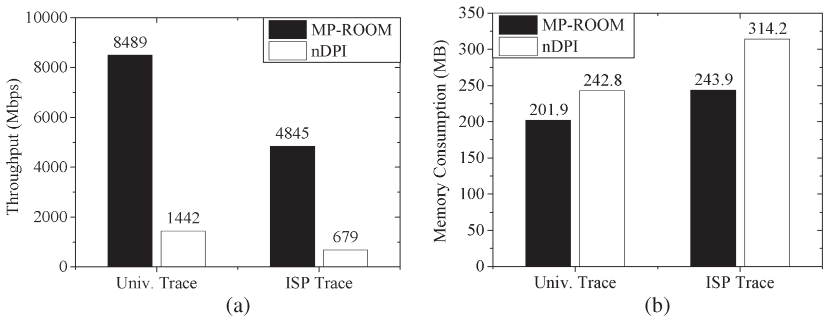

Fig. 9. Comprehensive comparison between MP-ROOM and nDPI. MPROOM achieves better throughput with lower memory consumptions on the real traces compared with nDPI. (a) Throughput comparison between MP-ROOM and nDPI. (b) Memory comparison between MP-ROOM and nDPI.

game, browser, etc. However, the rules of nDPI are not standard semantic-based, which harms the scalability of rules with the volume growth of traffic, and the rules are written in native code so it is hard to update the rule set without interrupting the system. Additional, the processing speed of nDPI is still slow.

Hierarchy organizations of rules have been studied in previous work [29], [30]. These techniques use exact string matching or hash comparing for the input payload instead of constructing a DFA-like structure. The matching speed of these techniques is not at the same level with DFA-based approach and the space complexity is not the focus of them. Thus, these techniques cannot be leveraged to achieve FGTI. Kruegel uses the hierarchy organization of rules to accelerate the matching speed, but it may suffer from the state space explosion [31]. Tongaonkar formulates the model of the hierarchy structure by using condition factorization and demonstrates the proposed automaton to be polynomial-size with optimizations [32]. [31], [32] focus on the hierarchy structure itself, since they assume that the width of the structure determines the space complexity. And MP-ROOM employs DFA to accelerate the matching speed inside matchers, so the number of matchers and the size of matchers both impact the memory consumption. Therefore, the heuristic algorithm of searching the optimal order in MP-ROOM not only considers the size of LMT, but also involves the size of the matchers, while it is not the focus in [32]. FDD is proposed to translate the firewall policies into a layered diagram, which is easier to analyze and optimize [19]. MP-ROOM translates the semantic-based rules into LMT in a similar way. However, they are designed for different purposes and completely orthogonal in different contexts as we mentioned in Section I.

DFA is usually used for matching [33]–[35] and an extend DFA structure to distinguish matching result is proposed in [22]. But these works focus on improving plain DFA structure and cannot handle the fine-grained rules directly. NetShield [15] which is involved in our comparisons makes a good start of organizing semantic-based rules, but MP-ROOM gains more from the optimal rule organization.

Discussion: Since MP-ROOM checks the payload of the packet in layer-7, privacy is the major concern of this kind of system. The end users worry that the Network Service Providers (NSPs) would snoop their private data through DPI technology. However, we argue that, MP-ROOM only focuses on the behaviors of users, not the detailed contents. For instance, NSPs can adjust the bandwidth according to the results of MP-ROOM/FGTI to achieve better voice chatting quality of an IM application. But the conversation is still private between end users for NSPs is not inspecting any of detailed contents. Technically, content providers often expose the controlling information of the application for easier decoding in server side, and encrypt the detailed contents in the payload of layer-7, which protects the privacy of users in another aspect.

The accuracy and coverage of the fine-grained rules determine the accuracy of the FGTI systems, since it performs precisely according to the rule set. There have been some previous work on generating the string/regex rules from training traffic automatically [36]. Although these works do not support FGTI rules directly, it can be a start point. We generate rules with the help of such tools and manual efforts in our experiments and will further look into this question in future work.

It is important to design a fast FGTI system to meet the increasing wire-speed in the core network, which has reached 40/100 Gb/s and even higher. There are two ways to improve the performance. First, some hardware-based approaches could achieve over 100 Gb/s throughput with FPGA or TCAM, but they could not support complex semantic-based rules. If we employ the ideas of MP-ROOM and other components into their hardware platforms, it is no doubt that the combined system can get a much higher performance. Second, some many core platforms such as Tilera TILE-Gx [37] have dozens of processing units, which make it possible to achieve 100 Gb/s in a single unit case. Also, the distributed systems with well-designed load balancing technique would accelerate the whole system indeed.

VIII. Conclusion

In this paper, we have proposed MP-ROOM, which constructs layered matching trees for semantic-based rules on multiple PDUs to accelerate the matching speed and to save the memory consumption. The optimal construction of LMT is demonstrated to be NP-hard, and therefore, a heuristic algorithm is developed. We also design and implement prototype systems for MP-ROOM and related works. The experiments under real traces and synthetic traces demonstrate that the performance-cost ratio of MP-ROOM is increased to 1.6 times of NetShield and 63 times of sequential method. Besides, a comprehensive evaluation shows that MP-ROOM prototype achieves 6.5 times higher throughput with only 80% memory cost compared with nDPI.

REFERENCES

[1] R. P. Mahowald and C. G. Sullivan, "Worldwide SaaS and cloud software 2012–2016 forecast and 2011 vendor shares," IDC Corporate USA, Framingham, MA, USA, 2011.
[2] T. Qiu *et al.*, "Packet doppler: Network monitoring using packet shift detection," in *Proc. ACM CoNEXT*, 2008, pp. 3:1–3:12.
[3] A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *Proc. 6th Int. Workshop PAM Netw.*, Mar. 2005, pp. 41–54.
[4] T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Commun. Surveys Tuts.*, vol. 10, no. 4, pp. 56–76, 2008.
[5] L. Peng *et al.*, "Traffic identification using flexible neural trees," in *Proc. IEEE IWQoS*, Jun. 2010, pp. 1–5.
[6] S.-H. Yoon, J.-S. Park, and M.-S. Kim, "Signature maintenance for internet application traffic identification using header signatures," in *Proc. IEEE NOMS*, Apr. 2012, pp. 1151–1158.
[7] Y. Choi, J. Y. Chung, B. Park, and J. Hong, "Automated classifier generation for application-level mobile traffic identification," in *Proc. IEEE NOMS*, Apr. 2012, pp. 1075–1081.
[8] J. Li, S. Zhang, Y. Lu, and J. Yan, "Real-time p2p traffic identification," in *Proc. IEEE GLOBECOM*, Dec. 2008, pp. 1–5.
[9] D. Sproul, NSP Success Hinges on Monetizing Pipe, 2012. [Online]. Available: http://mobiledevdesign.com/learning-resources/nsp-success-hinges-monetizing-data-pipe
[10] Z. Su, P. Ren, and Y. Chen, "Consistency control to manage dynamic contents over vehicular communication networks," in *Proc. IEEE GLOBECOM*, Dec. 2011, pp. 1–5.
[11] C. Hu *et al.*, "ANLS: Adaptive non-linear sampling method for accurate flow size measurement," *IEEE Trans. Commun.*, vol. 60, no. 3, pp. 789–798, Mar. 2012.
[12] H. Li and C. Hu, "ROOM: Rule organized optimal matching for fine-grained traffic identification," in *Proc. IEEE INFOCOM Mini-Conf.*, Turin, Italy, Apr. 2013, pp. 65–69.
[13] C. Hu *et al.*, "Disco: Memory efficient and accurate flow statistics for network measurement," in *Proc. IEEE ICDCS*, Jun. 2010, pp. 665–674.
[14] C. Hu *et al.*, "Accurate and efficient traffic monitoring using adaptive non-linear sampling method," in *Proc. IEEE INFOCOM*, Apr. 2008, pp. 26–30.
[15] Z. Li *et al.*, "Netshield: Massive semantics-based vulnerability signature matching for high-speed networks," in *Proc. ACM SIGCOMM*, 2010, pp. 279–290.
[16] Snort, Sourcefire 2013. [Online]. Available: http://snort.org
[17] V. Paxson, "Bro: A system for detecting network intruders in real-time," in *Proc. USENIX Security Symp.*, Berkeley, CA, USA, 1998, pp. 1–22.
[18] ntop, ndpi 2013. [Online]. Available: http://www.ntop.org/products/ndpi/
[19] M. G. Gouda and A. X. Liu, "Firewall design: Consistency, completeness and compactness," in *Proc. ICDCS*, Tokyo, Japan, Mar. 2004, pp. 320–327.
[20] S. Sahni and T. Gonzalez, "P-complete approximation problems," *J. Assoc. Comput. Mach.*, vol. 23, no. 3, pp. 555–565, Jul. 1976.
[21] Open Networking Foundation (ONF), Openflow Switch Specification Version 1.4.0 2013.
[22] G. Xia, X. Wang, and B. Liu, "SRD-DFA: Achieving sub-rule distinguishing with extended DFA structure," in *Proc. 8th IEEE Int. Conf., Dependable, Auton. Secure Comput.*, Dec. 2009, pp. 723–728.
[23] T. Nelms and M. Ahamad, "Packet scheduling for deep packet inspection on multi-core architectures," in *Proc. ACM/IEEE Symp. ANCS*, Oct. 2010, pp. 1–11.
[24] N. Schear, D. R. Albrecht, and N. Borisov, "High-speed matching of vulnerability signatures," in *Proc. 11th Int. Symp. Recent Adv. Intrusion Detection*, 2008, pp. 155–174.
[25] N. Borisov, D. J. Brumley, and H. J. Wang, "A generic application-level protocol analyzer and its language," in *Proc. 14th Annu. Netw. Distrib. Syst. Security Symp.*, 2007, pp. 216–231.
[26] W. Cui, M. Peinado, H. J. Wang, and M. E. Locasto, "Shieldgen: Automatic data patch generation for unknown vulnerabilities with informed probing," in *Proc. IEEE Symp. Security Privacy*, 2007, pp. 252–266.
[27] IPOQUE, Opendpi 2013. [Online]. Available: http://www.opendpi.org
[28] P. Hazel, PCRE Library 2012. [Online]. Available: http://www.pcre.org
[29] J.-H. Choi and M.-S. Kim, "Improved processing speed of traffic classification based on payload signature hierarchy," in *Proc. APNOMS*, 2013, pp. 1–3.
[30] S. Kawano, T. Okugawa, T. Yamamoto, T. Motono, and Y. Takagi, "High-speed DPI method using multi-stage packet flow analyses," in *Proc. 9th APSITT*, 2012, pp. 1–6.
[31] C. Kruegel and T. Toth, "Using decision trees to improve signature-based intrusion detection," in *Recent Advances in Intrusion Detection*. Berlin, Germany: Springer-Verlag, 2003, pp. 173–191.
[32] A. Tongaonkar, R. Sekar, and S. Vasudevan, "Fast packet classification using condition factorization," in *Applied Cryptography and Network Security*. Berlin, Germany: Springer-Verlag, 2009, pp. 417–436.
[33] R. Smith, C. Estan, and S. Jha, "XFA: Faster signature matching with extended automata," in *Proc. IEEE Symp. Security Privacy*, May 2008, pp. 187–201.
[34] Y. Xu, L. Ma, Z. Liu, and H. J. Chao, "A multi-dimensional progressive perfect hashing for high-speed string matching," in *Proc. ACM/IEEE ANCS*, 2011, pp. 167–177.
[35] K. Huang, D. Zhang, and Z. Qin, "Accelerating the bit-split string matching algorithm using bloom filters," *Comput. Commun.*, vol. 33, no. 15, pp. 1785–1794, Sep. 2010.
[36] Y. Wang *et al.*, "A semantics aware approach to automated reverse engineering unknown protocols," in *Proc. IEEE ICNP*, Austin, TX, USA, 2012, pp. 1–10.
[37] Tilera, Tilera Tile-gx 2013. [Online]. Available: http://www.tilera.com

Hao Li (S'13) received the B.S. degree in computer science from Xi'an Jiaotong University, Xi'an, China, in 2010. He is currently working toward the Ph.D. degree at the Department of Computer Science and Technology, Xi'an Jiaotong University. His research interests include network measurement and monitoring.

Chengchen Hu (S'04–M'09) received the B.S. degree from Northwestern Polytechnical University, Xi'an, China, and the Ph.D. degree from Tsinghua University, Beijing, China, in 2003 and 2008, respectively. From June 2008 to December 2010, he worked as an Assistant Research Professor with Tsinghua University. He is currently an Associate Professor with the Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, China. His main research interests include computer networking systems, and network measurement and monitoring. He severed in the organization committee and technical program committee of several conferences, e.g., IWQoS 2010, INFOCOM 2012/2013, GLOBECOM 2010–2012, ICC 2011–2013, etc.