ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

Optimization of Pattern Matching Algorithm for Network Intrusion Detection System

[I]Karthiga.R, [II]Suresh.P
[I,II]Dept. of Computer and Communication Engg., Sethu Institute of Tech., Virudhunagar, India

Abstract
NIDS needed memory-efficient pattern-matching algorithm which can significantly reduce the memory requirement. Focus on reducing the memory size of the exact string patterns. To observe that many string patterns are similar because of common sub-strings. Proposed a state-traversal mechanism on a merge_FSM while achieving the same purposes of pattern matching. Since the number of states in merg_FSM can be drastically smaller than the original FSM, it results in a much smaller memory size. The hardware needed to support the state-traversal mechanism is also limited. Proposed algorithm achieves 26% of memory reduction compared with the traditional AC algorithm for total string patterns. Applying this approach in NIDS, memory needed for implementing FSM will be reduced.

Keywords
Aho-Corasick(AC) algorithm, Finite State Machine(FSM), pattern matching, Network Intrusion Detection System(NIDS),Field Programmable Gate Array(FPGA)

I. Introduction

A. Pattern Matching

Pattern matching is the act of checking a perceived sequence of tokens for the presence of the constituents of some pattern. In contrast to pattern recognition, the match usually has to be exact. The patterns generally have the form of either sequences or tree structures. Uses of pattern matching include outputting the locations (if any) of a pattern within a token sequence, to output some component of the matched pattern, and to substitute the matching pattern with some other token sequence (i.e., search and replace).

1. Sequence Patterns

Sequence patterns (e.g., a text string) are often described using regular expressions and matched using techniques such as backtracking.

2. Tree Patterns

Tree patterns are used in some programming languages as a general tool to process data based on its structure, e.g., Haskell, ML and the symbolic mathematics language Mathematica have special syntax for expressing tree patterns and a language construct for conditional execution and value retrieval based on it.

B. NIDS

In computer security, a Network Intrusion Detection System (NIDS) is an intrusion detection system that attempts to discover unauthorized access to a computer network by traffic on the network for signs of malicious activity. A "Network Intrusion Detection System (NIDS)" monitors traffic on a network looking for suspicious activity, which could be an attack or unauthorized activity.

A large NIDS server can be set up on a backbone network, to monitor all traffic; or smaller systems can be set up to monitor traffic for a particular server, switch, gateway, or router. In addition to monitoring incoming and outgoing network traffic, a NIDS server can also scan system files looking for unauthorized activity and to maintain data and file integrity.

1. Need of Pattern Matching

Pattern matching is one of critical parts of Network Intrusion Detection Systems (NIDS). Pattern matching is computationally intensive. To handle an increasing number of attack signature patterns, a NIDS require a multi-pattern matching method that can meet the line-speed of packet transfer. The multi-pattern matching method should efficiently handle a large number of patterns with a wide range of pattern lengths and case insensitive pattern matches. It should also be able to process multiple input characters in parallel. In this paper, we propose a multi-pattern matching hardware accelerator based on Shift-OR pattern matching algorithm.

II. Literature Review

A.V.Aho et al[2] reviewed the AC algorithm. Among all memory architectures, the AC algorithm has been widely adopted for string matching in the algorithm can effectively reduce the number of state transitions and therefore the memory size. The state transition diagram derived from the AC algorithm where the solid lines represent the valid transitions while the dotted lines represent a new type of state transition called the failure transitions. The failure transition is explained as follows. Given a current state and an input character, the AC machine first checks whether there is a valid transition for the input character; otherwise, the machine jumps to the next state where the failure transition points. Then, the machine recursively considers the same input character until the character causes a valid transition. So AC machine may take more than one cycle to process an input character. This objective is to modify the AC algorithm so that storing only the state transition table of merg.FSM in memory while the overall system still functions correctly as the original AC state machine does.

Benjamin Brodie et al[5] presented and evaluated architecture for high throughput pattern matching of regular expressions. Using the approach that matches multiple patterns concurrently, responds rapidly to changes in the pattern set, and is well suited for synthesis in an ASIC or FPGA. This approach is based on a new and easily pipelined state-machine representation that uses encoding and compression techniques to improve density. Analysing the approach in terms of its throughput, density, and efficiency. The experimental results from an implementation in a commodity FPGA, showing better throughput and density than

the best known approaches.

.Baker et al[6] presented a methodology and a tool for automatic synthesis of highly efficient intrusion detection system using a high level, graph based partitioning methodology and tree based look ahead architectures. Intrusion detection for network security is a compute intensive application demanding high system performance. The tools implement and automate a customize flow for the creation of FPGA architectures using system level optimizations. Our methodology implemented with a tool suite we release for public use, allows for customized performance through more efficient communication and extensive reuse of hardware components for dramatic increases in area time performance.

H.J.Jung et al[8] examined string matching algorithms and their use for Intrusion Detection. In particular the efforts on providing worst-case performance that is amenable to hardware implementation. Contributing modifications to the Aho-Corasick string-matching algorithm that drastically reduce the amount of memory required and improve its performance on hardware implementations. It also shows that these modifications do not drastically affect software performance on commodity processors, and therefore may be worth considering in these cases as well. Proposed hash based approach for deep packet inspection using Bloom Filter. Which is used only for fixed length strings.

III. Existing System

In the past few years, many algorithms and hardware designs are proposed to accelerate pattern matching. The hardware approaches can be classified into two main categories, logic and memory architectures. The logic architectures[6] mostly use on chip logic resources of field-programmable gate array (FPGA) to convert regular expression pattern into parallel state machines or combinatorial circuits because FPGA allows for updating new attack patterns. Sidhu et al. proposed algorithm to compile regular expression patterns into combinatorial circuits based on Nondeterministic Finite Automaton (NFA). Hutchings et al. developed a module generator that shared common prefixes to reduce the circuit area on FPGA. The Aho–Corasick (AC) algorithm is the most popular algorithm which allows for matching multiple string patterns. The AC machine[2] first checks whether there is a valid transition for the input character; otherwise, the machine jumps to the next state where the failure transition points. Then, the machine recursively considers the same input character until the character causes a valid transition. So it produced some functional errors. But the proposed system is used to rectify these errors.

IV. Proposed System

Sequence patterns (e.g., a text string) are often described using regular expressions and matched using techniques such as backtracking.

A. Merge_FSM

In Fig.4.1 states 2 and 6 are similar because they have identical input transitions, identical failure transitions to state 0. Also, states 3 and 7 are similar.

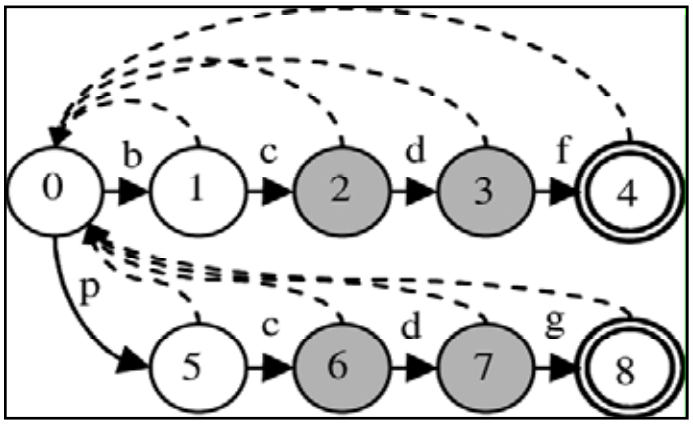

Fig. 4.1 State diagram of an AC machine.

Note that merging similar states results in an erroneous state machine. As shown in Fig. 4.2 the state machine merges the similar states 2 and 6 to become state 26, and merges the similar states 3 and 7 to become state 37.

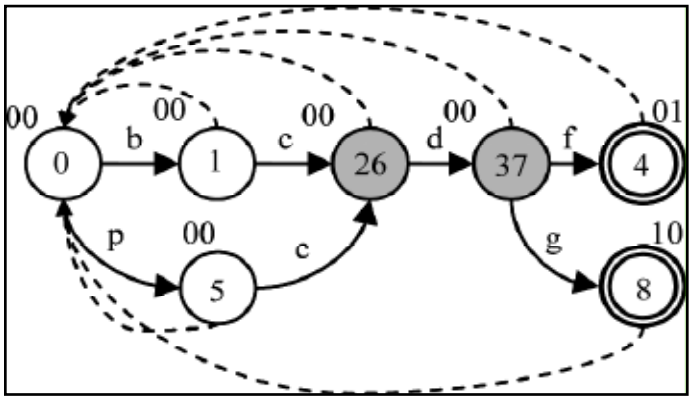

Fig 4.2 Merging similar states

B. State Traversal Mechanism On Merge_FSM

The overall architecture of state traversal machine is shown in Fig. 4.3. State Traversal Machine is used to guide the state machine to traverse on the Merg_FSM.STM Contains Following Bits.

- PathVec_ifFinal
- PreReg

Reuse those memory spaces storing zero vectors {00} to store useful path information called pathVec. First, each bit of the pathVec corresponds to a string pattern. Then, if there exists a path from the initial state to a final state, which matches a string pattern, the corresponding bit of the pathVec of the states on the path will be set to 1. Otherwise, they are set to 0. ifFinal, to indicate whether the state is a final state.

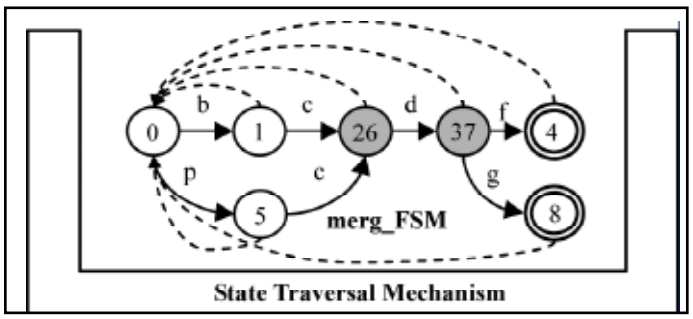

Fig. 4.3 Architecture of the state traversal machine.

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

PreReg is to trace the precedent pathVec in each state. The width of preReg is equal to the width of pathVec. Update these bits when merging pseudo equivalent statesThis example shows that if memorize the precedent state entering the merged states, it can differentiate all merged states.

C. Loop Back Problem

The loop back problem comes from merging common sub-patterns with different sequences. For example, the two patterns, "abcdef" and "wdebcg," have common sub patterns, "bc" and "de," which appear in different sequences.

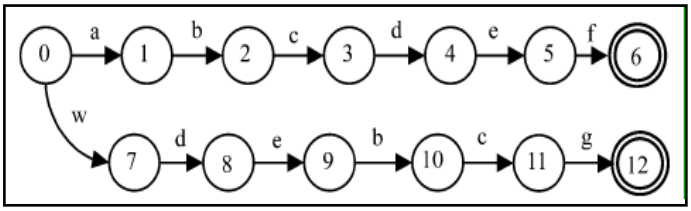

Fig. 4.4 AC State machine for the two patterns,"abcdef" and "wdebcg"

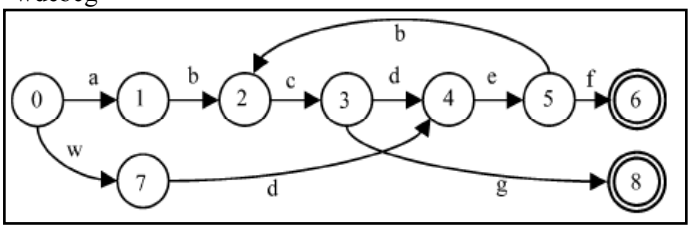

Fig. 4.5 Merging Pseudo-equivalent states with different sequences

Merging the pseudo-equivalent states will create a loop back transition from state 5 to state 2, as shown in Fig.4.5. The loop transition may cause false positive matching results. For example, the input string "abcdebcdef" will be mistaken as a match of the pattern "abcdef.".In other words, as long as the common substrings appear in sequence, merging the corresponding pseudo-equivalent states will not result in loop back transitions. Therefore, in this program, record and identify the orders of common sub-patterns. If the common sub-patterns appear in sequence, the corresponding pseudo-equivalent states can be merged without loop back problems. The following algorithm is used to find common substrings without the loop back problem.

Algorithm: Extract Substrings without loopback
Input: Two String Patterns
Output: Shared Substrings

Method:

- Using Longest common substring algorithm to extract common substring of the two string patterns
- Label the common substrings as new sequences
- Using LCS to find longest common subsequence among the two new sequences
- Output substrings of the same sequence

D. NIDS

NIDS is designed for passive monitoring system. It reads the packets from network interface by standard system facilities. Each Packet is checked against the NIDS rule set. The Rule set organized by two dimensional data structure chain, where each element is called as Chain Header. The input packet headers match with the chain header. If it matches the packet will be allowed otherwise it will be discarded.

V. Experimental Results

The results are compared with the methods of the AC algorithm and the bit split algorithm. In the first stage, obtain the string patterns from snort rule database. In the second stage, group 32 string patterns as a module based on the similarity of string patterns. In third stage, using LCS to extract substring which have the largest sharing gain.

Table I shows the results before and after integrating our algorithm to the AC algorithm. Columns one, two and three show the name of the rule set, the number of patterns, and the number of characters of the rule set. Columns four, five, and six show the number of state transitions, the number of states, and the memory size of the AC algorithm. Columns seven, eight, and nine show the results of our approach. Column ten shows the memory reduction compared to the AC algorithm. Because the memory requirement is proportional to the number of states, our algorithm has reduced memory size on the traditional AC algorithm. Using this algorithm, the number of transitions, states, memory size are given in that table. Integrating our algorithm to the AC algorithm the number of transitions, states, memory size are reduced.

Table II shows the results before and after applying our algorithm to the bit-split algorithm. Consider the same Oracle rule set in the first row of Table II. Applying the bit-split algorithm which splits the traditional AC state machine into 4 state machines. The size of memory is reduced after integrating our algorithm to the bit split algorithm. String patterns of Snort rule sets integrating our algorithm to the bit-split algorithm can achieve 26% of memory reduction.

Table 1: Experimental Results after Applying Our Algorithm to the AC Algorithm

Rule Sets	# of patterns	# of char	Aho-Corasick			Aho-Corasick+Our Algorithm			
			# of transitions	# of states	Memory(bytes)	# of transitions	# of states	Memory(bytes)	Memory Reduction
Oracle	337	11128	6793	6804	49267	4432	3846	30699	38%
Deleted	310	4636	3241	3251	22702	2541	2322	17776	22%
Exploit	160	2052	1561	1567	10545	1234	1135	8348	21%
Web-Misc	156	1644	1420	1425	9592	1305	1247	8892	7%
SMTP	104	989	715	719	4653	625	600	4109	12%
Misc	97	1403	1133	1137	7653	897	820	5846	24%
FTP	96	466	402	406	2517	385	374	2442	3%
Other Sets	957	14041	10912	10871	81768	9345	8777	70262	14%
Total Rules	2217	36359	26177	26180	188697	20764	19121	148374	21%
Reduction(%)					1			21%	

Table 2 Experimental Results after Applying Our Algorithm to the Bit-Split Algorithm

Rule Sets	# of patterns	# of char	Bit-Split Algorithm			Bit-Split Algorithm+Our Algorithm			
			# of transitions	# of states	Memory(bytes)	# of transitions	# of states	Memory(bytes)	Memory Reduction
Oracle	337	11128	21949	21993	159202	14437	12664	98400	38%
Deleted	310	4636	10575	10615	74085	8149	7381	53950	27%
Exploit	160	2052	5177	5201	34978	4286	4006	27509	21%
Web-Misc	156	1644	4536	4556	30646	4031	3766	25870	16%
SMTP	104	989	2253	2269	14665	2009	1932	12470	15%
Misc	97	1403	3866	3882	25149	3100	2879	19874	21%
FTP	96	466	994	1010	5980	948	929	5669	5%
Other Sets	957	14041	35471	35407	265921	30456	28749	218032	18%
Total Rules	2217	36359	84821	84933	610624	67416	62306	461773	26%
Reduction(%)					1			26%	

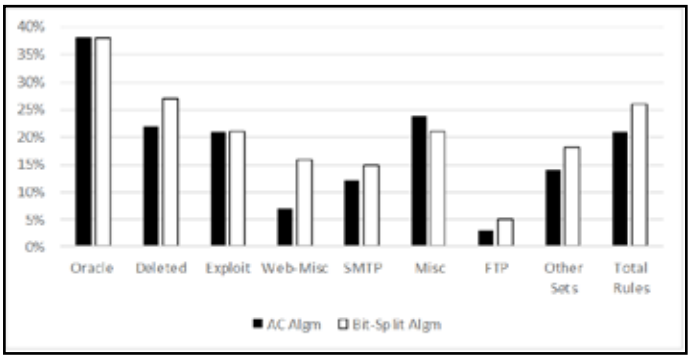

Fig 5.1 Comparison of Memory Reduction

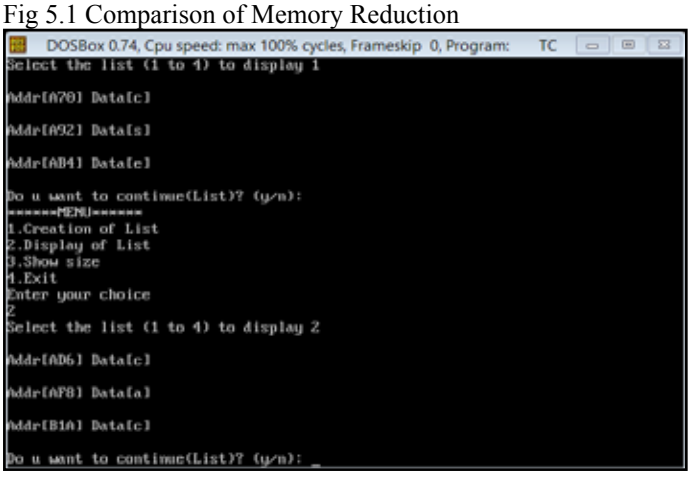

Fig.5.2 Data of list1 & list2

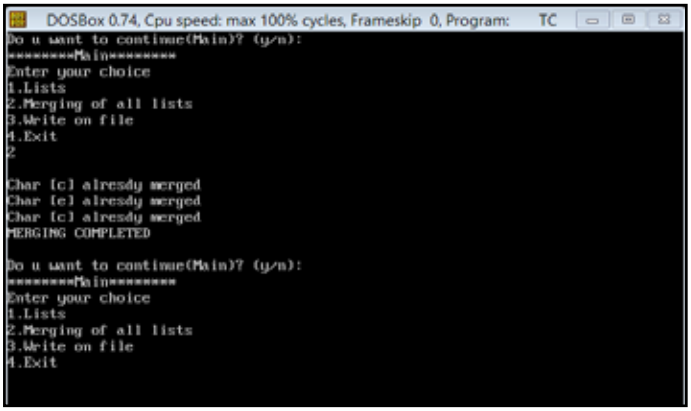

Fig.5.3 Merging of lists

The results of memory reduction are compared as shown in Fig.5.1 Comparison of Memory Reduction. Fig.5.2 shows an example of string patterns. The merging of similar string patterns show in Fig.5.3. After merging the length of the list will be reduced.

VI. Conclusion

Presented a memory-efficient pattern matching algorithm is done which can significantly reduce the number of states and transitions by merging pseudo-equivalent states while maintaining correctness of string matching. In addition, the new algorithm is complementary to other memory reduction approaches and provides further reductions in memory needs. The experiments demonstrate a significant reduction in memory footprint for data sets commonly used to evaluate IDS systems. In addition, it can gain 26% of memory reduction by integrating the approach to the bit-split algorithm which is the state-of-the-art memory-based approach.

REFERENCES

[1] *Cheng-Hung Lin and Shih-Chieh Chang, Member "Efficient Pattern Matching Algorithm for Memory -Architecture" ieee transactions on very large scale integration (vlsi) systems, vol. 19, no. 1, january 2011.*

[2] *A. V. Aho and M. J. Corasick "Efficient string matching: An AID to bibliographic search", Commun. ACM, vol. 18, no. 6, pp.333 -340 1975.*

[3] *M. Aldwairi , T. Conte and P. Franzon "Configurable string matching hardware for speeding up intrusion detection", Proc. ACM SIGARCH Comput. Arch. News, vol. 33, no. 1, pp.99 -107 2005.*

[4] *S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, "Deep packet inspection using parallel bloom filters," in Proc. 11th Symp. High Perform. Interconnects, Aug. 2003, pp. 44–53.*

[5] *B. Brodie, R. Cytron, and D. Taylor, "A scalable architecture for high-throughput regular-expression pattern matching," in Proc. 33rd Int. Symp. Comput. Arch. (ISCA), 2006, pp. 191–122.*

[6] *Z. K. Baker and V. K. Prasanna, "High-throughput linked-pattern matching for intrusion detection systems," in Proc. Symp. Arch. For Netw. Commun. Syst. (ANCS), Oct. 2005, pp. 193–202.*

[7] *Y. H. Cho and W. H. Mangione-Smith, "A pattern matching co-processor for network security," in Proc. 42nd IEEE/ ACM Des. Autom.Conf., Anaheim, CA, Jun. 13–17, 2005, pp. 234–239.*

[8] *H. J. Jung, Z. K. Baker, and V. K. Prasanna, "Performance of FPGA implementation of bit-split architecture for intrusion detection systems," presented at the 20th Int. Parallel Distrib. Process. Symp. (IPDPS), Rhodes Island, Greece, 2006*

[9] *C. R. Clark and D. E. Schimmel, "Scalable pattern matching*

ISSN : 2347 - 8446 (Online)
ISSN : 2347 - 9817 (Print)

on high speed networks," in Proc. 12th Ann. IEEE Symp. Field Program.Custom Comput. Mach. (FCCM), 2004, pp. 249–257.

[10] G. Dan, Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge, U.K.: Cambridge University Press, 1997.

R.Karthiga received her B.E., degree in Computer Science and engineering in SACS MAVMM engineering college, Madurai in 2012.She is currently pursuing her M.E. in computer and communication engineering in Sethu Institute of Technology,Tamil Nadu,India.

P.Suresh received his B.E., degree in Computer Science and Engineering in AKCE, Virudhunagar in 2001.He has received his M.E degree in Computer Science and Engineering in Anna University,Coimbatore in 2009.He is currently pursuing his Ph.D degree in Anna University, Chennai. His research interests include Datastructure and NIDS.