International Journal of Engineering Science Invention
ISSN (Online): 2319 – 6734, ISSN (Print): 2319 – 6726
www.ijesi.org Volume 2 Issue 6 ǁ June. 2013 ǁ PP.90-97

Hardware Based Multipattern Matching Using Non Deterministic Finite State Automata

D.Nithya, J.Sharmilee, Dr. T.Manigandan
(M.E .VLSI Design Nandha Engineering College Erode-52.)
(Associate Professor, Nandha Engineering College Erode-52.)
(Principal , P.A.College of Engg.& Technology, Pollachi).

ABSTRACT: *The covered encoding scheme for the TCAM based implementation of the Aho-corasick multipattern matching algorithm is used in Network Intrusion detection Systems (NIDS).The information of the failure transitions of the Aho-corasick Non deterministic Finite state Automata (NFA) is implicitly captured in the covered state encoding and the failure transitions are completely eliminated, the Aho-Corasick NFA can be implemented on a TCAM with smaller number of entries than other schemes. The modified Aho-Corasick NFA for multicharacter processing, which can be implemented on TCAM using the covered state encoding. The implementation of modified Aho-Corasick NFA using the Covered state encoding is superior to other schemes in both Memory requirements and lookup speed.. For Snort rule sets, the new algorithm achieves 21% of memory reduction compared with the traditional Aho–Corasick algorithm and can gain 24% of memory reduction by integrating the approach to The bit-split algorithm which is the state-of-the-Art memory-based approach.*

INDEXTERMS: ***Finite state Automata, TCAM, Multipattern matching.****(keywords)*

I. INTRODUCTION

For the past few years, a tremendous increase in the frequency and sophistication of attacks on the Internet. There have been notorious viruses/worms like Code Red, Nimda and Slammer etc. By exploiting the security flaws in operating systems, underlying network protocols and different software implementations, attackers bring down significant parts of the Internet in a matter of hours using distributed co-ordinated attacks aided with an ever increasing population of zombie machines.

With increased growth in malicious network activity, Network Intrusion Detection Systems (NIDS) are being devised and deployed to detect the presence of any malicious or suspicious content in packet data. Traditional software-based NIDS architecture fails to keep up with the throughput of high-speed networks because of the large number of patterns and complete payload inspection of packets. This has led to hardware-based schemes for multipattern matching.

To operate SNORT-like intrusion detection systems at multi-gigabit rates using hardware acceleration, one possibility is to use Ternary Content Addressable Memories (TCAM). TCAMs are widely used for IP header based processing such as longest prefix match. TCAMs can also be used effectively for the pattern matching functions needed in intrusion detection systems.

Ternary Content Addressable Memory (TCAM) is a type of memory that can perform parallel search at high speeds. A TCAM consists of a set of entries. The top entry of the TCAM has the smallest index and the bottom entry has the largest. Each entry is a bit vector of cells, where every cell can store one bit. Therefore, a TCAM entry can be used to store a string. A TCAM works as follows: given an input string, it compares this string against all entries in its memory in parallel, and reports one entry that matches the input.

High-speed pattern matching is required for a wide variety of other equally critical applications, including scanning through large data-sets (logs) for data mining operations, low latency XML switching, DNA sequence matching etc. Of greater interest is the use of patternMatching in next-generation network monitoring applications, including but not limited to, stateful packet inspection for QoS management, VOIP filtering, bandwidth metering, optimalcache replication etc. However, we limit our focus here to network-based IDS/IPS for virus/worm detection.

II. AHO CORASICK DETERMINISTIC FINITE STATE AUTOMATA

In the DFA, the dotted lines represent transitions, called by cross transitions, which are newly added by eliminating failure transitions. Shaded states represent the pattern matching states called output states. The trivial transitions going to the initial state are omitted in the figure.

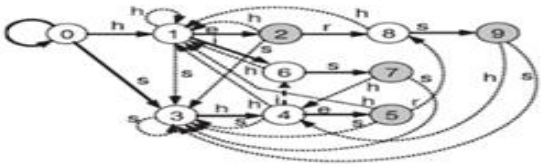

Figure 1.Aho-Corasick Finite Automata For DFA

The finite state machines for the set of patterns {he, she, his, hers} built by the AC algorithm is shown in Figure 4.1. The pattern matching can be performed using either the NFA or the DFA. To match a string, one starts from the initial state (usually 0). If a goto transition or a cross transition is matched with an input in the current state, the current state is moved along the matched transition. Otherwise, for a DFA-based matching, the current state goes back to the initial state and the matching process repeats for the next state.The AC DFA requires a large amount of memory in a straightforward RAM-based implementation that keeps a table of pointers to next states for every input since the table contains also trivial transitions which go back to the initial state.

III. TCAM BASED HARDWARE ARCHITECTURE

The architecture shown in Figure 2 consists of a TCAM, SRAM, and a logic. Each TCAM entry represents a lookup key, which consists of current state and input, and has corresponding data, which is the next state, in the SRAM whose address is given by the TCAM output. Two registers current state and input are initialized to the state 0 and the start data of the input buffer, respectively. If there is a matching entry for the state and input value in the TCAM, the TCAM outputs the index of the matching entry and then the SRAM outputs the next state data located in the corresponding location. Because a TCAM has "don't care" bits, multiple entries can be simultaneously matched and when this is the case, the index of the first matched entry is outputted. If there is no such match in the TCAM, the next state is the initial state.

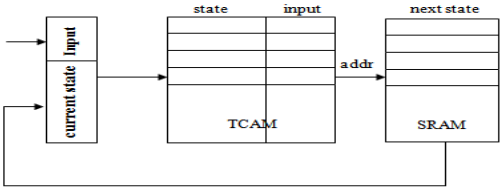

Figure 2. TCAM Based Hardware Architecture

The architecture consists of a TCAM, SRAM, and a logic. Each TCAM entry represents a lookup key, which consists of current state and input, and has corresponding data, which is the next state, in the SRAM whose address is given by the TCAM output. Two registers current state and input are initialized to the state 0 and the start data of the input buffer, respectively. If there is a matching entry for the state and input value in the TCAM, the TCAM outputs the index of the matching entry and then the SRAM outputs the next state data located in the corresponding location. Because a TCAM has "don't care" bits, multiple entries can be simultaneously matched and when this is the case, the index of the first matched entry is outputted. If there is no such match in the TCAM, the next state is the initial state.

At every state transition, an input is advanced to the next input and the next state value is stored into the current state register. Each TCAM entry represents a transition in the state machine. The number of TCAM entries is equal to the number of transitions and independent of the number of states. For a transition, g(s, i) = t where s is a current state, i is an input, and t is the next state, we will simply represent a pair of TCAM and SRAM entry by the combined form (s, i|t).

IV. MEMORY OPTIMIZATION

The next transitions to states at higher depths in the state machine could be reduced. If we suffix the state id representation of every state by the last character that caused a goto transition into that state, then we can remove the next transitions to states at depth two in the state machine. Hence, a state at depth one is represented as "i1,….., i8" where i1,…., i8 is the binary representation of the input character that causes a goto transition from the root node to that state. This state id representation would then match all those states in the state machine that is reached by a go to transition on character i1,….., i8.,from some other state. Thus all next transitions to states at depth two in the state machine are subsumed by the goto transition to depth two state, along similar lines of argument as above, and hence can be eliminated from the TCAM.

V. SEARCH SPEED ENHANCEMENT

Search Speed Enhancement is functionally similar to the state machine in Figure 2 except that the transitions are now on four characters each. We call this state Machine, a multi-character (compressed) state machine. The TCAM entries now contain four characters in the input field requiring 32 bits for their representation, and the input pointer is now incremented by 4 for every state transition. Hence we get a speedup of upto four, provided the input bus to the TCAM is wide enough.

VI. TCAM/SRAM ENTRIES

Each TCAM entry represents a transition in the state machine. The number of TCAM entries is equal to the number of transitions and independent of the number of states. SRAM address is given by the TCAM output. So the TCAM/SRAM entries plays a major role.

A. No Optimization

The diagram showing the no optimization scheme is given as Table 1

Table 1 TCAM / SRAM Entries for No Optimization

In the entries with no optimization, the left entries are goto transitions in the NFA and the right entries are cross transitions. The entries marked as "X" are cross transitions eliminated when the depth of the state encoding is increased. The entries marked as "√" are goto transitions whose current state field is replaced with the stateincluding "don't care" when the depth of the state encoding is increased. Note that the TCAM entries containing "don't care" must be located in the last positions since they have the lowest priority.

No optimization

cs	in	ns
0	h	1
0	s	3
1	e	2
1	i	6
2	r	8
3	h	4
4	e	5
6	s	7
8	s	9

cs	in	ns
1	h	1
1	s	3
2	h	1
2	s	3
3	s	3
4	h	1
4	s	3
4	i	6
5	h	1
5	s	3
5	r	8
6	h	1
7	h	4
7	s	3
8	h	1
9	h	4
9	s	3

B. Alicherry's Encoding

A novel method called Compact DFA is used to compress the DFA entries in the TCAM-based implementation. Although Compact DFA is originated from essentially the same idea as Alicherry's encoding which eliminates cross transitions by suffixing the state encoding by the prefix string from the root node to that state, it provides more efficient encoding scheme than Alicherry's encoding by constructing the common suffix tree and encoding the states, and it eliminates all the cross transitions and it is shown in Table 4.2. Compact DFA state encoding is still nonoptimal and its building algorithms are more or less complex since the Compact DFA is built from AC DFA.

Table 2 TCAM / SRAM Entries for Alicherry's Encoding Alicherry's Encoding

	cs	in	ns
√	1	e	2
√	1	i	6
	2	r	8
√	3	h	4
	4	e	5
	6	s	7
	8	s	9
x	4	i	6
	5	r	8
x	7	h	4
x	9	h	4
	*	h	1
	*	s	3

depth -1

	cs	in	ns
√	2e	r	8r
√	4h	e	5e
√	6i	s	7s
	8r	s	9s
x	5e	r	8r
	*h	e	2e
	*h	i	6i
	*s	h	4h
	**	h	1h
	**	s	3s

depth -2

cs	in	ns
8er	s	9rs
*he	r	8er
*sh	e	5he
*hi	s	7is
**h	e	2he
**h	i	6hi
**s	h	4sh
***	h	1-h
***	s	3-s

depth -3

VII. AC NON DETERMINISTIC FINITE STATE AUTOMATA

A proposed system is a new state encoding scheme in TCAM-based implementation of AC NFA. We can reduce the number of TCAM entries to the number of goto transitions in the NFA by fully utilizing the "don't care" feature of TCAM using a covered state encoding.

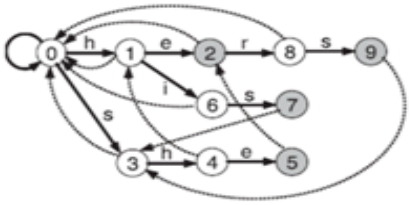

Figure 3. Goto and Failure Transitions in NFA

For an NFA-based matching, the current state is moved along its failure transition and the matching process repeats for the current input. The DFA examines each input only once while the NFA may examine each input several times along the failure transition path is shown in Figure 5.1. In matching a text string of length n, the DFA makes n state transitions and the NFA makes fewer than 2n state transitions.

VIII. COVERED STATE ENCODING

Since the AC NFA has a smaller number of transitions than the AC DFA, the NFA can be implemented with smaller TCAM entries than the DFA. The number of TCAM entriesin the NFA-based implementation is the sum of the number of goto transitions and the number of nontrivial failure transitions in the NFA.

In the NFA-based architecture, 1-bit field F indicating a failure transition is added in each SRAM entry. If an entry is associated with a failure transition, its F is 1 and its input field is "*"which can match with any input value. If the matched transition is a failure transition, or F= 1, an input is not advanced and current input is used again at the next matching. One character may be repeatedly processed along the states in the failure transition path until a non failure transition is matched (F= 0) or a state goes back to the initial state. The Table 3 shows the TCAM/SRAM entries in NFA.

Table 3 TCAM/SRAM entries in NFA

NFA			
cs	in	ns	F
8	s	9	0
6	s	7	0
4	e	5	0
3	h	4	0
2	r	8	0
1	e	2	0
1	i	6	0
0	h	1	0
0	s	3	0
9	*	3	1
7	*	3	1
5	*	2	1
4	*	1	1

	Covered state Encoding		
	cs	in	ns
0	8	s	9
1	6	s	7
2	4	e	5
3	3,7,9	h	4
4	2,5	r	8
5	1,4	e	2
6	1,4	in	6
7	all states	h	1
8	all states	s	3

cs: current state
in: input
ns: next state
F:Failure transitions

If we exploit the "don't care" feature of a TCAM, we can encode states so that a code covers some other codes. For example, code ⁎⁎⁎⁎ covers all of 4-bit codes and code 11⁎⁎covers four codes 1100, 1101, 1110, and 1111. We call a code each bit of which consists of 0 or 1 by a unique code, and a code each bit of which consists of 0, 1, or "⁎" by a cover code. Since the next state field in the SRAM cannot store "⁎" it should be represented by a unique code. However, the current state field in the TCAM can store a cover code.

The failure transition graph is a graph consisting of only failure transitions in the AC NFA. In a failure transition graph, the set of all the predecessors of a state s is denoted by PRED(s) and the set of all the successors of a state s is denoted by SUCC(s). Fig. 5.2 shows SUCC(s) and PRED(s) for a state s in a failure transition graph. If current state fields of TCAM entries associated with each state t are replaced with a cover code to cover not only state t but also all its predecessors PRED (s), we can simultaneously examine all the entries of s and SUCC(s) for a current state s since s is a predecessor of SUCC(s) and the failure transition entries are not needed any longer.

A. Covered State Encoding Algorithm

An algorithm for performing the covered state encoding for the AC NFA consists of four stages:

Step 1. Construct a failure tree.
Step 2. Determine the size (or dimension) code of each state.
Step 3. Assign a unique code and each state.
Step 4. Build the TCAM entries.

B. Evaluation of Covered State Encoding

In the Fig. 5.4, the state transitions for the input sequence "shershiss" is shown. The initial state is 0. After processing three input characters "she", the state becomes 5. Although there is no entry of state 5, the TCAM entry of state 2 (c _code= 100⁎) is matched with input r in state 5 (u _code= 1001) and the state becomes 8. After processing the following input sequence "shiss" from state 8, the state becomes 3. Boxes in Fig. 5.4 represent the cases that an input is matched with the TCAM entry of a failure transition state of the current state.

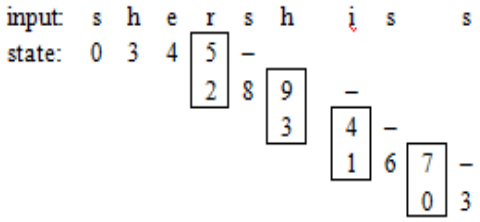

Figure 4. State Transitions for a Given Input Sequence.

The Snort rule set version 2.8 and ClamAV version 0.96 antivirus signature are used to evaluate the TCAM memory requirement of the covered state encoding. The Snort rule set has 5,169 patterns whose average length is 16.7 and ClamAV signatures have 30,385 patterns whose average length is 67.4. We compare the implementations of the AC DFAs using Alicherry's encoding of depth m (denoted by DFA-m) and the Compact DFA with the implementation of the AC NFA using the covered state encoding (denoted by AC NFA-c).Let T_g and T_f be the number of goto transitions and the number of failure transitions in the AC NFA, respectively. In the AC NFA-c, the TCAM entries consist only goto transitions and the number of TCAM entries is $T = T_g$. The number of TCAM entries in the Compact DFA is the same as that in the AC NFA-c since the Compact DFA eliminates all the cross transitions. Let $T_{c,i,}$ be the number of depth I transitions in the AC DFA. In the DFA-m, the number of TCAM entries is $T = Tg + \sum_{i=m+1}^{dmax} Tc,i$ where d_{max} is the maximum depth.

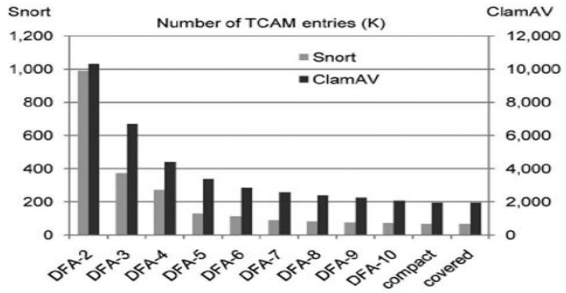

Figure 5 Comparison of the number of TCAM Entries

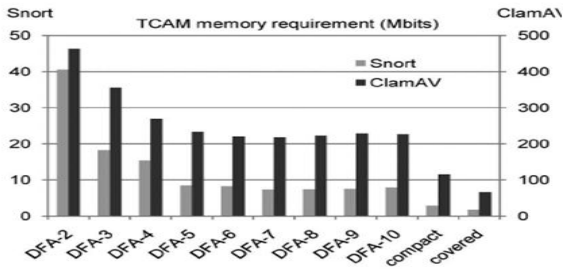

Figure 6 Comparison of the TCAM Memory Requirements

Fig. 5 shows the numbers of TCAM entries in various DFA-m ($2 \le m \le 10$) Compact DFA, and AC NFA-c for the Snort rule set and ClamAV signatures. As m increases, the number of TCAM entries in DFA-m approaches that of in the DFA-m, the number of bits required for the state encoding is $E = \log_2 S + 8(m - 1)$ bits where S is the number of states .In the AC NFA-c, the state code width is $E = \log_2 S + \delta$ where δ is the number of extra bits required in the covered state encoding and depends on the patterns. In the covered state encoding, the number of state bits E is equal to the number of states S at the worst case. A set of n strings $\{a_1b_1, a_2a_1b_2, \ldots, a_na_{n-1} \ldots a_1b_n\}$ where $bi \neq bj$ if $i \neq j$, is an example of the worst case. This case, however, is almost impossible for practical rule sets. The state code width in the Compact DFA is also represented by $E = \log_2 S + \delta$, where δ is the number of extra bits required in the Compact DFA. In the AC NFA-c, the number of extra bits δ (0, 5) is much smaller than log2 S. In the Compact DFA, however, the extra bit width δ (18, 30) is comparable to $\log_2 S$.

The width of a TCAM entry is W = E+ 8 since a TCAM entry consists of a current state and an 8-bit input data. The TCAM memory requirement is M = T .W = T . (E + 8) bits. Thus, the TCAM memory requirements of AC NFA-c, CompactDFA, and DFA-mare given by following equations:

$M_{covered} = M_{compact} = Tg .(\log_2 S + \delta + 8)$ (1)

$M_{DFA\text{-}m} = Tg + T = Tg + \sum_{i=m+1}^{dmax} Tc,i \ (\log_2 S + 8(m - 1) + 8)$ (2)

The SRAM memory requirement is T .E bits since an SRAM entry consists of the next state field. The DFA-1 is omitted in this figure due to its large value. This figure 5.6 shows that the TCAM memory requirement in the covered state encoding is much less than those in the other schemes. The code widths for Snort and ClamAV is shown in Table 5.3.In the Snort rule set, the TCAM memory requirement of DFA-m has the minimum value (=7.3 Mbit) when m = 7,which is 4.3 times of $M_{covered}$ (=7.3 Mbit) and the ClamAV signatures have the minimum $M_{DFA\text{-}m}$(=218 Mbit) when m =7, which is 3.3 times of $M_{covered}$ (= 66 Mbit). $M_{covered}$ is about 58 percent of $M_{compact}$ for both Snort and ClamAV.Thus, the TCAM memory requirement of AC NFA-c is the smallest. In the AC NFA-c, the memory utilization per character is 2.47 B/char for the Snort rule set and 4.04 B/char for ClamAV signatures.The commercially available TCAMs have the fixed entry widths which are multiples of a specific size D (either 36 or 40 bits). When the required TCAM entry width is W, the actual TCAM entry width is [W/ D).D. In the AC NFA-c, the actual TCAM entry widths are D for both Snort and ClamAV since the required entry widths are W = 17 þ +8 =25 for Snort and W = 26 + 8= 34 for ClamAV. In the DFAm (m - 3) and CompactDFA, the actual TCAM entry widths are 2D or more since the TCAM entry widths are larger than 40. Thus, the actual TCAM memory requirement in the AC NFA-c is also the smallest since the number of TCAM entries and the TCAM entry width in the AC NFA-c are smaller than in the other schemes

.

IX. MERGE _FSM

The merg_FSM is a different machine from the original state machine but with a smaller number of states and transitions.A direct implementation of merg_FSM has a smaller memory than the original state machine in the memory architecture. Our objective is to modify the AC algorithm so that we can store only the state transition table of merg_FSM in memory while the overall system still functions correctly as the original AC state machine does. The overall architecture of state traversal machine is shown in Fig. 7. The new state traversal mechanism guides the state machine to traverse on the merg_FSM and provides correct results as the original AC state machine.

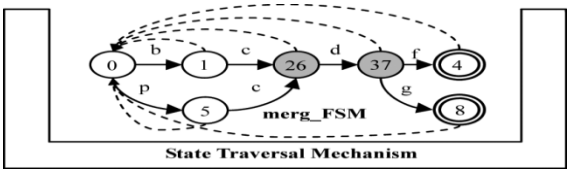

Figure 7.Architecture of **t**he state traversal machine.

State 26 represents two different states (state 2 and state 6) and state 37 represents two different states (state 3 and state 7).The figure 7 shows that directly merging similar states leads to an erroneous state machine. To have a correct result, when state 26 is reached, a mechanism is needed to understand in the original AC state machine whether it is state 2 or state 6. Similarly, when state 37 is reached, a mechanism is needed to know in the original AC state machine whether it is state 3 or state 7.In this example, state 2 or state 6 is differentiated if memorize the precedent state of state 26. If the precedent state of state 26 is state 1, in the original AC state machine, it is state 2. On the other hand, if the precedent state of state 26 is state 5, the original is state 6.

X. RESULTS & DISCUSSIONS

Outputs of these above mentioned was simulated using Modelsim 6.1f software.

A. SRAM

The TCAM simulation waveform is shown in Figure 8. For SRAM simulation different inputs have to be forced. Modelsim 6.3f is a software which is used to simulate the SRAM coding. SRAM has different inputs and outputs. By forcing the SRAM input value different SRAM output is obtained.

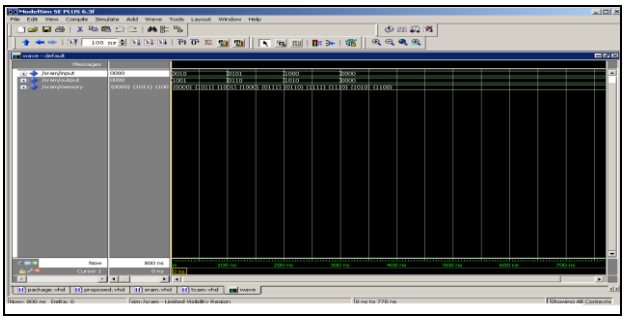

Figure 8. Simulated Output of SRAM

B.TCAM

The TCAM simulation waveform is shown in Figure 9.By forcing different values to the current state and input, different TCAM output is obtained. In this waveform, the current state is 1010 and the input is forced as e.

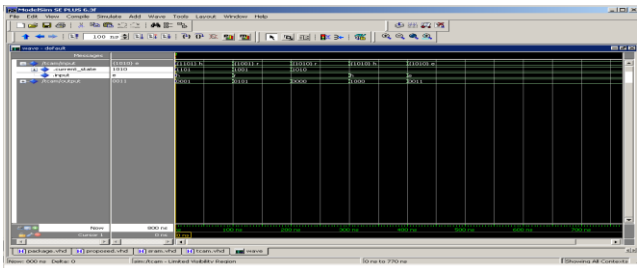

Figure 9. Simulated Output of TCAM C.Multipattern Matching

The Multipattern matching simulation waveform is shown in Figure 10. In Multipattern matching, the proposed clock is forced to a constant value and the proposed input is forced to certain 4-bit values. In this waveform, the clock value is forced to 1 and the proposed input is forced to apqr.

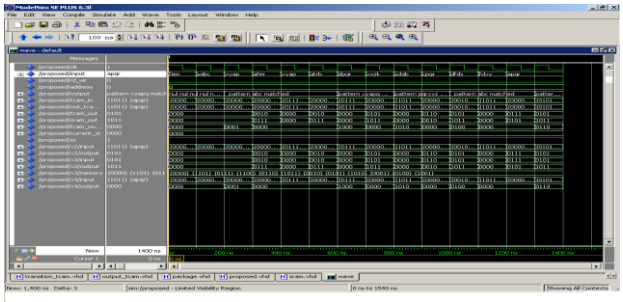

Figure 10. Simulated Output of Multipattern matching

The proposed output is the given pattern xyapq is matched. Based on the input values, the output pattern is either matched or not matched. So different input values are given and the output waveform is obtained.

XI. CONCLUSION

In this work, a covered state encoding scheme for the TCAM-based implementation of Aho-Corasickalgorithm, which is a multiple pattern matching algorithm widely used in network intrusion detection systems. The covered state encoding takes advantage of "don't care" feature of TCAMs and information of failure transitions is implicitly captured in the covered state encoding. The use of covered state encoding is the failure transitions do not need to be implemented as TCAM entries since all the states in the failure transition path can be simultaneously examined.

The covered state encoding requires the smaller number of TCAM entries than other schemes and the failure transitions are not needed. The time complexity of the algorithm building TCAM entries using the covered state encoding is O(n log n), where n is the number of states. Thus, the covered state encoding enables an efficient TCAM-based implementation of a multipattern matching algorithm. In Proposed system,the memory requirement is less and the pattern matching is high due to the Merge Finite State Automata and the pattern matching is done at high speed.

REFERENCES

[1]. Sangkyun Yun,"An efficient TCAM based implementation of Multipattern Matching using Covered state encoding",IEEE transactions on Computers, Vol 61,No.2,Feb.2012.[2] Dharmapurikar.S, Attig.M, and Lockwood.J, "Deep Packet Inspection Using Parallel Bloom Filters," IEEE Micro, vol. 24, no. 1, 220 IEEE TRANSACTIONS ON COMPUTERS, VOL. 61, NO. 2, FEBRUARY 2012.

[2]. Bremler-Barr.A, Hay.D, and Koral.Y, "CompactDFA: Generic State Machine Compression for Scalable Pattern Matching," Proc.IEEE INFOCOM, 2010.

[3]. Alicherry.M and Muthuprasanna.M, "Method and System for Multi-Character Multi-Pattern Pattern Matching," US Patent Application No. 20080046423, Feb. 2008.

[4]. Gould.M, Barrie. R, Williams.D, and de Jong.N, "Apparatus and Method for Memory Efficient, Programmable, Pattern Matching Finite State Machine Hardware," US Patent No.7082044B2,July2006.

[5]. L. Tan, B. Brotherton, and Sherwood.T, "Bit-Split String-Matching Engines for Intrusion Detection and Prevention," ACM Trans. Architecture and Code Optimization, vol. 3, no. 1, pp. 3-34, 2006.

[6]. [7] Gao.M, Zhang.K, and Lu.J, "Efficient Packet Matching for Gigabit Network Intrusion Detection Using TCAMs," Proc. 20th Int'l Conf. Advanced Information Networking and Applications (AINA), 2006.

[7]. Van Lunteren.J, "High-Performance Pattern-Matching for Intrusion Detection," Proc. IEEE INFOCOM, vol. 4, 2006.

[8]. Weinsberg.Y, Tzur-David.S, Dolev.D, and Anker.T, "High Performance String Matching Algorithm for a Network Intrusion Prevention System (NIPS)," Proc. IEEE High Performance Switching and Routing (HPSR), pp. 147-154, 2006.

[9]. Alicherry.M, Muthuprasanna . M, and Kumar .V, "High Speed Pattern Matching for Network IDS/IPS," Proc. 14th IEEE Int'l Conf. Network Protocols (ICNP), vol. 11, pp. 187-196, 2006.

[10]. Tuck.N, Sherwood.T, Calder.B, and Varghese.G, "Deterministic Memory-Efficient String Matching Algorithms for Intrusion Detection," Proc. IEEE INFOCOM, vol. 4, pp. 2628-2639, 2004.

[11]. Yu.F, Katz.R, and Lakshman.T, "Gigabit Rate Packet Pattern-Matching Using TCAM," Proc. 12th IEEE Int'l Conf. Network Protocols (ICNP '04), pp. 174-183, 2004.

[12]. Sourdis.I and Pnevmatikatos.D, "Pre-Decoded Cams for Efficient and High-Speed NIDS Pattern Matching," Proc. 12th Ann. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM),pp. 258-267, 2004.

[13]. Clark.C and Schimmel.D, "Scalable Pattern Matching for High Speed Networks," Proc. 12th Ann. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM), 2004.

[14]. Hutchings.B, Franklin.R, and Carver.D, "Assisting Network Intrusion Detection with Reconfigurable Hardware," Proc. 10th Ann. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM), pp. 111-120, 2002.

[15]. Cho. Y.H., Navab. S. and Mangione-Smith. W.H, "Specialized Hardware for Deep Network Packet Filtering," Proc. 12th Int'l Conf. Field-Programmable Logic and Applications (FPL), pp. 337-357,2002.

[16]. Aho.A and MCorasick.M, "Efficient String Matching: An Aid to Bibliographic Search," Comm. ACM, vol. 18, no. 6, pp. 333-340, 1975.

s: Part II—Results .