Deterministic Finite Automaton for Scalable Traffic Identification: the Power of Compressing by Range

Rafael Antonello, Stenio Fernandes, Djamel Sadok, Judith Kelner
Federal University of Pernambuco (UFPE)
Recife, Brazil

Géza Szabo
Ericsson Traffic Lab
Budapest, Hungary

Abstract— **Deep Packet Inspection (DPI) systems have been becoming an important element in traffic measurement ever since port-based classification was deemed no longer appropriate, due to protocol tunneling and misuses of well-defined ports. Current DPI systems express application signatures using regular expressions and it is usual to perform pattern matching through the use of Finite Automaton (FA). Although DPI systems are essentially more accurate, they are also resource-intensive and do not scale well with link speeds. Looking to this area of interest, this paper proposes a novel Deterministic Finite Automaton, called Ranged Compressed Deterministic Finite Automaton (RCDFA), that compresses transitions without additional memory lookups. Experimental results show that RCDFA yields space savings of 97% over the original DFA and up to 93% better compression when compared to the DFA's state-of-the-art compression techniques.**

Index Terms— **DFA Optimizations, Deep Packet Inspection, Performance Evaluation, Computer Networks**

I. INTRODUCTION

IN the past few years, network traffic characterization has become an important tool for accurate network management and traffic profiling. It is well known that port-based classification is inaccurate, due to traffic tunneling, for applications that use other ports assigned to well-known services in order to evade firewalls rules, such as P2P applications [4][7][5]. For that reason, traffic classification techniques have been recently relying on Deep Packet Inspection (DPI) engines. Such systems frequently perform a set of time-critical operations to verify certain application patterns or behaviors, while trying to minimize packet processing delays. Although DPI systems are essentially more accurate, they frequently perform a set of time-critical operations and are consequently resource-intensive. Therefore, if not proper designed, they may not scale well with link speeds. In general, a DPI system works as follows: first it has to collect packets from the network interface cards (NIC), create a data structure to represent incoming packets as network flows (usually as a hash table), and forward or store the received packets for further processing. After that it searches for well-known patterns within the packet payload (i.e. application signatures) for each flow. Pattern matching procedures in DPIs are usually performed at the user-space level and are highly processing intensive, which causes significant packet losses. In other words, even though NICs and Operating Systems' (OS) kernel can keep up with packets arriving at wire-speed, the pattern-matching component of the DPI system may not be able to deal with all the incoming packets without strangling the processor, thus incurring losses.

Currently, DPI systems express patterns using regular expressions [10]. Therefore, it is natural for them to perform pattern matching through the use of Finite Automaton (FA). State-space explosion of Deterministic FAs (DFA) may require an unacceptable amount of memory space [10]. Decreasing the complexity of matching procedures and reducing the memory consumption of DFAs are the main goals of research studies in this field. This paper proposes and evaluates a novel DFA that aims to decrease space requirements when used to perform pattern matching in DPI systems.

The contributions of this paper are two-fold: first, we have proposed a novel Deterministic Finite Automaton, called Ranged Compressed Deterministic Finite Automaton (RCDFA). RCDFA is based on the following key observation: several consecutive transitions lead to the same destination state. Smart transition representations result in huge space savings over a standard DFA. Second, we have developed an algorithm for converting FAs from the original DFA to RCDFA. This implies that previously developed and well-tested algorithms for parsing from a regular expression to Non-Deterministic FAs (NFA) and DFAs can be reutilized. We also evaluate and compare the performance of RCDFA to state-of-the-art DFA variations for traffic identification.

The remainder of this paper is organized as follows. Section II presents related work. Section III presents our new Automaton model. Section IV shows the methodology used on RCDFA evaluation and Section V presents experimental results. We discuss our findings in Section VI. Concluding remarks and suggestions for future work are presented in Section VII.

II. RELATED WORK

Although flexible and expressive, automata-evaluated regular expressions traditionally are memory-greedy and severely limit performance in most platforms. Developing DPI systems at multi-gigabit rates is a difficult task as they need to achieve high processing speeds while limiting memory consumption or access. Research studies have been adding some features to the original automata formalism in order to meet such speed and memory consumption requirements.

978-1-4673-0269-2/12/$31.00 ©2012 IEEE

In [11] Yu et al. proposed two rewrite rules that can dramatically reduce the size of the resulting DFAs. They developed techniques to combine multiple DFAs into a small number of groups in order to improve matching speeds. Kumar et al. [8] introduced a new representation for regular expressions, namely Delayed Input DFA (D^2FA). D^2FA is based on a technique used in the Aho-Corasick string matching algorithm. They observed that, in the case of practical rule-sets commonly used in network intrusion detection systems, many groups of states share sets of outgoing transitions. Therefore, in order to explore the redundancy present in these DFAs, they introduced a special type of transition, called default transitions. With such a modification, when matching an input string a default transition is used to determine the next state, whenever the current state has no outgoing edge labeled with the current input character. In [9], Kumar et al. proposed a new representation for the D^2FA, namely Content Addressed Delayed Input DFA (CD^2FA), which aims to improve its throughput. CD^2FA provides a compact representation of regular expressions that match the throughput of traditional uncompressed DFAs. Becchi et al. [3] introduced a general compression technique to reduce the number of transitions of a DFA with lower provable bounds on memory bandwidth, namely Fast Compression. Similar to D^2FA, this modification reduces the amount of memory needed to represent a DFA by exploiting its redundancy. In [6], Ficara et al. developed a new DFA variation called DeltaFA. DeltaFA's compression comes from the following observations: most default transitions are directed to states closer to the initial state; and, for any given input symbol most transitions are directed to the same state. Becchi et al. [2] proposed a hybrid automaton which addressed the exponential increase in the number of DFA states by combining the benefits of deterministic and non-deterministic finite automata. Basically, their automaton is a mix of Deterministic and Non-Deterministic Automata. In [10], Smith et al. proposed the Extended Finite Automata (XFAs), which augment traditional finite automata by using a temporary memory manipulated by instructions attached to states and edges. The author also presented a formal definition for their XFA and created a technique to build a XFA out of a regular expression.

Our work differs from the above-mentioned state-of-the-art models by exploring consecutive transitions in order to reduce space requirements. The central idea is simple, but very effective and not simplistic. Our model also proves to maintain a stable compression ratio when applied to a number of data sets, while previous work yields different results for datasets with different characteristics.

III. TECHNICAL BACKGROUND AND THE RANGED COMPRESSED DETERMINISTIC FINITE AUTOMATON (RCDFA)

FA formalism is a well-known and well-established theory. It was developed over decades and applied to several different fields as pattern recognition, lexical analysis in compilers, and recently to computer networks for network security and traffic classification. Although FA formalism is solid and general enough to deal with the above-mentioned applications, for some specific applications, it can exceed available resources, causing poor performance. One could make FA faster and improve resource consumption by reducing its generality, i.e., by modifying the formalism or the algorithms to adapt them to specific applications. This can create a FA variation, or even a new kind of abstract machine. In fact, some previous studies have created new FA variations. Strictly speaking, some of them are not FA variations, but new abstract machines, which use part of the FA theory to support themselves. Most previous research studies do not specify how to convert from a RE to its abstract machine. Instead, they use a FA as a base to create its abstract machine. From a practical view point, this is acceptable, as we are using a well-developed theory as base for a new and more specific one. However, we must keep in mind that these modifications are not standard FA and can have restricted use.

Following this trend, we looked into the original FA formalism and explored opportunities to reduce space requirements. We found some room for optimization by observing consecutive transitions leading to the same destination. Optimizing this aspect of a FA will decrease memory usage for storing transitions and will consequently decrease the memory footprint during the pattern matching procedure. Some previous work [2][3] applied a similar technique to export a FA to dot format[1] for later graphical representation conversion. However, they neither used it for compressing FA purposes nor described it as a new abstract machine model.

In this work, we aim to decrease the matching complexity and to provide memory savings on DFAs. Basically, we explore an algorithm to compress transitions without additional memory lookups. In other words, we aim at finding a good tradeoff between compression and matching speed. In addition, we tolerate the decrease of the model generality in order to obtain additional memory savings and performance gains. Therefore, our solutions are restricted to the traffic classification domain.

A. *Motivational Example*

Some previous studies focused on decreasing the number of transitions by looking for similar transitions in different states. For instance, D^2FA [8] tries to reduce the number of transitions by removing the ones common to pair of states and by introducing a default transition into it (default transitions are triggered without consuming an input symbol). Although that technique is efficient in compressing transitions, it also introduces additional memory accesses per input symbol.

In order to make things clearer, let's analyze the DFA created for recognizing the regular expression (regex) ^\x01[\x08\x09][\x03\x04] (from L7-Filter's FreeNet

[1]Dot Language. http://www.graphviz.org/doc/info/lang.html

application signature). The automaton presented in Figure 1 seems to be very simple, with 5 states and 10 transitions. However, it hides a pitfall, since some transitions are represented as intervals (the leftmost transitions). In fact, according to the automata theory, every standard DFA always has one transition for each alphabet symbol for every state. Therefore, supposing the DFA below uses the ASCII table as its input alphabet, it has 5 (number of states) * 256 (alphabet length) = 1280 transitions, although we only see 10.

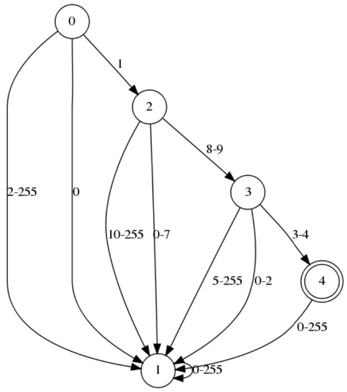

FIGURE 1 – DFA FOR FREENET REGEX

With a good understanding of the automaton complexity, we explored opportunities for improvements. Actually, the visual aid used to present the above DFA can be also adapted to compress the real automaton. Surprisingly, most previous studies depicted automata with some kind of visual compression, although no one used them as a real compressing technique. This could be partially due to the difficulty in finding a suitable memory layout for representing this new kind of automata. Figure 2 presents the same automaton, although it separates the traditional transitions from the ranged transitions (transitions for a char range). Regular transitions are in red (solid line) and ranged transitions in blue (dotted line). As we can see, this decreased the number of transitions from 1280 to 2 regular (or single) transitions and 8 ranged transitions.

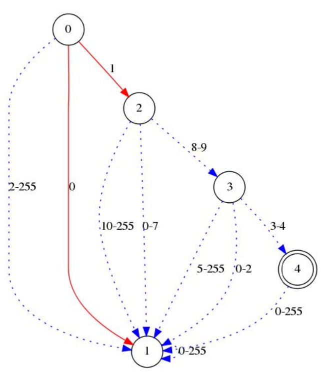

FIGURE 2 – DFA FOR FREENET REGEX

This way of representing transitions will lead to what we called a Ranged Compressed Deterministic Finite Automaton (RCDFA). RCDFA is a slightly different DFA model, although compatibility with the standard DFA is guaranteed by ensuring that both delta functions' results are identical. In the next subsections we describe the RCDFA, as well as the algorithm to convert from a DFA to a RCDFA.

B. *RCDFA Definition*

We describe the above-mentioned modification as a new kind of abstract machine (RCDFA). This new machine represents consecutive transitions going to the same destination state as a unique ranged transition. Basically we convert transitions for character ranges $c_m \ldots c_n$ for a state q_i, where $n \geq m$ and $\delta(q_i, c_j) = q_l$ for j varying from m to n to a unique ranged transition t_{m-n} to the state q_l. We slightly changed the FA formalism to deal with this new type of transitions. Therefore, the new RCDFA model is also a quintuple R=$(Q, \sum, \delta, q_0, F)$, where;

1. Q is a finite set of states;
2. $\sum$ is a finite set of input symbols;
3. $\delta : Q \times 2^{\Sigma} \rightarrow Q$ is a transitional function that takes a state and an input symbol "range" as arguments and returns a next state;
4. q_0 is the initial state that belongs to the $\sum$ set;
5. $F \subseteq Q$ is a set of final or accepting states.

C. *RCDFA and DFA Equivalence*

As mentioned before, RCDFA and DFA equivalence is enforced by ensuring that both Delta's functions have the same results for every state and symbol. Thus, we need to make sure that $\delta_{rcdfa}(s_i, (c_m, c_n)) = \delta_{dfa}(s_i, c_j)\ \forall s \in Q$ and $c \in \sum$ for j varying from m to n where $n \geq m$.

Figure 3 shows the algorithm for checking RCDFA and DFA equivalence. Initially, it iterates over all states of the RCDFA (line 2), then it verifies every transition of the current state (line 3). In line 4, it iterates over all symbols of the transition t (recall that transitions in the RCDFA are represented as a pair of symbols instead of a unique symbol). Lines 5 and 6 compare both Delta's function results, where if a different result is found, the function returns and the FAs are different. If no difference is found, the automata are equivalent.

```
1:  function checkEquivalence( DFA, RCDFA )
2:   for s=0 to GetNumberOfStates(RCDFA) do
3:    for each t(m,n) in GetTransitions(RCDFA)do
4:     for c = m to n do
5:      if GetNextState( DFA, c) ≠ GetNextState(
    RCDFA, (m,n)) then
6:         return false;
7:   return true;
8:  end function
```

FIGURE 3 – ALGORITHM FOR CHECKING DFA AND RCDFA EQUIVALENCE

At first look, the checking function demands one step for state (*N*), one for each alphabet symbol transition and one additional per symbol (C^2) in the state. Hence its time complexity would be O(N x C^2), O(n^3). However, each transition represents a range of symbols, and a range could be at maximum C symbols length. As a result, the time complexity actually is O(N x C), i.e. O(n^2).

D. Converting DFA to RCDFA

The algorithm to convert DFA to RCDFA is straightforward. In a nutshell, it receives as input an already computed DFA and then converts it to a RCDFA. It is also possible to derive a RCDFA directly from a regular expression. Figure 4 describes the conversion algorithm. In line 2 it iterates over all states present in the DFA received as parameter. Lines 3 to 20 initialize an array with one position for every symbol present in the input alphabet. Then, for every symbol in the alphabet, it creates a range transition if the subsequent symbols go to the same destination (lines 6 to 20). As far as we are concerned with complexity, the conversion algorithm requires one step per state (N) and two more per symbol (2C). This results in O(N X 2C) complexity, i.e. O(n^2).

```
 1:  function compressDFA( DFA )
 2:    for each state in DFA
 3:     for each symbol in alphabet do
 4:      mark[symbol] := not marked;
 5:     end for;
 6:     for each symbol in alphabet do
 7:       if mark[symbol] = not marked then
 8:        mark[symbol] := marked;
 9:        target:=GetNextState(DFA,state,
     symbol);
10:       ranged := false;
11:       begin_range = symbol;
12:       end_range = next symbol;
13:       while end_range < alphabet size and
14:        GetNextState(DFA, state, end_range) =
     target do
15:         mark[end_range] := marked;
16:         end_range := next symbol;
17:       end while;
18:       transitions_table[state] := new
     transition(begin_range, end_range);
19:      end if;
20:     end for;
21:     end for;
22:     end function
```

FIGURE 4 – ALGORITHM FOR CONVERTING DFA INTO RCDFA

E. RCDFA's Matching Process

The matching procedure is now quite different from the original DFA. With the RCDFA, the matching procedure looks to see if the input matches on a character range instead of a single character. Figure 5 shows the matching process for a RCDFA. $t_{rcdfa}(s, j)$ is the transition table mapping from a state *s* and *a* input char *j* to a next state *d*. First, the algorithm loads the information for the state *s* and then it looks for the next state. Basically, the lookup process is similar to the DFA; however the transition table's internal organization is totally different. It has transitions represented as ranges, therefore it verifies if *c* belongs to a range (n, m) (where $n \leq m$) instead of comparing with a single character transition.

```
1:  function RcdaLookup( s, c )
2:     read( s );
3:     d := t_rcdfa(s, c);
4:     return d;
5:  end function
```

FIGURE 5 – ALGORITHM FOR LOOKING UP ON A RCDFA

F. Combining Models

RCDFA is orthogonal to other models, i.e. it can also be combined with most of previously developed automaton models. Therefore, applying RCDFA over other automaton model could lead to additional compression. Although some previous techniques claim to be orthogonal to the others, we need to carefully analyze which techniques could be used this way. Misuses of such a tool can result in non-equivalent automata, i.e., different results for the automata's Delta functions. For example, both Fast Compression and D^2FA techniques reduce the automaton's transitions by adding default transitions to it. Those default transitions are organized by taking into account the likelihood of destination states of neighbors' state transitions. In fact, they use the insertion of default transitions for deleting transitions. Actually, one could consider them as the same technique with different policies for organizing default transitions and deleting labeled transitions. At this point it must be clear that those two techniques cannot be applied orthogonally one to another. Applying D^2FA over Fast Compression would disorganize the transition arrangements of the latter. We analyzed the RCDFA's orthogonality and found out it is very suitable for default transitions' based models. Consequently, RCDFA can be applied over D^2FA and Fast Compression with minor adaptations. We do not show the complete algorithm for converting between D^2FA/Fast Compression to RCDFA due to lack of space. Actually, RCDFA conversion algorithm needs only to take into account D^2FA/Fast Compression's default transition to be fully compatible. Figure 6 presents the difference between the conversion of DFA to RCDFA and D^2FA/Fast automata. Before line 21, the algorithm is the same as presented in Figure 5. After line 21, the algorithm had to be changed to deal with default transitions. In summary, this piece of code checks if there is a default transition for the current state. If so, it adds the default transition to the new automaton.

```
21:       defdst=GetDefaultTransitions(DFA,
    state)
22:       if ( defdst ≠ EMPTY )then
23:        defaulttransitions[state]=deftrans;
24:       endif;
25:      end for;
26:    end function
```

RCDFA is also orthogonal to DeltaFA and, even better, conversion from DeltaFA to a RCDFA does not require changes to the algorithm presented in Figure 5. Therefore, we only need to have a DeltaFA as an input instead of a standard DFA. The output is then a combination of DeltaFA and RCDFA.

The Deterministic Automata created by such combinations is summarized in TABLE I. Basically, we applied the ranged compression over the other three models, namely Fast Compression, D^2FA, and DeltaFA.

TABLE I – Combined Automata

Automaton's Name	Description
RcFast	Ranged compression applied over a Fast compressed automaton
RcD^2FA	Ranged compression applied over a D^2FA automaton
RcDelta	Ranged compression applied over a DeltaFA automaton

IV. METHODOLOGY

This section shows the methodology used for evaluating our new automaton model. We collect metrics directly from the Automaton, i.e., we convert a signature (from a given signature set) into an automaton which recognizes it. We then apply the compression algorithms creating each automaton model. Finally, we compute performance metrics.

TABLE II presents the factors and levels we used in our experiments. In summary, to test our model we used five different signature sets, namely L7-Filter, Bro, Snort-Web, Snort-ActiveX, and Snort-Spyware. TABLE III shows the most important parameters of each set, following the classification method proposed in [1]. L7-Filter base is the smallest one, but with moderate complexity. Bro is medium size, but with low complexity. SnortWEB also presents medium size although with high complexity. The largest base (SnortActiveX) is also very complex. Finally, SnortSpyware is not complex and is medium size. Those signature sets give us a good sample of real world expressions which DPI engines must tackle. These signatures were collected on October 2010.

TABLE II – Evaluation Factors and Levels

Factor	Levels
Signature Set	SnortWEB, SnortSpyware, SnortActiveX, Bro and L7-Filter
Automata model	RCDFA, Fast Compression, DeltaFA, and D^2FA

TABLE III – Signature sets' main characteristics

Sig-Set	Base Size	Sub-Pattern number	Overall complexity
L-7 Filter	Small	Medium	Moderate
Bro	Medium	Low	Low
Snort-Web	Medium	Medium	High
Snort-ActiveX	Large	High	High
Snort-Spyware	Medium	Medium	Low

We adopted the following metrics in our evaluation:

- **Total of transitions**: Number of automaton's transitions;
- **Single character transitions**: Transitions which cannot be collapsed with others forming character ranges;
- **Ranged transitions**: Transitions which can be triggered by character ranges;
- **Space reduction**: space reduction percentage over original DFA and other techniques;
- **Transitions per state**: the average number of transitions per state.

V. EXPERIMENTAL RESULTS

Firstly, we compare the total transitions number of each model (D^2FA, RCDFA, DeltaFA and Fast Compression). Secondly, we show the compression rate over the original DFA model. Then, we compute how much better RCDFA compress over D^2FA, DeltaFA, and Fast Compression. And, finally, we show the average number of transitions per state for each model.

A. *L7 - Filter*

For L7-Filter signatures, Fast Compression algorithm presented the largest number of transitions; around 1.4M transitions were used to represent all expressions whereas Fast yielded 900K. D^2FA utilized 500K transitions and RCDFA used only 55K transitions, where 17.5K were single transitions and 38.5K were ranged transitions.

Figure 7 shows the compression rates for every DFA modification. As we can see, DeltaFA technique had the worst result, since it reduced the DFA size in only 34.2%. Fast compression reduced the number of transitions in 59.2%. D^2FA achieved 76% and RCDFA was able to remove 97.4% of the original DFA's transitions. In fact, RCDFA compressed 96%, 93.8% and 89% better than DeltaFA, Fast Compression algorithm and D^2FA, respectively. RCDFA yielded far superior compression for the L7-Filter data set, which makes it more suitable for application/protocol identification signatures and more adequate for platforms where memory consumption is an issue.

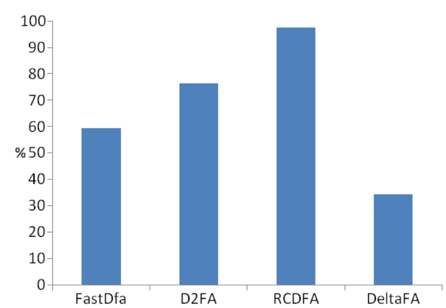

FIGURE 7 – COMPRESSION RESULTS FOR L7 FILTER

TABLE IV depicts the average number of transitions per state. As one could notice, standard DFAs always have $|\Sigma|$ symbols per state. For most DPI scenarios this is the ASCII table length (256 symbols). Therefore we take 256 symbols as our worst case. DeltaFA reduced this number to around 168 transitions in average. Fast Compression presented 104 transitions per state in average. D^2FA had around 60 transitions per state. Again, RCDFA has better results. It requires, in average, around six transitions per state.

TABLE IV – Average Number of Transitions per State

Model	StdDFA	FastDFA	D^2FA	RCDFA	DeltaFA
Number	256	104	60.2	**6.4**	168.4

B. Bro

This time, D^2FA had the biggest number of transitions, 137K. DeltaFA and Fast Compression had about half than D^2FA, 75K and 68K transitions respectively. RCDFA shows only 19K transitions, where over 8.5K were single transitions and ranged transitions accounted for 10.6K.

Compression comparison among all techniques is shown in Figure 8. For the Bro base, the compression rate is not too different than it was for L7-Filter. D^2FA had the smallest compression, around 82% followed by Delta with 90%. Fast Compression reduced the number of transitions in around 92%. Again, RCDFA performed well, presenting almost the same compression rate as for L7-Filter, 97.5%. For comparison, RCDFA compressed 86% better than D^2FA, 74% better than Fast compression and around 72% better than Fast algorithm.

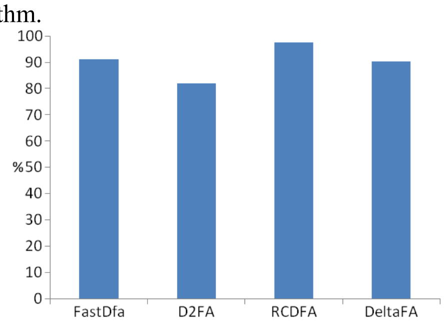

FIGURE 8 – COMPRESSION RESULTS FOR BRO

TABLE V shows the average number of transitions for each model. As we can we see, for Bro regex, all techniques greatly decreased this metric. D^2FA has the worst result, around 46 transitions per state, followed by DeltaFA with an average of 25 transitions. Fast utilized around 23 transitions per state. RCDFA reduced far better, it achieved similar results for this base, around 6 transitions per state.

TABLE V – Average Number of Transitions per State

Model	StdDFA	FastDFA	D^2FA	RCDFA	DeltaFA
Number	256	23	46	**6.4**	25.9

C. Snort-Web

For Snort-Web rules, D^2FA presented the highest number of transitions (572K) followed by DeltaFA with 456K transitions. Fast Compression had 339K transitions. RCDFA presented only 75K composed of 35K single transitions and 40K ranged transitions.

Figure 9 compares the compression ratio (CR) for each technique. D^2FA compressed the original DFA about 76%, followed by DeltaFA with 81%. Fast Compression reduced the number of transitions by 86.3 %. RCDFA achieved compression of around 97% (96.9%). Summarizing, RCDFA compressed 77.7% better than Fast Compression and around 83% compared to DeltaFA. It also outperformed D^2FA by around 86%. As far as we are concerned, in all bases analyzed so far, RCDFA has achieved a CR of around 97%.

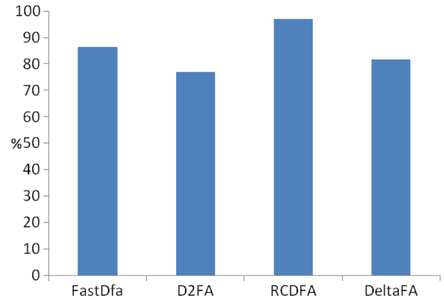

FIGURE 9 – COMPRESSION RESULTS FOR SNORTWEB

The average number of transitions per state is presented in TABLE VI. In this case, all techniques considerably reduced the average number of transitions per state. As expected, D^2FA had the worst result, around 59 transitions. DeltaFA yielded 47 and Fast Compression just about 35 transitions in average. RCDFA maintained its steady results presenting only 7 transitions per state in average.

TABLE VI – Average Number of Transitions per State

Model	StdDFA	FastDFA	D^2FA	RCDFA	DeltaFA
Number	256	35	59	**7.7**	47

D. Snort-ActiveX

For this base, the results were different than the previous ones. Fast Compression had almost the same number of transitions as RCDFA. The former had 5.6M transitions and the latter presented 6.6M. D^2FA and DeltaFA presented 34M and 27M transitions respectively, far greater than the other bases.

Following, Figure 10 depicts the compression comparison among the DFAs. At this time Fast Compression had slightly superior results compared to RCDFA, yielding 97.6% against 97% for RCDFA. D^2FA achieved 85% of reduction and DeltaFA had results of 88% for this signature set. For this base, Fast Compression algorithm performed 6% better than RCDFA, although RCDFA was still more efficient than D^2FA and DeltaFA by around 93% and 88%, in that order.

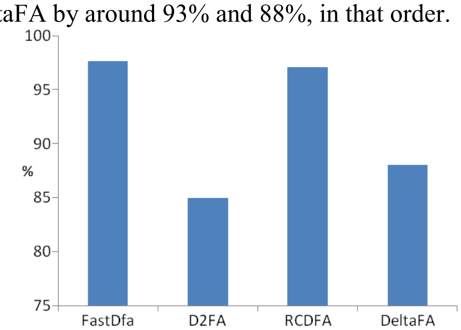

FIGURE 10 – COMPRESSION RESULTS FOR SNORT-ACTIVEX

TABLE VII shows the average number of transitions per state. In this case D^2FA had the worst result with 38 transitions, followed by DeltaFA with 30 transitions per state in average. Fast Compression and RCDFA presented almost the same results, 6 transitions for the former and around 7 for the latter.

TABLE VII – Average Number of Transitions per State

Model	StdDFA	FastDFA	D^2FA	RCDFA	DeltaFA
Number	256	**6**	38.3	**7.5**	30.5

E. Snort-Spyware

In this section, we show results from the last signature set, Snort-Spyware expressions. D^2FA once more presented the highest number of transitions, around 642K transitions, followed by DeltaFA (414K). Fast had over 251K transitions and RCDFA around 93K, distributed as follows: 40K single transitions and 52K ranged transitions.

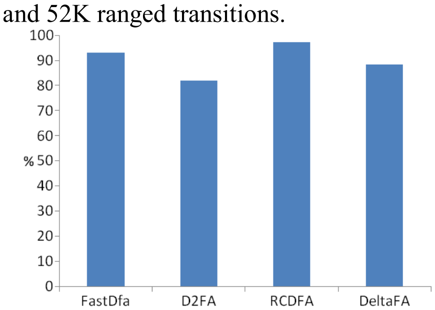

FIGURE 11 – COMPRESSION RESULTS FOR SNORT-SPYWARE

Figure 11 compares the compression rates for every kind of DFA. RCDFA was able to reduce original DFA by 97.3%, followed by the Fast model with 92.9%. DeltaFA and D^2FA had the lowest compression, 88.3% and 81.5%, in that order. In this base, RCDFA performed 63% better than the Fast technique and 77% better than the DeltaFA. RCDFA outperformed D^2FA by 85%. TABLE VIII presents the average number of transitions per state. Again, all techniques decreased considerately in this metric. D^2FA yielded 46 transitions per state followed by DeltaFA with 29 transitions. Fast Compression produced 18 transitions per state. RCDFA once again presented around 6 transitions per state in average.

TABLE VIII – Average Number of Transitions per State

Model	StdDFA	FastDFA	D^2FA	RCDFA	DeltaFA
Number	256	18	46.1	6.7	29.7

Due to space constraints, this paragraph presents the variation related results for the average number of transitions per state for all techniques and signature bases. In summary D^2FA, Fast Compression and DeltaFA presents at most 256 transitions per state and at least one for all signature bases. Their standard deviations ranged from 32 to 123 (in general, greater than 70). On the other hand, RCDFA has at most 37 and at least one to three transitions per state for all bases. Its standard deviation is very low, around 3 for every base.

F. RcFast, RcD2FA and RcDelta Resuls

This subsection presents the experimental results for all techniques used in conjunction with RCDFA.

TABLE IX presents the combined automata' transition reduction over standard DFA, i.e., how many transitions RCDFA reduced over other techniques compared to standard DFA. RcFast (Ranged Compression over Fast Compression) reduced from 98.5% to 99.4% when compared to the original DFA's transitions. RcD^2FA (Ranged compression over D^2FA) compressed around 99% for all signature bases. RcDelta (Ranged compression over DeltaFA) was able to reduce the original DFA from 97.8% to 98.6%. From these results, we argue that the best combination is RCDFA and Fast Compression. On average, together they are able to decrease the number of transitions in 99.16%.

TABLE IX – Combined Automata' Reduction over DFA

Model	L7	Bro	Snort Web	Snort ActiveX	Snort Spyware
RcFast	98.5%	99.3%	99.2%	99.4%	99.4%
RcD^2FA	99.1%	99.1%	99%	99%	99.2%
RcDelta	97.8%	98.6%	98.5%	98.6%	98.5%

TABLE X shows the combined automata' reduction over the already compressed automaton. For example, for RcFast this means how better the combined technique (RCDFA + Fast) performed over the Fast Compression alone. For almost all cases, the Ranged Compression combined with other techniques is able to decrease more than 90% over the single compressed technique. The worst result on average is for Snort-ActiveX base. As Ranged Compression had its worst results with this base, this was implied within the combined automata as well.

TABLE X – Combined Automata' Results over Compressed Technique

Model	L7	Bro	Snort Web	Snort ActiveX	Snort Spyware
RcFast	96.4%	93.5%	94.8%	75%	92.5%
RcD^2FA	96.5%	95.4%	95.8%	93%	95.6%
RcDelta	96.7%	86.1%	92.1%	88%	90.7%

VI. DISCUSSION

In the previous section we compared the transition's number and compression ratio for each signature base and automata model. RCDFA presents very good results for L7-Filter. This indicates applicability for detecting application and protocol. RCDFA also satisfactorily compresses signatures of IDS systems. It also presents compression of around 97% for IDS's signature sets. However, for Snort-ActiveX signatures, Fast algorithm performs 6% better than RCDFA. Scrutinizing this dataset, we noticed that these signatures have an elevated number of sub-patterns. Therefore, in datasets containing signatures with too many sub-patterns, Fast Compression presents additional compression and slightly better results. For all other scenarios, RCDFA outperforms Fast and D^2FA and even better, its compression rate remains stable around 97% when applied over datasets with different characteristics.

Additionally, we were able to apply RCDFA orthogonally to other techniques. From a practical point of view, techniques that rely on default transitions are very suitable for use with RCDFA. In such a case, the ones based on default transitions explore inter-state opportunities for compression and RCDFA would work on intra-state windows.

Regarding performance, it is a common belief that space savings are usually possible only in exchange of processing costs, but in DFAs, this is not always true. Evaluating performance in terms of memory accesses, standard DFA requires 1 memory access per input symbol, whereas D2FA and FastDFA require on average 2 accesses and DeltaFA requires 256 accesses. Actually, in [12], the authors showed that DeltaFA has performance losses of 99% in software implementation. On the other hand, RCDFA achieves good space compression while keeping one memory access per symbol. Therefore, RCDFA yields huge memory savings and its overall processing cost is comparable to the standard DFA (i.e., better than the state-of-the-art models). In addition, RCDFA has an advantage of improving the matching procedure performance by means of cache spatial locality. As RCDFA demands less memory space, all transitions will be closer to each other, therefore cache hit will also improve along with the overall performance.

Orthogonally, some studies tried to process more than one input character per lookup, these techniques are known as multi-stride automata. They can improve matching speed at the expense of an increased alphabet. RCDFA fits well in this scenario, as the more symbols an alphabet has, the more opportunities for ranged compression.

Looking at the experimental results, we can see that for RCDFA the average number of transitions is very low, around 6 transitions. This opens space for smart memory layouts for representing RCDFA's transitions and states. Naïve FA implementations would represent an automaton as a matrix mxn where m is |state| and n is |alphabet|. Additionally, each matrix element has length of ($\log_2$|alphabet| + size of pointer) bits (size of pointer is 32 or 64 bits depending of the hardware architecture). Obviously, for RCDFA this would result in memory space wasting as it uses only 6 transitions per state in average. Consequently, an RCDFA is not suitable for matrix based representations. Better choices would be linear and bitmapped memory layout. Particularly, as it has a really low number of transitions per states, linear encoding is a perfect match for representing RCDFA.

VII. CONCLUDING REMARKS AND FUTURE WORK

In this paper we proposed a new automaton model, RCDFA. We have thoroughly described it and presented an algorithm for converting an original DFA to RCDFA. We also ensured DFA and RCDFA equivalence. Additionally, we showed how to combine RCDFA with previously developed techniques. Finally, we evaluated RCDFA and compared it with the state-of-the-art automaton models for pattern matching. For the sake of fairness, the experimental evaluation was conducted using several well-known signature bases. According to the experimental results, RCDFA is able to compress DFA transitions in a stable rate of 97%. It also is able to reduce transitions up to 93% better than previous compression techniques. Additionally, by combining RCDFA with other compression techniques, we were able to reduce the number of standard DFA's transitions by up to 99.4%.

In the future, we aim to extend the work on optimizations in the RCDFA, by looking for matching speed improvements. Efficient ways of materialization of the RCDFA model, in terms of data structure representation, is also a good research challenge.

VIII. ACKNOWLEDGMENT

The authors would like to thank the Brazilian Research Funding Agency (CNPq) and Ericsson Research for supporting this work, which is part of the project UFP.37 (Broadband Traffic Measurements and Analysis – BTMA, Phase 3).

REFERENCES

[1] Antonello, R., Fernandes, S., Sadok, D., Kelner, J. "Characterizing Signature Sets for Testing DPI Systems", 3rd IEEE Management of Emerging Networks and Services Workshop - Globecom, Dec 2011

[2] Michela Becchi and Patrick Crowley. 2007. A hybrid finite automaton for practical deep packet inspection. In Proceedings of the 2007 ACM CoNEXT conference (CoNEXT '07). ACM, New York, NY, USA.

[3] Michela Becchi and Patrick Crowley. 2007. An improved algorithm to accelerate regular expression evaluation. In Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems (ANCS '07). ACM, New York, NY, USA, 145-154.

[4] Borgnat, P.; Dewaele, G.; Fukuda, K.; Abry, P.; Cho, K.; , "Seven Years and One Day: Sketching the Evolution of Internet Traffic," INFOCOM 2009, IEEE , vol., no., pp.711-719, 19-25 April 2009.

[5] Dreger, H., et al.. "Dynamic application-layer protocol analysis for network intrusion detection". 15th USENIX Security Symposium, 2006.

[6] Ficara, D., Di Pietro, A., Giordano, S., Procissi, G., Vitucci, F., Antichi, G. "Differential Encoding of DFAs for Fast Regular Expression Matching". IEEE/ACM Trans. on Networking Vol. 19, No.3, June 2011.

[7] Dusi, M.; Gringoli, F.; Salgarelli, L.; , "IP Traffic Classification for QoS Guarantees: The Independence of Packets," Computer Communications and Networks, 2008. ICCCN '08. Proceedings of 17th International Conference on , vol., no., pp.1-8, 3-7 Aug. 2008.

[8] Sailesh Kumar, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, and Jonathan Turner. 2006. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. SIGCOMM Comput. Commun. Rev. 36, 4 (August 2006), 339-350.

[9] Sailesh Kumar, Jonathan Turner, and John Williams. 2006. Advanced algorithms for fast and scalable deep packet inspection. In Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems (ANCS '06). ACM, New York, NY, USA, 81-92.

[10] Smith, R., Estan, C., Jha, S., Kong, S. "Deflating the big bang: fast and scalable deep packet inspection with extended finite automata". SIGCOMM Comput. Commun. Rev. 38, 4 (Oct. 2008), 207-218.

[11] Fang Yu, Zhifeng Chen, Yanlei Diao, T. V. Lakshman, and Randy H. Katz. 2006. Fast and memory-efficient regular expression matching for deep packet inspection. In Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems (ANCS '06). ACM, New York, NY, USA, 93-102.

[12] Tingwen Liu; Yifu Yang; Yanbing Liu; Yong Sun; Li Guo; , "An efficient regular expressions compression algorithm from a new perspective," INFOCOM, 2011 Proceedings IEEE , vol., no., pp.2129-2137, 10-15 April 2011.