Approximate Reduction of Finite Automata for High-Speed Network Intrusion Detection

Milan Češka, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Tomáš Vojnar

FIT, Brno University of Technology, IT4Innovations Centre of Excellence, Czech Republic

Abstract. We consider the problem of *approximate reduction of non-deterministic automata* that appear in hardware-accelerated network intrusion detection systems (NIDSes). We define an error *distance* of a reduced automaton from the original one as the probability of packets being incorrectly classified by the reduced automaton (wrt the probabilistic distribution of packets in the network traffic). We use this notion to design an *approximate reduction procedure* that achieves a great size reduction (much beyond the state-of-the-art language preserving techniques) with a controlled and small error. We have implemented our approach and evaluated it on use cases from SNORT, a popular NIDS. Our results provide experimental evidence that the method can be highly efficient in practice, allowing NIDSes to follow the rapid growth in the speed of networks.

1 Introduction

The recent years have seen a boom in the number of security incidents in computer networks. In order to alleviate the impact of network attacks and intrusions, Internet providers want to detect malicious traffic at their network's entry points and on the backbones between sub-networks. Software-based network intrusion detection systems (NIDSes), such as the popular open-source system SNORT [1], are capable of detecting suspicious network traffic by testing (among others) whether a packet payload matches a regular expression (regex) describing known patterns of malicious traffic. NIDSes collect and maintain vast databases of such regexes that are typically divided into groups according to types of the attacks and target protocols.

Regex matching is the most computationally demanding task of a NIDS as its cost grows with the speed of the network traffic as well as with the number and complexity of the regexes being matched. The current software-based NIDSes cannot perform the regex matching on networks beyond 1 Gbps [2, 3], so they cannot handle the current speed of backbone networks ranging between tens and hundreds of Gbps. A promising approach to speed up NIDSes is to (partially) offload regex matching into hardware [3–5]. The hardware then serves as a pre-filter of the network traffic, discarding the majority of the packets from further processing. Such pre-filtering can easily reduce the traffic the NIDS needs to handle by two or three orders of magnitude [3].

Field-programmable gate arrays (FPGAs) are the leading technology in high-throughput regex matching. Due to their inherent parallelism, FPGAs provide an efficient way of implementing *nondeterministic finite automata* (NFAs), which naturally arise from the input regexes. Although the amount of available resources in FPGAs is continually increasing, the speed of networks grows even faster. Working with multi-gigabit networks requires the hardware to use many parallel packet processing branches in a single FPGA [5]; each of them implementing a separate copy of the concerned NFA, and so reducing the size of the NFAs is of the utmost importance. Various language-preserving automata reduction approaches exist, mainly based on computing (bi)simulation relations on automata states (cf. the related work). The reductions they offer, however, do not satisfy the needs of high-speed hardware-accelerated NIDSes.

Our answer to the problem is *approximate reduction* of NFAs, allowing for a trade-off between the achieved reduction and the precision of the regex matching. To formalise the intuitive notion of precision, we propose a novel *probabilistic distance* of automata. It captures the probability that a packet of the input network traffic is incorrectly accepted or rejected by the approximated NFA. The distance assumes a *probabilistic model* of the network traffic (we show later how such a model can be obtained).

Having formalised the notion of precision, we specify the target of our reductions as two variants of an optimization problem: (1) minimizing the NFA size given the maximum allowed error (distance from the original), or (2) minimizing the error given the maximum allowed NFA size. Finding such optimal approximations is, however, computationally hard (**PSPACE**-complete, the same as precise NFA minimization).

Consequently, we sacrifice the optimality and, motivated by the typical structure of NFAs that emerge from a set of regexes used by NIDSes (a union of many long "tentacles" with occasional small strongly-connected components), we limit the space of possible reductions by restricting the set of operations they can apply to the original automaton. Namely, we consider two reduction operations: (i) collapsing the future of a state into a *self-loop* (this reduction over-approximates the language), or (ii) *removing states* (such a reduction is under-approximating).

The problem of identifying the optimal sets of states on which these operations should be applied is still **PSPACE**-complete. The restricted problem is, however, more amenable to an approximation by a *greedy algorithm*. The algorithm applies the reductions state-by-state in an order determined by a precomputed *error labelling* of the states. The process is stoppped once the given optimization goal in terms of the size or error is reached. The labelling is based on the probability of packets that may be accepted through a given state and hence over-approximates the error that may be caused by applying the reduction at a given state. As our experiments show, this approach can give us high-quality reductions while ensuring formal error bounds.

Finally, it turns out that even the pre-computation of the error labelling of the states is costly (again **PSPACE**-complete). Therefore, we propose several ways to cheaply over-approximate it such that the strong error bound guarantees are still preserved. Particularly, we are able to exploit the typical structure of the "union of tentacles" of the hardware NFA in an algorithm that is exponential in the size of the largest "tentacle" only, which is indeed much faster in practice.

We have implemented our approach and evaluated it on regexes used to classify malicious traffic in SNORT. We obtain quite encouraging experimental results demonstrating that our approach provides a much better reduction than language-preserving techniques with an almost negligible error. In particular, our experiments, going down to the level of an actual implementation of NFAs in FPGAs, confirm that we can squeeze into an up-to-date FPGA chip real-life regexes encoding malicious traffic, allowing them to be used with a negligible error for filtering at speeds of 100 Gbps (and even 400 Gbps). This is far beyond what one can achieve with current exact reduction approaches.

Related Work Hardware acceleration for regex matching at the line rate is an intensively studied technology that uses general-purpose hardware [6–14] as well as FPGAs [15–20, 3–5]. Most of the works focus on DFA implementation and optimization techniques.

NFAs can be exponentially smaller than DFAs but need, in the worst case, $\mathcal{O}(n)$ memory accesses to process each byte of the payload where n is the number of states. In most cases, this incurs an unacceptable slowdown. Several works alleviate this disadvantage of NFAs by exploiting reconfigurability and fine-grained parallelism of FPGAs, allowing one to process one character per clock cycle (e.g. [15, 16, 19, 20, 3–5]).

In [14], which is probably the closest work to ours, the authors consider a set of regexes describing network attacks. They replace a potentially prohibitively large DFA by a tree of smaller DFAs, an alternative to using NFAs that minimizes the latency occurring in a non-FPGA-based implementation. The language of every DFA-node in the tree over-approximates the languages of its children. Packets are filtered through the tree from the root downwards until they belong to the language of the encountered nodes, and may be finally accepted at the leaves, or are rejected otherwise. The over-approximating DFAs are constructed using a similar notion of probability of an occurrence of a state as in our approach. The main differences from our work are that (1) the approach targets approximation of DFAs (not NFAs), (2) the over-approximation is based on a given traffic sample only (it cannot benefit from a probabilistic model), and (3) no probabilistic guarantees on the approximation error are provided.

Approximation of DFAs was considered in various other contexts. Hyper-minimization is an approach that is allowed to alter language membership of a finite set of words [21, 22]. A DFA with a given maximum number of states is constructed in [23], minimizing the error defined either by (i) counting prefixes of misjudged words up to some length, or (ii) the sum of the probabilities of the misjudged words wrt the Poisson distribution over Σ^*. Neither of these approaches considers reduction of NFAs nor allows to control the expected error with respect to the real traffic.

In addition to the metrics mentioned above when discussing the works [23, 21, 22], the following metrics should also be mentioned. The Cesaro-Jaccard distance studied in [24] is, in spirit, similar to [23] and does also not reflect the probability of individual words. The edit distance of weighted automata from [25] depends on the minimum edit distance between pairs of words from the two compared languages, again regardless of their statistical significance. None of these notions is suitable for our needs.

Language-preserving minimization of NFAs is a **PSPACE**-complete problem [38, 26]. More feasible (polynomial-time) is language-preserving size reduction of NFAs based on (bi)simulations [27–30], which does not aim for a truly minimal NFA. A number of advanced variants exist, based on multi-pebble or look-ahead simulations, or on combinations of forward and backward simulations [31–33]. The practical efficiency of these techniques is, however, often insufficient to allow them to handle the large NFAs that occur in practice and/or they do not manage to reduce the NFAs enough. Finally, even a minimal NFA for the given set of regexes is often too big to be implemented in the given FPGA operating on the required speed (as shown even in our experiments). Our approach is capable of a much better reduction for the price of a small change of the accepted language.

2 Preliminaries

We use $\langle a, b\rangle$ to denote the set $\{x \in \mathbb{R} \mid a \leq x \leq b\}$ and $\mathbb{N}$ to denote the set $\{0, 1, 2, \dots\}$. Given a pair of sets X_1 and X_2, we use $X_1 \triangle X_2$ to denote their *symmetric difference*, i.e., the set $\{x \mid \exists! i \in \{1, 2\} : x \in X_i\}$. We use the notation $[v_1, \dots, v_n]$

to denote a vector of n elements, $\mathbf{1}$ to denote the all 1's vector $[1, \ldots, 1]$, $\boldsymbol{A}$ to denote a matrix, and $\boldsymbol{A}^\top$ for its transpose, and $\boldsymbol{I}$ for the identity matrix.

In the following, we fix a finite non-empty alphabet Σ. A *nondeterministic finite automaton* (NFA) is a quadruple $A = (Q, \delta, I, F)$ where Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $I \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of accepting states. We use $Q[A], \delta[A], I[A]$, and $F[A]$ to denote Q, δ, I, and F, respectively, and $q \xrightarrow{a} q'$ to denote that $q' \in \delta(q, a)$. A sequence of states $\rho = q_0 \cdots q_n$ is a *run* of A over a word $w = a_1 \cdots a_n \in \Sigma^*$ from a state q to a state q', denoted as $q \overset{w,\rho}{\rightsquigarrow} q'$, if $\forall 1 \leq i \leq n : q_{i-1} \xrightarrow{a_i} q_i$, $q_0 = q$, and $q_n = q'$. Sometimes, we use ρ in set operations where it behaves as the set of states it contains. We also use $q \overset{w}{\rightsquigarrow} q'$ to denote that $\exists \rho \in Q^* : q \overset{w,\rho}{\rightsquigarrow} q'$ and $q \rightsquigarrow q'$ to denote that $\exists w : q \overset{w}{\rightsquigarrow} q'$. The *language* of a state q is defined as $L_A(q) = \{w \mid \exists q_F \in F : q \overset{w}{\rightsquigarrow} q_F\}$ and its *banguage* (back-language) is defined as $L^\flat_A(q) = \{w \mid \exists q_I \in I : q_I \overset{w}{\rightsquigarrow} q\}$. Both notions can be naturally extended to a set $S \subseteq Q$: $L_A(S) = \bigcup_{q \in S} L_A(q)$ and $L^\flat_A(S) = \bigcup_{q \in S} L^\flat_A(q)$. We drop the subscript A when the context is obvious. A *accepts* the language $L(A)$ defined as $L(A) = L_A(I)$. A is called *deterministic* (DFA) if $|I| = 1$ and $\forall q \in Q$ and $\forall a \in \Sigma : |\delta(q, a)| \leq 1$, and *unambiguous* (UFA) if $\forall w \in L(A) : \exists! q_I \in I, \rho \in Q^*, q_F \in F : q_I \overset{w,\rho}{\rightsquigarrow} q_F$.

The *restriction* of A to $S \subseteq Q$ is an NFA $A_{|S}$ given as $A_{|S} = (S, \delta \cap (S \times \Sigma \times 2^S), I \cap S, F \cap S)$. We define the *trim* operation as $\mathit{trim}(A) = A_{|C}$ where $C = \{q \mid \exists q_I \in I, q_F \in F : q_I \rightsquigarrow q \rightsquigarrow q_F\}$. For a set of states $R \subseteq Q$, we use $\mathit{reach}(R)$ to denote the set of states reachable from R, formally, $\mathit{reach}(R) = \{r' \mid \exists r \in R : r \rightsquigarrow r'\}$. We use the number of states as the measurement of the size of A, i.e., $|A| = |Q|$.

A (discrete probability) *distribution* over a set X is a mapping $\Pr : X \to \langle 0, 1 \rangle$ such that $\sum_{x \in X} \Pr(x) = 1$. An n-state *probabilistic automaton* (PA) over Σ is a triple $P = (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \{\boldsymbol{\Delta}_a\}_{a \in \Sigma})$ where $\boldsymbol{\alpha} \in \langle 0, 1 \rangle^n$ is a vector of *initial weights*, $\boldsymbol{\gamma} \in \langle 0, 1 \rangle^n$ is a vector of *final weights*, and for every $a \in \Sigma$, $\boldsymbol{\Delta}_a \in \langle 0, 1 \rangle^{n \times n}$ is a *transition matrix* for symbol a. We abuse notation and use $Q[P]$ to denote the set of states $Q[P] = \{1, \ldots, n\}$. Moreover, the following two properties need to hold: (i) $\sum\{\boldsymbol{\alpha}[i] \mid i \in Q[P]\} = 1$ (the initial probability is 1) and (ii) for every state $i \in Q[P]$ it holds that $\sum\{\boldsymbol{\Delta}_a[i, j] \mid j \in Q[P], a \in \Sigma\} + \boldsymbol{\gamma}[i] = 1$ (the probability of accepting or leaving a state is 1). We define the *support* of P as the NFA $\mathit{supp}(P) = (Q[P], \delta[P], I[P], F[P])$ s.t.

$$\delta[P] = \{(i, a, j) \mid \boldsymbol{\Delta}_a[i, j] > 0\} \quad I[P] = \{i \mid \boldsymbol{\alpha}[i] > 0\} \quad F[P] = \{i \mid \boldsymbol{\gamma}[i] > 0\}.$$

Let us assume that every PA P is such that $\mathit{supp}(P) = \mathit{trim}(\mathit{supp}(P))$. For a word $w = a_1 \ldots a_k \in \Sigma^*$, we use $\boldsymbol{\Delta}_w$ to denote the matrix $\boldsymbol{\Delta}_{a_1} \cdots \boldsymbol{\Delta}_{a_k}$. It can be easily shown that P represents a distribution over words $w \in \Sigma^*$ defined as $\Pr_P(w) = \boldsymbol{\alpha}^\top \cdot \boldsymbol{\Delta}_w \cdot \boldsymbol{\gamma}$. We call $\Pr_P(w)$ the *probability* of w in P. Given a language $L \subseteq \Sigma^*$, we define the probability of L in P as $\Pr_P(L) = \sum_{w \in L} \Pr_P(w)$.

If Conditions (i) and (ii) from the definition of PAs are dropped, we speak about a *pseudo-probabilistic automaton (PPA)*, which may assign a word from its support a quantity that is not necessarily in the range $\langle 0, 1 \rangle$, denoted as the *significance* of the word below. PPAs may arise during some of our operations performed on PAs.

3 Approximate Reduction of NFAs

In this section, we first introduce the key notion of our approach: a *probabilistic distance* of a pair of finite automata wrt a given probabilistic automaton that, intuitively, represents the significance of particular words. We discuss the complexity of computing the probabilistic distance. Finally, we formulate two problems of *approximate automata reduction via probabilistic distance*.

3.1 Probabilistic Distance

We start by defining our notion of a probabilistic distance of two NFAs. Assume NFAs A_1 and A_2 and a probabilistic automaton P specifying the distribution $\mathrm{Pr}_P : \Sigma^* \to \langle 0, 1\rangle$. The *probabilistic distance* $d_P(A_1, A_2)$ between A_1 and A_2 wrt Pr_P is defined as

$$d_P(A_1, A_2) = \mathrm{Pr}_P(L(A_1) \bigtriangleup L(A_2)).$$

Intuitively, the distance captures the significance of the words accepted by one of the automata only. We use the distance to drive the reduction process towards automata with small errors and to assess the quality of the resulting automata.

The value of $\mathrm{Pr}_P(L(A_1) \bigtriangleup L(A_2))$ can be computed as follows. Using the fact that (1) $L_1 \bigtriangleup L_2 = (L_1 \setminus L_2) \uplus (L_2 \setminus L_1)$ and (2) $L_1 \setminus L_2 = L_1 \setminus (L_1 \cap L_2)$, we get

$$\begin{aligned} d_P(A_1, A_2) &= \mathrm{Pr}_P(L(A_1) \setminus L(A_2)) + \mathrm{Pr}_P(L(A_2) \setminus L(A_1)) \\ &= \mathrm{Pr}_P(L(A_1) \setminus (L(A_1) \cap L(A_2))) + \mathrm{Pr}_P(L(A_2) \setminus (L(A_2) \cap L(A_1))) \\ &= \mathrm{Pr}_P(L(A_1)) + \mathrm{Pr}_P(L(A_2)) - 2 \cdot \mathrm{Pr}_P(L(A_1) \cap L(A_2)). \end{aligned}$$

Hence, the key step is to compute $\mathrm{Pr}_P(L(A))$ for an NFA A and a PA P. Problems similar to computing such a probability have been extensively studied in several contexts including verification of probabilistic systems [34–36]. The below lemma summarises the complexity of this step.

Lemma 1. *Let P be a PA and A an NFA. The problem of computing $\mathrm{Pr}_P(L(A))$ is **PSPACE**-complete. For a UFA A, $\mathrm{Pr}_P(L(A))$ can be computed in **PTIME**.*

In our approach, we apply the method of [36] and compute $\mathrm{Pr}_P(L(A))$ in the following way. We first check whether the NFA A is unambiguous. This can be done by using the standard product construction (denoted as $\cap$) for computing the intersection of the NFA A with itself and trimming the result, formally $B = \mathit{trim}(A \cap A)$, followed by a check whether there is some state $(p, q) \in Q[B]$ s.t. $p \neq q$ [37]. If A is ambiguous, we either determinise it or disambiguate it [37], leading to a DFA/UFA A', respectively.[1] Then, we construct the trimmed product of A' and P (this can be seen as computing $A' \cap \mathit{supp}(P)$ while keeping the probabilities from P on the edges of the result), yielding a PPA $R = (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \{\boldsymbol{\Delta}_a\}_{a \in \Sigma})$.[2] Intuitively, R represents not only the words of $L(A)$ but also their probability in P. Now, let $\boldsymbol{\Delta} = \sum_{a \in \Sigma} \boldsymbol{\Delta}_a$ be the matrix that expresses, for any $p, q \in Q[R]$, the significance of getting from p to q via any $a \in \Sigma$. Further, it can be shown (cf. Appendix A.2) that the matrix $\boldsymbol{\Delta}^*$, representing the significance of going from p to q via any $w \in \Sigma^*$, can be computed as $(\boldsymbol{I} - \boldsymbol{\Delta})^{-1}$. Then, to get $\mathrm{Pr}_P(L(A))$, it suffices to take $\boldsymbol{\alpha}^\top \cdot \boldsymbol{\Delta}^* \cdot \boldsymbol{\gamma}$. Note that, due to the determinisation/disambiguation step, the obtained value indeed is $\mathrm{Pr}_P(L(A))$ despite R being a PPA.

[1] In theory, disambiguation can produce smaller automata, but, in our experiments, determinisation proved to work better.

[2] R is not necessarily a PA since there might be transitions in P that are either removed or copied several times in the product construction.

3.2 Automata Reduction using Probabilistic Distance

We now exploit the above introduced probabilistic distance to formulate the task of approximate reduction of NFAs as the following two optimisation problems. Given an NFA A and a PA P specifying the distribution $\mathrm{Pr}_P : \Sigma^* \to \langle 0, 1\rangle$, we define

- **size-driven reduction**: for $n \in \mathbb{N}$, find an NFA A' such that $|A'| \leq n$ and the distance $d_P(A, A')$ is minimal,
- **error-driven reduction**: for $\epsilon \in \langle 0, 1\rangle$, find an NFA A' such that $d_P(A, A') \leq \epsilon$ and the size $|A'|$ is minimal.

The following lemma shows that the natural decision problem underlying both of the above optimization problems is **PSPACE**-complete, which matches the complexity of computing the probabilistic distance as well as that of the *exact* reduction of NFAs [38].

Lemma 2. *Consider an NFA A, a PA P, a bound on the number of states $n \in \mathbb{N}$, and an error bound $\epsilon \in \langle 0, 1\rangle$. It is **PSPACE**-complete to determine whether there exists an NFA A' with n states s.t. $d_P(A, A') \leq \epsilon$.*

The notions defined above do not distinguish between introducing a *false positive* (A' accepts a word $w \notin L(A)$) or a *false negative* (A' does not accept a word $w \in L(A)$) answers. To this end, we define *over-approximating* and *under-approximating* reductions as reductions for which the additional conditions $L(A) \subseteq L(A')$ and $L(A) \supseteq L(A')$ hold, respectively.

A naïve solution to the reductions would enumerate all NFAs A' of sizes from 0 up to k (resp. $|A|$), for each of them compute $d_P(A, A')$, and take an automaton with the smallest probabilistic distance (resp. a smallest one satisfying the restriction on $d_P(A, A')$). Obviously, this approach is computationally infeasible.

4 A Heuristic Approach to Approximate Reduction

In this section, we introduce two techniques for approximate reduction of NFAs that avoid the need to iterate over all automata of a certain size. The first approach under-approximates the automata by removing states—we call it the *pruning reduction*—while the second approach over-approximates the automata by adding self-loops to states and removing redundant states—we call it the *self-loop reduction*. Finding an optimal automaton using these reductions is also **PSPACE**-complete, but more amenable to heuristics like greedy algorithms. We start with introducing two high-level greedy algorithms, one for the size- and one for the error-driven reduction, and follow by showing their instantiations for the pruning and the self-loop reduction. A crucial role in the algorithms is played by a function that labels states of the automata by an estimate of the error that will be caused when some of the reductions is applied at a given state.

4.1 A General Algorithm for Size-Driven Reduction

Algorithm 1 shows a general greedy method for performing the size-driven reduction. In order to use the same high-level algorithm in both directions of reduction (over/under-approximating), it is parameterized with three functions: label, reduce, and error. The real intricacy of the procedure is hidden inside these three functions. Intuitively, $\mathit{label}(A, P)$ assigns every state of an NFA A an approximation of the error that will be caused wrt the PA P when a reduction is applied at this state, while the purpose of $\mathit{reduce}(A, V)$ is to create a new NFA A' obtained from A by introducing some error at states from V.[3] Further, $\mathit{error}(A, V, \mathit{label}(A, P))$ estimates the error introduced by the application of $\mathit{reduce}(A, V)$, possibly in a more precise (and costly) way than by just summing the concerned error labels: Such a computation is possible outside of the main computation loop. We show instantiations of these functions later, when discussing the reductions used. Moreover, the algorithm is also parameterized with a total order $\preceq_{A,\mathit{label}(A,P)}$ that defines which states of A are processed first and which are processed later. The ordering may take into account the precomputed labelling. The algorithm accepts an NFA A, a PA P, and $n \in \mathbb{N}$ and outputs a pair consisting of an NFA A' of the size $|A'| \leq n$ and an error bound ϵ such that $d_P(A, A') \leq \epsilon$.

Algorithm 1: A greedy size-driven reduction

```
Input  : NFA $A = (Q, \delta, I, F)$, PA $P$, $n \geq 1$
Output: NFA $A'$, $\epsilon \in \mathbb{R}$ s.t. $|A| \leq n$ and
        $d_P(A, A') \leq \epsilon$
$V \leftarrow \emptyset$;
for $q \in Q$ in the order $\preceq_{A,label(A,P)}$ do
    $V \leftarrow V \cup \{q\}$; $A' \leftarrow reduce(A, V)$;
    if $|A'| \leq n$ then break ;
return $A'$, $\epsilon = error(A, V, label(A, P))$;
```

The main idea of the algorithm is that it creates a set V of states where an error is to be introduced. V is constructed by starting from an empty set and adding states to it in the order given by $\preceq_{A,\mathit{label}(A,P)}$, until the size of the result of $\mathit{reduce}(A, V)$ has reached the desired bound n (in our setting, reduce is always antitone, i.e., for $V \subseteq V'$, it holds that $|\mathit{reduce}(A, V)| \geq |\mathit{reduce}(A, V')|$). We now define the necessary condition for label, reduce, and error that makes Algorithm 1 correct.

Condition C1 *holds if for every NFA A, PA P, and a set $V \subseteq Q[A]$, we have that (a) $\mathit{error}(A, V, \mathit{label}(A, P)) \geq d_P(A, \mathit{reduce}(A, V))$, (b) $|\mathit{reduce}(A, Q[A])| \leq 1$, and (c) $\mathit{reduce}(A, \emptyset) = A$.*

C1(a) ensures that the error computed by the reduction algorithm indeed over-approximates the exact probabilistic distance, **C1**(b) ensures that the algorithm can (in the worst case, by applying the reduction at every state of A) for any $n \geq 1$ output a result $|A'|$ of the size $|A'| \leq n$, and **C1**(c) ensures that when no error is to be introduced at any state, we obtain the original automaton.

Lemma 3. *Algorithm 1 is correct if **C1** holds.*

4.2 A General Algorithm for Error-Driven Reduction

Algorithm 2: A greedy error-driven reduction.

```
Input  : NFA $A = (Q, \delta, I, F)$, PA $P$, $\epsilon \in \langle 0, 1 \rangle$
Output: NFA $A'$ s.t. $d_P(A, A') \leq \epsilon$
$\ell \leftarrow label(A, P)$;
$V \leftarrow \emptyset$;
for $q \in Q$ in the order $\preceq_{A,label(A,P)}$ do
    $e \leftarrow error(A, V \cup \{q\}, \ell)$;
    if $e \leq \epsilon$ then $V \leftarrow V \cup \{q\}$ ;
return $A' = reduce(A, V)$;
```

In Algorithm 2, we provide a high-level method of computing the error-driven reduction. The algorithm is in many ways similar to Algorithm 1; It also computes a set of states V where an error is to be introduced, but an important difference is that we compute an approximation of the error in each step and only add q to V if it does

[3] We emphasize that this does not mean that states from V will be simply removed from A—the performed operation depends on the particular reduction.

not raise the error over the threshold ϵ. Note that the error does not need to be monotone, so it may be advantageous to traverse all states from Q and not terminate as soon as the threshold is reached. The correctness of Algorithm 2 also depends on **C1**.

Lemma 4. *Algorithm 2 is correct if* ***C1*** *holds.*

4.3 Pruning Reduction

The pruning reduction is based on identifying a set of states to be removed from an NFA A, under-approximating the language of A. In particular, for $A = (Q, \delta, I, F)$, the pruning reduction finds a set $R \subseteq Q$ and restricts A to $Q \setminus R$, followed by removing useless states, to construct a reduced automaton $A' = \mathit{trim}(A_{|Q\setminus R})$. Note that the natural decision problem corresponding to this reduction is also **PSPACE**-complete.

Lemma 5. *Consider an NFA A, a PA P, a bound on the number of states $n \in \mathbb{N}$, and an error bound $\epsilon \in \langle 0, 1\rangle$. It is* ***PSPACE****-complete to determine whether there exists a subset of states $R \subseteq Q[A]$ of the size $|R| = n$ such that $d_P(A, A_{|R}) \leq \epsilon$.*

Although Lemma 5 shows that the pruning reduction is as hard as a general reduction (cf. Lemma 2), the pruning reduction is more amenable to the use of heuristics like the greedy algorithms from §4.1 and §4.2. We instantiate reduce, error, and label in these high-level algorithms in the following way (the subscript p means *pruning*):

$$\mathit{reduce}_p(A, V) = \mathit{trim}(A_{|Q\setminus V}), \quad \mathit{error}_p(A, V, \ell) = \min_{V' \in \lfloor V \rfloor_p} \sum \{\ell(q) \mid q \in V'\},$$

where $\lfloor V \rfloor_p$ is defined as follows. Because of the use of trim in reduce_p, for a pair of sets V, V' s.t. $V \subset V'$, it holds that $\mathit{reduce}_p(A, V)$ may, in general, yield the same automaton as $\mathit{reduce}_p(A, V')$. Hence, we define a partial order $\sqsubseteq_p$ on 2^Q as $V_1 \sqsubseteq_p V_2$ iff $\mathit{reduce}_p(A, V_1) = \mathit{reduce}_p(A, V_2)$ and $V_1 \subseteq V_2$, and use $\lfloor V \rfloor_p$ to denote the set of minimal elements wrt V and $\sqsubseteq_p$. The value of the approximation $\mathit{error}_p(A, V, \ell)$ is therefore the minimum of the sum of errors over all sets from $\lfloor V \rfloor_p$.

Note that the size of $\lfloor V \rfloor_p$ can again be exponential, and thus we employ a greedy approach for guessing an optimal V'. Clearly, this cannot affect the soundness of the algorithm, but only decreases the precision of the bound on the distance. Our experiments indicate that for automata appearing in NIDSes, this simplification has typically only a negligible impact on the precision of the bounds.

For computing the state labelling, we provide the following three functions, which differ in the precision they provide and the difficulty of their computation (naturally, more precise labellings are harder to compute): label_p^1, label_p^2, and label_p^3. Given an NFA A and a PA P, they generate the labellings ℓ_p^1, ℓ_p^2, and ℓ_p^3, respectively, defined as

$$\ell_p^1(q) = \sum \left\{ \Pr_P(L^\flat_A(q')) \;\middle|\; q' \in \mathit{reach}(\{q\}) \cap F \right\},$$

$$\ell_p^2(q) = \Pr_P\left(L^\flat_A(F \cap \mathit{reach}(q))\right), \qquad \ell_p^3(q) = \Pr_P\left(L^\flat_A(q).L_A(q)\right).$$

A state label $\ell(q)$ approximates the error of the words removed from $L(A)$ when q is removed. More concretely, $\ell_p^1(q)$ is a rough estimate saying that the error can be bounded by the sum of probabilities of the banguages of all final states reachable from q (in the worst case, all those final states might become unreachable). Note that $\ell_p^1(q)$ (1) counts the error of a word accepted in two different final states of $\mathit{reach}(q)$ twice, and (2) also considers words that are accepted in some final state in $\mathit{reach}(q)$ without going through q. The labelling ℓ_p^2 deals with (1) by computing the total probability of the banguage of the set of all final states reachable from q, and the labelling ℓ_p^3 in addition also deals with (2) by only considering words that traverse through q (they can still be accepted in some final state not in $reach(q)$ though, so even ℓ_p^3 is still imprecise). Note that if A is unambiguous then $\ell_p^1 = \ell_p^2$.

When computing the label of q, we first modify A to obtain A' accepting the language related to the particular labelling. Then, we compute the value of $\Pr_P(L(A'))$ using the algorithm from §3.1. Recall that this step is in general costly, due to the determinisation/disambiguation of A'. The key property of the labelling computation resides in the fact that if A is composed of several disjoint sub-automata, the automaton A' is typically much smaller than A and thus the computation of the label is considerable less demanding. Since the automata appearing in regex matching for NIDS are composed of the union of "tentacles", the particular A's are very small, which enables efficient component-wise computation of the labels.

The following lemma states the correctness of using the pruning reduction as an instantiation of Algorithms 1 and 2 and also the relation among ℓ_p^1, ℓ_p^2, and ℓ_p^3.

Lemma 6. *For every $x \in \{1, 2, 3\}$, the functions $reduce_p$, $error_p$, and $label_p^x$ satisfy **C1**. Moreover, consider an NFA A, a PA P, and let $\ell_p^x = label_p^x(A, P)$ for $x \in \{1, 2, 3\}$. Then, for each $q \in Q[A]$, we have $\ell_p^1(q) \geq \ell_p^2(q) \geq \ell_p^3(q)$.*

4.4 Self-loop Reduction

The main idea of the self-loop reduction is to over-approximate the language of A by adding self-loops over every symbol at selected states. This makes some states of A redundant, allowing them to be removed without introducing any more error. Given an NFA $A = (Q, \delta, I, F)$, the self-loop reduction searches for a set of states $R \subseteq Q$, which will have self-loops added, and removes other transitions leading out of these states, making some states unreachable. The unreachable states are then removed.

Formally, let $sl(A, R)$ be the NFA (Q, δ', I, F) whose transition function δ' is defined, for all $p \in Q$ and $a \in \Sigma$, as $\delta'(p, a) = \{p\}$ if $p \in R$ and $\delta'(p, a) = \delta(p, a)$ otherwise. As with the pruning reduction, the natural decision problem corresponding to the self-loop reduction is also **PSPACE**-complete.

Lemma 7. *Consider an NFA A, a PA P, a bound on the number of states $n \in \mathbb{N}$, and an error bound $\epsilon \in \langle 0, 1 \rangle$. It is **PSPACE**-complete to determine whether there exists a subset of states $R \subseteq Q[A]$ of the size $|R| = n$ such that $d_P(A, sl(A, R)) \leq \epsilon$.*

The required functions in the error- and size-driven reduction algorithms are instantiated in the following way (the subcript sl means *self-loop*):

$$reduce_{sl}(A, V) = trim(sl(A, V)), \quad error_{sl}(A, V, \ell) = \sum \{\ell(q) \mid q \in \min(\lfloor V \rfloor_{sl})\},$$

where $\lfloor V \rfloor_{sl}$ is defined in a similar manner as $\lfloor V \rfloor_p$ in the previous section (using a partial order $\sqsubseteq_{sl}$ defined similarly to $\sqsubseteq_p$; in this case, the order $\sqsubseteq_{sl}$ has a single minimal element, though).

The functions $label_{sl}^1$, $label_{sl}^2$, and $label_{sl}^3$ compute the state labellings ℓ_{sl}^1, ℓ_{sl}^2, and ℓ_{sl}^3 for an NFA A and a PA P defined as follows:

$$\ell_{sl}^1(q) = weight_P(L_A^\flat(q)), \qquad \ell_{sl}^2(q) = \Pr_P\left(L_A^\flat(q).\Sigma^*\right),$$

$$\ell_{sl}^3(q) = \ell_{sl}^2(q) - \Pr_P\left(L_A^\flat(q).L_A(q)\right).$$

Above, $weight_P(w)$ for a PA $P = (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \{\boldsymbol{\Delta}_a\}_{a \in \Sigma})$ and a word $w \in \Sigma^*$ is defined as $weight_P(w) = \boldsymbol{\alpha}^\top \cdot \boldsymbol{\Delta}_w \cdot \mathbf{1}$ (i.e., similarly as $\mathrm{Pr}_P(w)$ but with the final weights $\boldsymbol{\gamma}$ discarded), and $weight_P(L)$ for $L \subseteq \Sigma^*$ is defined as $weight_P(L) = \sum_{w \in L} weight_P(w)$.

Intuitively, the state labelling $\ell^1_{sl}(q)$ computes the probability that q is reached from an initial state, so if q is pumped up with all possible word endings, this is the maximum possible error introduced by the added word endings. This has the following sources of imprecision: (1) the probability of some words may be included twice, e.g., when $L^\flat_A(q) = \{a, ab\}$, the probabilities of all words from $\{ab\}.\Sigma^*$ are included twice in $\ell^1_{sl}(q)$ because $\{ab\}.\Sigma^* \subseteq \{a\}.\Sigma^*$, and (2) $\ell^1_{sl}(q)$ can also contain probabilities of words that are already accepted on a run traversing q. The state labelling ℓ^2_{sl} deals with (1) by considering the probability of the language $L^\flat_A(q).\Sigma^*$, and ℓ^3_{sl} deals also with (2) by subtracting from the result of ℓ^2_{sl} the probabilities of the words that pass through q and are accepted.

The computation of the state labellings for the self-loop reduction is done in a similar way as the computation of the state labellings for the pruning reduction (cf. §4.3). For a computation of $weight_P(L)$ one can use the same algorithm as for $\mathrm{Pr}_P(L)$, only the final vector for PA P is set to $\mathbf{1}$. The correctness of Algorithms 1 and 2 when instantiated using the self-loop reduction is stated in the following lemma.

Lemma 8. *For every $x \in \{1, 2, 3\}$, the functions $reduce_{sl}$, $error_{sl}$, and $label^x_{sl}$ satisfy **C1**. Moreover, consider an NFA A, a PA P, and let $\ell^x_{sl} = label^x_{sl}(A, P)$ for $x \in \{1, 2, 3\}$. Then, for each $q \in Q[A]$, we have $\ell^1_{sl}(q) \geq \ell^2_{sl}(q) \geq \ell^3_{sl}(q)$.*

5 Reduction of NFAs in Network Intrusion Detection Systems

We have implemented our approach in a Python prototype named APPREAL (APProximate REduction of Automata and Languages)[4] and evaluated it on the use case of network intrusion detection using SNORT [1], a popular open source NIDS.

5.1 Network Traffic Model

The reduction we describe in this paper is driven by a probabilistic model representing a distribution over Σ^*, and the formal guarantees are also wrt this model. We use *learning* to obtain a model of network traffic over the 8-bit ASCII alphabet at a given network point. Our model is created from several gigabytes of network traffic from a measuring point of the CESNET Internet provider connected to a 100 Gbps backbone link (unfortunately, we cannot provide the traffic dump since it may contain sensitive data).

Learning a PA representing the network traffic faithfully is hard. The PA cannot be too specific—although the number of different packets that can occur is finite, it is still extremely large (a conservative estimate assuming the most common scenario Ethernet/IPv4/TCP would still yield a number over $2^{10,000}$). If we assigned non-zero probabilities only to the packets from the dump (which are less than 2^{20}), the obtained model would completely ignore virtually all packets that might appear on the network, and, moreover, the model would also be very large (millions of states), making it difficult to use in our algorithms. A generalization of the obtained traffic is therefore needed.

A natural solution is to exploit results from the area of PA learning, such as [39, 40]. Indeed, we experimented with the use of ALERGIA [39], a learning algorithm that constructs a PA from a prefix tree (where edges are labelled with multiplicities) by merging

[4] https://github.com/vhavlena/appreal/tree/tacas18

nodes that are “similar.” The automata that we obtained were, however, *too* general. In particular, the constructed automata destroyed the structure of network protocols—the merging was too permissive and the generalization merged distant states, which introduced loops over a very large substructure in the automaton (such a case usually does not correspond to the design of network protocols). As a result, the obtained PA more or less represented the Poisson distribution, having essentially no value for us.

In §5.2, we focus on the detection of malicious traffic transmitted over HTTP. We take advantage of this fact and create a PA representing the traffic while taking into account the structure of HTTP. We start by manually creating a DFA that represents the high-level structure of HTTP. Then, we proceed by feeding 34,191 HTTP packets from our sample into the DFA, at the same time taking notes about how many times every state is reached and how many times every transition is taken. The resulting PA P_{HTTP} (of 52 states) is then obtained from the DFA and the labels in the obvious way.

The described method yields automata that are much better than those obtained using ALERGIA in our experiments. A disadvantage of the method is that it is only semi-automatic—the basic DFA needed to be provided by an expert. We have yet to find an algorithm that would suit our needs for learning more general network traffic.

5.2 Evaluation

We start this section by introducing the experimental setting, namely, the integration of our reduction techniques into the tool chain implementing efficient regex matching, the concrete settings of APPREAL, and the evaluation environment. Afterwards, we discuss the results evaluating the quality of the obtained approximate reductions as well as of the provided error bounds. Finally, we present the performance of our approach and discuss its key aspects. Due to the lack of space, we selected the most interesting results demonstrating the potential as well as the limitations of our approach.

General setting. SNORT detects malicious network traffic based on *rules* that contain *conditions*. The conditions may take into consideration, among others, network addresses, ports, or Perl compatible regular expressions (PCREs) that the packet payload should match. In our evaluation, we always select a subset of SNORT rules, extract the PCREs from them, and use NETBENCH [20] to transform them into a single NFA A. Before applying APPREAL, we use the state-of-the-art NFA reduction tool REDUCE [41] to decrease the size of A. REDUCE performs a language-preserving reduction of A using advanced variants of simulation [31] (in the experiment reported in Table 3, we skip the use of REDUCE at this step as discussed in the performance evaluation). The automaton A^{RED} obtained as the result of REDUCE is the input of APPREAL, which performs one of the approximate reductions from §4 wrt the traffic model P_{HTTP}, yielding A^{APP}. After the approximate reduction, we, one more time, use REDUCE and obtain the result A'.

Settings of APPREAL. In the use case of NIDS pre-filtering, it may be important to never introduce a false negative, i.e., to never drop a malicious packet. Therefore, we focus our evaluation on the *self-loop reduction* (§4.4). In particular, we use the state labelling function $label^2_{sl}$, since it provides a good trade-off between the precision and the computational demands (recall that the computation of $label^2_{sl}$ can exploit the “tentacle” structure of the NFAs we work with). We give more attention to the *size-driven reduction* (§4.1) since, in our setting, a bound on the available FPGA resources is typically given and the task is to create an NFA with the smallest error that fits inside. The order $\preceq_{A,\ell^2_{sl}}$ over states used in §4.1 and §4.2 is defined as $s \preceq_{A,\ell^2_{sl}} s' \Leftrightarrow \ell^2_{sl}(s) \leq \ell^2_{sl}(s')$.

Evaluation environment. All experiments run on a 64-bit Linux Debian workstation with the Intel Core(TM) i5-661 CPU running at 3.33 GHz with 16 GiB of RAM.

Description of tables. In the caption of every table, we provide the name of the input file (in the directory `regexps/tacas18/` of the repository of AppReal) with the selection of Snort regexes used in the particular experiment, together with the type of the reduction (size- or error-driven). All reductions are over-approximating (self-loop reduction). We further provide the size of the input automaton $|A|$, the size after the initial processing by Reduce ($|A^{\textsc{Red}}|$), and the time of this reduction ($time(\textsc{Reduce})$). Finally, we list the times of computing the state labelling $label^2_{sl}$ on $A^{\textsc{Red}}$ ($time(label^2_{sl})$), the exact probabilistic distance ($time(\text{Exact})$), and also the number of *look-up tables* ($LUTs(A^{\textsc{Red}})$) consumed on the targeted FPGA (Xilinx Virtex 7 H580T) when $A^{\textsc{Red}}$ was synthesized (more on this in §5.3). The meaning of the columns in the tables is the following:

$\boldsymbol{k}/\boldsymbol{\epsilon}$ is the parameter of the reduction. In particular, $\boldsymbol{k}$ is used for the size-driven reduction and denotes the desired reduction ration $k = \frac{n}{|A^{\textsc{Red}}|}$ for an input NFA $A^{\textsc{Red}}$ and the desired size of the output n. On the other hand, $\boldsymbol{\epsilon}$ is the desired maximum error on the output for the error-driven reduction.

$|\boldsymbol{A}^{\textsc{App}}|$ shows the number of states of the automaton $A^{\textsc{App}}$ after the reduction by AppReal and the time the reduction took.

$|\boldsymbol{A'}|$ contains the number of states of the NFA A' obtained after applying Reduce on $A^{\textsc{App}}$ and the time used by Reduce at this step.

Error bound shows the estimation of the error of A' as determined by the reduction itself, i.e., it is the probabilistic distance computed by the function $error$ in §4.

Exact error contains the values of $d_{P_{HTTP}}(A, A')$ that we computed *after* the reduction in order to evaluate the precision of the result given in **Error bound**. The computation of this value is very expensive ($time(Exact)$) since it inherently requires determinisation of the whole automaton A. We do not provide it in Table 3 (presenting the results for the automaton $A_{\texttt{bd}}$ with 1,352 states) because the determinisation ran out of memory (the step is not required in the reduction process).

Traffic error shows the error that we obtained when compared A' with A on an HTTP traffic sample, in particular the ratio of packets misclassified by A' to the total number of packets in the sample (242,468). Comparing **Exact error** with **Traffic error** gives us a feedback about the fidelity of the traffic model P_{HTTP}. We note that there are no guarantees on the relationship between **Exact error** and **Traffic error**.

LUTs is the number of LUTs consumed by A' when synthesized into the FPGA. Hardware synthesis is a costly step so we provide this value only for selected NFAs.

Approximation errors

Table 1 presents the results of the self-loop reduction for the NFA $A_{\texttt{mal}}$ describing `http-malicious` regexes. We can observe that the differences between the upper bounds on the probabilistic distance and its real value are negligible (typically in the order of 10^{-4} or less). We can also see that the probabilistic distance agrees with the

Table 1: Results for the `http-malicious` regex, $|A_{\mathtt{mal}}| = 249$, $|A^{\textsc{Red}}_{\mathtt{mal}}| = 98$, $time(\textsc{Reduce}) = 3.5\,\text{s}$, $time(label^2_{sl}) = 38.7\,\text{s}$, $time(\text{Exact}) = $ 3.8–6.5 s, and $LUTs(A^{\textsc{Red}}_{\mathtt{mal}}) = 382$.

(a) size-driven reduction

k	$\lvert A^{\textsc{App}}_{\mathtt{mal}}\rvert$	$\lvert A'_{\mathtt{mal}}\rvert$	**Error bound**	**Exact error**	**Traffic error**	**LUTs**
0.1	9 (0.65 s)	9 (0.4 s)	0.0704	0.0704	0.0685	—
0.2	19 (0.66 s)	19 (0.5 s)	0.0677	0.0677	0.0648	—
0.3	29 (0.69 s)	26 (0.9 s)	0.0279	0.0278	0.0598	154
0.4	39 (0.68 s)	36 (1.1 s)	0.0032	0.0032	0.0008	—
0.5	49 (0.68 s)	44 (1.4 s)	2.8e-05	2.8e-05	4.1e-06	—
0.6	58 (0.69 s)	49 (1.7 s)	8.7e-08	8.7e-08	0.0	224
0.8	78 (0.69 s)	75 (2.7 s)	2.4e-17	2.4e-17	0.0	297

(b) error-driven reduction

ϵ	$\lvert A^{\textsc{App}}_{\mathtt{mal}}\rvert$	$\lvert A'_{\mathtt{mal}}\rvert$	**Error bound**	**Exact error**	**Traffic error**
0.08	3	3	0.0724	0.0724	0.0720
0.07	4	4	0.0700	0.0700	0.0683
0.04	35	32	0.0267	0.0212	0.0036
0.02	36	33	0.0105	0.0096	0.0032
0.001	41	38	0.0005	0.0005	0.0003
1e-04	47	41	7.7e-05	7.7e-05	1.2e-05
1e-05	51	47	6.6e-06	6.6e-06	0.0

traffic error. This indicates a good quality of the traffic model employed in the reduction process. Further, we can see that our approach can provide useful trade-offs between the reduction error and the reduction factor. Finally, Table 1b shows that a significant reduction is obtained when the error threshold ϵ is increased from 0.04 to 0.07.

Table 2 presents the results of the size-driven self-loop reduction for NFA $A_{\mathtt{att}}$ describing `http-attacks` regexes. We can observe that the error bounds provide again a very good approximation of the real probabilistic distance. On the other hand, the difference between the probabilistic distance and the traffic error is larger than for $A_{\mathtt{mal}}$. Since all experiments use the same probabilistic automaton and the same traffic, this discrepancy is accounted to the different set of packets that are incorrectly accepted by $A^{\textsc{Red}}_{\mathtt{att}}$. If the probability of these packets is adequately captured in the traffic model, the difference between the distance and the traffic error is small and vice versa. This also explains an even larger difference in Table 3 (presenting the results for $A_{\mathtt{bd}}$ constructed from `http-backdoor` regexes) for $k \in \langle 0.2, 0.4\rangle$. Here, the traffic error is very small and caused by a small set of packets (approx. 70), whose probability is not correctly captured in the traffic model. Despite this problem, the results clearly show that our approach still provides significant reductions while keeping the traffic error small: about a 5-fold reduction is obtained for the traffic error 0.03 % and a 10-fold reduction is obtained for the traffic error 6.3 %. We discuss the practical impact of such a reduction in §5.3.

Table 2: Results for the `http-attacks` regex, size-driven reduction, $|A_{\mathtt{att}}| = 142$, $|A^{\textsc{Red}}_{\mathtt{att}}| = 112$, $time(\textsc{Reduce}) = 7.9\,\text{s}$, $time(label^2_{sl}) = 28.3\,\text{min}$, $time(\text{Exact}) = $ 14.0–16.4 min.

k	$\lvert A^{\textsc{App}}_{\mathtt{att}}\rvert$	$\lvert A'_{\mathtt{att}}\rvert$	**Error bound**	**Exact error**	**Traffic error**
0.1	11 (1.1s)	5 (0.4s)	1.0	0.9972	0.9957
0.2	22 (1.1s)	14 (0.6s)	1.0	0.8341	0.2313
0.3	33 (1.1s)	24 (0.7s)	0.081	0.0770	0.0067
0.4	44 (1.1s)	37 (1.6s)	0.0005	0.0005	0.0010
0.5	56 (1.1s)	49 (1.2s)	3.3e-06	3.3-06	0.0010
0.6	67 (1.1s)	61 (1.9s)	1.2e-09	1.2e-09	8.7e-05
0.7	78 (1.1s)	72 (2.4s)	4.8e-12	4.8e-12	1.2e-05
0.9	100 (1.1s)	93 (4.7s)	3.7e-16	1.1e-15	0.0

Performance of the approximate reduction

Table 3: Results for the `http-backdoor` regex, size-driven reduction, $|A_{\texttt{bd}}| = 1,352$, $time(\textsc{Reduce}) = 8.1\,\mathrm{min}$, $time(label^2_{sl}) = 19.9\,\mathrm{min}$, $LUTs(A^{\textsc{Red}}_{\texttt{bd}}) = 2,266$.

k	$\lvert A^{\textsc{App}}_{\texttt{bd}}\rvert$	$\lvert A'_{\texttt{bd}}\rvert$	Error bound	Traffic error	LUTs
0.1	135 (1.2 m)	8 (2.6 s)	1.0	0.997	202
0.2	270 (1.2 m)	111 (5.2 s)	0.0012	0.0631	579
0.3	405 (1.2 m)	233 (9.8 s)	3.4e-08	0.0003	894
0.4	540 (1.3 m)	351 (21.7 s)	1.0e-12	0.0003	1063
0.5	676 (1.3 m)	473 (41.8 s)	1.2e-17	0.0	1249
0.7	946 (1.4 m)	739 (2.1 m)	8.3e-30	0.0	1735
0.9	1216 (1.5 m)	983 (5.6 m)	1.3e-52	0.0	2033

In all our experiments (Tables 1–3), we can observe that the most time-consuming step of the reduction process is the computation of state labellings (it takes at least 90 % of the total time). The crucial observation is that the structure of the NFAs fundamentally affects the performance of this step. Although after REDUCE, the size of $A_{\texttt{mal}}$ is very similar to the size of $A_{\texttt{att}}$, computing $label^2_{sl}$ takes more time (28.3 min vs. 38.7 s). The key reason behind this slowdown is the determinisation (or alternatively disambiguation) process required by the product construction underlying the state labelling computation (cf. §4.4). For $A_{\texttt{att}}$, the process results in a significantly larger product when compared to the product for $A_{\texttt{mal}}$. The size of the product directly determines the time and space complexity of solving the linear equation system required for computing the state labelling.

As explained in §4, the computation of the state labelling $label^2_{sl}$ can exploit the "tentacle" structure of the NFAs appearing in NIDSes and thus can be done component-wise. On the other hand, our experiments reveal that the use of REDUCE typically breaks this structure and thus the component-wise computation cannot be effectively used. For the NFA $A_{\texttt{mal}}$, this behaviour does not have any major performance impact as the determinisation leads to a moderate-sized automaton and the state labelling computation takes less than 40 s. On the other hand, this behaviour has a dramatic effect for the NFA $A_{\texttt{att}}$. By disabling the initial application of REDUCE and thus preserving the original structure of $A_{\texttt{att}}$, we were able to speed up the state label computation from 28.3 min to 1.5 min. Note that other steps of the approximate reduction took a similar time as before disabling REDUCE and also that the trade-offs between the error and the reduction factor were similar. Surprisingly, disabling REDUCE caused that the computation of the exact probabilistic distance became computationally infeasible because the determinisation ran out of memory.

Due to the size of the NFA $A_{\texttt{bd}}$, the impact of disabling the initial application of REDUCE is even more fundamental. In particular, computing the state labelling took only 19 min, in contrast to running out of memory when the REDUCE is applied in the first step. Note that the size of $A_{\texttt{bd}}$ also slows down other reduction steps (the greedy algorithm and the final REDUCE reduction). We can, however, clearly see that computing the state labelling is still the most time-consuming step.

5.3 The Real Impact in an FPGA-Accelerated NIDS

Further, we also evaluated some of the obtained automata in the setting of [5] implementing a high-speed NIDS pre-filter. In that setting, the amount of resources available for the regex matching engine is 15,000 LUTs[5] and the frequency of the engine is 200 MHz. We synthesized NFAs that use a 32-bit-wide data path, corresponding to processing 4 ASCII characters at once, which is—according to the analysis in [5]—the best trade-off between the utilization of the chip resources and the maximum achievable frequency. A simple analysis shows that the throughput of one automaton is 6.4 Gbps, so in order to reach the desired link speed of 100 Gbps, 16 units are required, and 63 units are needed to handle 400 Gbps. With the given amount of LUTs, we are therefore bounded by 937 LUTs for 100 Gbps and 238 LUTs for 400 Gbps.

We focused on the consumption of LUTs by an implementation of the regex matching engines for `http-backdoor` ($A^{\textsc{Red}}_{\mathtt{bd}}$) and `http-malicious` ($A^{\textsc{Red}}_{\mathtt{mal}}$).

- **100 Gbps**: For this speed, $A^{\textsc{Red}}_{\mathtt{mal}}$ can be used without any approximate reduction as it is small enough to fit in the available space. On the other hand, $A^{\textsc{Red}}_{\mathtt{bd}}$ without the approximate reduction is way too large to fit (at most 6 units fit inside the available space, yielding the throughput of only 38.4 Gbps, which is unacceptable). The column **LUTs** in Table 3 shows that using our framework, we are able to reduce $A^{\textsc{Red}}_{\mathtt{bd}}$ such that it uses 894 LUTs (for $\boldsymbol{k}$ = 0.3), and so all the needed 16 units fit into the FPGA, yielding the throughput over 100 Gbps and the theoretical error bound of a false positive $\leq 3.4 \times 10^{-8}$ wrt the model P_{HTTP}.
- **400 Gbps**: Regex matching at this speed is extremely challenging. The only reduced version of $A^{\textsc{Red}}_{\mathtt{bd}}$ that fits in the available space is the one for the value $\boldsymbol{k} = 0.1$ with the error bound almost 1. The situation is better for $A^{\textsc{Red}}_{\mathtt{mal}}$. In the exact version, at most 39 units can fit inside the FPGA with the maximum throughput of 249.6 Gbps. On the other hand, when using our approximate reduction framework, we are able to place 63 units into the FPGA, each of the size 224 LUTs ($\boldsymbol{k}$ = 0.6) with the throughput over 400 Gbps and the theoretical error bound of a false positive $\leq 8.7 \times 10^{-8}$ wrt the model P_{HTTP}.

6 Conclusion

We have proposed a novel approach for approximate reduction of NFAs used in network traffic filtering. Our approach is based on a proposal of a probabilistic distance of the original and reduced automaton using a probabilistic model of the input network traffic, which characterizes the significance of particular packets. We characterized the computational complexity of approximate reductions based on the described distance and proposed a sequence of heuristics allowing one to perform the approximate reduction in an efficient way. Our experimental results are quite encouraging and show that we can often achieve a very significant reduction for a negligible loss of precision. We showed that using our approach, FPGA-accelerated network filtering on large traffic speeds can be applied on regexes of malicious traffic where it could not be applied before.

In the future, we plan to investigate other approximate reductions of the NFAs, maybe using some variant of abstraction from abstract regular model checking [42], adapted for the given probabilistic setting. Another important issue for the future is to develop better ways of learning a suitable probabilistic model of the input traffic.

[5] We omit the analysis of flip-flop consumption because in our setting it is dominated by the LUT consumption.

References

1. The Snort Team: Snort (`http://www.snort.org`).
2. Becchi, M., Wiseman, C., Crowley, P.: Evaluating Regular Expression Matching Engines on Network and General Purpose Processors. In: Proc. of ANCS'09, ACM (2009) 30–39
3. Kořenek, J., Kobierský, P.: Intrusion Detection System Intended for Multigigabit Networks. In: Proc. of DDECS'07, IEEE (2007)
4. Kaštil, J., Kořenek, J., Lengál, O.: Methodology for Fast Pattern Matching by Deterministic Finite Automaton with Perfect Hashing. In: Proc. of DSD'07, IEEE (2009) 823–829
5. Matoušek, D., Kořenek, J., Puš, V.: High-Speed Regular Expression Matching with Pipelined Automata. In: Proc. of FPT'16, IEEE (2016) 93–100
6. Kumar, S., Dharmapurikar, S., Yu, F., Crowley, P., Turner, J.S.: Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection. In: Proc. of SIGCOMM'06, ACM (2006) 339–350
7. Tan, L., Sherwood, T.: A High Throughput String Matching Architecture for Intrusion Detection and Prevention. In: Proc. of ISCA'05, IEEE (2005) 112–122
8. Kumar, S., Turner, J.S., Williams, J.: Advanced Algorithms for Fast and Scalable Deep Packet Inspection. In: Proc. of ANCS'06, ACM (2006) 81–92
9. Becchi, M., Crowley, P.: A Hybrid Finite Automaton for Practical Deep Packet Inspection. In: Proc. of CoNEXT'07, ACM (2007)
10. Becchi, M., Crowley, P.: An Improved Algorithm to Accelerate Regular Expression Evaluation. In: Proc. of ANCS'07, ACM (2007) 145–154
11. Kumar, S., Chandrasekaran, B., Turner, J.S., Varghese, G.: Curing Regular Expressions Matching Algorithms from Insomnia, Amnesia, and Acalculia. In: Proc. of ANCS'07, ACM (2007) 155–164
12. Yu, F., Chen, Z., Diao, Y., Lakshman, T.V., Katz, R.H.: Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection. In: Proc. of ANCS'06, ACM (2006) 93–102
13. Liu, C., Wu, J.: Fast Deep Packet Inspection with a Dual Finite Automata. IEEE Transaction on Computers **62**(2) (2013) 310–321
14. Luchaup, D., De Carli, L., Jha, S., Bach, E.: Deep Packet Inspection with DFA-trees and Parametrized Language Overapproximation. In: Proc. of INFOCOM'14, IEEE (2014) 531–539
15. Mitra, A., Najjar, W.A., Bhuyan, L.N.: Compiling PCRE to FPGA for Accelerating SNORT IDS. In: Proc. of ANCS'07, ACM (2007) 127–136
16. Brodie, B.C., Taylor, D.E., Cytron, R.K.: A Scalable Architecture for High-Throughput Regular-Expression Pattern Matching. In: Proc. of ISCA'06, IEEE (2006) 191–202
17. Clark, C.R., Schimmel, D.E.: Efficient Reconfigurable Logic Circuits for Matching Complex Network Intrusion Detection Patterns. In: Proc. of FPL'03. Volume 2778 of LNCS, Springer (2003) 956–959
18. Hutchings, B.L., Franklin, R., Carver, D.: Assisting Network Intrusion Detection with Reconfigurable Hardware. In: Proc. of FCCM'02, IEEE (2002) 111–120
19. Sidhu, R.P.S., Prasanna, V.K.: Fast Regular Expression Matching Using FPGAs. In: Proc. of FCCM'01, IEEE (2001) 227–238
20. Puš, V., Tobola, J., Košař, V., Kaštil, J., Kořenek, J.: Netbench: Framework for Evaluation of Packet Processing Algorithms. Proc. of ANCS'11, ACM/IEEE (2011) 95–96
21. Maletti, A., Quernheim, D.: Optimal Hyper-Minimization. CoRR **abs/1104.3007** (2011)
22. Gawrychowski, P., Jez, A.: Hyper-Minimisation Made Efficient. In: Proc. of MFCS'09. Volume 5734 of LNCS, Springer (2009) 356–368

23. Gange, G., Ganty, P., Stuckey, P.J.: Fixing the State Budget: Approximation of Regular Languages with Small DFAs. In: Proc. of ATVA'17. Volume 10482 of LNCS, Springer (2017) 67–83
24. Parker, A.J., Yancey, K.B., Yancey, M.P.: Regular Language Distance and Entropy. CoRR **abs/1602.07715** (2016)
25. Mohri, M.: Edit-Distance of Weighted Automata. In: Proc. of CIAA'02. Volume 2608 of LNCS, Springer (2002) 1–23
26. Malcher, A.: Minimizing Finite Automata is Computationally Hard. Theoretical Computer Science **327**(3) (2004) 375–390
27. Hopcroft, J.E.: An $N.logN$ Algorithm for Minimizing States in a Finite Automaton. Technical report (1971)
28. Paige, R., Tarjan, R.E.: Three Partition Refinement Algorithms. SIAM Journal of Computing **16**(6) (1987) 973–989
29. Bustan, D., Grumberg, O.: Simulation-based Minimization. ACM Transactions on Computational Logic **4**(2) (2003) 181–206
30. Champarnaud, J., Coulon, F.: NFA Reduction Algorithms by Means of Regular Inequalities. Theoretical Computer Science **327**(3) (2004) 241–253
31. Mayr, R., Clemente, L.: Advanced Automata Minimization. In: Proc. of POPL'13, ACM (2013) 63–74
32. Etessami, K.: A Hierarchy of Polynomial-Time Computable Simulations for Automata. In: Proc. of CONCUR'02. Volume 2421 of LNCS, Springer (2002) 131–144
33. Clemente, L.: Büchi Automata Can Have Smaller Quotients. In: Proc. of ICALP'11. Volume 6756 of LNCS, Springer (2011) 258–270
34. Vardi, M.Y.: Automatic Verification of Probabilistic Concurrent Finite State Programs. In: Proc. of SFCS'85. IEEE (1985) 327–338
35. Baier, C., Kiefer, S., Klein, J., Klüppelholz, S., Müller, D., Worrell, J.: 'Markov Chains and Unambiguous Büchi Automata. In: Proc. of CAV'16, Volume 9779 of LNCS, Springer (2016) 23–42
36. Baier, C., Kiefer, S., Klein, J., Klüppelholz, S., Müller, D., Worrell, J.: Markov Chains and Unambiguous Büchi Automata. CoRR **abs/1605.00950** (2016)
37. Mohri, M.: A disambiguation Algorithm for Finite Automata and Functional Transducers. In: Proc. of CIAA'12. Volume 7381 of LNCS, Springer (2012) 265–277
38. Jiang, T., Ravikumar, B.: Minimal NFA Problems Are Hard. SIAM Journal on Computing **22**(6) (1993) 1117–1141
39. Carrasco, R.C., Oncina, J.: Learning Stochastic Regular Grammars by Means of a State Merging Method. In: Proc. of ICGI'94. Volume 862 of LNCS, Springer-Verlag (1994) 139–152
40. Thollard, F., Clark, A.: Learning Stochastic Deterministic Regular Languages. In: Proc. of ICGI'04. Volume 3264 of LNCS, Springer-Verlag (2004) 248–259
41. Mayr, R., et al.: Reduce: A Tool for Minimizing Nondeterministic Finite-Word and Büchi Automata. `http://languageinclusion.org/doku.php?id=tools` [Online; accessed 2017-09-30].
42. Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T.: Abstract Regular (Tree) Model Checking. STTT **14**(2) (2012) 167–191
43. Hogben, L.: Handbook of Linear Algebra. 2nd ed. CRC Press (2013)

A Proofs of Lemmas

A.1 Proof of the First Part of Lemma 1—PSPACE for NFAs

(Sketch) The membership in **PSPACE** can be shown as follows. The computation described at the end of §3.1 corresponds to solving a linear equation system. The system has an exponential size because of the blowup caused by the determinisation/disambiguation of A required by the product construction. The equation system can, however, be constructed by a **PSPACE** transducer. Moreover, as solving linear equation systems can be done in the complexity class **NC**, i.e., using a poly-logarithmic space transducer, one can combine these two transducers to obtain a **PSPACE** algorithm. The hardness is obtained by a reduction from the (**PSPACE**-complete) universality of NFAs. Intuitively, we can build a PA P defining a probability distribution over Σ^* assigning each $w \in \Sigma^*$ a non-zero probability. Then, $L(A) = \Sigma^*$ iff $\Pr_P(L(A)) = 1$. □

A.2 Proof of the Second Part of Lemma 1—PTIME for UFAs

We modify the proof from [36] into our setting. We assume a UFA $A = (Q, \delta, I, F)$ and a PA $P = (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \{\boldsymbol{\Delta}_a\}_{a \in \Sigma})$. First, note that $\Pr_P(L(A))$ can be seen as a restriction of $\Pr_P$ to $L(A)$. We construct a PPA R obtained by a trim product of P and A. Then, we construct $|R|$-row square matrix[6] $\boldsymbol{E}$ describing the transitions of R as

$$\boldsymbol{E}[(q_P, q_A), (q'_P, q'_A)] = \sum_{a \in \Sigma} \left(\boldsymbol{\Delta}_a[q_P, q'_P] \cdot |\{(q_A, a, q'_A)\} \cap \delta| \right). \tag{1}$$

Note that $\rho(\boldsymbol{E}) < 1$, where $\rho(\boldsymbol{E})$ is the *spectral radius* of $\boldsymbol{E}$. Intuitively, because we removed the redundant nodes from the product of P and A, the significance of paths between two nodes of length n for $n \to \infty$ is 0 (for a precise proof see [36]). We further use the following standard result in linear algebra: if $\rho(\boldsymbol{E}) < 1$, then (i) the matrix $\boldsymbol{I} - \boldsymbol{E}$ is invertible and (ii) the sum of powers of $\boldsymbol{E}$, denoted as $\boldsymbol{E}^*$, can be computed as $\boldsymbol{E}^* = \sum_{i=0}^{\infty} \boldsymbol{E}^i = (\boldsymbol{I} - \boldsymbol{E})^{-1}$ [43] (note that matrix inversion can be done in polynomial time).

The matrix $\boldsymbol{E}^*$ represents the reachability between nodes of R, i.e., $\boldsymbol{E}^*[r, r']$ is the sum of significances of all (possibly infinitely many) paths from r to r' in R. When related to P and A, $\boldsymbol{E}^*$ represents reachability in P wrt the language of A, in particular,

$$\boldsymbol{E}^*[(q_P, q_A), (q'_P, q'_A)] = \sum \left\{ \boldsymbol{\Delta}_w[q_P, q'_P] \;\middle|\; q_A \overset{w}{\rightsquigarrow} q'_A \in A, w \in \Sigma^* \right\}. \tag{2}$$

Using this matrix we can compute $\Pr_P(L(A))$ as

$$\Pr_P(L(A)) = \boldsymbol{\alpha}_R^\top \cdot \boldsymbol{E}^* \cdot \boldsymbol{\gamma}_R, \tag{3}$$

where $\boldsymbol{\alpha}_R$ is a vector of initial weights of R and $\boldsymbol{\gamma}_R$ is a vector of final weights of R. □

[6] We implicitly assume a bijection between states of the product R and $\{1, \ldots, |Q[R]|\}$.

A.3 Proof of Lemma 2

Membership in **PSPACE**: We non-deterministically generate an automaton A' with n states and test (in **PSPACE**, as shown in Lemma 1) that $d_P(A, A') \leq \epsilon$. This shows the problem is in **NPSPACE** $=$ **PSPACE**.

PSPACE-hardness: We use a reduction from the problem of checking universality of an NFA $A = (Q, \delta, I, F)$ over Σ, i.e., from checking whether $L(A) = \Sigma^*$, which is **PSPACE**-complete. First, for a reason that will become clear below, we test whether A accepts all words over Σ of length 0 and 1, which can be done in polynomial time. Next, let P be a one-state PA that assigns a nonzero probability to each word from Σ^*, e.g., $P = \big(\mathbf{1}, [\mu], \{[\mu]_a\}_{a \in \Sigma}\big)$ where $\mu = \frac{1}{|\Sigma|+1}$. We can now show that $L(A) = \Sigma^*$ iff there is a 1-state NFA A' s.t. $d_P(A, A') \leq 0$. The implication from left to right is clear. The reverse implication: We have tested that $\{\varepsilon\} \cup \Sigma \subseteq L(A)$. Therefore, if there is a 1-state automaton A' s.t. $d_P(A, A') \leq 0$, then A' must also accept all words over Σ of length 0 and 1, and hence $L(A') = \Sigma^*$. This, together with the fact that P assigns a nonzero probability to each word from Σ^*, implies $L(A) = \Sigma^*$. □

A.4 Proof of Lemmas 3 and 4

Follows straightforwardly from Condition **C1**. □

A.5 Proof of Lemma 5

Membership in **PSPACE**: We non-deterministically generate a subset R of $Q[A]$ having n states and test (in **PSPACE**, as shown in Lemma 1) that $d_P(A, A_{|R}) \leq \epsilon$. This shows the problem is in **NPSPACE** $=$ **PSPACE**.

PSPACE-hardness: We reduce from the **PSPACE**-complete problem of checking universality of an NFA $A = (Q, \delta, I, F)$. Let us consider some symbol $x \notin \Sigma$. Then, we construct an NFA A' s.t. $L(A') = x^*.L(A)$ and we set $n = |A'| + 1$. Further, let P be a PA representing a distribution that is defined for each $w \in (\Sigma \cup \{x\})^*$ as

$$\mathrm{Pr}_P(w) = \begin{cases} \mu^{|w'|+1} & w = x^n.w', w' \in \Sigma^*, \mu = \frac{1}{|\Sigma|+1}, \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Consequently, we construct an $(n+1)$-state NFA B accepting the language $x^n.\Sigma^*$ defined as follows $B = (Q_B, \delta_B, \{q_1\}, \{q_{n+1}\})$ where $Q_B = \{q_1, \ldots, q_{n+1}\}$ and $\delta_B = \{q_i \xrightarrow{x} q_{i+1} \mid 1 \leq i \leq n\} \cup \{q_{n+1} \xrightarrow{a} q_{n+1} \mid a \in \Sigma\}$. Using this automaton, we construct an NFA $C = A' \cup B$ where the union of two NFAs is defined as $A_1 \cup A_2 = (Q[A_1] \uplus Q[A_2], \delta[A_1] \uplus \delta[A_2], I[A_1] \uplus I[A_2], F[A_1] \uplus F[A_2])$. Further, if there exists a set $R \subseteq Q[A'] \uplus Q[B]$ of the size $|R| = n$ s.t. $d_P(C, C_{|R}) \leq 0$, then $L(B) = \Sigma^*$ Since $|Q[A']| = n-1$, at least one state from $Q[B]$ is not in $Q[C_{|R}]$ and therefore, every state from $Q[C]$ in $C_{|R}$ cannot both reach a final state and be reachable by some initial state. However, we still have $x^n.\Sigma^* \subseteq L(C_{|R})$ and thus $L(A) = \Sigma^*$. Conversely, if $L(A) = \Sigma^*$, there there exists a set $R \subseteq Q[A'] \uplus Q[B]$ of the size $|R| = n$ s.t. $d_P(C, C_{|R}) \leq 0$. □

A.6 Proof of Lemma 6

First, $|\mathit{reduce}_p(A, Q[A])| = 0$ and $\mathit{reduce}_p(A, \emptyset) = A$. Next, for each $q \in Q[A]$ we prove inequalities $\ell_p^1(q) \geq \ell_p^2(q) \geq \ell_p^3(q)$. The first one follows from the fact that if the banguages of reachable final states are not disjoint, in the case of ℓ_p^1 we may sum probabibilities of the same words multiple times. The second inequality follows from the inclusion $L_A^\flat(q).L_A(q) \subseteq L_A^\flat(F \cap \mathit{reach}_A(q))$.

For a second part of the proof, we consider some set of states $V \subseteq Q[A]$ and an arbitrary $V' \subseteq Q[A]$ s.t. $V' \sqsubseteq_p V$. Further, we have

$$\begin{aligned} L(A) \triangle L(\mathit{reduce}_p(A, V)) &= L(A) \triangle L(\mathit{reduce}_p(A, V')) \\ &= L(A) \setminus L(\mathit{reduce}_p(A, V')) \\ &\subseteq \bigcup_{q \in V'} L_A^\flat(q).L_A(q). \end{aligned} \tag{5}$$

Finally, using Equation 5 we obtain

$$\begin{aligned} d_P(A, \mathit{reduce}_p(A, V)) &= \mathrm{Pr}_P(L(A) \setminus L(\mathit{reduce}_p(A, V'))) \\ &\leq \sum_{q \in V'} \mathrm{Pr}_P(L_A^\flat(q).L_A(q)) = \mathit{error}_p(A, V, \ell_p^3), \end{aligned} \tag{6}$$

which concludes the proof. □

A.7 Proof of Lemma 7

Membership in **PSPACE** can be proved in the same way as in the proof of Lemma 5.

PSPACE-hardness: We reduce from the **PSPACE**-complete problem of checking universality of an NFA $A = (Q, \delta, I, F)$. First, we check whether $I[A] \neq \emptyset$. Next, we consider a one-state PA P that assigns a nonzero probability to each word from Σ^*, e.g., $P = \big(\mathbf{1}, [\mu], \{[\mu]_a\}_{a \in \Sigma}\big)$ where $\mu = \frac{1}{|\Sigma|+1}$. Now, we have $L(A) = \Sigma^*$ iff there exists a set of states $R \subseteq Q$ of the size $|Q|$ such that $d_P(A, sl(A, R)) \leq 0$. □

A.8 Proof of Lemma 8

First, $|\mathit{reduce}_{sl}(A, Q[A])| = 1$ and $\mathit{reduce}_{sl}(A, \emptyset) = A$. Next, for each $q \in Q[A]$ we prove inequalities $\ell_{sl}^1(q) \geq \ell_{sl}^2(q) \geq \ell_{sl}^3(q)$. We come out from the equation $\mathit{weight}_P(w) = \mathrm{Pr}_P(w.\Sigma^*)$. From this equality, we obtain

$$\sum_{w \in L_A^\flat(q)} \mathit{weight}_P(w) = \sum_{w \in L_A^\flat(q)} \mathrm{Pr}_P(w.\Sigma^*), \tag{7}$$

which implies $\mathit{weight}_P(L_A^\flat(q)) \geq \mathrm{Pr}_P\left(L_A^\flat(q).\Sigma^*\right)$.

For the second part, let us consider some set of states $V, V' \subseteq Q[A]$ s.t. $V' \sqsubseteq_{sl} V$. Then, we can estimate the symmetric difference of the laguages of the original and the reduced automaton as

$$\begin{aligned} L(A) \triangle L(\mathit{reduce}_{sl}(A, V)) &= L(A) \triangle L(\mathit{reduce}_{sl}(A, V')) \\ &\subseteq \bigcup_{q \in V'} L_A^\flat(q).\Sigma^* \setminus \bigcup_{q \in V'} L_A^\flat(q).L_A(q). \end{aligned} \tag{8}$$

Finally, using the previous inclusion, we can estimate the probabilistic distance of A and $\mathit{reduce}_{sl}(A, V)$ as

$$\begin{aligned} d_P(A, \mathit{reduce}_{sl}(A, V)) &\leq \Pr_P\left(\bigcup_{q \in V'} L^{\flat}_A(q).\Sigma^* \setminus \bigcup_{q \in V'} L^{\flat}_A(q).L_A(q) \right) \\ &\leq \sum_{q \in V'} \Pr_P\left(L^{\flat}_A(q).\Sigma^*\right) - \sum_{q \in V'} \Pr_P\left(L^{\flat}_A(q).L_A(q)\right) \qquad (9) \\ &= \mathit{error}_{sl}(A, V, \ell^3_{sl}), \end{aligned}$$

which concludes the proof. □