2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)

Deep Packet Inspection in FPGAs via Approximate Nondeterministic Automata

Milan Češka, Vojtěch Havlena, Lukáš Holík, Jan Kořenek,
Ondřej Lengál, Denis Matoušek, Jiří Matoušek, Jakub Semrič, and Tomáš Vojnar

Brno University of Technology, Faculty of Information Technology, IT4I Centre of Excellence, Czech Republic

***Abstract*—Deep packet inspection via regular expression (RE) matching is a crucial task of network intrusion detection systems (IDSes), which secure Internet connection against attacks and suspicious network traffic. Monitoring high-speed computer networks (100 Gbps and faster) in a single-box solution demands that the RE matching, traditionally based on *finite automata* (FAs), is accelerated in hardware. In this paper, we describe a novel FPGA architecture for RE matching that is able to process network traffic beyond 100 Gbps. The key idea is to reduce the required FPGA resources by leveraging approximate nondeterministic FAs (NFAs). The NFAs are compiled into a multi-stage architecture starting with the least precise stage with a high throughput and ending with the most precise stage with a low throughput. To obtain the reduced NFAs, we propose new approximate reduction techniques that take into account the profile of the network traffic. Our experiments showed that using our approach, we were able to perform matching of large sets of REs from SNORT, a popular IDS, on unprecedented network speeds.**

I. INTRODUCTION

Intrusion Detection Systems (IDSes), such as SNORT [1], SURICATA [2], or BRO [3], are widely used to secure Internet connection against attacks and malicious traffic. One of the prominent approaches for IDSes is *deep packet inspection* (DPI), which is based on matching *regular expressions* (REs) describing attack patterns against network traffic. Despite the recent rapid increase of encrypted traffic on the Internet, RE-based DPI is still in high demand since IDSes are often deployed at network entry points after decryption.

Due to the increasing number of security vulnerabilities and network attacks, the number of REs in IDSes is constantly growing. At the same time, the speed of networks is growing too—telecommunication companies started to deploy 100 Gbps links, the 400 Gbps Ethernet standard has recently been ratified [4], and large data centers already call for a 1 Tbps technology. Consequently, despite many proposed optimisations, existing IDSes are still far from being able to process the traffic in current high-speed networks at the line speed. The best software-based solution we are aware of is the one in [5], which can achieve a 100 Gbps throughput using BRO on a cluster of servers with a well-designed distribution of network traffic. Processing network traffic at such speeds in single-box IDSes is far beyond the capabilities of software-based solutions—hardware acceleration is needed.

A well-suited technology for accelerating IDSes is that of *field-programmable gate arrays* (FPGAs). They provide high computing power and flexibility for network traffic processing, and they are increasingly being used in data centers [6], [7] for this purpose. The flexibility of FPGAs allows them to match REs at speeds over 100 Gbps [8]. Such high speeds, however, put excessive demands on the resources of FPGAs. The sets of the matched REs are complex, large, and still growing, and matching on the speeds of tens and hundreds of Gbps requires massive parallelization. For instance, in the HW architecture that we propose in Section II, processing 100 Gbps input network traffic requires 64 concurrently functioning RE matching units (of 8-bit input width operating at 200 MHz) and processing 400 Gbps requires even 256 units. These demands easily exceed the size of any available FPGA chip. Reducing the consumed resources is thus of paramount importance.

The FPGA matching units traditionally implement *finite automata* (FAs), either *deterministic* (DFAs) or *nondeterministic* (NFAs). In this paper, we focus on NFAs since they are often much smaller than the corresponding DFAs and can be efficiently mapped into FPGAs as shown, e.g., by [9]–[13].

The main conceptual difference from other works on efficient synthesis of NFAs into FPGAs (described in more detail in the related work section), which allows us to achieve much higher matching speeds, lies in leveraging *reduced* NFAs that *over-approximate* the language of the original NFA—we design novel reduction techniques that provide high-quality tradeoffs between the precision and reduction factors.

Subsequently, in order to utilise the resources of an FPGA in the best possible way, we propose a *multi-stage* architecture of the RE matching engine. Consider an NFA $\mathcal{A}$ that recognizes the language L of a given set of REs. The proposed architecture is composed of several stages where the first stage contains many, concurrently running, copies of small NFAs that crudely over-approximate L, and further stages contain a smaller number of copies of NFAs that are larger but more precise. The throughput of the stages decreases: the first stage needs to be able to process the network traffic at the line speed while the subsequent stages can handle less. The task of every stage is to decrease the amount of traffic entering the next stage by removing some of the packets that are guaranteed not to be in L. The last stage contains either $\mathcal{A}$, or, in the case of insufficient FPGA resources, a reduced version of $\mathcal{A}$ that over-approximates L (in which case the final processing step takes place in software). The NFAs used in the over-approximating stages need to discard a significant portion of their input from further processing while keeping all of the suspicious traffic. In

2576-2621/19/$31.00 ©2019 IEEE
DOI 10.1109/FCCM.2019.00025

IEEE computer society

the worst case, i.e., when every input packet is a member of L, this is, naturally, impossible. Fortunately, in standard traffic on a backbone network, only a small portion of the packets is normally in L (the extreme case when a significant proportion of network traffic is in L usually corresponds to a DoS attack, which needs to be handled by other means than IDSes).

Our approximate NFA reduction takes an advantage of particularities of standard network traffic. Namely, given an NFA constructed from the REs of interest, we label its states with their *significance*—the likelihood that they will be used during processing a packet—, and then simplify the least significant parts of the automaton. The simplification is implemented by *pruning* and *merging* of the insignificant states. The significance of a state is determined using *training traffic*, a finite sample of "standard" traffic from the network node where the IDS is to be deployed (it may be necessary to generate a new design once in a while). The reduction scales well—the worst-case time complexity of the most expensive step, computing the state labelling, is $\mathcal{O}(n^2k)$ where n is the number of states of the NFA and k is the size of the training traffic (since the automata are usually sparse, the quadratic factor is rarely an issue on real-world examples).

We implemented the proposed approach and evaluated it on REs taken from the IDS SNORT and other resources. We were able to obtain a substantial reduction of the size of the NFAs while keeping the number of false positives low. When used within the multi-stage architecture, we were able to perform RE matching at 100 and 400 Gbps on sets of REs whose sizes were far beyond the capabilities of existing solutions.

The contributions of this paper are the following:

1) We propose a new HW architecture for matching REs in high-speed computer networks using several stages of NFAs that over-approximate the matched REs.
2) We developed new scalable over-approximation techniques that take into account the profile of network traffic and thus provide great reductions of the NFAs while keeping an acceptable error.
3) We present experimental results proving that our approach can indeed be successfully used to implement single-box hardware-accelerated IDSes at speeds of 100 Gbps or even 400 Gbps, which is, taking into account the size and complexity of the considered REs, far beyond the capabilities of current solutions.

Related Work

Many different architectures for resource-efficient mapping of NFAs for fast RE matching into FPGAs have been designed, starting with the work of Prasana and Sidhu [9]. Later, Clark and Schimmel [10] reduced hardware resources by using a shared decoder of input symbols, and the architecture of Sourdis *et al.* [11] implements an NFA transition table by a pre-decoded CAM. Lin *et al.* [12] introduced an architecture that allows hardware resources to be shared for matching REs with the same prefixes, infixes, and suffixes. Furthermore, the work of Yun and Lee [13] improved the encoding of states using the so-called at-most-two-hot encoding.

To increase the RE matching speed, some architectures make the NFAs process multiple bytes of the input per clock cycle. Prasanna *et al.* [14] introduced spatial stacking for multi-character matching, but the high fan-out of the automaton significantly decreases its frequency, already for matching 8 bytes per clock cycle. Achieving the 100 Gbps throughput, which requires more than 64 bytes to be processed at once, is thus not possible. Becchi and Crowley [15] introduced a multi-striding technique, which is widely used to increase the throughput of many NFA-based RE matching architectures. Multi-striding alone, however, cannot increase the processing speed to 100 Gbps because with the length of the stride, the NFA grows rapidly, and the frequency drops dramatically [8].

We build on our previous work [8] introducing parallel pipelined automata, which can scale the throughput of NFA-based RE matching to over 100 Gbps. The processing speed is increased at the cost of a linear growth of hardware resources because the architecture consists of many automata connected in one processing pipeline. In a recent work, a novel FPGA architecture that significantly improved the throughput of DFA-based RE matching has been proposed by Yang *et al.* [31]. The architecture achieves a throughput of 140 Gbps on a single FPGA chip for IDS modules where the underlying *DFAs* have up to 10k states (corresponding to 34 REs of SNORT). Our new multi-stage architecture, which leverages approximate *NFAs*, fundamentally improves the results of [8] and [31]. We can achieve, on a similar chip, a throughput beyond 200 Gbps for much more complicated sets of REs (e.g., SNORT's `spyware` module with 461 REs where the underlying NFA has ∼10k states and the corresponding DFA is prohibitively large—our attempt of its determinisation depleted the available memory (32 GiB) after reaching 616k states).

The works closest to our NFA reduction techniques are [16] and [17]. In [16], the authors address the issue of software-based acceleration of matching REs describing network attacks in SNORT. To reduce the number k of membership tests needed for matching a packet against k distinct DFAs, [16] builds a "search tree" with the k DFAs in its leaves and with the inner nodes occupied by preferably small DFAs that over-approximate the union of their children (the precise DFA accepting the union is prohibitively large). Matching a packet then means to propagate it down the tree as long as it belongs to languages of the DFA nodes. The over-approximating DFAs are constructed using a similar notion of significance of states as in our approach. The differences from our work are the following: (1) [16] needs several membership tests per packet, (2) it targets only software and its hardware implementation would be too complex, (3) it does not consider reduction of NFAs, and (4) it uses only a pruning-based reduction (we also employ merging and simulation-based reductions).

Our previous work in [17] also targets approximate reductions of NFAs used in HW-accelerated IDSes. In contrast to the approach in this paper, which uses a sample of the network traffic to determine the states to remove, [17] uses a probabilistic model of the traffic in the form of a probabilistic automaton, which is used to label states of the NFA by probabilities of

activation. The method has the following drawbacks: (1) the model can be constructed from a sample of traffic only semi-automatically, with an aid of a network expert, and (2) it scales to NFAs of up to only around 1,300 states, whereas our NFAs are sometimes an order of magnitude larger. Moreover, [17] considers only a single-stage architecture.

Simulation-based reduction is a standard language-preserving NFA reduction technique, which, apart from its basic form [18] (used in [19] in the context of FPGA-accelerated IDSes), comes in a number of advanced variants (e.g. look-ahead, multi-pebble, or mediated [20]–[22]), many of them implemented in the tool REDUCE [23]. Although REDUCE is a part of our workflow, the power of simulation reduction alone is by far insufficient for our purposes.

Finally, we compare our approach with RE-matching techniques that use modern general-purpose GPUs. As in FPGAs, we can distinguish architectures leveraging both DFAs and NFAs. Prominent GPU architectures based on DFAs include Gregex [28], hierarchical parallel machines [29], and a recent work employing algorithm/implementation co-optimization based on a GPU performance model [30]. These architectures are able to perform RE matching at the theoretical throughput of 100–150 Gbps. Their practical performance is, however, limited by the packet transfer throughput (e.g., the throughput of [28] was 25 Gbps on NVIDIA GTX260), and, indeed, the complexity of RE modules they can handle due to determinization.

An RE matching architecture for GPUs based on NFAs was proposed in iNFAnt [27] and further improved in [32]. In contrast to the aforementioned DFA-based solutions, this architecture can handle complex RE modules where the underlying NFAs have over 10 thousand states. The performance of this work is, however, significantly inferior to our FPGA-based architecture. For example, for a category of SNORT modules where the underlying NFAs have 3–18 thousand states, [32] reports an overall throughput of 1–2 Gbps on NVIDIA Tesla c2050. Our experiments in Section IV show that, on SNORT modules with similar sizes, our approach is able to achieve a throughput of over 100 Gbps.

Further disadvantages of using GPUs are that (i) they are notorious energy hogs, e.g., the NVIDIA Tesla K20c (used by [30]) card's TDP is 225 W, which is significantly higher than the 75 W consumed by our FPGA-based architecture and (ii) they impose a high latency on packet processing (in the order of hundreds of ms compared to $\sim$10 μs of our solution).

II. ARCHITECTURE

In this section, we first describe the basic architecture of the RE matching engine from [8], which is used as a building block of our multi-stage architecture described later.

A. Regular Expression Matching Engine

The architecture proposed in [8] uses *pipelining* to enable RE matching in high-speed networks on a single FPGA. An example of an instance of this architecture consisting of $k = 3$ pipelined FAs and a single packet buffer is shown in Fig. 1a.

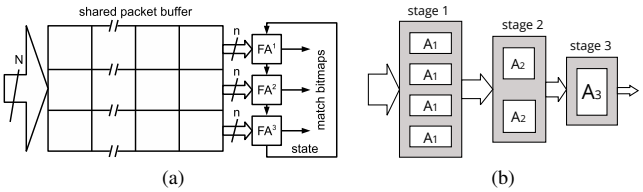


Fig. 1. (a) The architecture with $k = 3$ pipelined FAs sharing a single packet buffer able to store N-bit data words (columns), each of which consists of k independent n-bit blocks (rows). (b) An example of the multi-stage architecture with 3 stages.

The packet buffer stores every N-bits-wide input data word as k independent data blocks of n bits (i.e., $N = k \cdot n$). Each of the k FAs can read data blocks on the corresponding row of the buffer and also receives the configuration of the previous FA in the pipeline that processed the previous block of the packet (or the initial configuration if the FA is processing the first block of a packet). Based on these inputs, the FA either computes the next configuration or determines an RE that matches the packet. These pieces of information are sent to the next FA in the pipeline until the last block of the packet is processed, in which case a bitmap encoding all matching REs is output. As only one of the k pipelined FAs can perform RE matching in a given packet at any moment, the whole pipeline can theoretically perform RE matching in k packets in parallel.

The FA that encodes the complete set of REs is replicated k times. For instance, to achieve the throughput of 100 Gbps, we used the replication factor $k = 16$ in [8]. Such a massive replication costs a lot of FPGA resources, which becomes the major bottleneck of the approach. On the other hand, the replication is also a unique opportunity for optimizations because every reduction of the FA's size means a 16-times higher reduction in the whole architecture. From this point of view, using NFAs is preferable to DFAs because, especially in the combination with language-preserving simulation-based reduction, they are much more succinct than DFAs, and can also be mapped into FPGAs more efficiently than DFAs. They are, however, still prohibitively large in many practical cases; this calls for better reduction methods. In Section III, we present two such methods in the form of approximate reductions, which do not preserve the language of the FA. The reduced FAs are used within our multi-stage architecture (built on top of the pipelined approach of [8]), which allows us to flexibly adjust the tradeoff between precision and consumed resources. The multi-stage architecture is described next.

B. Multi-Stage Regular Expression Matching Architecture

We now propose a concept of a *multi-stage* RE matching unit that uses aggressive approximate reductions, which do not preserve the language of the NFA, to utilise FPGA resources efficiently. The architecture of the RE matching unit is composed of several stages (see Fig. 1b for an example of a 3-stage architecture). Every stage in the architecture contains an instantiation of the RE matching engine described in Section II-A.

The idea is that each of the stages will use different NFAs—starting with a bigger number of smaller and imprecise NFAs and proceeding to smaller numbers of larger but more precise NFAs—to decrease, in a resource-efficient way, the number of packets entering the subsequent stage. Consider an NFA $\mathcal{A}$ that recognizes the language L defined by the REs in a given SNORT module. The first stage of the architecture contains many copies of a small NFA $\mathcal{A}_1$, which over-approximates L, i.e., apart from all packets in L, it also matches some packets not in L. All matched packets are then sent to Stage 2, which contains less copies of a larger NFA $\mathcal{A}_2$, which over-approximates L, but more precisely than $\mathcal{A}_1$ (which is the reason it is larger—less precise approximations of L obtained by our reduction are usually smaller than more precise ones).

The number of copies of $\mathcal{A}_2$ can be smaller due to the fact that the traffic entering it is just a fraction of the input traffic since Stage 1 has removed a significant number of packets from further processing. Each subsequent stage contains an even smaller number of even more precise (and, therefore, larger) NFAs. The final stage contains either copies of $\mathcal{A}$, in which case the output of the RE matching unit is exactly the packets from L, or as precise over-approximation of L as possible given the available resources, in which case the last remaining false positives need to be removed in software.

Our approximate reduction methods (described in detail in Section III) output a Pareto frontier of NFAs $\mathbb{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_k\}$ obtained from the input NFA $\mathcal{A}$. Each NFA $\mathcal{A}_i$ comes with two parameters: (1) its size given as the number of *look-up tables* (LUTs) obtained from its HW synthesis, denoted by the function $LUT : \mathbb{A} \to \mathbb{N}$, and (2) the probability that it accepts an input packet, denoted by the function $Acc : \mathbb{A} \to [0, 1]$. Note that lower is better for both parameters.

Given the set $\mathbb{A}$, we aim at obtaining a configuration of the multi-stage architecture that is as small and precise as possible. This gives rise to the following two optimisation problems:

- OPT_{RSC}: minimise the amount of resources (denoted by the variable $\langle\text{RSC}\rangle$) used by the RE matching unit given a maximum speed of traffic on its output X and
- OPT_{out}: minimise the speed of the traffic on the output of the last n-th stage of the RE matching unit (denoted by the variable $\langle\text{out}_n\rangle$) given a fixed amount of resources Y.

The optimization problems can be formalised (and then solved by a constraint solver) using the following constraints.

The first constraint formalises that, in an n-stage architecture, the output of each stage (denoted using a real-valued variable $\langle\text{out}_i\rangle$) is the fraction of the unit's input traffic $\langle\text{out}_0\rangle$ given by the acceptance probability of the used NFA $\mathcal{A}_j$ (the 0/1 variable $\langle_i\text{uses}_j\rangle$ denotes that Stage i uses the NFA $\mathcal{A}_j$):

$$\forall 1 \leq i \leq n : \langle\text{out}_i\rangle = \langle\text{out}_0\rangle \cdot \sum_{\mathcal{A}_j \in \mathbb{A}} \langle_i\text{uses}_j\rangle \cdot Acc(\mathcal{A}_j). \quad (1)$$

The second family of constraints formalises that every stage uses precisely one version of the NFA from $\mathbb{A}$:

$$\forall 1 \leq i \leq n : 1 = \sum_{\mathcal{A}_j \in \mathbb{A}} \langle_i\text{uses}_j\rangle. \quad (2)$$

TABLE I
(A) PARAMETERS OF THE NFAS. (B) POSSIBLE CONFIGURATIONS OF THE MULTI-STAGE UNIT (OUTPUT OF EACH CONFIGURATION IS 10 GBPS).

	LUT	Acc
$\mathcal{A}_1$	100	0.5
$\mathcal{A}_2$	200	0.2
$\mathcal{A}_3$	1,000	0.1

(a)

#	Stg. 1	Stg. 2	Stg. 3	LUTs
1	16×$\mathcal{A}_3$	—	—	16,000
2	16×$\mathcal{A}_2$	4×$\mathcal{A}_3$	—	7,200
3	16×$\mathcal{A}_1$	8×$\mathcal{A}_3$	—	9,600
4	16×$\mathcal{A}_1$	8×$\mathcal{A}_2$	4×$\mathcal{A}_3$	7,200

(b)

Finally, the third constraint formalises the total resources $\langle\text{RSC}\rangle$ used by the architecture:

$$\langle\text{RSC}\rangle = \sum_{1 \leq i \leq n} \left\lceil \frac{\langle\text{out}_{i-1}\rangle}{TP} \right\rceil \cdot \sum_{\mathcal{A}_j \in \mathbb{A}} \langle_i\text{uses}_j\rangle \cdot LUT(\mathcal{A}_j). \quad (3)$$

We suppose that TP is the *throughput* of the NFAs (obtained as the multiple of the NFA's input bit-width and the clock frequency). The total resources are then computed in such a way that the i-th stage uses $\lceil \langle\text{out}_{i-1}\rangle / TP \rceil$ NFAs $\mathcal{A}_j$ (since it needs to be able to process all of the output traffic from the previous stage), each taking $LUT(\mathcal{A}_j)$ LUTs.

The considered optimisation problems can then be formalised as follows:

$$OPT_{RSC} : \text{Minimise } \langle\text{RSC}\rangle \text{ subject to } \langle\text{out}_n\rangle \leq X,$$
$$OPT_{out} : \text{Minimise } \langle\text{out}_n\rangle \text{ subject to } \langle\text{RSC}\rangle \leq Y.$$

Here, X is the maximal permissible output traffic, and Y is the maximal number of LUTs available in the architecture.

Example: Consider the scenario with 100 Gbps input delivered over a 512-bit interface on 200 MHz. Moreover, assume that the data width of the NFAs is set to 32 bits, their throughput is, therefore, 6.4 Gbps, so 16 of them are needed to process the input 100 Gbps, and, further, assume that there are 10,000 available LUTs. Suppose that we are given a precise NFA $\mathcal{A}$ for which our reduction procedure—based on a sample of network traffic—yields a set of approximated NFAs $\mathbb{A} = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3 = \mathcal{A}\}$ (i.e., $\mathcal{A}_3$ is the precise NFA $\mathcal{A}$) such that their parameters are as given in Table Ia. Ignoring the (negligible) overhead of stage interconnection, the possible configurations of the multi-stage unit are presented in Table Ib. Note that Configuration 1 (which uses a single stage with no approximation) cannot fit within the available resources. □

III. APPROXIMATE NFA REDUCTION

A *nondeterministic finite automaton* (NFA) over a finite alphabet Σ is a quadruple $\mathcal{A} = (Q, \delta, q_I, F)$ where Q is a finite set of *states*, $\delta \subseteq Q \times \Sigma \times Q$ is a *transition relation*, $q_I \in Q$ is the *initial state*, and $F \subseteq Q$ is a set of *final states*. For $q \in Q$ and $a \in \Sigma$, we write $\delta(q, a)$ to denote the set $\{q' \in Q \mid (q, a, q') \in \delta\}$. It is lifted to sets $S \subseteq Q$ as $\delta(S, a) = \bigcup_{q \in S} \delta(q, a)$ and to words as $\hat{\delta}(q, \varepsilon) = \{q\}$ and $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$ for $w \in \Sigma^*$.

Our hardware architecture uses NFAs to accept packets based on their prefixes, and so we use a slightly non-standard notion of the *language* of an NFA. Namely, the language of a state q is the set of words $L_{\mathcal{A}}(q) = \{w_1.w_2 \in \Sigma^* \mid \exists q_f \in$

$F:\ q_f \in \hat{\delta}(q, w_1)\}$, i.e., words whose prefix can take $\mathcal{A}$ from q to some accepting state, followed by an arbitrary suffix. The language of $\mathcal{A}$ is then defined as $L(\mathcal{A}) = L_{\mathcal{A}}(q_I)$.

A. Pruning Reduction

Our first NFA reduction is the so-called *pruning reduction*. The reduction removes from the automaton a set R of states considered as *insignificant*, together with all their adjacent transitions. At the same time, in order to overapproximate the original language, all states that are not removed and that have a transition going to a removed state are made final. Below, we call such states *border states*, forming a set B.

More precisely, let $\mathcal{A} = (Q, \delta, q_I, F)$ be an NFA over Σ, and let $R \subseteq Q$, $q_I \notin R$, be a set of states to be removed (we will later discuss how to find such a set). Let $B = \{q \in Q \setminus R \mid \exists a \in \Sigma:\ \delta(q, a) \cap R \neq \emptyset\}$ be the set of border states corresponding to R. The operation of pruning from $\mathcal{A}$ the states from R produces the NFA $\mathcal{A}_R = (Q' = Q \setminus R, \delta', q_I, F')$ where $\delta' = \delta \cap (Q' \times \Sigma \times Q')$ and $F' = (F \cap Q') \cup B$.

The pruning reduction over-approximates the original language, i.e., $L(\mathcal{A}) \subseteq L(\mathcal{A}_R)$. The obtained NFA can, of course, be potentially further reduced by exact, simulation-based reductions [23].

The trade-off between reduction and accuracy that the pruning reduction offers depends on the choice of the set R of the states to be removed. We therefore try to compose R from such states of $\mathcal{A}$ that have the least influence on the acceptance/rejection of packets in typical traffic. For that, we use a representative sample $\mathcal{S}$ of the network traffic in the form of a multiset of packets. We label each state $q \in Q$ of the input NFA $\mathcal{A}$ by its *significance*: the number $\ell(q)$ of packets from $\mathcal{S}$ over which the state q can be reached in $\mathcal{A}$ (if there are multiple ways of reaching q over the same packet, we do not distinguish them). Formally, $\ell(q) = \sum_{w \in \{w_1.w_2 \in \mathcal{S} \mid q \in \hat{\delta}(q_I, w_1)\}} \mathcal{S}(w)$ where $\mathcal{S}(w)$ is the number of occurrences of the packet w in the multiset $\mathcal{S}$.

The error caused by the pruning reduction based on a set of states R wrt a sample $\mathcal{S}$ can be bounded in terms of significance of the border states B corresponding to R. Indeed, only the packets accepted at some border state can get wrongly accepted. Formally, $error_{ac}(\mathcal{S}, \mathcal{A}, \mathcal{A}_R) \leq \sum_{q \in B} \ell(q)$ where $error_{ac}(\mathcal{S}, \mathcal{A}, \mathcal{A}_R) = \sum_{w \in L(\mathcal{A}_R) \setminus L(\mathcal{A})} \mathcal{S}(w)$ is the exact error caused by the reduction on the sample $\mathcal{S}$.

To specify the desired reduction, we use a target *reduction ratio* $\theta \in (0, 1]$ meaning that the automaton should be reduced to $m = \lceil \theta \cdot |Q| \rceil$ states. To obtain m states while minimising the error, we fill R with $|Q| - m$ least significant states[1].

The significance of all states is computed efficiently in time $\mathcal{O}(kn^2)$ where $n = |Q|$ and $k = \sum_{w \in \mathcal{S}} |w| \cdot \mathcal{S}(w)$ is the overall length of $\mathcal{S}$. For that, we can use the subset construction known from determinisation of NFAs, just run on particular packets $w \in \mathcal{S}$. Namely, each state's significance is initially set to zero. Then, we run $\mathcal{A}$ over every $w = a_1 \dots a_l \in \mathcal{S}$, computing consecutive sets of states Q^i that are possibly reached after processing the prefix $a_1 \dots a_i$, and, at the end of the run, we increment by one the significance of each of the states encountered on the way. Formally, for each $w \in \mathcal{S}$, we start with $Q^0 = \{q_I\}$, and, subsequently, compute $Q^{i+1} = \delta(Q^i_1, a_{i+1})$ for all $1 < i < l$. The significance of all states in the set $\bigcup_{i=0}^{l} Q^i$ is then incremented by one.

B. Merging Reduction

Our second reduction, called a *merging reduction*, is motivated by an observation that, in typical traffic, packets that start with a prefix of a certain kind (i.e., they are from some language L) almost always continue by an infix w that follows a predetermined pattern (a concrete word or a sequence of characters from predetermined character classes). We say that, in a sample $\mathcal{S}$, the pattern of w is *predetermined* by L. The part of the automaton that, after reading the prefix from L, tests whether the infix fits the pattern can be significantly simplified by collapsing it into a single state with a self-loop over all the symbols that label the original transitions while causing a small error only: Indeed, it is unlikely that a packet with an infix other than the predetermined one will appear after the given prefix. Note that the pruning reduction discussed previously is not suitable for simplifying the states that test the pattern as they may be of an arbitrarily high significance.

The operation of merging a sub-automaton based on a set S of states means to (1) redirect the targets of transitions entering S to a new state s, (2) reconnect all transitions leaving S to start from s instead, (3) make s final iff any of the states in S is final, and (4) remove the states of S. Note that, like pruning, merging also over-approximates the language by allowing any permutation of the infix pattern.

Our detection of the parts of automata to be merged—typically, sequences of states—is based on a notion of *distance* defined for a pair of states q and r as $dist(q, r) = \max\left(\frac{\ell(r)}{\ell(q)}, \frac{\ell(q)}{\ell(r)}\right)$ if they are neighbours (i.e., $r \in \delta(q, a)$ or $q \in \delta(r, a)$ for some a) and as $dist(q, r) = \infty$ otherwise. Intuitively, a small $dist(q, r)$ means that $\ell(q)$ and $\ell(r)$ are similar, which typically happens if most of the packets reaching q continue to r or vice versa.[2] Symbols on transitions from q to r hence form a predetermined pattern of length 1. Therefore, merging q and r, and thus over-approximating the pattern, should cause a small error only. Merging of longer patterns is then achieved by merging multiple patterns of length 1.

The merging reduction is parameterised by a *distance ceiling* D—we merge states with distance below D. Formally, the sets of states to be merged are defined as the equivalence classes of the smallest equivalence $\sim_D \subseteq Q \times Q$ that contains all pairs (q, r) with $dist(q, r) \leq D$ (in other words, $\sim_D$ is the reflexive transitive closure of $\{(q, r) \mid dist(q, r) \leq D\}$).

[1] This strategy can cause a larger error when an originally non-accepting border state of a high significance is forced to become accepting by some insignificant accepting successor state. This could be avoided, e.g., by preferring pruning final states without a significant successor border state. In our experiments, we, however, sufficed with the simple strategy.

[2] In theory, it does not have to be the case, as $dist(q, r)$ may be polluted by packets reaching and leaving q and r from and to other states, but it is mostly the case in practice.

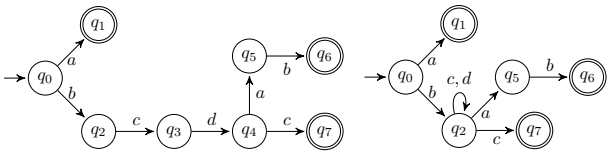


Fig. 2. An input NFA (left) and an NFA obtained by merging (right).

TABLE II
SIZES OF THE CONSIDERED NFAS.

NFA	States	Transitions
`backdoor`	3,898	100,301
`l7-all`	7,280	2,647,620
`pop3`	923	209,467
`sprobe`	168	5,108
`spyware`	12,809	279,334

The merging reduction does not provide theoretical guarantees, and it is to a large degree based on empirical experience, in which its parametrisation with D allows one to control the ratio between reduction and error well. There are, however, cases in which merging leads to an undesirable loss of precision even with a small D. To limit such effects, we restrict merging by an additional parameter, the *frequency ceiling* $F \in (0, 1]$. We prohibit merging of states with $\mathit{freq}(q) = \frac{\ell(q)}{|\mathcal{S}|} > F$, that is, those whose frequency in $\mathcal{S}$ is larger than the ceiling. Formally, given D and F, we merge the equivalence classes of $\sim_D \cap \{(q, r) \mid \mathit{freq}(q) \leq F \wedge \mathit{freq}(r) \leq F\}$.

An example of the merging reduction is shown in Fig. 2, assuming $\mathit{dist}(q_2, q_3) < D$, $\mathit{dist}(q_3, q_4) < D$, and that all other distances are greater than D. To make the states q_2, q_3, and q_4 less critical, we further assume that $\mathit{freq}(q_2) < F$, $\mathit{freq}(q_3) < F$, $\mathit{freq}(q_4) < F$. Intuitively, this means that only some packets continue over b from q_0 to q_2. Roughly all of those packets then continue until they reach q_4, where another important split of the acceptance happens. The prefix b, however, predetermines the subsequent occurrence of cd. By simplifying the automaton and allowing any string from $\{c, d\}^*$ to appear after b, no significant error arises since most packets will anyway contain cd after b only.

IV. EVALUATION

In this section, we present an experimental evaluation of the proposed approach on real RE-matching instances.

A. Considered REs

We experimented with a set of REs describing protocols and attacks obtained from the L7 classifier for the Linux Netfilter [24] framework and from the SNORT tool [1]. From the L7 classifier describing L7 protocols, we used all rules, giving us a set of REs denoted as `l7-all` below. From the SNORT tool, we used the following set of REs: `backdoor`, `pop3`, and `spyware-put` (abbreviated as `spyware` below), describing attacks on selected protocols. We also used nine rules, denoted as `sprobe`, proposed for lawful interception in cooperation with our national police. We used the NETBENCH tool [25] for (i) translating REs to NFAs and (ii) the synthesis of reduced NFAs to VHDL. The sizes of the NFAs obtained by translating the considered REs are shown in Table II.

B. Evaluation Data

The sample of network traffic that we used for our experiments was obtained from two measuring points of a nationwide Internet provider connected to a 100 Gbps backbone link. The training data used for labelling the automata contained ∼1M packets sampled from the captured traffic during the time of 19.5 min containing 509M packets. The testing data used for the subsequent evaluation consisted of ∼21M packets sampled from the captured traffic containing ∼210M packets. The testing data was sampled over the time of 105 hours (over 4 days) such that every hour 1M packets were captured.

C. Evaluation Environment

We implemented the proposed techniques in a Python prototype [33]. In the experiments, we ran merging with the frequency ceiling $F = 0.1$ and the distance ceiling $D = 1.005$. FPGA synthesis was done using Xilinx Vivado v.2018.1.

D. Running Time

Our reduction techniques are light-weight, and, in contrast to existing approaches [17], they are capable of reducing large NFAs appearing in real-world RE matching problems for real network scenarios. (Recall that the largest automata we consider have over 12k states or over 2.5M transitions.) Using the training data containing 1M packets, we needed about 15 min to derive the state labelling function ℓ for the largest considered NFAs. The runtime of the other parts of the reduction process was then negligible. Also note that, for a given NFA, the labelling can be performed only once for various values of the reduction ratio θ.

E. Research Questions

We are interested in the following two key research questions related to the proposed approach:

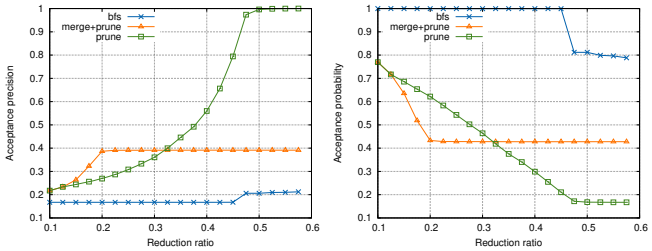
R1: Are our reduction techniques able to provide useful trade-offs between the reduction error and the reduction ratio?

R2: Can the reduced NFAs be compiled into a multi-stage architecture with throughput of 100 Gbps and beyond?

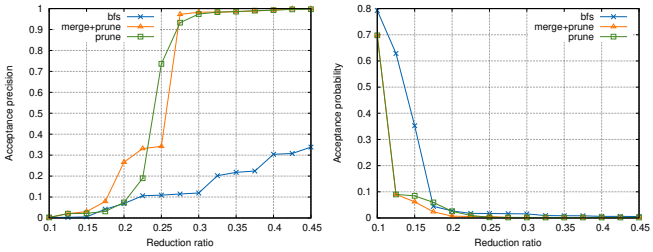
R1: *Reduction Trade-offs*

In our experiments with the reduction techniques, we consider both the pruning and merging reductions. The merging reduction is, however, always combined with a subsequent pruning reduction as a standalone use of merging turned out not to be effective. Moreover, as a baseline, we also consider a so-called *bfs-reduction*. It does not use any training traffic and works by simply pruning away states that are far from the initial state. All of the approximate reductions are followed by the exact simulation-based reduction [20] whenever the tool REDUCE [23] implementing this reduction is capable of handling the approximate NFA (it fails on large NFAs).

We consider two metrics characterizing the reduced automata. The first metric is the *acceptance precision* $AP =$



(a) AP for `l7-all`. (b) $Prob$ for `l7-all`.



(c) AP for `backdoor`. (d) $Prob$ for `backdoor`.

Fig. 3. Results for `l7-all` and `backdoor`.

$\frac{A_{TP}}{A_{FP}+A_{TP}}$ where A_{TP} denotes acceptance true positives (the packet is accepted and should have been accepted) and A_{FP} denotes acceptance false positives (the packet is accepted and should *not* have been accepted). This metric expresses the ratio of correctly accepted packets to all accepted packets from the testing traffic sample and hence characterizes the error caused by the approximation. Our second metric is the *acceptance probability* $Prob = \frac{A_{TP}+A_{FP}}{|\mathcal{S}|}$ (where $|\mathcal{S}|$ denotes the size of the input network traffic sample) that captures what fraction of the input traffic is accepted by the reduced NFA and passed to the next stage, i.e., how much the NFA reduces the flow of packets to be further processed. $Prob$ is an important metric for building efficient multi-stage architectures.

Figs. 3 and 4 show the trade-offs achieved by our different reduction strategies (namely, *bfs*, *prune*, and *merge-prune*) on challenging RE matching problems.

Figs. 3a and 3b show results for the NFA describing `l7-all`. We observe that the particular reduction techniques provide a different quality of the trade-offs. In particular, *bfs* is not capable of producing any useful approximation. Further, we can observe that *merge-prune* dominates for reduction ratios lower than 0.3, but it is significantly outperformed by *prune* for higher ratios.

The figures show that these trends are preserved for both metrics. Note that the original NFA accepts around 17 % of the traffic, and using the *prune* technique, we obtain a reduced NFA having only a half of the states with almost the same acceptance probability $Prob$.

The reduction trade-offs that we obtain for the NFA of the `backdoor` attack are plotted in Figs. 3c and 3d. The *bfs* reduction is again significantly outperformed by both the *prune* and *merge-prune* methods when AP is considered. Note that these two techniques provide reductions that achieve almost a 100 % AP using only 35 % of the states of the original NFA.

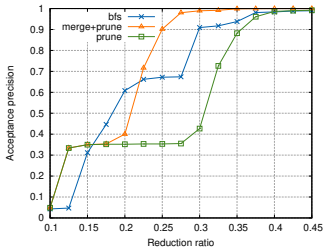


Fig. 4. AP for `spyware`.

As the NFA accepts only 0.2 % of the traffic, we can obtain the accepting probability $Prob$ that is close to this value using only 22 % of its states regardless of the reduction used. We observed similar trends also for the `pop3` and `sprobe` attacks (not presented here due to lack of space) where almost a 0 % $Prob$ was achieved using only 20 % and 25 % states, respectively.

Finally, we report the results for `spyware` REs in Fig. 4. We can see that, wrt AP, *prune* lags behind the other two techniques for reduction ratios between 0.15 and 0.35. For higher reductions, all techniques provide AP close to 100 %. Similar trends can be observed also for the $Prob$ metric (not presented here). Note that the original NFA accepts about 3.5 % of the input traffic only.

The experiments conducted within **R1** clearly demonstrate that the proposed reduction techniques are able to provide high-quality trade-offs between the precision and the reduction factors. They also show that our techniques can considerably outperform the baseline *bfs-reduction* and can handle very complex NFAs (having more than 10k states), where existing methods (such as those in [17] and [23]) fail.

R2: *Compiling the Multi-Stage Architecture*

We will use the reduced NFAs from **R1** to obtain instantiations of the multi-stage architecture that can be used in FPGA-based IDSes to effectively decrease the amount of traffic that the software part of the IDS needs to process. We synthesise our designs for a card with the Xilinx Virtex UltraScale+ VU9P FPGA chip, which contains 1,182k LUTs (other resources are in our case always dominated by the number of LUTs). From our experience, it is possible to use up to 70 % of the LUTs available on the FPGA and successfully route designs at the considered frequency (200 MHz), which leaves us with 827k LUTs that we can use. Moreover, the components that we use for receiving packets and transfering them to the CPU consume around 90k LUTs, we are therefore left with 737k LUTs for RE matching.

In every stage of the RE matching unit, we use the pipelined NFAs architecture described in Section II-A instantiated with 8-bit data-width of the NFAs, which gave us the best results. Therefore, the throughput (parameter TP in Section II-B) of every NFA at 200 MHz is 1.6 Gbps. The optimisation problem of compiling the multi-stage architecture leads to a mixed integer quadratically constrained quadratic program, which we solve using the Gurobi solver [26] (this step took at most 2 s).

TABLE III
OPTIMAL MULTISTAGE ARCHITECTURE FOR THE BACKDOOR MODULE.

Precise				
speed	1 stg	2 stg	3 stg	4 stg
100	236k	56k	50k	50k
200	473k	113k	99k	96k
400	946k	223k	194k	186k

TABLE IV
OPTIMAL MULTISTAGE ARCHITECTURE FOR THE SPYWARE MODULE.

Precise					4 % of traffic			
speed	1 stg	2 stg	3 stg	4 stg	1 stg	2 stg	3 stg	4 stg
100	5M	444k	296k	296k	227k	61k	65k	69k
200	10M	809k	513k	513k	453k	122k	126k	133k
400	20M	1.5M	945k	945k	907k	242k	247k	261k

Our goal is to obtain single-box IDSes using a combination of hardware preprocessing and a software IDS. This means that the task of the hardware accelerator is to decrease the amount of the traffic entering the software part as much as possible while keeping all suspicious packets. Ideally, the final stage of the RE matching unit would be the precise NFA, so the hardware accelerator would output precisely the packets that match the given set of REs. Due to the size of the precise NFAs, this is, however, often infeasible, so we also consider the setting that decreases the traffic as much as possible.

Below, we provide results of our experiments for some of the considered sets of REs. We tried to compile architectures for the speeds 100, 200, and 400 Gbps using 1–4 stages.

1) backdoor: In Table III, we present results of optimal architectures for the SNORT's `backdoor` module. We present only results for the precise setting, as our multi-stage architectures can handle 400 Gbps traffic using the available resources (737k LUTs). Note that the single stage architecture (i.e., the "1 stg" column), can process traffic up to 200 Gbps only (the precise NFA consumes 3,695 LUTs). In order to process 400 Gbps, it is necessary to use the multi-stage architecture, in which case four stages give the best results.

2) spyware: Our results for the SNORT's `spyware` module are shown in Table IV. This module is much more complex than `backdoor`, since its precise NFA takes ∼78k LUTs. Therefore, for the throughputs of 100 Gbps and 200 Gbps, the multi-stage architec- ture is needed. For 400 Gbps, we were not able to obtain a precise configuration; we were, however, able to obtain multi-stage configurations (the best one with 2 stages) that decrease the amount of traffic sent to the CPU below 4 % (i.e., 16 Gbps). Although 16 Gbps is on the edge of capabilities of current SW-based IDSes, we stress that the final-stage reduced NFA has AP very close to 100 % (Fig. 4) and thus only a small fraction of packets are misclassified.

3) l7-all: Our most challenging example is from the `l7-all` RE set. Although the size of the precise automaton is not as large as for `spyware` (the precise NFA consumes 27,650 LUTs), it is less amenable for approximate reduction because, in contrast to SNORT modules, it contains REs that are matched by many packets. The results for the `l7-all` RE set are shown in Table V. Our best solution reduces the input traffic from 100 Gbps to 17 Gbps and uses 597 kLUTs in two stages. As in the previous case, the final-stage reduced NFA has almost 100 % precision and thus only a small fraction of packets are misclassified.

TABLE V
OPTIMAL MULTISTAGE ARCHITECTURE FOR THE L7-ALL MODULE.

Precise					17 % of traffic			
speed	1 stg	2 stg	3 stg	4 stg	1 stg	2 stg	3 stg	4 stg
100	1.8M	894k	880k	880k	1.1M	597k	648k	701k

4) sprobe and pop3: The sets of REs for `sprobe` and `pop3` are, on the other hand, quite less challenging. The precise NFAs consume only 195 and 1,721 LUTs, respectively, so we can easily obtain a precise design at 400 Gbps with only a single stage using ∼50k and ∼440k LUTs, respectively.

The experiments conducted within **R2** clearly demonstrate the practical potential of our approach. The key observation is that the resource reductions provided by the particular multi-stage architectures directly depend on the characteristics of the underlying NFAs (both the precise NFA and the reduced variants) and the typical traffic. Apart from the size of the precise NFA, there are two crucial characteristics: (1) whether the number of packets accepted by the precise NFA is low and (2) whether the reduction can compress the NFA while not increasing the number of accepted packets too much. If both these conditions are met (as for `backdoor` and `spyware`), we observe drastic resource savings allowing us to achieve throughput of the resulting IDSes going beyond 100 Gbps, which is unprecedented for REs of such size and complexity. On the other hand, if the original NFA is large, accepts many packets, and highly precise reductions achieve only moderate reductions (as for `l7-all`), the resulting multi-stage architecture provides only moderate savings and ensuring 100 Gbps remains at the edge of what we can achieve.

V. CONCLUSION

We have leveraged techniques for approximate reduction of NFAs to allow RE matching of a set of SNORT/Netfilter modules on speeds significantly beyond the capabilities of the state-of-the-art single-box solutions, namely 100 and even 400 Gbps. The use of the approximate reduction allowed us to significantly decrease the size of the NFAs while keeping the number of false positives low (e.g. for SNORT's `spyware` module, we obtained a reduction to 28 % of the original size while keeping the error below 2 %). Moreover, the use of the multi-stage architecture better utilises FPGA resources (e.g., using 3 stages, we were able to reduce the resources consumed by `spyware` down to ∼5 % of the original without introducing any error). As far as we know, our technique is the first solution that can be used to obtain single-box IDSes detecting large sets of complex REs at speeds over 100 Gbps.

Acknowledgement: We thank Vlastimil Košař for his comments on an earlier draft of the paper and Martin Žádník for providing us with the backbone network traffic. This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science — LQ1602.

REFERENCES

[1] M. Roesch et al., SNORT http://www.snort.org

[2] Matt Jonkman et al., "SURICATA," Emerging Threats, 2017. [Online]. Available: {http://suricata-ids.org}

[3] Vern Paxson et al., "The BRO Network Security Monitor," 2018. [Online]. Available: {http://www.bro.org}

[4] "IEEE Standard for Ethernet - Amendment 10: Media Access Control Parameters, Physical Layers, and Management Parameters for 200 Gb/s and 400 Gb/s Operation," *IEEE Std 802.3bs-2017*, pp. 1–372, 2017.

[5] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney, "The NIDS Cluster: Scalable, Stateful Network Intrusion Detection on Commodity Hardware," in *RAID'07*. Springer, 2007, pp. 107–126.

[6] A. Caulfield *et al.*, "A cloud-scale acceleration architecture," in *MICRO'16*, 2016.

[7] A. Putnam *et al.*, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," in *ISCA'14*. IEEE Press, 2014, pp. 13–24.

[8] D. Matousek, J. Korenek, and V. Pus, "High-speed Regular Expression Matching with Pipelined Automata," in *FPT'16*. IEEE, 2016.

[9] R. Sidhu and V. K. Prasanna, "Fast Regular Expression Matching Using FPGAs," in *FCCM'01*. IEEE Computer Society, 2001, pp. 227–238.

[10] C. R. Clark and D. E. Schimmel, "Efficient Reconfigurable Logic Circuits for Matching Complex Network Intrusion Detection Patterns," in *FPL'03*. Springer, 2003, pp. 956–959.

[11] I. Sourdis, J. Bispo, J. M. P. Cardoso, and S. Vassiliadis, "Regular Expression Matching in Reconfigurable Hardware," *Journal of Signal Processing Systems*, vol. 51, no. 1, pp. 99–121, 2008.

[12] C. Lin, C. Huang, C. Jiang, and S. Chang, "Optimization of Pattern Matching Circuits for Regular Expression on FPGA," *IEEE Trans. VLSI Syst.*, vol. 15, no. 12, pp. 1303–1310, 2007.

[13] S. Yun, K. Lee, "Optimization of Regular Expression Pattern Matching Circuit Using At-Most Two-Hot Encoding on FPGA," *FPL'10*, 2010.

[14] Y. H. Yang and V. Prasanna, "High-Performance and Compact Architecture for Regular Expression Matching on FPGA," *IEEE Transactions on Computers*, vol. 61, no. 7, pp. 1013–1025, 2012.

[15] M. Becchi and P. Crowley, "Efficient Regular Expression Evaluation: Theory to Practice," in *ANCS'08*. ACM, 2008, pp. 50–59.

[16] D. Luchaup, L. D. Carli, S. Jha, and E. Bach, "Deep Packet Inspection with DFA-trees and Parametrized Language Overapproximation," in *INFOCOM'14*. IEEE, 2014, pp. 531–539.

[17] M. Češka, V. Havlena, L. Holík, O. Lengál, and T. Vojnar, "Approximate Reduction of Finite Automata for High-Speed Network Intrusion Detection," in *TACAS'18*. Springer, 2018, pp. 155–175.

[18] D. Bustan and O. Grumberg, "Simulation Based Minimization," in *CADE'17*. Springer, 2000, pp. 255–270.

[19] V. Košař, M. Žádník, and J. Kořenek, "NFA Reduction for Regular Expressions Matching Using FPGA," in *FPT'13*, 2013, pp. 338–341.

[20] L. Clemente and R. Mayr, "Advanced Automata Minimization," in *POPL'13*. ACM Trans. Comput. Log., 2013, pp. 63–74.

[21] K. Etessami, "A Hierarchy of Polynomial-Time Computable Simulations for Automata," in *CONCUR'02*. Springer, 2002, pp. 131–144.

[22] P. A. Abdulla, Y. Chen, L. Holík, and T. Vojnar, "Mediating for Reduction (On Minimizing Alternating Büchi Automata)," *Theoretical Computer Science*, vol. 552, pp. 26 – 43, 2014.

[23] R. Mayr *et al.*, "REDUCE: A Tool for Minimizing Nondeterministic Finite-Word and Büchi Automata," http://languageinclusion.org.

[24] R. Russel et al., "Netfilter," 2018. [Online]. Available: http://netfilter.org

[25] V. Pus, J. Tobola, V. Kosar, et al., "Netbench: Framework for Evaluation of Packet Processing Algorithms," in *ANCS'11*. ACM/IEEE, 2011.

[26] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2018. [Online]. Available: http://www.gurobi.com

[27] N. Cascarano, P. Rolando, F. Risso, and R. Sisto. "iNFAnt: NFA pattern matching on GPGPU devices," *SIGCOMM Comput. Commun. Rev.* vol. 40, issue 5, pp. 20-26, 2010.

[28] L. Wang, S. Chen, Y. Tang, and J. Su, "Gregex: GPU Based High Speed Regular Expression Matching Engine," in *IMIS'11*, pp. 366-370, IEEE, 2011,

[29] C. Lin, C. Liu and S. Chang, "Accelerating Regular Expression Matching Using Hierarchical Parallel Machines on GPU," in *GLOBECOM'11*, pp. 1-5, IEEE, 2011.

[30] C.L. Hsieh, L. Vespa, and N. Weng, "A high-throughput DPI engine on GPU via algorithm/implementation co-optimization", Journal of Parallel and Distributed Computing, 88, pp. 46-56, 2016.

[31] J. Yang, L. Jiang, X. Bai, H. Peng and Q. Dai, "A High-Performance Round-Robin Regular Expression Matching Architecture Based on FPGA", in *ISCC'18*, pp. 1-7, IEEE, 2018.

[32] M. Avalle, F. Risso, and R. Sisto, "Scalable Algorithms for NFA Multi-Striding and NFA-Based Deep Packet Inspection on GPUs", IEEE/ACM Transactions on Networking, vol. 24, no. 3, pp. 1704-1717, 2016.

[33] J. Semrič et al., "AHOFA", https://github.com/jsemric/ahofa, 2018.