An Optimized Approach for Intrusion Detection System.

Madhuri Shinde, Pallavi Shinde, Shrutika Surwade, Anuja Umbarde

Students, Dept. of Computer Engineering, K K Wagh College of Engineering, Nashik, Maharashtra, India

---***---

Abstract - *Intrusion Detection System is a security system that monitors computer systems and network traffic as well as analyses that traffic for possible hostile attacks originating from outside the organization and also for system misuse or attacks originating from inside the organization. IDS uses pattern matching algorithms, which is based on constructing and traversing deterministic finite automaton (DFA) that represents patterns. IDSs needs to deal with hundreds of patterns that requires to store very large DFA, and which do not fit in memory. This results in major bottleneck on throughput of IDS .Previously TCAM (Ternary Content Addressable Memory) was used which is hardware device that enable longest prefix matching operation, But they are costly and power consumption is very high. Therefore performance is degraded. To overcome this issue, Compact DFA technique uses software based approach. These method reduces size of Aho-Corasick Deterministic finite automata which is generated from regular expressions/patterns and uses this compressed Deterministic finite automata in pattern matching process for speed up. The compression approach uses bit reduction method which decreases the state storage size by representing all transitions to that state through single storage unit.*

***Key Words*: Aho-Corasick, NIDS, Pattern Matching, OpenMP, CompactDFA.**

1. INTRODUCTION

Pattern matching algorithms lie at the core of all contemporary Intrusion Detection Systems, making it intrinsic to reduce their speed and memory requirements. The most common algorithm used today is the seminal Aho-Corasick (AC) algorithm, which uses a Deterministic Finite Automaton (DFA) to represent the pattern set. Pattern matching algorithms, such as an Aho-Corasick, are often used as a first step to identify regular expressions, which become popular in recent signature-sets. Each regular expression contains one or more string tokens, which can be extracted offline. Thus, the DFA may be used to represent these anchors and to inspect the input traffic these anchors and to inspect the input traffic. Recently, there were many efforts to compress the Aho-Corasick DFA and by that to improve the algorithm's performance While most of these works either suggest dedicated hardware solutions or introduce non-constant higher processing time, this present a generic DFA compression algorithm and store the resulting DFA in an off-the-shelf hardware. Novel algorithm works on a large class of *Aho-Corasick like DFAs*, whose unique properties are defined in the sequel. This algorithm reduces the rule set to *only one rule per state*.

1.1 Aho-Corasick

Aho-corasick algorithm is used to construct DFA and that DFA is used for searching. It has three functions that are goto function, failure function, output function.

1.2 Compact DFA

Compact DFA technique shows a reduction of the pattern matching problems, Better Performance, Less Resources and minimize the memory for searching. It Increase throughput. It uses a following stages:

Stage I: State Grouping

To group states, calculate two parameters for each state: its *common suffix* (*CS*) and its *longest common suffix* (*LCS*).The motivation behind these parameters, using wasteful code that encodes each state with its label

Stage II: Common Suffix Tree

Encode the rules with smaller number of bits and transform them from being defined on suffixes to being defined on prefixes.

Stage III: State and Node Encoding

In the third step encode the Common Suffix Tree and then the states and rules.

1.3 CompactDFA for total memory minimization

Minimizing the number of rules increases the number of bits used to encode a state. For some applications, however, to minimize the total memory requirement, namely, the product of the number of rules and the total number of bits required to encode a rule. Hence, prefer using less bits per state code, even at the cost of slightly increasing the number of rules .One way to reduce memory requirements is to encode by one rule only the rules whose next state is of depth at most *D*, where *D* is the parameter of the algorithm. Most of the rules are due to transitions to a state with small depth, implying that compressing them might be enough. In addition, even though only a small fraction of the rules corresponds to states with large depth, they might increase the code width significantly due to the structure of the Common Suffix Tree. The optimal D for total memory requirement minimization can be found simply by checking linearly possible *D* values and choosing the optimal one.
Time and memory requires for searching and storing encoded rules is less for Compact DFA. Compact DFA technique shows a reduction of the pattern matching problems, Better Performance, Less Resources and minimize the memory for searching. It Increase throughput.

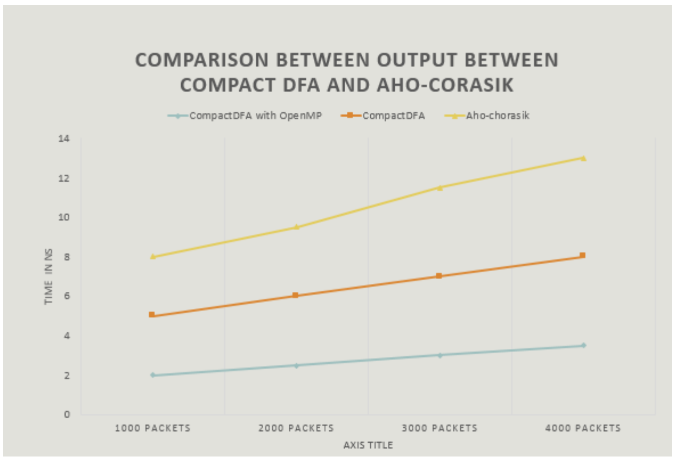

Chart -1: Comparison between output of Aho-corasik ,compact DFA and CompactDFA with OpenMP.

Here,we are showing the result of Aho-chorasik algorithm, Aho-corasick algorithm with OpenMP, Compact DFA and Compact DFA with OpenMP.

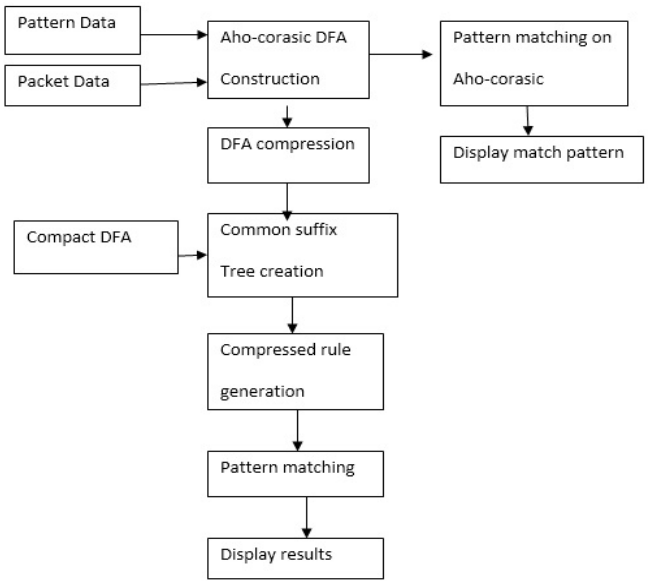
Pattern Data
Packet Data
Aho-corasic DFA Construction
Pattern matching on Aho-corasic
Display match pattern
DFA compression
Compact DFA
Common suffix Tree creation
Compressed rule generation
Pattern matching
Display results

Fig -1: Block diagram of Compact DFA.

3. CONCLUSIONS

Compact DFA gives a reduction of the pattern matching problem which results the reduced area, better performance ,less number of resources .In general DFA may require up to 2n states but after minimization DFA requires n states. Compact DFA, gets as an input a DFA and produces compressed rule sets; each compressed rule defines the set of states that the rule applies to using a common prefix.

ACKNOWLEDGEMENT

The authors would like to thank Mrs. I .Priyadarshani, for their guidance and support during our graduate study.

REFERENCES

[1] M. Alicherry, M. Muthuprasanna, and V. Kumar, “High speed pattern matching for network IDS/IPS,” in *IEEE ICNC*
[2] T. Song, W. Zhang, D. Wang, and Y. Xue, “A memory efficient multiple pattern matching architecture for network security,”.
[3] H. Song, F. Hao, M. Kodialam, and T.-V. Lakshman, “IPv6 lookups using distributed and load balanced bloom filters for 100gbps core router line cards,” .

[4] N. Tuck, T. Sherwood, B. Calder, and G. Varghese, "Deterministic memory efficient string matching algorithms for intrusion detection,"

BIOGRAPHIES

Student: Madhuri V Shinde.
K K Wagh College of Engineering,
Nashik.
Computer Engineering.

Student: Pallavi S Shinde.
K K Wagh College of Engineering,
Nashik.
Computer Engineering.

Student: Shrutika Surwade.
K K Wagh College of Engineering,
Nashik.
Computer Engineering.

Student: Anuja Umbarde
K K Wagh College of Engineering,
Nashik.
Computer Engineering.