Poster: Novel Opportunities in Design of Efficient Deep Packet Inspection Engines

Anton Chekashev
ITMO University

Vitalii Demianiuk
Ariel University

Kirill Kogan
Ariel University

Abstract—**Deep Packet Inspection (DPI) is an essential building block implementing various services on data plane [5]. Usually, DPI engines are centered around efficient implementation of regular expressions both from the required memory and lookup time perspectives. In this paper, we explore and generalize original approaches used for packet classifiers [7] to regular expressions. Our preliminary results establish a promising direction for the efficient implementation of DPI engines.**

I. Introduction and motivation

Deep packet inspection (DPI) is a fundamental infrastructure in security network services detecting pathological traffic patterns in the network. DPI for network security combines early IDS (*intrusion detection system*) and latter IPS (*intrusion prevention system*). IDS detects threats and triggers alerts. IPS takes measurements to prevent possible threats (e.g., by terminating suspicious connections). Snort [11] is widely used open-source IDS and IPS, Bro [6] is widely used open-source IDS. An additional application of DPI is data loss prevention (DLP) [2] focusing on sensitive data leaving an intranet. Also there are other applications of DPI as bandwidth management [3], copyright enforcement [9], etc. The central part of DPI engines is an implementation of the lookup mechanism finding a matched pattern for incoming traffic. Usually, patterns in DPI are represented by regular expressions whose efficiency should be deterministic and independent of traffic arrivals; otherwise, performance gap can be used for DDOS attacks [1]. For instance, Bro [6] and Snort [11] are both vulnerable to this kind of attacks [8]. Hence, lookup in DPI engines should be implemented at line rate.

One possible representation of looked up patterns is a combined deterministic finite automaton (DFA) of multiple regular expressions representing pathological patterns. In this case, to find matched regular expressions, it is sufficient to make one traversal over a looked-up input string. Unfortunately, such representations can be infeasible since the size of such combined DFA can be exponential on the total number of represented regular expressions. To reduce memory requirements for representations of regular expressions, one can consider usage of non-deterministic automatons (NFA) since the number of states in a representing NFA is proportional to the sum of lengths of all given regular expressions. In this case, during a lookup, we have to maintain a set of all active states that can be reached by the current prefix of a looked-up input string; during character processing, we should apply state transitions for all these active states. Hence, the lookup time heavily depends on the number of all active states, which in the worst case can be equal to the number of states in the whole representing NFA that make representations based on NFAs also infeasible. To address limitations of representations based on DFAs and NFAs, [4] proposed a Hybrid-FA representation combining advantages of both DFA and NFA reprsentations. Unfortunately, the number of active states can be still big even for Hybrid-FAs that can be a reason for significant perfomance degradation of the lookup time. Moreover [1] introduced algorithmic complexity attacks on Hybrid-FA.

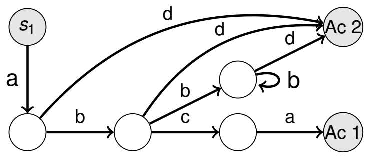
II. Proposed solution

Design principles of DPI engines that mitigate lookup time and required memory is still an intriguing open problem. We explore novel representations of multiple regular expressions addressing a fundamental tradeoff between required memory and lookup time. Based on these principals, someone can develop DPI engines with the following characteristics:

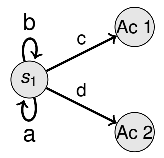
- **High-speed processing**. An incoming string is looked up into a constant number of DFAs that guarantees efficient lookup time both in average and worst cases since each match to DFA is a single traversal along an input string.
- **Low-memory cost**. All constructed DFAs will consist of a small number of states confirming feasibility of the proposed representation supporting scalable number of pathological patterns.

There are two types of regular expressions representing pathological patterns. The regular expressions of the first type check if an input string s matches represented patterns; expressions of the second type check if s has a substring matching a represented pattern. In this paper, we are mostly focusing on the first type of regular expressions; in real scenarios, they correspond to small DFAs. The major problem is how to support a big number of such patterns in small memory and with moderate lookup time.

In DPI engines (like Snort [11]) most regular expressions of the first type are *value-disjoint*, i.e cannot match the same input string. We exploit this property and design efficient DPI engines based on the proposed below design principles. Actually, we generalize the original ideas that we used in our SIGCOMM paper [7] to represent value-disjoint packet classifiers. In this case, we can split processing into two consecutive stages: (1) find a “potentially” matched regular expression r representing a single pathological pattern; (2) perform a false-positive check verifying if a found r matches an incoming input string.



(a) combined incomplete DFA



(b) combined seDFA

Fig. 1. Combined representations of the two regular expressions: *^abca* and *^ab*d*; AC 1 is a terminal state for *^abca*, AC 2 is a terminal state for *^ab*d*.

Algorithm 1 StateShrink($\mathcal{R}$)

Input: a set of regular expressions $\mathcal{R}$
Output: seDFA D representing $\mathcal{R}$

```
1: $\mathcal{N} \leftarrow$ a set of all NFAs representing regular expression in $\mathcal{R}$
2: combine all NFAs in $\mathcal{N}$ into a single incomplete DFA $D$
3: for every state $s \in D$ do
4:     if only one terminal $t \in D$ is reacheable from $s$ then
5:         remove all transitions outgoing from $s$
6:         shrink $s$ and $t$
7: repeat
8:     was_merge = False
9:     for every pair of states $u, v \in D, u \neq v$ do
10:        if every pair of transitions outgoing from $u$ and $v$ by the
           same symbol lead to the same state then
11:            shrink $u$ and $v$
12:            was_merge = True
13: until was_merge = False
```

To implement the first phase, we introduce a notion of a *semi-equivalent* automaton (seDFA) which is an incomplete DFA[1] with the following properties: (1) each represented regular expression has a separate terminal state in seDFA; (2) if an input string s is matched by a regular expression x, s leads to the terminal state corresponding to x in seDFA. If an input string s is not matched by any represented regular expression, the lookup result of s in D can be anything. To find a matching regular expression for an input string s, we need to lookup s in seDFA followed by the false-positive check with the regular expression corresponding to the terminal state in seDFA matching s. The false-positive check can be implemented by DFA, NFA or any other representation satisfying memory and lookup time constraints. Due to value-disjointness only a single pattern can be matched false-positively and this is a reason why an additional lookup will be required to verify this single matched pattern.

In practice, the size of seDFA is significantly smaller than the size of the corresponding combined incomplete DFA. The intuition behind this that a lot of space is required to guarantee correct results for unmatched strings. For instance, in Figure 1, the number of states in seDFA is smaller than the number of states in the combined incomplete DFA by 4 states since existence of c or d in an input string is sufficient to determine the regular expression for a false-positive check.

Problem 1. *For a given set of regular expressions, construct seDFA minimizing the number of states in seDFA.*

The main idea of seDFA representations is similar to the

[1]A DFA D is incomplete if for some states and symbols corresponding transitions may be undefined. If during lookup in D, there is no transition for a current symbol of an input string, the lookup terminates.

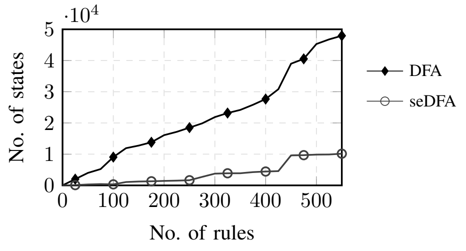


Fig. 2. The number of states in the combined representation as a function of the number of represented order-independent regular expressions.

representations proposed in [7]. However, minimizing the sizes of these representations are completely different problems: [7] finds subsets of classification fields preserving order independence, and we construct seDFA. In the preliminary study, we propose StateShrink heuristic solving Problem 1 (see Algorithm 1). First, StateShrink combines all NFAs representing given regular expressions into a common incomplete DFA D using the powerset construction [10] (line 2). Then, StateShrink shrinks every non-terminal state s into terminal t if t is a unique terminal in D that is reachable from s (lines 4-9). Finally, StateShrink iteratively shrinks pairs of states in D while possible as shown in lines 10-18 of Algorithm 1.

Theorem 1. StateShrink *constructs a valid seDFA.*

Even such simple heuristics construct efficient seDFA for real-world regular expressions from a Snort database: in Figure 2, the number of states in seDFA representing 500 value-disjoint rules is 5 time smaller than the number of states in the corresponding combined DFA. In the future study, we propose more efficient heuristics constructing combined seDFA and *semi-equivalent* representations in the case when not all regular expressions are value-disjoint.

Acknowledgments: This work was supported by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber directorate in the Prime Minister's Office.

References

[1] Y. Afek, A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral. Making dpi engines resilient to algorithmic complexity attacks. *IEEE/ACM Transactions on Networking*, 24(6):3262–3275, December 2016.
[2] S. Alneyadi, E. Sithirasenan, and V. Muthukkumarasamy. A survey on data leakage prevention systems. *J. Netw. Comput. Appl.*, 62(C):137–152, Feb. 2016.
[3] H. Asghari, M. Eeten, J. Bauer, and M. Mueller. Deep packet inspection: Effects of regulation on its deployment by internet providers. *SSRN Electronic Journal*, 01 2013.
[4] M. Becchi and P. Crowley. A hybrid finite automaton for practical deep packet inspection. In *CoNEXT*, pages 1:1–1:12, 2007.
[5] A. Bremler-Barr, Y. Harchol, D. Hay, and Y. Koral. Deep packet inspection as a service. CoNEXT '14, page 271–282, 2014.
[6] The Bro network security monitor. http://www.bro.org.
[7] K. Kogan, S. I. Nikolenko, O. Rottenstreich, W. Culhane, and P. T. Eugster. Exploiting order independence for scalable and expressive packet classification. *Trans. Networking*, 24(2):1251–1264, 2016.
[8] W. Lee, J. B. D. Cabrera, A. Thomas, N. Balwalli, S. Saluja, and Y. Zhang. Performance adaptation in real-time intrusion detection systems. In *Recent Advances in Intrusion Detection*, 2002.
[9] M. Mueller, A. Kuehn, and S. Santoso. Policing the network: Using dpi for copyright enforcement. *Surveillance & Society*, 9, 06 2012.
[10] M. O. Rabin and D. S. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2):114–125, 1959.
[11] Snort: The open source network intrusion detection system. http://www.snort.org.