ROOM: Rule Organized Optimal Matching for Fine-Grained Traffic Identification

Hao Li, Chengchen Hu
MOE Key Lab for Intelligent Networks and Network Security
Department of Computer Science and Technology
Xi'an Jiaotong University

Abstract—**Fine-grained traffic identification (FGTI) reveals the context/purpose of each packet that flows through the network nodes/links. Instead of only indicating the application/protocol that a packet is related to, FGTI further maps the packet to a meaningful user behavior or application context. In this paper, we propose a Rule Organized Optimal Matching (ROOM) for fast and memory efficient fine-grained traffic identification. ROOM splits the identification rules into several fields and elaborately organizes the matching order of the fields. We formulate and model the optimal rule organization problem of ROOM mathematically, which is demonstrated to be NP-hard, and then we propose an approximate algorithm to solve the problem with the time complexity of $O(N^2)$ (N is the number of fields in a rule). In order to perform evaluations, we implement ROOM and related work as real prototype systems. Also, real traces collected in wired Internet and mobile Internet are used as the experiment input. The evaluations show very promising results: 1.6X to 104.7X throughput improvement is achieved by ROOM in the real system with acceptable small memory cost.**

I. Introduction

Application traffic identification is a fundamental technology for network monitoring and measurement [1], which empowers people to better monitor, manage and control the networks. The traditional traffic identification, called Coarse-Grained Traffic Identification (CGTI) [2]–[5], only reports at the protocol or application granularity, *e.g.*, the traffic belongs to MSN or skype, but it fails to provide Fine-Grained Traffic Identification (FGTI) information for nowadays' complicated use, *e.g.*, tweeting using iPhone with Chrome.

FGTI system has to employ semantic-based rules instead of regular expression-based (regex-based) rules as 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 all the values in specified segments (*a.k.a.*, fields) of the packet matches with the rule. This requirement of first splitting the packet into different fields brings higher complexity. In addition, FGTI also employs more rules than CGTI to demonstrate more specific behaviors of application. Even only considering the simpler regex-based rules, it is demonstrated that the combination of a selected rule set with 794 regular expressions consumes 5.29GB memory [6].

FGTI is also different with the Intrusion Detection system (IDS). Only few packets related to intrusions would be matched in IDS, but every packet should have a match in FGTI system. Besides, IDS system will not do any further identification for a "bad" flow but just reset the connection, while FGTI system checks every single packet of a flow even it has matched for certain behaviors. Therefore, the previous studies cannot be directly used for FGTI.

This paper is supported by 973 plan (2012CB315901), 863 plan (SS2013AA010601), NSFC (61272459, 61221063), the Fundamental Research Funds for Central Universities.

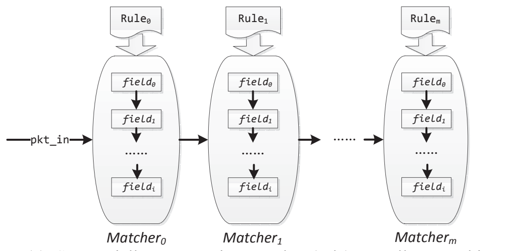
In this paper, we investigate efficient method to support semantic based rules for FGTI so as to provide high identification throughput with controlled memory consumption. We observe the feature that segments (*a.k.a.*, fields) in different matching rules of FGTI can share the same signature. Based on this observation, we propose a Rule Organized Optimal Matching (ROOM) to eliminate the matching on redundant fields. As a result, we exploit a better tradeoff between the time complexity and the memory complexity. 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 do the matching. We make the following contributions in this paper:

- We have proposed ROOM to construct a Layered Matching Tree (LMT), which reduces the space complexity and improve throughput for FGTI matching.
- We have demonstrated that the construction problem of an optimal LMT is NP-hard and we proposed an approximation algorithm to solve the problem.
- We have implemented a real system of ROOM for performance testing, which achieves 5Gbps matching throughput in average with only 20MB memory cost. Compared with previous work, ROOM improves the performance-cost ratio by 1.5 times to 23 times.

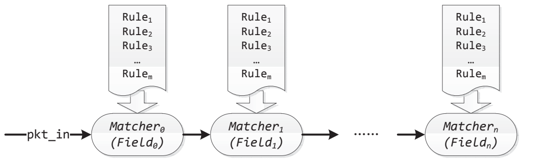
The remainder of the paper is organized as follows: In Section II, we present our basic idea of this paper. We describe the detailed design of ROOM in Section III. In Section IV, a prototype of ROOM is built and evaluated under real traces. And finally in Section V, we conclude the paper.

II. Basic Idea

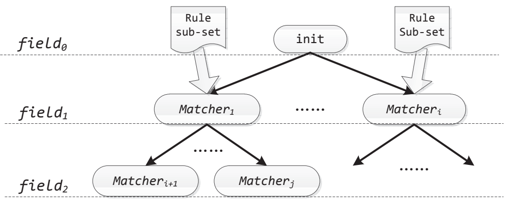
In the literature, there are two ways to match a large number of semantic-based rules in CGFI and IDS systems [6, 7]. As shown in Fig. 1(a), the first one constructs one matcher for each rule, by checking each field of the rule sequentially [7]. Matchers are checked one by one as well. The system moves to the next matcher if no match in current matcher, and stops when one of the matchers (rule) is hit. Obviously, the matching



(a) Sequentially constructing matcher (rule) as well as matching.



(b) All the rules can be splitted to the fields for DFA combination.



(c) The whole rule set is divided into sub-sets according to fields.

Fig. 1. Indicative comparisons of different matching systems.

speed is the major concern 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 first divided according to their fields, and then regular expressions in the same field can be merged to construct one DFA-based matcher, as shown in Fig. 1(b). The system stops when a matcher failed and moves to next matcher if any rule is hit [6]. Although DFA matcher saves the matching time, there are two main concerns: 1) memory explosion caused by the combination of values in rules brings uncertain risk. 2) 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 severe with the growing number of both rules and fields.

The idea of the proposed ROOM in this paper is depicted in Fig. 1(c). First, the whole rule set is divided into sub-sets according to fields, which is similar with the second method, to maintain the matching speed of DFA-based matcher. Besides, the transitions between matchers remove the extra overhead for intermediate results in the second method, which may lead to a higher performance. Second, the next matcher is determined by the matching result of the current matcher, that every matcher is constructed on a rule sub-set, which leads to lower risk on memory explosion.

To give an intuitive explanation on how ROOM works, we suppose a simple rule set in Table I, where a column represents a packet field and a row is a matching rule. 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. Suppose that a packet holding "$weibo.cn$" in its "Host" field and "$/ttt/gettimeline.php$" in its "URI" field comes into ROOM based system. In the first step, ROOM searches "Host" field of the packet in $Matcher_{0,0}$ and selects the left branch

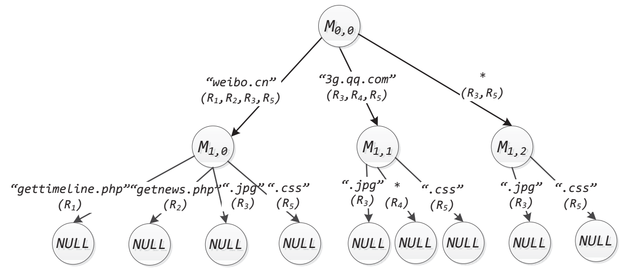


Fig. 2. Matchers constructed by ROOM on the rule set in Table I.

to activate the $Matcher_{1,0}$, and ROOM matches "URI" field of packet with this matcher to hit on the left branch. For $Matcher_{1,0}$ is on the last field, we can get R_1 as the final hit rule. As a result, the key problem is how to construct LMT based on the rule set, which will be present in Section III.

RuleID	Field == "Host"	Field == "URI"
1	weibo.cn	gettimeline.php
2	weibo.cn	getnews.php
3	*	.jpg
4	3g.qq.com	*
5	*	.css

TABLE I
SIMPLIFIED RULE SET SAMPLE

III. CONSTRUCTING THE MATCHERS OF ROOM

A. Primary Matcher Construction

The matchers are sorted into a LMT as shown in the example of Fig. 2. In each layer, the matchers are constructed according to a field. In a matching field i, there can be several matchers and we use $M_{i,k}$ to denote the kth matcher for the ith field[1]. We define a matcher as a two-tuple $(ProtocolField, InitRuleSet)$, where $InitRuleSet$ describes the whole set of the matcher and its descendant, and $ProtocolField$ determines which part (field) of $InitRuleSet$ will be used for the matcher. If the maximum number of fields is 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 $M_{1,j}$, denoted by $H_{1,j}$. 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:

[1]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.

$H_{0,0} \to \{F_{1,j}, V_{1,j}\} \to \{H_{1,j}, S_{1,j}\} \to \cdots \to \{H_{i,k}, S_{i,k}\} \to \{F_{i+1,l}, V_{i+1,l}\} \to \{H_{i+1,l}, S_{i+1,l}\} \to \ldots$.

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.

B. Organize the Order of Rule Fields

The change of the field order in the aforementioned matcher construction algorithm will lead to different LMTs. In the example of Fig. 2, the construction takes "Host" as the first field and "URI" as the second. If we disorder the two fields, LMT will be in the form of Fig. 3, which has 5 matchers in the second layer. The memory consumptions of these two LMTs are obviously different that Fig. 3 has 94 nodes in total while Fig. 2 has 65 only.

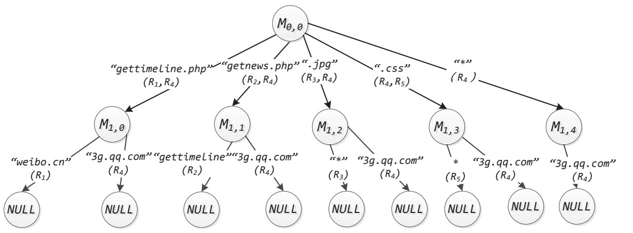


Fig. 3. Disordered Layered Matching Tree

This example brings a question: what is the optimal order of the fields? 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.

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

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 to Quadratic Assignment Problem (QAP) [8]. QAP considers allocating n facilities to n locations. There are three costs in the problem: one is the cost function of distance between locations, the second is the function of flow between facilities and the last is 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 our ROP to QAP, 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), (3) can be transferred to (2) by mapping the variables and functions: $N \to n, D \to f, I \to d, T_N \to S_n, \Psi \to \phi$. f and d are given in QAP, but D and I in ROP need to be calculated from the rule set. It is demonstrated in [9] that the calculation of the memory consumption about merging of DFAs can be completed in Polynomial time. So the reduction can be done in polynomial time. Since QAP is known as an NP-hard problem [8], ROP is also NP-hard with a polynomial reduction to QAP. As a result, we need to propose an approximate algorithm to organize the rule field order so as to get the optimal memory consumption.

The number of matchers in a certain layer impacts its complexity, that more matchers leads to many smaller DFAs instead of a few large DFAs which brings less overhead. So it tends to choose fields bringing more branches for higher layers. However, such kind of field may also bring huge overhead for current layer, which is a trade-off. Besides, we can see that R_3 and R_5 are replicated more than once in Fig. 2. It results from the "$*$" cases of the first field which brings copying work for these two rules. The effect of such kind of redundancy depends on two factors: the complexity of the redundant values brought by "$*$" cases, and the number of branches, to which all "$*$" rules would be copied. The product of these two factors describes the redundant effect brought by "$*$" cases.

In general, we aim to choose a field, which brings more branches for next layer, lower complexity for current layer and less redundancy to LMT. Here we first define a notation C, where $C(RuleSet)$ represents the number of nodes of the matcher constructed by rules in $RuleSet$ 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.

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

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 number of branches. Now we can use $W_{field,i}$ to 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 chooses the field holding G_{min}. As this calculation has to be done each time choosing a next field, the time complexity is $O(N^2)$ where N represents the number of the fields.

In the case of rules in Table I, for the first layer, we compute $C_{\text{“}Host\text{”}} = 18$, $C_{\text{”}URI\text{”}} = 30$. Then, $W_{\text{“}Host\text{”},0} = (C(\{\text{“}.jpg\text{”}, \text{“}.css\text{”}\}) \times 2 = 8 \times 2$, and $W_{\text{“}URI\text{”},0} = (C(\{\text{“}3g.qq.com\text{”}\}) \times 4 = 10 \times 4$. Then we can get $G_{\text{“}Host\text{”},0} = (C_{\text{“}Host\text{”},0} \times W_{\text{“}Host\text{”},0})/|V_{\text{“}Host\text{”},0}| = (18 \times 16)/3 = 96$, and $G_{\text{“}URI\text{”},0} = 240$ by the same way. So “Host” is chosen for the first layer. Then we calculate the G from the rest fields for the next layer . Finally, we get an optimal fields' order: “Host” and “URI”, which is just the order we depict in Fig. 2.

C. Further Optimizations

We propose two more optimizations to further save the memory. As shown above, a matcher is defined as a two-tuple $(ProtocolField, InitRuleSet)$. It is possible that multiple matchers share same two-tuple. In this case, we merge such matchers and their descendants as well.

On the other hand, copying “$*$” case impacts the overhead a lot. As we can see from Fig.2, LMT copies R_3 and R_5 to every matcher in “Host” field, because a packet matches “weibo.cn” in this field may finally match R_3 or R_5 although they don't depend on the field. However, we find some fields provide semantic-proof to exclude this possibility. For instance, “Host” is a field to describe the destination host domain in HTTP request. If a packet matches a certain “Host” value, it must be related to this certain host server, which excludes the rules holding “$*$”. We define Critical Field (CF) simply as follows: a field that can make a partition of the rule set in this layer by using its semantic. For application layer protocols are generally based on strong semantics, it is easy to find a CF in an application protocol, *e.g.*, “Host” in HTTP protocol, “DomainName” in DNS protocol *etc*.

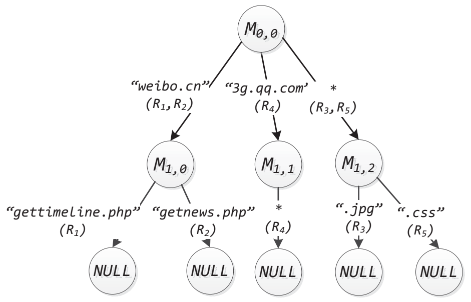


Fig. 4. Optimal Rule Set Organization

The optimal LMT for rules in Table I is shown in Fig. 4. Compared with the LMT in Fig. 2, it now only contains four matchers without any copying of “$*$” cases as the original one.

IV. Evaluations

A. Experimental Settings

In this section, we compare the performance of ROOM with related work. To the best of our knowledge, NetShield [6] achieves the best performance with semantic-based rules, which has the similar structure as Fig. 1(b), and we first involve it in the comparison. We also implement sequential matching method for comparison whose architecture is depicted in Fig. 1(a). Each of these three methods is implemented into a prototype system for evaluations. The evaluations are performed on a platform with Intel Xeon E5606 (2.13 Ghz), 32 GB memory and Linux 2.6 kernel. Each system is with single thread for clean comparisons.

	Univ. trace	ISP trace	Synthetic trace
Capture time	05/22/2012	12/20/2010	04/05/2012
Duration	67 min.	40 min.	N/A
Trace size	19GB	4.9GB	731MB
Mean pkt. length	793 B	572 B	342 B
# of behaviors	27360	99641	143922

TABLE II
Specification of Selected Traces

There are 262 experimental rules, which contains 6 independent fields. The rule set can identify both wired Internet and mobile Internet application behaviors. The rules for wired Internet traffic are selected from Snort [10] and are rewritten to be semantic-based rules. The rules for mobile Internet traffic are provided by KOP (Keen on Packet) DPI [11], which cover the behaviors of 13 popular mobile applications for their iOS, Symbian and Android versions, *e.g.*, chatting of skype in iOS, refreshing of Weibo in Android, *etc*.

Two real traces and one synthetic trace 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 real trace is collected in Hangzhou, Zhejiang province of China, from a Radio Network Controller (RNC) of a leading ISP in China, denoted as “ISP trace”. A third experimental trace utilized is an artificial and random mix of a set of “atom-behavior” (single behavior of a certain application) traces generated by mobile devices like phones and pads. The synthetic trace is employed to simulate various situations in real world and to confirm the identification accuracy with the ground truth. Table II lists the features of the three traces.

During the evaluations, we have checked three metrics of the matching systems: identification throughput, memory cost and performance-cost ratio (which is defined as the normalized throughput divided by the normalized memory cost).

B. Experimental Results

1) Throughput: When testing the throughput, we pre-load the traces into the memory to remove the bottleneck in hard disk I/O. Fig. 5 illustrates the tested throughput. ROOM performs 6.28Gbps, 3.19Gbps and 2.736 Gbps throughput on the univ. trace, the ISP traces and the synthetic trace, respectively. As shown in Table III, ROOM provides a speedup from 1.61 to 2.08 over NetShield and a speedup from 24.8 to 104.7 over sequential matching. In fact, two facts affect the throughput. The first effect comes from the number of behaviors. A large number of behaviors obviously make the sequential matching slow as demonstrated in the figure. However, the behavior number does not affect ROOM and NetShield much. The throughput differences in different traces for ROOM and NetShield are caused by the mean packet length, which is the second fact to the throughput. For all the methods, smaller mean packet length leads to higher throughput since matching is performed on each packet.

	Univ. trace	ISP trace	Synthetic trace
ROOM	1.0	1.0	1.0
NetShield	1.61	1.85	2.08
Sequential	104.7	39.8	24.8

TABLE III
THROUGHPUT SPEEDUP OF ROOM OVER OTHER METHODS

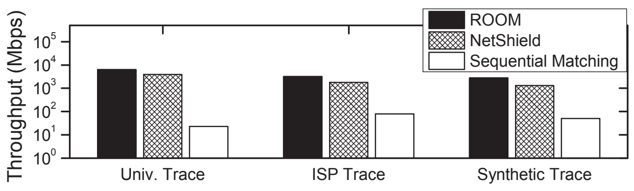


Fig. 5. Matching throughput plotted with logarithmic coordinates.

2) Memory Consumption: There are two categories of memory consumptions: one is the flow structure for maintaining the flow information and the other is the matching data structures representing different rule set organizations. The first part consumption is the same for the three methods so we only check the second one. The memory for matching data structure can be further divided into static memory usage and dynamic memory usage. The former is used to store the matcher and the latter is for caching the temporary intermediate results. Table IV shows the memory usage. The results indicate that ROOM costs a little bit more static memory. The experimental rule set is not very complex which leads no exponential growth inside matchers, so the effect brought by "$*$" case is the dominant factor for the static consumption. When processing the matching, ROOM shows its advantage in dynamic usage over NetShield for it has no intermediate results. When it comes to synthetic trace which is more complicated, ROOM performs better than NetShield in total space complexity. It can be predicted that if the rule set gets more complicated as well as the trace, ROOM can get even better performance. The comparison ratios (the overall memory cost of ROOM divided by the cost of Netshield) are 1.1065, 1.0848 and 0.9627 for the Univ. trace, ISP trace and Synthetic trace, respectively.

3) Performance-Cost Ratio: To give a fair comparison of the three methods over both throughput performance and memory consumption, we check the performance-cost ratio R, which is defined as the normalized throughput divided by the normalized memory cost.

$$R = \frac{T/T_{max}}{M/M_{max}}, \tag{7}$$

where T is the achieved throughput and M is the consumed memory cost. T and M are normalized by the maximum throughput T_{max} and memory cost M_{max}. The definition keeps the value of R in the range of $[0, 1]$ and larger R means larger efficiency. Table V illustrates such kind of performance gain of throughput over memory cost for ROOM, NetShield and sequential matching. The results confirm the proposed idea in Section II: ROOM provides the best tradeoff between the processing speed and the memory cost.

		Univ. trace	ISP trace	Synthetic trace
ROOM	Static	4.3MB	4.3MB	4.3MB
	Dynamic	27.9MB	20.0MB	11.2MB
NetShield	Static	2.7MB	2.7MB	2.7MB
	Dynamic	26.4MB	19.7MB	13.4MB
Sequential	Static	1.1MB	1.1MB	1.1MB
	Dynamic	18.6MB	12.2MB	4.4MB

TABLE IV
MEMORY CONSUMPTION

	Univ. trace	ISP trace	Synthetic trace
ROOM	1.0000	0.6731	0.9031
NetShield	0.6872	0.4074	0.4172
Sequential	0.0156	0.0308	0.1025

TABLE V
PERFORMANCE-COST RADIO

V. CONCLUSIONS

In this paper, we have proposed ROOM, which constructs a layered matching tree for semantic-based rules to accelerate the matching speed and to save the memory consumption. The optimal construction of the LMT is demonstrated to be NP-hard and therefore an approximate algorithm is developed. We also implement prototypes for ROOM, NetShield and sequential matching. The experiments under two real traces and one synthetic trace demonstrate that ROOM improves the performance-cost ratio by 1.5 times to 2.4 times than NetShield and 16 times to 23 times than sequential method. As the throughput evaluation in Section IV is single-thread-based, and it can be easily extended to a multi-thread implementation to achieve higher performance. We leave it as future work. In addition, ROOM may check the payload in application layer, so privacy could be a concern. However, ROOM is provided as a tool for service and content providers to extract insensitive information, and as a result they can provide better services to users, *e.g.*, application promotion and acceleration.

REFERENCES

[1] Tongqing Qiu, Jian Ni, Hao Wang, Nan Hua, Y. Richard Yang, and Jun Jim Xu. Packet doppler: network monitoring using packet shift detection. In *ACM CoNEXT*, pages 3:1–3:12, New York, NY, USA, 2008. ACM.

[2] Andrew W. Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *6th International Workshop on Passive and Active Network Measurement (PAM)*, March 2005.

[3] T.T.T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys Tutorials*, 10(4):56 –76, quarter 2008.

[4] Yeongrak Choi, Jae Yoon Chung, Byungchul Park, and J.W. Hong. Automated classifier generation for application-level mobile traffic identification. In *IEEE NOMS*, pages 1075 –1081, April 2012.

[5] A. Dainotti, F. Gargiulo, L.I. Kuncheva, A. Pescape and, and C. Sansone. Identification of traffic flows hiding behind tcp port 80. In *IEEE ICC*, pages 1 –6, may 2010.

[6] Zhichun Li, Gao Xia, Hongyu Gao, Yi Tang, Yan Chen, Bin Liu, Junchen Jiang, and Yuezhou Lv. Netshield: massive semantics-based vulnerability signature matching for high-speed networks. In *ACM SIGCOMM*, pages 279–290, New York, NY, USA, 2010. ACM.

[7] ntop. ndpi. "http://www.ntop.org/products/ndpi/".

[8] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the Association of Computing Machinery*, 23:555C565, 1976.

[9] Yanbing Liu, Li Guo, Muyi Guo, and Ping Liu. Accelerating dfa construction by hierarchical merging. In *IEEE ISPA*, pages 1 –6, may 2011.

[10] Sourcefire. Snort. "http://snort.org".

[11] ANTS. KOP DPI. "https://github.com/antsgroup/KOP-DPI-System".