Space-Time Tradeoff in Regular Expression Matching with Semi-Deterministic Finite Automata*

Yi-Hua E. Yang and Viktor K. Prasanna
Ming Hsieh Dept. of Electrical Eng., University of Southern California

***Abstract*—Regular expression matching (REM) with nondeterministic finite automata (NFA) can be computationally expensive when a large number of patterns are matched concurrently. On the other hand, converting the NFA to a deterministic finite automaton (DFA) can cause *state explosion*, where the number of states and transitions in the DFA are exponentially larger than in the NFA. In this paper, we seek to answer the following question: to match an arbitrary set of regular expressions, is there a finite automaton that lies between the NFA and DFA in terms of computation and memory complexities? We introduce the *semi-deterministic finite automata* (SFA) and the *state convolvement test* to construct an SFA from a given NFA. An SFA consists of a fixed number (p) of *constituent DFAs* (c-DFA) running in parallel; each c-DFA is responsible for a subset of states in the original NFA. To match a set of regular expressions with n overlapping symbols (that can match to the same input character concurrently), the NFA can require $O(n)$ computation per input character¸ whereas the DFA can have a state transition table with $O(2^n)$ states. By exploiting the *state convolvements* during the SFA construction, an equivalent SFA reduces the computation complexity to $O(p^2/c^2)$ per input character while limiting the space requirement to $O(p^2 \times (n/p)^c)$ states, where $c \geq 1$ is a small design constant. Although the problem of constructing the optimal (minimum-sized) SFA is shown to be NP-complete, we develop a greedy heuristic to quickly construct a near-optimal SFA in time and space quadratic in the number of states in the original NFA. We demonstrate our SFA construction using real-world regular expressions taken from the Snort IDS.**

***Index Terms*—Regular expression, deep packet inspection, NFA, DFA, graph coloring, Aho-Corasick algorithm, state explosion, space-time tradeoff**

I. Introduction

Regular expressions and finite automata are classic topics in languages [11, 13]. Recently, regular expression matching (REM) has become key components in a number of important applications ranging from network filtering to text processing. Prominent examples include network intrusion detection, Extensible Markup Language (XML) processing and Biosequence analysis. In particular, popular network intrusion detection software [1, 2, 4] performs REM over a large number of patterns for deep packet inspection (DPI).

However, implementing a large-scale REM efficiently can be a hard problem. Suppose there are $k > 1$ regular expressions (*regexes*) to match, where the i-th regex, $i = 1 \dots k$, has n_i characters. Converting the k regexes into an NFA using the modified McNaughton-Yamada construction [26] will produce $n \triangleq \sum_{i=1}^{k} n_i$ NFA states, which must be examined in parallel for each input character. This results in high computation complexity in the NFA-based REM. While the NFA can be compiled into accelerated hardware architectures [12, 27] to achieve high throughput performance, the hardware-based solution usually lacks flexibility and is hard to update.

Alternatively, the NFA can be converted to a DFA, where the transitions between all valid combinations of the NFA states are compiled into a deterministic state transition table (STT). The DFA performs one STT access per input character and can be easily implemented on a processor-based architecture. Nevertheless, DFA-based REM still has several problems:

1) *State-space explosion*. To represent all possible state combinations in the original NFA, the number of states in the minimal equivalent DFA can be $O(2^n)$ in the worst case [21].
2) *State-transition explosion*. To make state transitions deterministic, any character class (*e.g.*, $[\mathtt{a}-\mathtt{z}]$) must be split into individual character labels (*e.g.*, $(\mathtt{a}|\mathtt{b}\dots|\mathtt{z})$), which can increase the number of (non-initial) state transitions in the STT by $O(|\Sigma|)$.
3) *Minimization cost*. Although the minimal DFA can have $N < O(2^n)$ states, minimizing the DFA can take $O(N' \log N')$ time [15], where N' is the pre-minimized DFA size and is much closer to $O(2^n)$.
4) *Compression cost*. Although the STT can be subsequently compressed by exploiting similarity between various state transitions, a good compression algorithm usually has time complexity at least $O(N^2)$.
5) *Memory access latency*. In practice the STT accesses performed by a DFA usually have very poor locality due to random and input-dependent state traversals. This results in poor cache performance and long memory access latencies, which can in tun make the matching throughput low and input-dependent.

In this work, we ask the following question: what would be the tradeoff architecture between an NFA and its equivalent DFA? More specifically, we wish to find a finite automata architecture which provides the space-time tradeoff between the NFA and DFA, where *space* is defined as the amount of static memory required to store and run the finite machine, and *time* is the amount of computation and communication required by the state machine to process each input character. We establish the formalism of the *convolvement analysis*, with which we may construct a (family of) *semi-deterministic finite automaton* (SFA) from an arbitrary NFA (particularly one

* Supported by U.S. National Science Foundation under grant CCR-0702784.

constructed for REM). The SFA allows us to tradeoff the NFA-like conciseness for some DFA-like computation efficiency. Specifically, our contributions include the following:

1) We establish a formal *convolvement analysis* to describe the relations between various states in an NFA.
2) With the help of the convolvement analysis, we analytically show that the number of additional DFA states required is between quadratic and exponential in the number of *pair-wise conflicting* states in the NFA.
3) We propose the *semi-deterministic finite automaton* (SFA) and an algorithm to construct a *minimal SFA*. The minimal SFA construction problem is shown to be equivalent to the graph coloring problem.
4) Analytically, we show the SFA has the time complexity of $O(p^2/c^2)$ per input character, where $p \ll n$ is the number of *constituents* in the SFA and $c > 1$ is a small design constant. We also derive SFA's space complexity to be $O(p^2 \times (\frac{n}{p})^c)$, and show how to change the "design constant" c for the space-time tradeoff.
5) We implement a prototype SFA and verify its correctness with a non-trivial set of complex real-life patterns and various input streams.

Section II reviews the related work. Section III gives the definitions of NFA and DFA and describes both the state-space and state-transition explosions. In Section IV, we define various types of *state convolvements* and use them to quantify the state-space explosion. Section V introduces the SFA architecture and the minimal SFA construction algorithm. Section VI analyzes the performance and space-time tradeoff of SFA and proposes several performance optimizations. Experiments on a prototype SFA implementation is described in Section VII, before the conclusion in. Section VIII.

II. Related Work

NFA-based REM was initially implemented in a circuit-based architecture [12]. Sidhu and Prasanna [23] later proposed to construct NFA-based REM circuits on field-programmable gate arrays (FPGA)[23]. Optimizations such as input/output pipelining [16], common-prefix extraction [10, 16], temporal [25] and spatial [26] multi-character matching, and centralized [10, 20] and memory-assisted [26] character decoding, were applied to further improve the matching throughput and resource utilization. While achieving high throughput (~10 Gbps) over a large number (>1k) of patterns, the Achilles' heel of NFA-based REM on FPGA is its circuit-based construction, which is difficult to realize and update.

DFA-based REM utilizes relatively simple operations on processor-based architectures. However, state-space explosion of the resulting DFA can greatly increase the amount of required memory, which could in turn impact the memory performance and even the practical feasibility of the solution. Various techniques were proposed to compress the state transition table (STT) of the DFA with nondeterministic features [8, 14, 17, 18, 24], achieving moderately high matching throughput [9]. These techniques, however, can be computationally expensive and are often designed *a-posteriori* to a particular set (or type) of regexes.

Partitioning a large-scale REM into smaller problems is a well-known technique. In [1], DFAs are constructed from small groups of regexes in an ad-hoc and "lazy" manner. In [28], a trial-and-error algorithm was proposed to group a set of $m > 1$ regexes into $g \in (0, m)$ groups. In both [17] and [7], techniques were proposed to split each regex into a prefix and a suffix, where all prefixes are compiled into a single composite DFA, which triggers a suffix matching when a boundary states is reached.

High-performance REM software was proposed in [22], which matches patterns in a memory-based approach similar to the Generalized Aho-Corasick algorithm [19]. While reporting very high throughput, their software accepts only a simplified set of regex constructions; it does not allow unions of arbitrary sub-regexes, and accepts Kleene closures only over single *any-value* characters.

III. Theoretical Foundations

A. NFA and DFA-based REM

Definition 1: An *NFA-based REM* for regex *R* is a nondeterministic finite automaton $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ constructed for matching *R* in a continuous input stream:

- Σ is a finite *alphabet* which composes *R*;
- Q is a finite set of *NFA states*;
- $\delta : Q \times \Sigma \rightarrow 2^Q$ is the nondeterministic transition function, where 2^Q denotes the *power set* of Q;
- $q_0 \in Q$ is the *initial state* of the finite state control; and
- $F \subseteq Q$ is the set of *final NFA states*.

In addition, an NFA matching multiple regexes $\{R_1, \ldots, R_n\}$ concurrently can be constructed by combining the NFAs $\{\mathcal{M}_i = (Q_i, \Sigma, \delta_i, q_{0,i}, F_i), i = 1 \ldots n\}$ as follows:

1) Combine the initial states $\{q_{0,1}, \ldots, q_{0,n}\}$ of all NFAs into a single initial state q_0.
2) Let $(Q, \delta, F) = (\bigcup_{i=1}^n Q_i, \bigcup_{i=1}^n \delta_i, \bigcup_{i=1}^n F_i)$.

Definition 2: An *DFA-based REM* for regex *R* is a deterministic finite automaton $\mathcal{M}' = (Q', \Sigma, \delta', q_0, F')$ converted from the NFA-based REM for *R* by the standard subset construction [6]:

- $Q' \subseteq 2^Q$ is a finite set of *DFA states*;
- $\delta' : Q' \times \Sigma \rightarrow Q'$ is the deterministic transition function;
- $F' \subseteq Q'$ is the set of *final DFA states*.
- Σ and q_0 are the same as in the original NFA;

B. State-space and State-transition Explosions

The *state explosion problem* caused by the NFA-to-DFA conversion has been the focus of many studies [21, 24, 28]:

- It was shown by construction in [21] that, for every $n > 1$, there is an n-state NFA of binary alphabet whose equivalent DFA has exactly 2^n states.
- The DFA for a real-life regex ($/.*\texttt{AUTH}\backslash\texttt{s}[\hat{}\backslash\texttt{n}]\{\texttt{100}\}/$) was shown in [28] to require exponentially more number of states than the NFA matching the same regex.

- In [24], the state explosion due to combining (*union*-ing) two individual DFAs was explained with state and path ambiguity in the original DFAs.

For the purpose of formalizing our *semi-deterministic finite automaton* (SFA) and the *convolvement analysis*, we offer yet another view of this classic problem. We note that each of these various views of the state explosion problem may represent one facet of the complex nature of finite automata. While our view was useful in formalizing the SFA construction, we make no effort nor claim in giving a unifying treatment of the state explosion problem.

Different from earlier works, however, we divide the state explosion into two types: a *state-space explosion*, and a *state-transition explosion*. In the discussions below, we assume an NFA-based REM architecture constructed by the modified McNaughton-Yamada (MMY) algorithm [27]. The MMY algorithm constructs an NFA which is both uniform and modular. In particular, it is guaranteed that every state in the resulting NFA is associated with (reached via) one labeled transition, whose label can be either a single character value or an arbitrary character class.

1) State-space Explosion: We will use an example to help us understand the cause of the state-space explosion. Suppose there are n non-initial states in an NFA. Let q_0 be the initial state which accepts the *any-value* (wildcard) character class and ϵ-transitions to itself and to q_1; let every other state q_i, $1 \leq i < n$, make an ϵ-transition to q_{i+1} and let q_n be the final state. Also suppose q_1 accepts the character class $[\mathtt{ac}]$ and $q_2 \ldots q_n$ accept the character class $[\mathtt{ab}]$. Literally, the NFA matches the simple regex $/.*[\mathtt{ac}][\mathtt{ab}]\{n-1\}/$.[1] It can be shown that by controlling the input string to be a sequence of alternative a's and b's, *every* possible combination of the n states $\langle q_1, \ldots, q_n \rangle$ can be activated. Thus the subset construction will produce a DFA with at least 2^n states, each of which as a unique vector of n binary (NFA state) values. The exponential state-space explosion can be seen by the fact that adding any state to the NFA (by repeating the character class $[\mathtt{ab}]$ an extra time, for example) increases the number of DFA states by 2^n.

In general, most regex patterns do not suffer from the worst-case state-space explosion because many combinations of $q_1 \ldots q_n$ cannot be valid DFA states. In our example, suppose an extra state is added to the NFA where q_n is *concatenated* by an q_{n+1} which accepts the character class $c_{n+1} = [\mathtt{cd}]$. The resulting NFA matches the regex $/.*[\mathtt{ac}][\mathtt{ab}]\{n-1\}[\mathtt{cd}]/$. Since c_{n+1} is disjoint from $c_2 \ldots c_n$, it is guaranteed that q_{n+1} and $q_2 \ldots q_n$ can never be activated by the same input character, and adding q_{n+1} to the NFA will only increase the number state in the equivalent DFA by 2 (where $\langle q_1, q_2 \ldots, q_{n+1} \rangle = \langle 0, 0 \ldots, 1 \rangle$ or $\langle 1, 0 \ldots, 1 \rangle$) instead of 2^n. More generally, if it can be guaranteed that q_{n+1} and some $q_{i_1} \ldots q_{i_h}$, $h \geq 1$, are never active concurrently, then adding q_{n+1} to the NFA will increase the number state in the equivalent (minimal) DFA by *at most* 2^{n-h} states, greatly reducing the state-space explosion that would have occurred otherwise.

As another example, suppose the state q_{n+1} accepts the character class $c_{n+1} = [\mathtt{a}]$. The resulting NFA now matches $/.*[\mathtt{ac}][\mathtt{ab}]\{n-1\}\mathtt{a}/$. In this case, it can be seen that whenever q_{n+1} is active, so is q_1. Since only $q_2 \ldots q_n$ can have non-trivial combinations with q_{n+1}, adding q_{n+1} increases the number of states in the DFA by 2^{n-1} instead of 2^n. Again, in general, if it can be guaranteed that whenever q_{n+1} is active, some $q_{i_1} \ldots q_{i_h}$, $h \geq 1$, are also active, then adding q_{n+1} to the NFA will also increase the number of states in the equivalent DFA by at most 2^{n-h}.

From the two case examples above, we see that the state-space explosion is caused by states that do *not* have a reduced value set when *conditioned on* the values of some other state. In other words, if $\forall i = 1 \ldots n$, either $\{q_{n+1} | q_i = 1\} \equiv \{0\}$ or $\{q_{n+1} | q_i = 1\} \equiv \{1\}$, then adding q_{n+1} to the NFA with (non-initial) states $\{q_1 \ldots q_n\}$ will induce *no* state-space explosion in the equivalent DFA. On the other hand, state-space explosion can be caused by an q_{n+1} where $\{q_{n+1} | q_i = 1\} \equiv \{0, 1\}$ for some $1 \leq i \leq n$.

2) State-transition Explosion: In our NFA-based REM architecture, every state can accept a character class consisting of one or more value points in the alphabet. This helps to reduce simple (single-character) unions of any $v \geq 1$ states into a single NFA state. For example, with the ability to accept character classes, the regex $/.*[\mathtt{ac}][\mathtt{ab}]\{n-1\}/$ can be matched by an n-state NFA. Without character classes, the regex would have to be rewritten as $/.*(\mathtt{a}|\mathtt{c})(\mathtt{a}|\mathtt{b})\{n-1\}/$ and matched by a $2n$-state NFA. In addition, the number of inter-state ϵ-transitions is also doubled. In general, expanding a character class of size v to individual character values increase the number of NFA states and state transitions by v times.

During the subset construction, various character values will be used to find the next set of active NFA states, a process which could inherently split the labeled transitions (and the corresponding NFA states). Suppose an NFA state $q_i, 1 \leq i \leq n$, accepts the character class $[\mathtt{ab}]$. If q_i is activated together with state $q_j, j \neq i$, when reached via character a, but with another state $q_k, k \neq i$, via the character b, then the labeled transition $[\mathtt{ab}]$ in the original NFA must be split into two distinct state transitions (one for a and one for b) in the equivalent DFA. In the worst case, the number of state transitions in the NFA can be increased by $|\Sigma|$ times (Σ being the alphabet) in the equivalent DFA.

IV. Convolvement Analysis

The two case examples in Section III-B1 hint on the formalism of our convolvement analysis. We use the term "convolvement" to describe the possible values of an NFA state when it is conditioned on the value of another NFA state. To this end, we devise a *convolvement test* to find out the convolvement relations between any pair of states in an

[1] Although this is a highly artificial example, it is not without practical significance. For example, many identification systems have ID formats starting with an English alphabet followed by a fixed number of alphanumeric characters. To match such an ID of length n in a continuous stream of input characters, the regex $/.*[\mathtt{a-z}][\mathtt{a-z0-9}]\{n-1\}/$ will be used, with the exact same pattern as the example.

arbitrary NFA. The convolvement relations are then used to help us construct a minimal SFA.

A. State Convolvement in NFA

Conceptually, the *convolvement* between two NFA states describes the concurrent activeness of the two states for all input sequences. We define three types of convolvement relations: "*conform*" ($\shortparallel$), "*convolve-to*" ($\triangleright$), and "*conflict*" ($\bowtie$).

Definition 3: Let $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ be an NFA. Two states $x, y \in Q$ may have three types of *convolvement*:

- We say x *conforms with* y, or $x \shortparallel y$, *iff*

$$\forall s \in \Sigma^+,\ q_0 \overset{s}{\vdash} x \implies q_0 \overset{s}{\nvdash} y$$

- We say x *convolves to* y, or $x \triangleright y$, *iff*

$$\forall s \in \Sigma^+,\ q_0 \overset{s}{\vdash} x \implies q_0 \overset{s}{\vdash} y$$

- We say x *conflicts with* y, or $x \bowtie y$, *iff*

$$\begin{aligned} \exists s_1, s_2, s_3 \in \Sigma^+ \quad \ni \quad & (q_0 \overset{s_1}{\vdash} x \wedge q_0 \overset{s_1}{\vdash} y) \\ \wedge \quad & (q_0 \overset{s_2}{\vdash} x \wedge q_0 \overset{s_2}{\nvdash} y) \\ \wedge \quad & (q_0 \overset{s_3}{\nvdash} x \wedge q_0 \overset{s_3}{\vdash} y) \end{aligned}$$

where $q_0 \overset{s}{\vdash} q$ means s is a sequence of labels (input characters) on some transition path from q_0 to q.
Recall that starting from q_0, the set of sequences of input characters that reach (activate) any NFA state form a *regular set* [6]. In other words, each $q \in Q$ can be used to define a regular set $\mathcal{L}(q) \triangleq \{s \in \Sigma^+ \mid q_0 \overset{s}{\vdash} q\}$. Thus Definition 3 basically says that, between any pair of NFA states $x, y \in Q$, (1) $x \shortparallel y$ when the regular sets defined by x and y have empty intersection; (2) $x \triangleright y$ when the regular set defined by x is a subset of the regular set defined by y; and (3) $x \bowtie y$ when the regular sets defined by x and y have non-empty intersection, while neither is a subset of the other.

Note that a state always convolves to itself ($\forall x, x \triangleright x$). We can derive the following equations from the convolvement relations:

$$x \triangleright y \wedge y \triangleright z \implies x \triangleright z \tag{1}$$

$$x \triangleright y \wedge y \shortparallel z \implies x \shortparallel z \tag{2}$$

The convolvement relation between any pair of states $x, y \in Q$ in $\mathcal{M}$ can be determined by Algorithm 1.

Algorithm 1 RE-NFA State Convolvement Test

Input: NFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$, $x, y \in Q \backslash \{q_0\}$.
Output: Convolvement relation between x and y.

1) Create a "reverse" NFA of $\mathcal{M}$ with respect to x, $\hat{\mathcal{M}}^x = (Q, \Sigma, \hat{\delta}, x, \{q_0\})$, where
   a) $\hat{\delta}$ is the "reverse" of δ:
   $$\forall q_i, q_j \in Q, \sigma \in \Sigma,\ \delta(q_i, \sigma) = q_j \Rightarrow \hat{\delta}(q_j, \sigma) = q_i$$
   b) x is the initial state of $\hat{\mathcal{M}}^x$; and
   c) $\{q_0\}$ is the (singleton) set of final state of $\hat{\mathcal{M}}^x$.
2) Create a reverse NFA $\hat{\mathcal{M}}^y = (Q, \Sigma, \hat{\delta}, y, \{q_0\})$ with respect to y in a similar manner.
3) Initialize two *convolvement registers*, $r_x = 0, r_y = 0$.
4) Do breadth-first traversals in $\hat{\mathcal{M}}^x$ and $\hat{\mathcal{M}}^y$ from states x and y, respectively. Let the *traversal depth* d be the number of labeled transitions traversed.
5) For each $d > 0$ before both $\hat{\mathcal{M}}^x$ and $\hat{\mathcal{M}}^y$ reach q_0:
   a) Let $\{(\mathcal{K}_i^x, \mathcal{K}_i^y), i = 1 \ldots t\}$ be t pairs of state subsets of Q where $\mathcal{K}_i^x$ and $\mathcal{K}_i^y$ are reached concurrently by the same input sequence $s \in \Sigma^d$ from x and y, respectively, using the reverse transition function $\hat{\delta}$:
   $$\exists s \in \Sigma^d,\ x \underset{\hat{\delta}}{\overset{s}{\vdash}} \mathcal{K}_i^x \Longleftrightarrow y \underset{\hat{\delta}}{\overset{s}{\vdash}} \mathcal{K}_i^y.$$
   If no such state subset pair exists (*i.e.*, $t = 0$), then x *conform with* y. *Fin.*
   b) $\forall i = 1 \ldots t$, let Λ_i^x and Λ_i^y be the set of transition labels from $\mathcal{K}_i^x$and $\mathcal{K}_i^x$, respectively, through the reverse transition function $\hat{\delta}$:
   $$\Lambda_i^x = \{\alpha \in \Sigma \mid \exists q \in \mathcal{K}_i^x \ni \hat{\delta}(q, \alpha) \neq \varnothing\}$$
   $$\Lambda_i^y = \{\beta \in \Sigma \mid \exists q \in \mathcal{K}_i^y \ni \hat{\delta}(q, \beta) \neq \varnothing\}$$
   If $\Lambda_i^x \cap \Lambda_i^y \neq \varnothing$, then:
      i) If $\Lambda_i^x \nsubseteq \Lambda_i^y$, then set $r_x = 1$ ($x \ntriangleright y$).
      ii) If $\Lambda_i^y \nsubseteq \Lambda_i^x$, then set $r_y = 1$ ($y \ntriangleright x$).
   c) If $r_x \cdot r_y = 1$, then x *conflict with* y. *Fin.*
   d) Else, increment d by 1 and repeat Step 5.
6) Both traversals in $\hat{M}^x$ and $\hat{M}^y$ reach q_0, and $r_x \cdot r_y = 0$:
   a) If $r_x = 1$, then y *convolves to* x. *Fin.*
   b) If $r_y = 1$, then x *convolves to* y. *Fin.*
   c) If $r_x = r_y = 0$, then x and y are equivalent. *Fin.*

B. State Convolvement vs. State-space Explosion

We will now associate the state convolvements in an NFA with the state-space explosion in its equivalent DFA. To simplify the discussion, we define the concept of *neighbor set* of states in the NFA.

Definition 4: Let $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ be an NFA. The *neighbor set* $\mathcal{N}_s \in 2^Q$ of $\mathcal{M}$ for any sequence of characters $s \in \Sigma^+$ is the largest set of NFA states reached concurrently from q_0 after consuming s as input. In other words,

$$\forall s \in \Sigma^+,\ \mathcal{N}_s \triangleq \max\{q \in Q \mid q_0 \overset{s}{\vdash} q\}.$$

In simple terms, we can regard each neighbor set as a valid combination of NFA states which are activated concurrently by the same input string.
The subset construction algorithm basically converts an NFA $\mathcal{M}$ to a DFA $\mathcal{M}'$ by finding the complete set of neighbor sets for all possible sequences of input characters. There is thus a one-to-one mapping between the neighbor sets $\{\mathcal{N}_s | s \in \Sigma^+\}$ of $\mathcal{M}$ and the set of states Q' of $\mathcal{M}'$.

Theorem 1: When converting an NFA $\mathcal{M}$ to its equivalent DFA $\mathcal{M}'$, there is state-space explosion in $\mathcal{M}'$ if and only if

there are state conflict in $\mathcal{M}$.

Proof: We will prove the proposition in three parts: (1) having no convolvement in $\mathcal{M}$ leads to no state-space explosion in $\mathcal{M}'$; (2) having only the "convolve-to" type of convolvements in $\mathcal{M}$ also does not incur state-space explosion in $\mathcal{M}'$; and (3) having the "conflict" type of convolvements in $\mathcal{M}$ incurs state-space explosion in $\mathcal{M}'$.

First, we assume there is no state convolvement in $\mathcal{M}$ other than with the initial state. In other words, every pair of non-initial states in $\mathcal{M}$ conform with each other. Since no two states of M can be active concurrently, each neighbor set of $\mathcal{M}$ consists of only one state in Q, and $\mathcal{M}'$ is free of state-space explosion.

Second, we assume all the convolvements in $\mathcal{M}$ are convolve-to relations. Due to the property in Eqs. 1 and 2, the convolve-to relations can be used to organize all states in $\mathcal{M}$ (together with the initial state) in a multi-dimensional tree, where every neighbor set corresponds to a single (partial) path, with respect to the reversed convolve-to relation, from the tree root (which is also the initial state in $\mathcal{M}$). Since there are exactly N partial paths in a tree of N nodes, the number of neighbor sets and thus the number of states in the equivalent DFA is the same as the number of states in the NFA. No state-space explosion occurs.[2]

Third, we assume there are conflict relations between every pair of states in $Q^{(h)} \subset Q$, where $|Q^{(h)}| = h > 1$. We will bound the number of possible neighbor sets caused by the number (h) of *pair-wise conflicting* states in the NFA. Assume $h = 2$ in the simplest case, the two conflicting states $q_{i_1}, q_{i_2} \in Q^{(2)}$ will generate exactly three neighbor sets, $\langle q_{i_1} \rangle$, $\langle q_{i_2} \rangle$ and $\langle q_{i_1}, q_{i_2} \rangle$. Increasing h to 3 by adding an extra state q_{i_3} conflicting with both q_{i_1} and q_{i_2} will produce at least three and at most four extra neighbor sets. The three extra neighbor sets are $\langle q_3 \rangle$, $\langle q_{i_1}, q_{i_3} \rangle$ and $\langle q_{i_2}, q_{i_3} \rangle$, caused by the conflict relations between q_{i_3} and both q_{i_1} and q_{i_2}. Depending on whether q_{i_3} also conflicts with $\langle q_{i_1}, q_{i_2} \rangle$ (the neighbor set constructed by q_{i_1} and q_{i_2}), an additional neighbor set, $\langle q_{i_1}, q_{i_2}, q_{i_3} \rangle$, may be added.

In general, if there are h pair-wise conflicting states producing $\mathcal{H}$ neighbor sets in $Q^{(h)}$, then adding another state $q_{i_{h+1}}$ conflicting to all of $\{q_{i_1}, \ldots, q_{i_h}\} \in Q^{(h)}$ can produce at least $h + 1$ extra neighbor sets due to the *known* conflict relations between $q_{i_{h+1}}$ and $q_{i_1} \ldots q_{i_h}$. On the other hand, adding $q_{i_{h+1}}$ can produce at most $\mathcal{H} + 1$ extra neighbor sets due to the *potential* conflict relations between $q_{i_{h+1}}$ and the $\mathcal{H}$ valid combinations (neighbor sets) of $q_{i_1} \ldots q_{i_h}$.

Thus we can calculate the minimum number of neighbor sets in $Q^{(h)}$ as $3 + (2 + 1) + (3 + 1) + \ldots + h = \frac{h \times (h+1)}{2}$, whereas the maximum number of neighbor sets in $Q^{(h)}$ as $3 + (3 + 1) + (7 + 1) + \ldots + 2^{h-1} = 2^h - 1$. ■

It is interesting that our quantification of the state-space explosion through the convolvement analysis coincides with the "quadratic to exponential explosion" derived in [28] through case studies . Our theory shows that the amount of state-space explosion when converting an NFA to an equivalent DFA is indeed bounded between quadratic and exponential in the number of *pair-wise conflicting* NFA states.

[2] Dictionary-based string matching using the Aho-Corasick algorithm [5] is a canonical example of this scenario.

V. Semi-deterministic Finite Automata

A. The SFA Architecture

Definition 5: A *semi-deterministic finite automaton* (SFA) *with* p *constituents*, $\mathcal{M}^{(p)}$, is a collection of p *constituent DFA*s (c-DFA), $\mathcal{M}_i = (Q_i, \Sigma, \delta_i^{(p)}, q_{0,i}, F_i, \pi_i)$, $i = 1 \ldots p$, such that

1) Q_i is the set of *states* of the i-th c-DFA;
2) Σ is the common finite *alphabet*;
3) $\delta_i^{(p)} : Q_i \times \Sigma \to (Q_1, \ldots, Q_p)$ defines the *constituent transition function* for the i-th c-DFA;
4) $q_{0,i} \in Q_i$ is the *initial state* in the i-th c-DFA;
5) $F_i \subseteq Q_i$ is the set of *final states* in the i-th c-DFA; and
6) $\pi_i : Q_i \to \mathbb{Z}$ is the *state priority function* for the i-th c-DFA.

Basically, an SFA is $p > 0$ c-DFAs running collaboratively and in parallel. Each c-DFA has a constituent transition function ($\delta_i^{(p)}$) that is locally deterministic but globally nondeterministic: At any state in the c-DFA, an input character can induce up to one *local transition* to another state in the current c-DFA, and up to $p - 1$ *global transitions* to one state in each of the other $(p - 1)$ c-DFAs.

Because a c-DFA is by itself deterministic, at any time only one state in each c-DFA can be active (although multiple c-DFAs in the SFA can each have one active state concurrently). With Definition 5 alone, however, it is possible for two transitions from c-DFA $\mathcal{M}_i$ and $\mathcal{M}_j$, $i \neq j$, to activate two states in the same c-DFA $\mathcal{M}_k$ concurrently. In order to maintain the deterministic property of each c-DFA, however, the following condition must always be observed.

Condition 1: The transitions in an SFA must respect either one of the following rules:

1) It is not possible for two transitions to activate different states in a single c-DFA concurrently.
2) If two transitions activate different states in a single c-DFA concurrently, the two states must have different priorities, and the lower-priority state is ignored.

The priorities of the states in each c-DFA are specified by the state priority function $\pi_i(q)$ in Definition 5.

Both NFA and DFA are special cases of SFA. An NFA with p states can be regarded as an SFA with p c-DFAs, each of which contains only the initial state plus one state from the NFA. A DFA with n states can be regarded as an SFA with one c-DFA which contains all n states in the DFA. Conversely, SFA can be seen as a tradeoff between the compute-intensive NFA and the space-intensive DFA.

B. Minimal SFA Construction

As shown in Theory 1 (Section III-B), state conflicts in an NFA $\mathcal{M}$ cause state-space explosion in the equivalent DFA $\mathcal{M}'$. One way to *avoid* the state-space explosion is to construct a "multi-constituent" finite automaton where conflicting states

in the NFA are separated into different constituents. If each constituent is then converted to a DFA, then no state-space explosion would occur. A *minimal* SFA is such a collection of constituent DFAs where the number of constituents is minimized for a given NFA.

Definition 6: An SFA $\mathcal{M}^{(p)}$ is *minimal* for an NFA $\mathcal{M}$ if and only if

1) $\mathcal{M}^{(p)}$ and $\mathcal{M}$ are equivalent (accept the same language).
2) Total number of non-initial states in $\mathcal{M}^{(p)}$ is the same as that in $\mathcal{M}$ (no state-space explosion).
3) $\mathcal{M}^{(p)}$ has the least number of c-DFAs (p is minimal).

Three steps need to be performed to convert an NFA into a minimal SFA. First, the NFA states must be put into *compatible groups*, where the states within each group can only conform with or convolve to each other. This is called the *compatible state grouping* and can be modeled as the classic vertex coloring problem on a graph where each NFA state is a vertex and each state conflict relation is an edge.

Second, the states in each compatible group will be used to construct a *constituent DFA* (c-DFA). Constituent transitions from any c-DFA state will be assigned the same sets of target states as the nondeterministic transitions from the corresponding NFA state. By construction, if two states in the same c-DFA are activated concurrently, then one state must convolve to the other. Suppose the two states are x and y, where $x \triangleright y$. We will assign to x a higher priority than y, since x being active implies a more specific condition than y being active.

Third, for each pair of states $x \triangleright y$ in a c-DFA, if y has an output transition to another c-DFA, then the output transition must be copied to x if x does not also transition to a higher-priority state in the particular c-DFA.

Together, the three steps above ensure that Condition 1 in Section V-A is observed. A formal description of the minimal SFA construction is given below.

Definition 7: For a given NFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$, a *minimal SFA for* $\mathcal{M}$, $\mathcal{M}^{(p)} \triangleq \{\mathcal{M}_1, \ldots, \mathcal{M}_p\}$ where $\forall i = 1 \ldots p : \mathcal{M}_i = (Q_i, \Sigma, \delta_i^{(p)}, q_{0,i}, F_i, \pi_i)$, can be constructed using the following procedure:

1) Perform compatible state grouping for $\mathcal{M}$
   a) $\forall x, y \in Q \backslash \{q_0\}$, find the compatibility between x and y using Algorithm 1.
   b) Create state conflict graph $G_\mathcal{M} = \{V_\mathcal{M}, E_\mathcal{M}\}$:
      i) Every $q \in Q$ is mapped to a $v_q \in V_\mathcal{M}$.
      ii) Every "conflict" between $x, y \in Q \backslash \{q_0\}$ is mapped to an edge $e_{a,b} = \{v_x, v_y\} \in E_\mathcal{M}$.
   c) Perform vertex coloring on $G_\mathcal{M}$, partitioning $V_\mathcal{M}$ into subsets $V_{\mathcal{M},1}, \ldots, V_{\mathcal{M},p}$ (p colors).
2) Construct p c-DFAs, $\mathcal{M}_i = (Q_i, \Sigma, \delta_i^{(p)}, q_{0,i}, F_i, \pi_i)$, $i = 1 \ldots p$, such that
   a) $Q_i = \{q \in Q \mid v_q \in V_{\mathcal{M},i}\}$;
      $F_i = \{q \in F \mid v_q \in V_{\mathcal{M},i}\}$.
   b) $\forall q \in Q_i$ and $\sigma \in \Sigma$,
   $$\delta_i^{(p)}(q, \sigma) \triangleq \{x \in \delta(q, \sigma) \mid \forall y \in \delta(q, \sigma) : x \triangleright y\}$$
   c) $q_{0,i} \in Q_i$ is a *copy* of q_0.[3]
3) $\forall i = 1 \ldots p$, $\pi_i(\cdot)$ is defined as follows:
   a) $\pi_i(q_{0,i}) = 0$ (lowest priority for initial states)
   b) $\forall q \in Q_i$, $\pi_i(q) = 1$ *iff* $\forall t \in Q_i \backslash \{q_{0,i}\}, q \not\triangleright t$.
   c) $\forall x, y \in Q_i$, $x \triangleright y \Rightarrow \pi_i(x) > \pi_i(y)$.
4) $\forall i = 1 \ldots p$, update $\delta_i^{(p)}$ as follows:
   $$\forall q \in Q_i, \sigma \in \Sigma,\ \delta_i^{(p)}(q, \sigma) = \bigsqcup_{q \triangleright t} \delta_i^{(p)}(t, \sigma)$$
   where $\forall x, y \in Q_i$, $\pi_i(x) > \pi_i(y) \implies x \sqcup y = x$.

Essentially, the minimal SFA construction exploits the high-order properties (*i.e.*, state compatibility) of an NFA to reduce the computation complexity of regular expression matching (REM) from processing a large number of NFA states individually to processing potentially much fewer c-DFAs in parallel. The compatible state grouping ensures that (1) no state-space explosion will occur in any c-DFA and (2) the number of c-DFAs is minimized.

In the best case where there is no conflict between any pair of states in the NFA, the minimal SFA construction will result in a single c-DFA with the same number of states as the NFA. For example, it can be shown that in the case of dictionary-based string matching (DBSM) problem,[4] the minimal SFA construction will reduce to the Aho-Corasick algorithm, where each "convolve-to" relation in the NFA translates to a "failure function" in the Aho-Corasick DFA [5].

In the worst case where there are conflicts between all pairs of states in the NFA, the minimal SFA construction will result in an SFA with the same number of c-DFAs as the number of states in the NFA. Since every state *can* be active both *with* and *without* any other state in the NFA, an NFA is the only architecture that can fully describe the automaton without introducing additional states.

The minimal SFA construction described in this section will fall in the middle, where the number of conflicts is relatively few compared with the number of NFA states, but still large enough to cause serious state-space explosion.

VI. SFA Performance Analysis

A. Execution Models of SFA

To understand the performance implication and the space-time tradeoff of SFA, we will first look at SFA's execution model. An SFA is globally partitioned into multiple constituents running in parallel like an NFA (as in Figure 1(a)), while each constituent is locally supported by a constituent state transition table (c-STT) and functions like a DFA (as in Figure 1(b)).

Upon receiving each input character, a c-DFA makes one access to its local c-STT to find a output transition vector of

[3] This is only to satisfy the formalism of $\mathcal{M}_i$ being a DFA. In practice, $q_{0,i}$ is always ignored (due to its lowest priority) if another state in $\mathcal{M}_i$ is active.

[4] Continuously matching a string "$\alpha_1 \ldots \alpha_1$" of length $l > 0$ against a stream of input characters can be formulated as matching the regex $/.*\alpha_1 \ldots \alpha_1/$ against the same input. Thus DBSM can be modeled as REM.

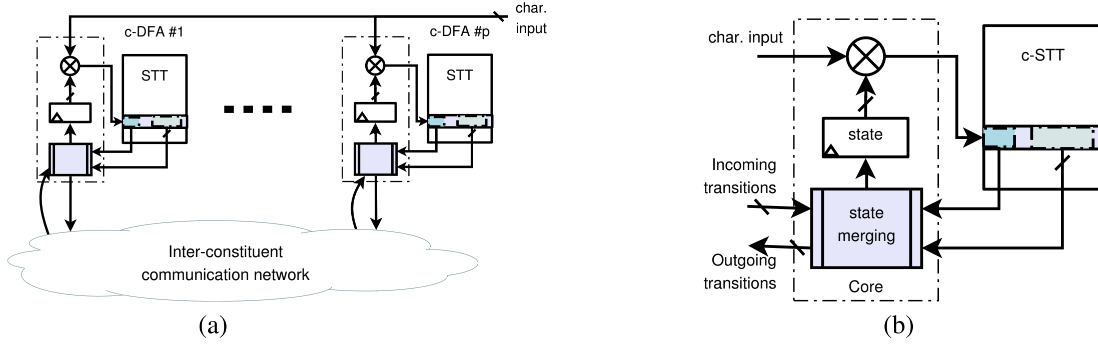

Fig. 1. Execution model of the (a) SFA, and the (b) c-DFA.

length p, where p is the number of constituents in the SFA. Each member in the vector corresponds to an output transition to one target c-DFA. Then the non-zero members in the vector are sent to the corresponding target c-DFAs via the inter-constituent communication network. In the worst case, the inter-constituent communication cost is $p \times (p-1)$, although usually most of the values in the output transition vectors will be zero.

Each c-DFA will then collect the up to p incoming output transitions and "merge" them by keeping only the one that activates the highest-priority state. Note that the state priority information can be calculated statically. For example, a higher priority state can always be assigned a greater state number than a lower-priority state. In this case, the merging becomes a simple `MAX` operation. Alternatively, special bit-level encoding can be used to transform the state merging into bit-wise `OR` operations. The worst-case computation cost across all c-DFAs is $O(p^2)$, assuming every c-DFA makes an output transition to every other c-DFA.

The memory requirement of a minimal SFA can be calculated as follows. Since there is no space-state explosion during the construction of a minimal SFA, the total number of states will be equal to the number of states in the original NFA (n). However, each output transition is now a p-value vector (for every possible input character). Thus for a fixed alphabet Σ, the memory required by all c-STTs in the SFA is $|\Sigma| \times pn = O(pn)$.

B. *Combining C-DFAs for Space-Time Tradeoff*

We note that for a given set of regexes, p (the number of c-DFAs in the minimal SFA) is a fixed value. In other words, there is *nothing* we can do to reduce the value of p in a minimal SFA without experiencing some degree of state-space explosion. It turns out that with the SFA architecture, however, we can easily tradeoff the memory efficiency of the minimal SFA for reduced time complexity in a "partially exploded" SFA.

Suppose we have a minimal SFA $\mathcal{M}^{(p)}$ with p c-DFAs, $\mathcal{M}_i,\ i = 1 \ldots p$. The SFA has a worst-case computation complexity $O(p^2)$, which is the best we can achieve without suffering from any state-space explosion. To reduce this worst-case complexity, we will *combine* two smallest c-DFAs into one. This can be done by taking the product of the state sets of the two c-DFAs. For example, assume without loss of generality the c-DFA $\mathcal{M}_i$ has n_i states and $n_1 \leq \ldots \leq n_p$. To remove one c-DFA from the SFA, we can combine $\mathcal{M}_1$ and $\mathcal{M}_2$ into an $\mathcal{M}_{1,2}$ with $n_1 \times n_2$ states. To remove two c-DFAs from the SFA, we can combine $\mathcal{M}_1$ and $\mathcal{M}_4$ into an $\mathcal{M}_{1,4}$ with $n_1 \times n_4$, as well as $\mathcal{M}_2$ and $\mathcal{M}_3$ into an $\mathcal{M}_{2,3}$ with $n_2 \times n_3$ states. Alternatively, we can also combine $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$ into an $\mathcal{M}_{1,2,3}$ with $n_1 \times n_2 \times n_3$ states. Both ways of combining will produce the same number of (combined) c-DFAs. Which one is more favorable depends on the relative sizes of $(n_1 \times n_4 + n_2 \times n_3)$ and $(n_1 \times n_2 \times n_3 + n_4)$. This combining process can be performed continuously until a satisfactory space-time tradeoff point is reached.

To analytically derive the effect on space-time tradeoff by the c-DFA combination process, assume we reduced the number of c-DFAs from p in the minimal SFA $\mathcal{M}^{(p)}$ to p/c in the resulting (partially exploded) SFA $\mathcal{M}^{(p,c)}$. With only p/c constituents, $\mathcal{M}^{(p,c)}$ will have worst-case computation complexity $O(p^2/c^2)$. On the other hand, assume on average each c-DFA in the minimal SFA consists of $O(n/p)$ states, then combining c of such c-DFAs into a product c-DFA will result in $O((n/p)^c)$ combined states per combined c-DFA. Recall that each state requires $|\Sigma| \times p$ memory space to store all the possible output transitions. Thus the total memory required by all (p/c) combined c-DFAs is $O(|\Sigma| \times p \times \frac{p}{c} \times (\frac{n}{p})^c) \approx O(p^2 \times (\frac{n}{p})^c)$ for a fixed Σ and a relatively small (design) constant c.

C. *Performance Optimizations*

It can be shown that in the worst case, each input character can induce up to $p(p-1)$ state transitions crossing the constituents of an SFA. Depending on the interconnect architecture of the underlying system, the large number of inter-constituent messages can proportionally increase the latency of the state transitions. Below we outline two optimization techniques to reduce the inter-constituent communication and to mitigate the overhead to synchronization various c-DFAs.

1) Pattern-directed Staging: We define *staging* as the process of partitioning an SFA into multiple *stages*, each of which is as a subset of "closely-related" c-DFAs in the SFA.

Within each stages, various c-DFAs are allowed to make inter-constituent transitions just like in a non-staged SFA. Across stages, however, state transitions can only go from a lower stage to a higher stage, but not the other way. Such staging eliminates the synchronization overhead between any two c-DFAs in two different stages, as long as the higher stage does not run ahead of the lower one (for example, if all stages are processed in a pipelined fashion).

A simple way of staging is to partition the c-DFAs along regex boundaries. Since state transitions always occur within each regular expression independently, this approach allows various stages to proceed alongside one another without interaction. On the other hand, it can also proportionally increase the number of constituents in the SFA. Conceptually, it is similar to the pattern grouping proposed in [28].

More sophisticated match staging can be performed by examining the "convolve-to" relations (in addition to the state conflicts) during compatible state grouping. More specifically, two states will be preferred in the same stage (although possibly different c-DFAs) if they convolve to the same third state. Note that simply cutting regular expressions in halves (as in [7, 17]) does not generate valid c-DFAs and thus cannot be used for match staging the SFA.

2) Constituent DFA Optimization: As shown in Figure 1(b), other than the "stage merging" feature, each c-DFA has the same architecture as a memory-based DFA. Thus conventional DFA optimizations, including both state space minimization and state transition table compression, can also be applied to the c-DFA. Such optimizations can be particular useful for the combined c-DFAs, where taking the product of the state sets of two c-DFAs directly can result in many redundant states.

VII. Prototype Implementation and Evaluation

To verify the proposed SFA architecture, we compiled a small set of relatively complex regexes extracted from various Snort [4] rulesets into SFAs. Table I lists the regexes used for our tests. We produced a series of 6 SFAs, each with a different number of regexes (from 2 to 12) and constituents (from 6 to 10), as shown in Table II. Each larger SFA is a superset of the smaller SFAs. This allows us to evaluate the performance of SFA with respect to number of c-DFAs as well as matched regexes.

We implemented the SFAs on a dual-core x86-64 PC with AMD Athlon64 X2 processor and 2 GB DDR-333 memory. The program is dual-threaded to match the dual-core platform it runs on. Due to the few number of cores, multiple constituents in an SFA are statically mapped to one core. While this may have resulted in some load unbalance during run time, we note that *our purpose is* not *to produce the highest performance implementation of SFA, but to verify its functional correctness and demonstrate the performance characteristics*.

To reduce inter-constituent communication overhead, we divided all c-DFAs into two stages: an *initial stage* and a *final stage*, with the following guidelines:

- All c-DFAs in the initial stage are assigned to core 0. All c-DFAs in the final stage are assigned to core 1.

TABLE I
List of regexes compiled into the 10-constituent SFA

Rule #	Pattern of Regular Expression (RE)
109	^NetBus\s+\d+\x2E\d+
147	^GateCrasher\s+v\d+\x2E\d+\x2C\s+Server\s+On-Line\x2E\x2E\x2E
6036	minicommand\s+fileserver\s+ready\.\r\n
6048	^100013Agentsvr\x5E\x5EMerlin
1778	^STAT\s+[^\n]*\x3f
2597	^Authorization\x3a(\s*|\s*\r?\n\s+)Basic\s+=
553	^USER\s+(anonymous|ftp)[^\w]*[\r\n]
11192	^Content-Type\x3a[\x20\x09]+application\/octet-stream
12593	-chrome\s*javascript
13246	\x3Ctitle\x3ETroya\s+\x2D\s+by\s+Sma\s+Soft\x3C\x2Ftitle\x3E
12044	Transfer-Encoding\x3a\s*chunked.*\n\r?\n
10453	DivXProGainBundle\s+registration

TABLE II
Basic SFA statistics for various sets of regexes.

# regexes	# states	# (initial+final) c-DFAs
2	41	(2 + 4) = 6
4	96	(3 + 5) = 8
6	131	(3 + 6) = 9
8	190	(3 + 7) = 10
10	244	(3 + 7) = 10
12	308	(3 + 7) = 10

- Global state transitions are allowed from one c-DFA to another in the same stage, or from an initial-stage c-DFA to a final-stage c-DFA.
- Global state transitions are *not* allowed from a final-stage c-DFA to an initial-stage c-DFA.

With this arrangement, messages of the inter-constituent state transitions are only sent from core 0 to core 1, making the two cores implicitly synchronized as long as core 1 does not run ahead of core 0.

Note also the manual SFA construction that we used for the tests does *not* produce minimal SFAs. For simplicity, we first constructed a close-to-minimal SFA with 10 c-DFAs for the entire set of 12 regexes (where 3 belong to the initial stage and 7 to the final stage). Then we removed 2 regexes at a time to produce each of the smaller SFAs. A c-DFA is only removed after all states inside it are removed. Sub-minimality can occur when multiple c-DFAs in one of the smaller SFAs could have been merged into fewer c-DFAs without inducing any state-space explosion.

We compared the throughput performance of our SFA prototype with the equivalent NFA-based REM by Perl-Compatible Regular Expression (PCRE) [3]. Figure 2 shows the comparison results. For fair comparison, we executed two copies of PCRE on the dual-core system and aggregated their throughput. For the input stream, we concatenated all the minimum-length strings generated by each set of regexes. A string generated by an RE is said to be "minimum-length" when the pattern enclosed in each Kleene closure is repeated only once. Such an input stream would result in a throughput performance close to the worst case for both SFA and NFA.

As shown in Figure 2, the SFA-based REM consistently outperforms the NFA-based REM in terms of matching through-

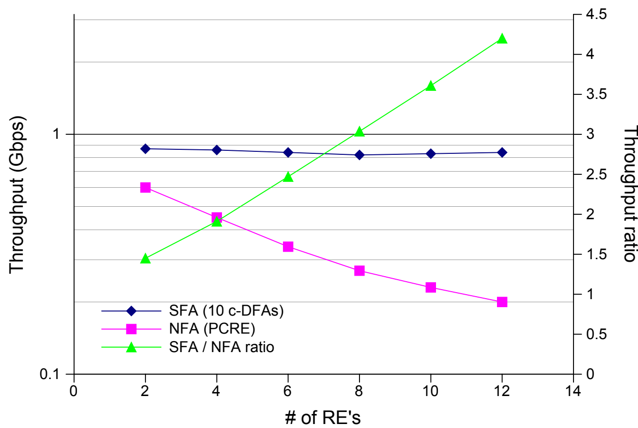

Fig. 2. Worst-case throughput of SFA vs. NFA matching various numbers of regexes concurrently.

put. Two points from the graph worth further discussion:

1) The more regexes are matched concurrently, the greater advantage SFA enjoys over NFA. This should not be surprising since the number of c-DFAs in the SFA increases at a much slower rate than the number of states in the NFA. Specifically, in an SFA, if state x convolves to state y, then only one state needs to be set active at run time. In the equivalent NFA, however, both states must be set active and checked at run time.
2) The performance of SFA hardly reduces from matching a set of 2 regexes to a set of 12 regexes. This is likely due to the non-minimal SFA we constructed manually. In fact, all the three SFAs for the 8-, 10-, and 12-regex sets have the same number of c-DFAs, and thus almost the same performance.

Alternatively, compiling all 12 regexes into a single DFA would have produced over a million states due to the relatively large number (>20) of overlapping Kleene closures .

VIII. Conclusion

The semi-deterministic finite automaton (SFA) offers an effective tradeoff between the computation complexity of NFA and the space complexity of DFA for regular expression matching (REM). We introduced a novel convolvement analysis for REM, which allowed us to identified and quantify the state-space explosion when converting an NFA to a DFA. The SFA architecture is globally similar to the NFA while locally similar to a DFA. An SFA-based REM solution can be mapped onto a multi-core execution model. The throughput advantage of SFA over NFA increases proportionally with respect to the number of regular expressions matched concurrently. On the other hand, state-space explosion of the DFA can be either completely avoided or controlled with a design constant.

References

[1] Bro Intrusion Detection System. http://bro-ids.org.
[2] Clam AntiVirus. http://www.clamav.net/.
[3] PCRE: Perl Compatible Regular Expression. http://www.pcre.org/.
[4] SNORT. http://www.snort.org/.
[5] Alfred V. Aho and Margaret J. Corasick. Efficient String Matching: An Aid to Bibliographic Search. *Communications of the ACM*, 18(6):333–340, 1975.
[6] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall, Inc., 1972.
[7] Michela Becchi and Patrick Crowley. A Hybrid Finite Automaton for Practical Deep Packet Inspection. In *ACM CoNEXT*, pages 1–12, 2007.
[8] Michela Becchi and Patrick Crowley. An Improved Algorithm to Accelerate Regular Expression Evaluation. In *Proc. of Sym. on Archit. for Netw. and Comm. Systems (ANCS)*, pages 145–154, 2007.
[9] Michela Becchi, Charlie Wiseman, and Patrick Crowley. Evaluating Regular Expression Matching Engines on Network and General Purpose Processors. In *Proc. of Sym. on Archit. for Netw. and Comm. Systems (ANCS)*, 2009.
[10] João Bispo, Ioannis Sourdis, João M. P. Cardoso, and Stamatis Vassiliadis. Regular expression matching for reconfigurable packet inspection. In *Proc. of IEEE Intl. Conf. on Field Programmable Technology (FPT)*, pages 119–126, December 2006.
[11] J.A. Brzozowski. Derivatives of Regular Expressions. *J. ACM*, 11:481–494, 1964.
[12] R. W. Floyd and J. D. Ullman. The Compilation of Regular Expressions into Integrated Circuits. *Journal of ACM*, 29(3):603–622, 1982.
[13] V.M. Glushkov. The Abstract Theory of Automata. *Russian Math. Surveys*, 16:1–53, 1961.
[14] J. Grosch. Efficient generation of lexical analyzers. *Software–Practice & Experience*, 19(11):1089–1103, 1989.
[15] John Hopcroft. An n log n algorithm for minimizing states in a finite automaton. Technical report, Stanford University, 1971.
[16] B. L. Hutchings, R. Franklin, and D. Carver. Assisting Network Intrusion Detection with Reconfigurable Hardware. In *Proc. of Sym. on Field-Programmable Custom Computing Machines*, page 111, 2002.
[17] Sailesh Kumar, Balakrishnan Chandrasekaran, Jonathan Turner, and George Varghese. Curing Regular Expressions Matching Algorithms from Insomnia, Amnesia, and Acalculia. In *Proc. of Sym. on Archit. for Netw. and Comm. Systems (ANCS)*, pages 155–164, 2007.
[18] Sailesh Kumar, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, and Jonathan Turner. Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection. *SIGCOMM Computer Communication Review*, 36(4):339–350, 2006.
[19] Tsern-Huei Lee. Generalized aho-corasick algorithm for signature based anti-virus applications. In *Proc. Intl. Conf. on Computer Communications and Networks (ICCCN)*, 2007.
[20] Cheng-Hung Lin, Chih-Tsun Huang, Chang-Ping Jiang, and Shih-Chieh Chang. Optimization of Regular Expression Pattern Matching Circuits on FPGA. In *Proc. of Conference on Design, Automation and Test in Europe (DATE)*, pages 12–17, 2006.
[21] A. R. Meyer and M. J. Fischer. Economy of description by automata, grammars, and formal systems. In *Proc. of Sym. on Switching and Automata Theory (SWAT '71)*, pages 188–191, Washington, DC, USA, 1971. IEEE Computer Society.
[22] Davide Pasetto, Fabrizio Petrini, and Virat Agarwal. Tools for very fast regular expression matching. *Computer*, 43:50–58, 2010.
[23] R. Sidhu and V.K. Prasanna. Fast Regular Expression Matching Using FPGAs. In *Proc. of IEEE Sym. on Field-Programmable Custom Computing Machines*, pages 227–238, 2001.
[24] Randy Smith, Cristian Estan, and Somesh Jha Shijin Kong. Deflating the Big Bang: Fast and Scalable Deep Packet Inspection with Extended Finite Automata. In *ACM SIGCOMM*, August 2008.
[25] Norio Yamagaki, Reetinder Sidhu, and Satoshi Kamiya. High-Speed Regular Expression Matching Engine Using Multi-Character NFA. In *Proc. of Intl. Conf. on Field Programmable Logic and Applications (FPL)*, pages 697–701, Aug. 2008.
[26] Yi-Hua E. Yang, Weirong Jiang, and Viktor K. Prasanna. Compact Architecture for High-Throughput Regular Expression Matching on FPGA. In *Proc. of Sym. on Archit. for Netw. and Comm. Systems (ANCS)*, November 2008.
[27] Yi-Hua E. Yang and Viktor K. Prasanna. Software Toolchain for Large-Scale RE-NFA Construction on FPGA. *Intl. Journal of Reconfigurable Computing*, 2009:10, 2009.
[28] Fang Yu, Zhifeng Chen, Yanlei Diao, T. V. Lakshman, and Randy H. Katz. Fast and Memory-Efficient Regular Expression Matching for Deep Packet Inspection. In *Proc. of Sym. on Archit. for Netw. and Comm. Systems (ANCS)*, pages 93–102, 2006.