Another CDFA based Multi-Pattern Matching Algorithm and Architecture for Packet Inspection

Tian Song  
School of Computer Science, Beijing Institute of Technology, Beijing, China, 100081  
songtian@bit.edu.cn

Dongsheng Wang  
Department of Computer Science and Technology, Tsinghua University, Beijing, China, 100084  
wds@tsinghua.edu.cn

Abstract—Multi-pattern matching algorithm and architecture is critical for packet inspection based network security applications, especially for high speed network or large pattern sets. This paper presents a method to optimize the potential memory usage of DFA based algorithms for multi-pattern expression matching by the combining DFA’s paths, named isomorphic path combination (IMPC). To achieve IMPC, a novel multi-pattern matching algorithm, called ACS, is proposed, which is based on CDFA. Compared to the algorithms on DFA, our method can reduce 78.6% states for Snort pattern set, which results in one of the most memory efficient methods. The most important is that our method is a kind of optimization and can be embedded to other architectures as the second step for better results. Finally the architecture based on ACS is proposed and the experimental results show that 47.6% to 84.0% memory space can be saved for different size of pattern sets as compared to the best known architectures. The method is another one based on CDFA. It means that CDFA may be a more proper model for multi-pattern matching than other FAs.

Keywords— pattern matching; NIDS; CDFA; string matching;

1. INTRODUCTION

With the development and wide application of network, the game between attacks and defenses has also evolved. Intrusion detection/prevention systems (IDS/IPS), virus scanners, spam filters and other content inspection applications are developed to scan the payload to find malicious codes. So, unified threat management (UTM) has been recently introduced to combine all the above functions to detect all kinds of attacks in network.

To unload the burden of payload inspection for UTM like systems, specific hardware with stable performance is required to accelerate computation intensive operations, such as packet classification, multi-string matching and regular expression matching. At the same time, for more than 1-10Gbps wire speed, those architectures are also critical to overall performance.

For payload inspection, systems are required to monitor every network packet in real time, and match with a developing big pattern set. It consumes a lot of time and memory space. Generally UTM-like systems needed to simultaneously match tens of thousands signatures or more at the speed of multiple gigabits per second. Therefore, effective matching algorithms and architectures are very important.

This paper continues the research to show that multi-pattern matching algorithms and architectures can be highly optimized by an elegant idea based on Cached DFA. Cached DFA was firstly proposed by T. Song [1], which is a simple extended model of DFA by adding one or more buffers (the word “cache” is used in our work). The extension is elegant and promising as a better basic theory for pattern matching algorithms. ACC algorithm based on CDFA can greatly reduce more than 90% memory usage by removing almost all cross transitions of DFA for not-anchored matching, with the overhead of only 32-bit memory. [1] The idea is to exploit one unit memory to dynamically generate cross transitions. In this paper, we present another idea based on CDFA to combine existed transitions.

Since there are many good algorithms proposed recently, our idea is not to propose a stand-alone one but a method can be embedded to the good ones as a second step. In our work, we take advantage CDFA for the purpose, which can further reduce more than 70% memory requirement.

The key contributions can be summarized as follows.

- The lower boundary of traditional DFA based pattern matching algorithms is presented and analyzed.
- Isomorphic path combination (IMPC), an idea to optimize pattern matching algorithms, is addressed.
- Cached DFA (CDFA) based method is designed to achieve IMPC. Operational details are also addressed.
- A novel pattern matching algorithm, ACS, which are based on CDFA and IMPC is proposed. The related hardware design model is also presented.
- Experimental results show that 78.6% states can be saved by using ACS algorithm than DFA based solution.

The rest of the paper is organized as follows. Section 2 provides a brief overview of related works. Then section 3 describes the analysis of the DFA based solution to pattern matching and addresses the issues to be solved. Section 4 begins our method by reviewing the model of CDFA and our idea, namely isomorphic path combination. In section 5, ACS algorithm and related hardware are presented, which are based on CDFA for IMPC. Consequent evaluation results are given in section 6. In the last section, conclusions are drawn.
II. RELATED WORKS

In recent years, several multi-pattern (string/regex) matching methods and architectures have been proposed for network security. Their bases can be classified into four categories: FPGA, CAM, bloom filter and DFA.

FPGA based pattern matching architectures were firstly developed because FPGA is suitable for fast prototyping [2-5]. They have one common feature that all patterns are synthesized as programmable logics on FPGA. However, FPGA-based solutions encounter the issue of dynamically updating, which can interrupt the normal working of systems.

CAM and TCAM are other devices that can be applied to construct pattern matching architectures. Fang Yu[6] proposed an TCAM based design to handle pattern matching and packet classification at gigabit rate. Pattern set are stored in the CAM or TCAM memory.

The theory of bloom filter is the third basis of pattern matching architectures [7-8]. Multi-hardware based bloom filters can match patterns at high speed. However these methods are related to the longest pattern in one set. Thus pattern set with longer pattern may result in larger chip area and more power consumption.

Recently, DFA based pattern matching architectures have gained the favor of researchers, because they have stable (deterministic) matching speed regardless of the size of pattern set, the length of pattern and the relationship between patterns and input texts [9-11]. Aho-Corasick[17], AC, was the classic algorithm based on DFA. Pattern set is also stored in memory with proper format, which is fit for dynamically updating.

However the memory requirement of AC like algorithms is large with the increasing of pattern set. Lin Tan [9] tried to solve this issue by using many tiny split state machines that represent patterns. Jan v. Lunteren [11] uses the method of priorities and pattern set partition. Cheng-Hung Lin[20], and Michela Becchi[21] both tried to merge states in DFA.

To solve the issue from the view of basic model, several extended DFAs were also proposed, such as D’FAs, XFA and CDFAs[1, 12]. The research based on CDFA is furthered in this paper. We will show that CDFA has the potential to further optimize the memory usage on the result of previous work.

III. PROBLEM ANALYSIS

A. Pattern Matching Problem and DFA

For network security, pattern matching is widely used with different definitions in different cases. For some applications, such as packet classification by checking packet headers, the location of a given pattern may be anchored, i.e., a matching only occurs when the pattern begins at a predefined location within the text to be matched. This case is called anchored matching. In other cases, patterns may begin anywhere in the text for the cases such as payload checking or spam filtering. Accordingly, it is called anywhere matching.

DFA based pattern matching algorithms for hardware designs are widely used, because it can provide stable matching speed regardless of the relationship between patterns and input text. This advantage can defend the attacks aiming at pattern matching engines. In our work, we take the precondition that architectures on DFA accept 8-bit character at one time.

B. Classification of Transition Rules

DFAs are built separately for anywhere matching and anchored matching problems. Taking set of {SIG, SSH} as an example, Figure 1 shows the DFA of anywhere matching, and Figure 2 shows the DFA of anchored matching.

For anywhere matching, there are total 16 transitions in its DFA. Considered the different functions, we classify all these 16 transitions into four categories: basic transitions, cross transitions, failure transitions and restartable transitions. The definitions can be found in our previous works [1]. For anchored matching, as Figure 2 shows, its DFA consists of 9 transitions. According to the classification of transitions, there are only basic transitions (numbered 1-5) and failure transitions (numbered 6-9).

In our previous works, all restartable, failure and cross transitions can be efficiently eliminated from CDFA. In the rest of our work, the basic transitions and states are our targets. It is obviously that reducing the state number can dramatically reduce the total memory used in the architectures.

C. Lower Boundary of Anywhere Matching

For anywhere matching, basic and cross transitions can cause memory’s explosion. A natural method is to partition the pattern set into many unrelated smaller ones which are handled by individual DFAs [9-10]. Here we give some results of Snort to evaluate this method and find their essences in figure 3.

Snort[15] is an open source lightweight network intrusion detection system with thousands of patterns. One version pattern set of Snort v2.3.3 with more than 3000 patterns is used. After eliminating duplicated patterns, there are 1785 different
ones left. Only strings are used in this statistics for simplicity. It is important to know that our method can be also used in regular expression matching.

When pattern set is partitioned into smaller ones, the dichotomy based on pattern numbers is used without any selection. Subset number is the number of smaller pattern sets and the results are the sum of all the subsets. We do not include failure and restartable ones in our statistics, since they are at most 256 with priority method.[10]

Figure 3 show the trend of transition number of Snort. We can find that the number of cross transitions changes greatly, while the number of basic transitions remains stable.

The trends are intuitive for a given pattern set. The basic transitions are stable because the states and the framework of DFA are determined only by given pattern sets. However, the cross transitions represent the common sub-patterns within the pattern sets. Therefore, with more subsets (less patterns in each subset), the number of cross transitions decreases rapidly. In an extreme case when there is only one pattern in one subset, the number of total transitions may result in a lower boundary.

For anywhere matching problem, the lower boundary occurs when no cross transitions exists, which equals to the corresponding value of anchored matching problem. We use transitions per character (trans per char or TPC for short) as the metric, and the lower boundary is shown in formula 1. Figure 4 gives our data of Snort set compared to Jan v. Lunteren’s in which optimized set partitioning method is used.

\[
\text{LowBoundary} = \frac{\text{basic \_ trans}}{\text{total \_ chars}} \tag{1}
\]

Because basic transitions form the framework of DFA and determine the lower boundary for a given pattern set, they are hard to optimize. Lower boundary determines the scale of DFA. All optimizations can only be taken with the restriction. How to use fewer states to build the DFA is an important issue.

In our paper, we mainly solve the issue by using CDFA based ACS algorithm, which can break through the lower boundary of DFA for pattern matching problems, regardless of anywhere or anchored matching.

IV. IMPC IDEA

A. Isomorphic Path Combination

For a given pattern set, the framework of DFA, which is defined as the states and transitions regardless of the relationship between patterns or within patterns, can be built only by basic transitions. Taking pattern set \{pattern, betters\} as an example, the traditional DFA for accepting it is shown as Figure 5 (a).

In Figure 5 (a), there is one longest common sub-pattern “ter” for both of the patterns, which consumes 10 states (S9 to S13 and S2 to S6) and 8 transitions. Obviously, S2 to S6 represents the same sub-pattern with S9 to S13, named isomorphic path, which shows the possibility for optimization.

A DFA can be defined as \( M = \{K, \Sigma, \delta, s_0, F\} \) [18], where \( \Sigma \) is the alphabet and \( \delta \) is the transition function for transitions. The isomorphic path can be defined as follows.

Definition: (Isomorphic path) \( S_i, S_{i+1}, ..., S_{i+m} \) (states) and \( S_j, S_{j+1}, ..., S_{j+m} \) are isomorphic if and only if, for \( \forall c \in \Sigma \), \( p=0,1, ..., m, q=0,1, ..., m \), \( \delta(S_{i+p}, c) = S_{i+q} \) and \( \delta(S_{j+p}, c) = S_{j+q} \) are both correct at the same time. The common path from \( S_i \) to \( S_{i+m} \) (or from \( S_j \) to \( S_{j+m} \)) is defined as isomorphic path.

The idea to reduce basic transitions by taking advantage of isomorphic paths is then called IsoMorphic Path Combination (IMPC for short). Figure 5 (b) shows the ideal (optimized) DFA to accept the pattern set, which can combine the isomorphic paths. However the ideal DFA cannot work because the transition from S6 to S9 by accepting “s” should be conditional, otherwise “patters” can be recognized as the result pattern incorrectly.
The reason why we cannot combine isomorphic path by traditional DFA directly lies in two facts. One is that DFA does not have the ability to memorize the history. The other is due to the fact that the next state is only determined by current state and input character. Thus, in S6, there is no history path to be referred and the transfer from S6 to S9 cannot be conditional.

To explore the potential of reducing the number of basic transitions, CDFA model is revisited.

B. CDFA Model

As illustrated in Figure 6(a), DFA is a simple and concise model. The transitions are stored in transition rules (tran-rules) memory and accessed by tran-rules selector. The next state (to be stored in state register) is determined only by input character and current state (stored in state register).

CDFA is an extended DFA by using certain number of registers as cache (only one register is used in this work), as Figure 6(b) shows. Some information may be stored in cache and the next cache is determined by current cache, input character and current state. The next state is also determined by the input character, current state and current cache.

With cached state, CDFA extends DFA with the capability of memorizing history. The general framework of CDFA is better than DFA in two aspects, which have been mentioned in section 4.1, that is, CDFA can memorize the history and take advantage of conditional transition by using cache. Thus isomorphic path can be combined based on it.

C. CDFA for IMPC

Our method of using CDFA to combine isomorphic path acts the same as the ideal DFA (shown in Figure 5 (b)).

For pattern set {pattern, betters}, the CDFA for IMPC is shown in Figure 7, in which a register ($P$ in the figure) is added as cache. It has the same structure as the ideal DFA, which is compressed by combining isomorphic path.

The actions of CDFA are mostly similar to the ones of DFA. In cycle 1, CDFA accepts input character “b” and the current state transfers to S8. In cycle 2, “e” is accepted and the current state transfers to S2. Additionally, S8 needs to be stored to the cache of CDFA (SP). From cycle 2 to cycle 5, CDFA acts the same as DFA. In cycle 7, the next state is determined by input character (“s”) and the content of cache which is fetched from SP. If the cache content is S8, the next state will be S9, otherwise transitions with lower priorities (failure and restartable transitions) will be further considered. The transition table is showed in Table 1.

From the example, we can find that states of CDFA for IMPC have three operations. For each state, we should know what to do. Thus three denotations are used. We classify states of CDFA for IMPC into three types: converging states, common states and diverging states, as shown in Figure 8

- **Converging states** are those in front of combined isomorphic path. Besides state’s transitions, these states should store the state number in cache for future use, such as S1 and S8 in Figure 7.
- **Common states** are the traditional states as the ones in DFA. In these states, cache is not operational, such as S2, S3, S4 and S5 as Figure 7.
- **Diverging states** are the ones that determine the next state by inputting character and the content of cache. In these states, the content of cache is fetched and cache is also cleared. S6 belongs to this type. For the transitions, they require one more filed to save the possible content of cache, such as R9 and R10.

In CDFA for IMPC, states are colored to distinguish different types, yellow for converging states, pink for diverging states and white for common states. The colors of states are painted during the CDFA building. The executive component performs the corresponding operations by current state’s color.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Current State</th>
<th>Cached State</th>
<th>Input Char</th>
<th>Next State</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>S0</td>
<td>*</td>
<td>“p”</td>
<td>S1</td>
<td>2</td>
</tr>
<tr>
<td>R2</td>
<td>S1</td>
<td>Store S1</td>
<td>“a”</td>
<td>S2</td>
<td>2</td>
</tr>
<tr>
<td>R3</td>
<td>S2</td>
<td>*</td>
<td>“a”</td>
<td>S3</td>
<td>2</td>
</tr>
<tr>
<td>R4</td>
<td>S3</td>
<td>*</td>
<td>“i”</td>
<td>S4</td>
<td>2</td>
</tr>
<tr>
<td>R5</td>
<td>S4</td>
<td>*</td>
<td>“e”</td>
<td>S5</td>
<td>2</td>
</tr>
<tr>
<td>R6</td>
<td>S5</td>
<td>*</td>
<td>“i”</td>
<td>S6</td>
<td>2</td>
</tr>
<tr>
<td>R7</td>
<td>S6</td>
<td>*</td>
<td>“b”</td>
<td>S8</td>
<td>2</td>
</tr>
<tr>
<td>R8</td>
<td>S8</td>
<td>Store S8</td>
<td>“e”</td>
<td>S2</td>
<td>2</td>
</tr>
<tr>
<td>R9</td>
<td>S8</td>
<td>S8</td>
<td>“s”</td>
<td>S9</td>
<td>2</td>
</tr>
<tr>
<td>R10</td>
<td>S6</td>
<td>S1</td>
<td>“n”</td>
<td>S7</td>
<td>2</td>
</tr>
<tr>
<td>R11</td>
<td>*</td>
<td>*</td>
<td>“b”</td>
<td>S8</td>
<td>1</td>
</tr>
<tr>
<td>R12</td>
<td>*</td>
<td>*</td>
<td>“p”</td>
<td>S1</td>
<td>1</td>
</tr>
<tr>
<td>R13</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>S0</td>
<td>0</td>
</tr>
</tbody>
</table>
D. Implicit State Coloring

There are two ways to represent the states’ colors. One is to explicitly use another several bits for each state, which may cause memory overhead. The other is to take advantage of current information and to implicitly color the states. We use the later method.

The implicit state coloring is based on the fact that the serial numbers of states are not strictly fixed, that is, the same DFA(CDFA) may have many ways to number the states for a given structure. Although Figure 9 (a) and (b) use different ways to number states, they have the same function which accepts pattern set \{\text{pattern, betters}\}.

Based on the feature of state numbering, we color the states of CDFA in Figure 7 in the following ways.

The new numbering with implicit state color and the old numbering both require 4 bits to represent one state. No additional bit for coloring is needed in this example. In practice, one additional bit may be required at most for each state in different pattern sets.

V. ACS ALGORITHM AND ARCHITECTURE

This section will present ACS algorithm, which exploits the CDFA model with implicit state coloring for IMPC to realize a fast and storage-efficient pattern matching solution. The algorithm is similar to AC algorithm addition with the method of how to find isomorphic paths. We do not aim to find all isomorphic paths but the efficient ones.

A. Rules for Finding Isomorphic Path

For easy implementation, some rules are given for finding isomorphic paths. They are not strictly prerequisite for IMPC but the experienced ones for simplifying the issue. The basic idea is that all isomorphic paths are not overlapped and not confused for judging the next step on the diverging state.

- R1: The first character of all patterns is never counted as part of isomorphic path.
- R2: For each converging state, there is only one corresponding diverging state. That is, for an isomorphic path, the only exit corresponds to all entrances.
- R3: For one pattern, there may be many potential isomorphic paths with other patterns, while only those chosen to be combined are called isomorphic paths.
- R4: For one pattern, there may be several isomorphic paths to be combined with other patterns. However, any two of them are not overlapped.
- R5: For one pattern, any of its isomorphic paths does not include another one.
- R6: Along the isomorphic paths, there is no branch until the diverging state.
- R7: Potential isomorphic paths can be overlapped and included by others. The algorithm for choosing isomorphic paths from potential ones is discussed in next section.

All the above rules are used in later experiments.

<table>
<thead>
<tr>
<th>New numbering</th>
<th>Corresponding old states</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial state</td>
<td>4'b0000</td>
</tr>
<tr>
<td>White</td>
<td>S2, S3, S4, S5, S7, S9</td>
</tr>
<tr>
<td>Pink</td>
<td>S6</td>
</tr>
<tr>
<td>Yellow</td>
<td>S1, S8</td>
</tr>
</tbody>
</table>

Figure 8: Three types of states

CDFA of \{\text{pattern, betters}\} has 10 states whose old serial number ranges from 0 to 9. To represent the 10 states, 4 bits are needed. Based on the requirements of IMPC, 2 states are converging states (yellow), 1 state is diverging state (pink) and others are common states. There is no state that belongs to both converging state and diverging state. Thus all states are needed to pain three colors.

Given the initial state is also S0, Huffman code is used in our numbering, as Table 2 shows.

Figure 9: Different numbering of the same CDFA
B. Greedy Algorithm for IMPC

Based on the rules of finding isomorphic paths, heuristic algorithms are used to choose isomorphic paths from potential ones to minimize the cost of CDFA.

We know that if an n-character-length isomorphic path is combined, n+1 states and n transitions are saved. The minimum cost of CDFA represented by states and transitions is the maximum number of states and transitions that can be saved by IMPC.

For example, pattern set \{pattern, betters, latten\} has several potential isomorphic paths, such as “tter” for “pattern” and “betters”, “tte” for all the three patterns. However according to the rules of finding isomorphic paths, “tter” and “tte” cannot be accepted and combined at the same time since they are overlapped with each other. Thus heuristic algorithms are required to make this decision.

In our work, we use greedy algorithm [19] for IMPC to choose isomorphic paths from potential ones.

For a given pattern set, suppose there are m potential isomorphic paths with their lengths \(l_1, l_2, \ldots, l_m\), and for the \(i\)th isomorphic path, there is \(t_i\) patterns that share it. Thus, for \(i\)th isomorphic path, total \((l_i+1)×t_i\) states and \(l_i×t_i\) transitions can be saved. Suppose that \(l_i>t_i\), it is named as value. However, some of the \(m\) potential isomorphic paths may conflict with others, just as the “tter” and “tte” in the above example.

Our greedy combination algorithm is applied as follows.

**Step 1**: Find the isomorphic path with maximum value among all potential ones and combine it. If the maximum value is 0, the algorithm is over.

**Step 2**: Find all the potential isomorphic paths that may conflict with the above one and delete them all(conflict means overlapped or contained).

**Step 3**: Reevaluate all values of left potential isomorphic paths, and go to Step 1.

Based on greedy algorithm for IMPC, only “tte” of the above example will be combined since its value is 9 while the value of “tter” is 8.

C. The Correctness of ACS

The path combination in DFA is a little complicated because of branch path and cross transitions. To classify all the situations are our future works. In this paper, to guarantee the correctness of ACS algorithm, we only choose the simplest path to combine, i.e., the isomorphic paths are not overlapped (R4 in section 5.1), not branched (R6) and not concluded by itself or others (R5).

Based on those rules, the isomorphic paths are the linear ones with only one corresponding diverging state each (R2). All path combinations are like the example in section 4.3.

D. Regular Expression Matching

In our work, patterns mean strings or regular expressions. For regular expression matching, our method can be used after the DFA is built up as a second step for optimization.

If no traditional DFA are used for matching regular expression, our method can be also used if the extended DFA keeps the features of DFA, that is, the next state is decided only by the current state and the incoming character.

In conclusion, our method is only related to optimize built DFA not related to match string or regular expression. It is the reason why our method can be embedded to other algorithms for better results.

![Pattern matching architecture based on CDFA](image)

**Figure 10**: Pattern matching architecture based on CDFA

E. ACS Related Architecture

Based on the features of ACS that fewer states and transitions are generated, pattern matching architecture can be further optimized. The new architecture is similar to the original one except the additional components for cache and state coloring. Theoretically, ACS can be applied to any pattern matching architecture with little modification.

In our work, we present an architecture based on ACS, which can be used with cache or without it. The architecture comes from the bitmap compression algorithm[11]. The architecture is shown in Figure 10, in which shadowed parts are used for CDFA while others are the original ones for DFA.

In DFA, the next state is determined only by current state and input character. In Figure 10, the upper half shows this procedure of bitmap compression in hardware. In this architecture, for each state in DFA, its next states are sequentially stored in “next state memory”. To access the next state, each state needs two additional properties. One is the base address that stands for the beginning address in “next state memory” and the other is a bitmap that has 256 bits to represent the accepted input characters from the state[11].

With these two properties, the next state can be accessed by current state and input by adding several operations. For ACS algorithm, all states are colored to identify three types of states. Especially for diverging states, the next state is not only determined by current state and input but also cached state. Therefore an additional “pre-index memory” is needed to transfer cached state (must be converging states) to corresponding diverging state (pink state). The transfer is unique because each converging state only corresponds to one diverging state which is mentioned in section 5.1. Compared to the accessed pink state, the current state (pink state) may know whether it is
the one corresponding to cached state. Thus the next state can be determined by the help of cached state.

Suppose there are \( n \) states and \( m \) transitions in traditional DFA for a given pattern set. The memory required in the architecture based on bitmap compression (upper half in Figure 10) is as follows (in bytes).

\[
\text{Mem}_{\text{DFA}} = n \left( \log_2 m \right) / 8 + 32 + m \quad (2)
\]

If ACS is used, there are \( n' \) states and \( m' \) transitions that can be saved. The memory required is as follows (in bytes).

\[
\text{Mem}_{\text{ACS}} = (n - n') \left( \log_2 (m - m') \right) / 8 + 33 + m - m' \quad (3)
\]

In equation 3, the maximum overhead of ACS (“pre-index memory” in Figure 10) is used, that is, all states are yellow states (in fact, yellow states are part of all states).

VI. PERFORMANCE EVALUATIONS AND RESULTS

A. Methodology

Because CDFA extends DFA model, it can be applied to all the algorithms and architectures based on DFA. Thus to evaluate ACS, two different methodologies are used in this section. The first way is to compare the real value of CDFA with the one of DFA. The second is to show the proportion that can be saved by using ACS, which will be valuable for the evaluation of known DFA based algorithms and architectures. The results make sense to both anywhere matching and anchored matching.

Two pattern sets are used for experiments. One is the pattern set of Snort, and details are given in section 3.3. The other is a set of email addresses (address set for short).

Address set comes from the internet containing 100k email addresses. We take it to reinforce the results of large scale pattern sets. Address set has 1.51M characters, in which average pattern length is 15 byte. One domain has about 10k addresses, while others contain 1 to 5k.

The pattern matching architecture is implemented as a module by verilog language on Xilinx FPGA.

B. Minimum Length of IMPC

Different minimum length (\( \text{minlen} \)), which is permitted length of IMPC, may affect the results of IMPC. For Snort pattern set, results are given in Figure 11.

In Figure 11, when \( \text{minlen} \) equals 2, only 21.4% states are used in CDFA to represent the original DFA (78.6% saved). We can find that there are fewer states if the shorter \( \text{minlen} \) is applied. It is obvious that more isomorphic paths can be found with shorter \( \text{minlen} \).

Similar result is shown in Figure 12 for address set. Only 8.9% states are used in CDFA when \( \text{minlen} \) equals 2 (91.1% saved). The result of address set is much better than Snort set, because there are more isomorphic paths in address set, such as domains of email addresses.

To evaluate the effect of CDFA for IMPC for smaller pattern sets, we partition pattern set into many smaller ones with the method in section 3.3. \( \text{minlen} \) of CDFA ranges from 2 to 8 and the number of subsets for Snort set is from 1 to 32.

The result of more subsets is the sum of the results in each subset. Our purpose of this representation is to compare the results with the one of Snort set. Individual result of each small set can be calculated.

Figure 13 show the trend of this case. We find that ACS with smaller pattern sets can result in worse results. It is reasonable that isomorphic paths can be more easily found in a larger pattern set than in smaller ones. In practice, the proper minimum length of IMPC is up to the requirement of memory usage.

C. Memory Requirement

Memory requirement of pattern set on CDFA may be various in different architectures. Here, the architecture of section 5.4 is used to show the efficiency.

Because the efficiency of architectures can affect the memory requirements, only proportions of saved memory are given as results. That is, if the architecture is memory efficient itself, the memory usage on it cannot reflect the real effect of algorithm. Thus, we take the methodology that proportions of saved memory by IMPC compared to the original architecture are the real effect of our algorithm.

Here \( \text{minlen} \) equals 2. Figure 14 shows the proportion of saved memory by using ACS. 63.2% memory can be saved with our method when Snort set acts as a whole. With smaller pattern sets (16 subsets of Snort set), still 47.6% memory can be saved. For larger pattern set, 84.0% memory can be saved in address set. From the statistics, we can find that ACS algorithm can save more memory if the pattern set is larger. It is also a feature to be used for bigger pattern set.
In our work, we intend not to compare our architecture with other ones. It is because our architecture is not the key of our work. The main point is that ACS algorithm and the example implementation can be used and embedded to other algorithms or architectures as a second optimization step for less memory usage if and only if those algorithms and architectures have the DFA or extended DFA as their basic model.

VII. CONCLUSIONS

In our works, pattern matching issues for deep packet inspection are analyzed in depth. For both anywhere and anchored matching, a lower boundary of cost for building DFA is presented. However for the lower boundary, there is still great potential for further optimization. The idea to break through the lower boundary, named isomorphic path combination (IMPC), is presented. It can correctly function on Cached DFA (CDFA) model. Furthermore, ACS algorithm based on CDFA for IMPC is proposed.

Experimental results show that ACS can reduce 78.6% states for Snort pattern set and 91.1% states for address set. As to memory requirement, 47.6% to 84.0% memory can be saved considering different size of pattern sets.

It is important to know that, the path combinational method can be embedded as a second step to almost all the algorithms based on DFA for better results.

ACKNOWLEDGMENT

The authors would like to thank Prof. Yibo Xue from Tsinghua University, and all reviewers that give in-depth, helpful and positive suggestions to the paper.

REFERENCES