Finite Automata in Pattern matching for Hardware based NIDS Applications – a Tutorial and Survey

C. Jasmine*, Dr. T. Latha**
Department of Electronics and communication Engineering,
*Mar Ephraem College of Engineering and Technology, Marthandam, India,
**St. Xavier’s Catholic College of Engineering, Nagercoil, India,

Abstract: Automata theory is the widely used theory for pattern matching because of its simplicity, speed and robustness. Since 1975, automata theory has been widely used for pattern matching with variety of improvements over the traditional matching. This paper presents a survey on the efficient usage of automata theory in the pattern matching applications, mostly hardware architectures for Network Intrusion Detection Systems (NIDS). Among the hardware architectures, memory architectures are widely used for their flexibility and scalability. In the memory architecture, the string patterns are compiled into Finite Automata and the corresponding state transition table is stored in the memory. If the number of patterns increases, the size of the Automata will increase. This results in more memory consumption. In this paper, we review the compilation of patterns into finite Automata and also the various state reduction techniques associated with it for reducing the memory requirement. Also we propose a new searching method based on the out degree based indexing to avoid the limitations of Content Addressable Memory and the hash based algorithms.

Keywords: Automata, Pattern, Network Intrusion Detection, Transition Table.

I. INTRODUCTION

Finite Automata or the state machine is a mathematical model to design computer software and sequential logic circuits. Finite Automata are useful to model many important hardware and software applications. The various applications like the Network Intrusion Detection System (NIDS), Bio Informatics uses the Deterministic Finite Automata (DFA) for compiling large set of patterns. In the NIDS application, the patterns refers to the malicious attack patterns that harms the system or the entire network. In the field of Bioinformatics, the DNA sequences are the patterns.

The Network Intrusion Detection System aims at detecting the malicious network packets by inspecting the contents of the packet against the malicious patterns. A pattern is a group of characters that exist along with the malicious programs. Pattern matching is the process of matching the incoming packet contents against the known patterns of the malwares.

Figure 1 A Finite Automata

The security attacks are increasing nowadays. Everyone is aware of the Twitter and Yahoo mail server attacks, happened recently. Numerous viruses, worms and Trojans are giving threats to the internet. To accommodate huge amounts of patterns and to meet the high-speed requirements of current networks, many hardware architectures are proposed to accelerate pattern matching. Among hardware architectures, memory architectures have been widely adopted because of their flexibility and scalability. There are
two steps in designing memory architectures for pattern matching. The first step is to compile attack patterns into a deterministic finite automaton (DFA). Figure 1 shows three compiled patterns, “pcd”, “fqh” and “bcm”. 1 is the starting state and 7, 4 and 10 are the ending states. The solid lines show the transition between the states and the dotted lines shows the failure transitions. All the failure transitions are not given for simplicity.

II. Finite Automata Based String Matching Algorithms

In this section, we review the various string matching algorithms based on Finite Automata Theory especially for String matching in Network Intrusion System Applications. Five popular string matching algorithms has been reviewed. The performance of the 5 algorithms are discussed in Section 3 and the conclusion is given in Section 4.

2.1 Aho Corasick Algorithm

The Aho Corasick Algorithm[10] is popular algorithm for string matching [1],[2],[3],[4],[5],[6]. Because the number of states is reduced in Aho Corasick Algorithm than the standard DFA, Which in turn reduces the memory size. Consider the following transition diagram for matching the strings “bcdf” and “pcd”.

The valid transitions are represented by the solid lines and the failure transitions are represented by the dotted lines. Consider the string “bcp”. The DFA starts from the state 0. By taking the input symbol ‘b’, it moves to the state 1. From state 1, by taking the input symbol ‘c’, it moves to the state 2. From state 2, there is no valid transition for the symbol ‘p’. Hence the DFA takes the failure transition and goes to the state 0 and start reading the next input. The DFA never terminates when it reads an invalid symbol. The final states are compiled patterns[12],[13],[14],[15] and also, to increase the searching speed[16],[17],[18].
represented by double circles. 
The corresponding state transition table for the Figure 3.1 is shown in Table 3.1. Here the match vector indicates whether the string matches and the number in the match vector indicates the number given for the pattern.01 for the first pattern and 10 for the second pattern.

<table>
<thead>
<tr>
<th>Input</th>
<th>Next State</th>
<th>Failure State</th>
<th>Match Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>b</td>
<td>S1</td>
<td>00</td>
</tr>
<tr>
<td>S1</td>
<td>p</td>
<td>S5</td>
<td>00</td>
</tr>
<tr>
<td>S2</td>
<td>c</td>
<td>S2</td>
<td>00</td>
</tr>
<tr>
<td>S3</td>
<td>d</td>
<td>S3</td>
<td>00</td>
</tr>
<tr>
<td>S4</td>
<td>f</td>
<td>S4</td>
<td>00</td>
</tr>
<tr>
<td>S5</td>
<td>c</td>
<td>S6</td>
<td>00</td>
</tr>
<tr>
<td>S6</td>
<td>d</td>
<td>S7</td>
<td>00</td>
</tr>
<tr>
<td>S7</td>
<td>g</td>
<td>S8</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 3.1 Transition Table for AC Machine

The Aho Corasick Algorithm suffers from a high memory access requirement [7]. In worst case, the average number of memory access required per input character is 2. Hence the system throughput decreases. Another problem with the Aho Corasick algorithm is that, parallelism is not possible. The processing of one input character per clock will create bottle neck for the gigabit networks.

2.2 Jump Ahead Aho Corasick Algorithm [JACK]

The throughput of the Aho Corasick algorithm can be improved by scanning multiple characters at a time, making it parallel. In [7], the Aho Corasick algorithm was designed to read ‘k’ input characters at a time.

In our NFA, we consider the group of k characters as a single symbol. This NFA jumps k characters ahead and hence the name Jump Ahead Aho Corasick Algorithm. The Figure 4.1 Shows the NFA using JACK Algorithm. Here k=4. After reading 4 characters, NFA jumps to the next state. If the character group is less than 4, it is taken as the tail transition and is represented as dotted lines. In this algorithm, if the string appears in middle of the character group, it will not be detected. For example, if the input string is, “thechanges”, the NFA can’t detect the string “chan”, because the k characters in that input string will be taken as “the”, “hang” and”es”. In order to avoid this, k machines are deployed each of which scans the text with one byte offset. These machines are virtual machines. In this way, by changing the state after reading 4 characters instead of single character, the memory usage is reduced.

2.3 Parallel Failure less Aho Corasick Algorithm [PFAC]

The PFAC in [8],[9] increases the throughput of the string matching to an extent. This algorithm is commonly used in Graphical Processing Unit [GPU]. In this a GPU thread is allocated to each byte of the input stream, which identifies the virus pattern starting at the starting location of the GPU thread. In the conventional AC state machine [10], if there is a failure transition, it backtracks in order to read the next input string. If we use PFAC, there is no need to backtrack, since there is no failure transition. Hence all the failure transitions of the Aho Corasick Algorithm can be removed, which reduces the memory size. Since, the failure transitions are removed, it is called Failure less AC state machine.

GPU’s uses shared memory and the size of the shared memory is limited and so we can’t store all the virus patterns. Hence, the virus patterns are grouped by their prefix similarity and the prefixes are shared, which
reduces the size of the memory. The Figure 5.1 shows the failure less AC state machine for the pattern, “bede”, “bbda” and “bcfg”.

![Figure 5 PFAC State Machine](image)

Figure 5 PFAC State Machine

Whenever, there is an invalid transition, the execution of the thread will stop. Consider the input string, “bcm”, after reading the input ‘c’ and reaching the state 2, there is no valid transition for the symbol ‘m’. Hence the thread will terminate immediately.

Hence the thread will terminate immediately. The Figure 5.2 shows the PFAC algorithm and threads. The boundary detection problem is eliminated in PFAC algorithm [8]. In PFAC, both the worst case and average case performance are good while compared to the conventional AC algorithm.

![Figure 6 PFAC Threads](image)

Figure 6 PFAC Threads

In PFAC algorithm, each byte of an input stream is handled by a separate thread. Whenever, there is an invalid transition, the execution of the thread will stop. Consider the input string, “bcm”, after reading the input ‘c’ and reaching the state 2, there is no valid transition for the symbol ‘m’. Hence the thread will terminate immediately. The Figure 5.2 shows the PFAC algorithm and threads. The boundary detection problem is eliminated in PFAC algorithm [8]. In PFAC, both the worst case and average case performance are good while compared to the conventional AC algorithm.

2.4 Bit Split Algorithm

Aho Corasick state machine requires 256 outgoing edges for pointing the next states. The nodes nearer to the root need more than 200 next pointers where the nodes near the leaf needs one or two pointers. Hence many pointers are wasted. In [2],[6] the Bit Split Algorithm is proposed to reduce the wastage of pointers. Here the Aho Corasick state machine is split into an optimum number of binary state machines[6].The Aho-Corasick state machine can be split into several state machines so that one state machine takes care of a subset of input bits. This will reduce the number of outgoing edges, which in turn reduce the memory requirement. Because the bit-split algorithm removes most of the wasted edges, the total storage required is much smaller than that of the starting machine. But the Bit slit algorithm in [2],[6] are not designed to handle more number of input patterns. The Figure shows the bit split algorithm with Binary FSM.

![Figure 7 Bit Split Architecture](image)

Figure 7 Bit Split Architecture

In this algorithm, the input packet characters are converted into binary streams of 8 bits. The ith bit of each character is given to a binary FSM which performs the string matching. Finally the intersection of matches is taken and the Match ID could be generated using Multiplexer.
### 2.5 Merge FSM Algorithm

The virus string patterns may have common substrings which results in similar transitions while compiling in to a finite state machine. These states are not equivalent states as we saw in the section 2. So these states can’t be merged directly.

![Figure 8 FSM to Merge](image)

In [11], the algorithm to merge those states has been proposed. Consider the AC state machine In [11], the algorithm to merge those states has been proposed. Consider the AC state machine for the virus patterns, “amfg” and “bmfm” in Figure 6.1. Here, the state 2 and state 6 are similar, since they have identical transitions m and identical failure transition to state0. Similarly, the state 3 and state 7 are similar. When we merge these two states, it will give erroneous output. Therefore these states are called Pseudo equivalent states [11]. These Pseudo equivalent states can be merged and named as state 26 and state 37 where, state 26 indicates state 2 and state 6 and the state 37 indicates the two different states, state 3 and state 7 and the resulting Merge Finite State machine [FSM] is shown in the Figure 6.2.

![Figure 9 Merged FSM](image)

This FSM will accept the string “amfn” mistakenly. So this FSM may produce false positive results. In order to avoid this error, when the state 26 is reached, the AC state machine must know whether it is state 2 or state 6. This can be done by keeping the precedent state of the state 26, ie, the precedent state of the state 26 is the state 1. The AC state machine should take state2 from state26 if the precedent state is 2 and it should take state 6 if the precedent state is 5. So memorizing the precedent state is needed. The precedent state can be identified by using the path vector variable pathvec and the ifFinal bits for the non merged states, the path vector will be the number given for that pattern, matched by taking that particular path and for the merged states, the path vector will be the union of the path vectors of the precedent states. Another variable named, prereg is used to trace the previous path vector in each state. The prereg is calculated by performing a bitwise AND operation on the path vector of the next state and the current state.

During transition, if the bits of the prereg becomes zero, the machine will take failure transition. the ifFinal bit of the ending state will be 1 and 0 for the other states. The pathVec_ifFinal pair is given like 01_1 in the merge_FSM. In this way, the number of states are reduced and which in turn reduces the memory required to store the virus pattern compiled in the form of AC state machine.

### III. SEARCHING METHODS

The malicious patterns which are compiled as Finite Automata are stored in the form of transition tables in the memory. Whenever the input packet arrives, it is needed to compare the input pattern with the patterns we have stored in the form of transition tables. For that the mostly used searching methods are the Content Addressable Memory (CAM) based searching and the Hash based searching. Both searching methods has been discussed below.
3.1 CAM based searching method

The data stored in the memory can be accessed by using the address of specific locations. But this kind of access by using the memory address will be a challenging task for the applications which needs faster access. The time taken to access an item from the memory can be reduced considerably if we access it by using the content of the item rather than address. Such kind of memory is called Content Addressable Memory (CAM). Rather than Random Access Memory (RAM), where the user access data by providing address, Content Addressable Memory (CAM) uses the content and provides the address. The CAM searches in one clock cycle throughout the memory and returns the address [19]. There is another type of Cam called Ternary CAM (TCAM). The term “ternary” refers to the memory’s ability to store and query data using three different inputs: 0, 1 and X. The “X” input, which is often referred to as a “don’t care” or “wildcard” state, enables TCAM to perform broader searches based on pattern matching, as opposed to binary CAM, which performs exact-match searches using only 0s and 1s. The Figure 10 shows the conceptual view of a CAM.

3.2 Limitations of CAM

The low memory density of CAMs while compared to SRAMs and DRAMs and its costly implementation makes it not suitable for the applications where more memory usage is not a big issue. Also, the CAM consumes more power while compared to RAMs due to its parallel searching and match line sensing.

3.3 Hash based searching method

The hash based searching methods are used to search a particular memory address without collisions. Hash functions are used in hash tables to quickly locate a data record. Hash functions are widely used in Bloom Filters, invented during 1970’s to test whether an element belongs to a set or not. Bloom filters are applied in networking [21] for network packet routing applications. Bloom filters suffered due to false positives as the number of items increased. Also there is no possibility of deletion in bloom filters. In order to avoid this, counting bloom filter [22] came into existence. Later perfect hashing has been proposed in [20] for Graphic Processing Units. But the computational complexity of perfect hashing is more. In order to avoid these limitations of CAM and Perfect Hashing methods, we are proposing a new searching method based on out degree based indexing.

IV. PROPOSED SEARCHING METHOD

The proposed method is based on out degree based indexing. In this method, the known malicious patterns are compiled using the Merge FSM algorithm and the corresponding transition table is stored in the Memory. From the Table 2, the Merge FSM algorithm is known to provide comparatively better throughput and less memory utilization than other similar algorithms. Consider the automata shown in the Figure 11, having three patterns “ab”, “ac” and “de”. The state S0 is
having out _degree as 2, S1 is having 2, S4 is having 1 and S2, S3, and S5 are not having any out _degree. Out _degree is the number of outgoing edges from a particular node.

Table.2

<table>
<thead>
<tr>
<th>Platform</th>
<th>Memory Efficiency</th>
<th>Memory Utilization (Memory/Char)</th>
<th>Throughput (Gbps)</th>
<th>Memory (Kb)</th>
<th>Char num</th>
<th>Approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPGA</td>
<td>0.27</td>
<td>2867</td>
<td>0.45</td>
<td>188</td>
<td>39.3k</td>
<td>AC Algorithm[10]</td>
</tr>
<tr>
<td>FPGA</td>
<td>0.67</td>
<td>227.50</td>
<td>0.45</td>
<td>41k</td>
<td>25k</td>
<td>JAC[7]</td>
</tr>
<tr>
<td>FPGA</td>
<td>0.46</td>
<td>~0.00</td>
<td>0.45</td>
<td>~2.5k</td>
<td>41k</td>
<td>PEAC[8,9]</td>
</tr>
<tr>
<td>FPGA</td>
<td>0.88</td>
<td>148</td>
<td>0.45</td>
<td>3.6k</td>
<td>3.6k</td>
<td>BtSFSm[2,6]</td>
</tr>
</tbody>
</table>

The proposed architecture is shown in the Figure 12. It consists of a search table, next state memory and a search control block. The search table consists of the number of outgoing edges and the search address of the next state memory which is holding the character and the corresponding next state. For the automata shown in the Figure 11, the search table is shown in the Table 3 and the next state memory is shown in the Table 4. Initially the current _state is 0. Here, depending upon the out _degree, the search address is fixed. So when we want to search for a particular pattern, for example the pattern “ab”, the search table shows the out _degree for the initial state S0 is 2 and the corresponding search address is 0.

Table 3 Search Table

<table>
<thead>
<tr>
<th>state</th>
<th>Outgoing_edge</th>
<th>Search_addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

Figure 11 Input Automata

Table 4 Next state memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Char_out</th>
<th>Next_state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>d</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>s</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

So the first character “a” of the input pattern “ab” is directly compared with the character present in the searching address 0 and the corresponding next _state has been identified. Now the next _state becomes the current _state for the next comparison. This current _state provides the address of the search table. During this repeated process, if the outgoing _edge is 0, then it is known that, the particular state is the ending state and a match is found, telling that it is a malicious pattern.

http://piserjournal.org/
If the pattern to be searched is “de”, then we need one more block called as the search control block shown in Figure 13. Initially the current state is 0 and the input Character is “d”. The address 0 which is given by the current state is having the out_degree 2 and the search address as 0. Now the input character “d” is compared with the character present in the search address 0 of the next state memory. But the character present in that address is “a”. So the searching is now done by the search control block. In the search control block, there is a counter which counts whenever there is a comparison, done by the comparator. If the output of the first comparison is not equal, then the counter will be incremented. The output of the counter is then compared with the out_degree of the state 0. If the count is less than the out_degree, then the search address is incremented and now the search address is 1 and now the input character “d” is compared with the character present in the address 1 of the next state memory and there is a match and the corresponding next state 4 is taken and the current state is updated. Now the searching is done in the search table with the address 4 given by the current state and the corresponding out_degree is 0. This indicates that the pattern “de” matches and hence it is a malicious pattern. Here the searching is purely based on the number of outgoing edges and there is no need for a CAM or a complex hash function for searching purpose.

V. DISCUSSION

From the review of the five Finite Automata Based searching algorithms, the Merge FSM seems to be giving better results. The performance comparison is done and it is shown in Table 2. Here the number of characters used for string matching, memory occupied by the string patterns, throughput, memory utilization which is the measure of number of memory bytes occupied by the individual characters and the memory Efficiency of the above five Algorithms has been compared. In the proposed method, the malicious patterns are compiled using Merge_FSM algorithm and stored in the form of transition table as next state memory. Instead of using the CAM which is a costly memory and the complex perfect hashing, we have proposed a effective searching method based on the out degree of the states in the FSM. Our searching method is less complex than the conventional hashing.
techniques and cheaper than the CAM. The proposed architecture has been implemented in FPGA.

VI. CONCLUSION

In this paper, we have presented a tutorial and survey on the Finite Automata Based string matching for Hardware based NIDS. The Five commonly used algorithms have been compared and the performance, based on various parameter measures is also given. Finite Automata based string matching technique is attracting the researchers since it helps in reducing memory size and also it is suitable for Memory Architectures. Also we have proposed a new searching method based on the out degree of the states in the FSM. This proposed searching method is cheaper when compared to CAM based searching and it is simpler when compared to the perfect hashing and other complex hashing methods.

REFERENCES


[20] Cheng- Hung Lin, Cheng Hsung Liu, Shih Chieh Chang, Wing-Kai hon, Memory efficient Pattern Matching Architectures using
Perfect Hashing on Graphic Processing Units, 2012 Proceeding, IEEE Infocom.
