Compact DFA: A Variable Stride Pattern Matching Algorithm to Perform Pattern Matches Using HEXA

J. DURGADEVI
M.Tech Student in Department of CSE
D.N.R. College of Engineering and Technology
Andhra Pradesh, India

M. S. V. V. RAMESH
Assistant Professor in Department of CSE
D.N.R. College of Engineering and Technology
Andhra Pradesh, India

D. D. D. SURIBABU
HOD & Associate Professor in Department of CSE
D.N.R. College of Engineering and Technology
Andhra Pradesh, India

Abstract—In any network identifying the intruders while packet transferring is done by using pattern matching. In every intrusion detection system different pattern matching approaches are used. One of the approach is construction of DFA to identify the exact pattern in the system. But memory usage and memory bandwidth are the bottleneck for the DFA construction. In this paper we propose an algorithm which identifies the pattern as variable strides i.e., it uses the block oriented approach instead of bit oriented process. It is a multiple pattern matching algorithm with minimum memory usage. With including the algorithm, we propose a compact DFA which does not use addition memory for traversing in the graph to identify the pattern. Using all these approaches the throughput of the system can be increased in many folds at minimum cost.

Index Terms—Packet inspection, Intruders, pattern matching.

I. INTRODUCTION

At the time of communication data transferred in a network increases then the number of intruders are also increases to perform malicious operations on the data packet. So protection of the data at the time of communication in a network is important and it is done by Intrusion Detection System in a network. But the Intrusion Detection System uses the pattern (string) matching to identify the malicious attacks on data. Intrusion Detection System identifies the incoming packets in a network and matches the pattern with the previous stored patterns. If any malicious patterns are identified then it automatically intimates to the administrator to perform proper action.

Pattern matching used in every contemporary Intrusion Detection System and it acts like heart of IDS. The common algorithm used in pattern matching is Aho-croasicks, it uses DFA to identify the patterns. To construct the DFA, first predefined regular expressions are stored offline and those regular expressions are acts as regular sets. The regular expression engine verifies the new incoming patterns with these old patterns by using the DFA.

To construct and traverse the DFA mainly five components are used. Those are set of states, starting state, transition function, alphabet set and final state which are represented as (Q, q₀, δ, Σ, q_f). The final state is the acceptance state. To identify the pattern in a DFA, first the traversing starts with the starting state by reading the starting symbol from the pattern based on the rules generated by the regular expression engine. Based on the rules the string matching process done by moving from previous state to the next state.

To move from one state to another based on the rules, the DFA construction uses pointers. But the pointer uses more memory for storing the graph data values. So to minimize the memory requirement a compact DFA is used in this process, which is a compressed DFA. This compressed DFA uses the implicit information to identify the next node in a graph without using the explicit information. Due to this the memory requirement at the time of traversing a DFA to identify the string will minimized. With including this, we present one more extension to our basic scheme. Our first extension is Compact DFA used for total memory minimization. It aims at minimizing the product of the number of rules and the code width, rather than only the number of rules. The second extension is the variable-stride DFA algorithm which is used to speed up the identification process of strings by inspecting more than one variable at a time.

The variable stride DFA algorithm is a multi pattern matching algorithm. It reads variable number of bytes from a data stream to identify the pattern in single step. Normally it uses the block oriented approach instead of bit oriented approach while traversing in a DFA at the time of pattern matching. To identify multiple bytes of input
stream in each step it uses the intra stream parallelism approach. This process minimizes the memory usage without increasing the cost and with optimized throughput for the system. With including this intra stream parallelism it uses the winnowing algorithm. This winnowing algorithm is used to divide the pattern into variable number of streams based on the matching patterns. All these processes are very useful to traverse the DFA with minimum memory requirement.

II. RELATED WORK

A. Constructing Condensed DFA:-

In this module, we propose a new method to construct a compact DFA called HEXA (History based Encoding, Execution, and Addressing). Other methods of DFA uses different bits to construct the structured graph but HEXA uses fixed and constant number of bits to represent a graph. Due to these the memory required while traversing in one state to another is minimized. The required total memory is also minimized, because it uses auxiliary memory which is a small fraction of total memory.

The HEXA does not take the single bit as a reference to traverse from one state to another state using the transition function. The general HEXA transition function starts from the current state while reading the block of matching pattern moves to another state. In general normal graphs reads root node as first if it does not match then it moves to left and then right node. But as like normal trie it does not follow this approach. To understand the HEXA process considers an example graph.

Let us consider a tree which is constructed based on the rule set derived from a DFA transition table rules. For example the table consist of five regular expressions which are 0*, 1*, 10*, 101*, 00*, 10*, and 101*. If we construct a tree it consists of five nodes those are p1, p2, p3, p4 and p5. Generally normal DFA starts with p1 to match the pattern for any string. It follows the next current state nodes to traverse strings as follows:

1. p1p2  2. p1p2p4  3. p1p2p5
   4. p1p4  5. p1p2p3

To verify the string 001 the normal DFA starts with p1 by reading the symbol 0 from the transition. It reads every single symbol to match the total pattern. If we want to match the long prefix also it internally verifies individual symbols to match the longest pattern which matches with the string. It is a cost and time consuming process and also requires more memory.

HEXA uses special states based on the input stream presented on the rule set. Based on this it easily matches the exact first block stream which matches the pattern in a single step. This indicates that the HEXA uses the internal information only for identifying the complete pattern without using additional memory. For example to match the string 001 it starts with 00 rule presented in the DFA table. So it directly verifies the block string in a single transition of DFA.

The identifiers used in HEXA are unique. HEXA uses a hash function which is used to identify the block stream in a single move. It performs one to one mapping operation to match the block stream. It does not requires additional memory to store these values. It normally uses an array to store the values instead of pointers. So the memory requirement for pattern matching is minimized while using HEXA instead of DFA.

B. User Details Verification Using DFA:-

In this Module the User details in a network are verified to make sure that the user is a valid user. The user details are verified in the three ways. One is by using username, other is by using user id and the last one is by using ip address. By using any one of this the pattern matching processes is done.

In general, the pattern matching technique for string verification constructs the DFA for the entered system rules and then it processes the given string in that DFA. If the string processing is ended at the final accepting state of the constructed DFA the user is valid user, otherwise the user is not a valid member in the network. In this module we also verify two types of compact DFA’s for getting good throughput.

C. State Rule Encoding for DFA:-

In this module, to increase the performance of the algorithm based on size k each stride is taken with predetermined k size. There is one problem arise with this process i.e., in some cases accepting state is skipped over the transition on a stride in the automata. To overcome this problem variable strides are taken to accept the acceptance state at the correct situation.

Initially define set H with internal sets which are encoded. For every set there is an acceptance state s which belongs to S and the transition length is from k to s i.e., kS(i) where in i \in [1, \ldots, k]. Finally H is having the size greater than 1 as set of all such states.

In the state grouping algorithms the pre-processing step is segmentation of sequence of blocks by considering all the patterns. In the segmentation the first block is called as the head block and the last block is known as the tail block. There are two types of patterns which are called coreless patterns and indivisible patterns. The initial patterns are long patterns which contain delimiters and other type does not have delimiters respectively.
The following properties are used in our segmentation scheme:

**Property 1:** Any segmented block size is in the range \([1, w]\). The sizes of Tail block are in the range \([k - 1, w + k - 2]\), the sizes of indivisible pattern are in the range \([1, w + k - 2]\) and the sizes of coreless pattern are in the range \([w + k - 1, 2w + k - 2]\). Moreover, in our algorithm every pattern this property to segment the patterns.

**Property 2:** The core block patterns contains exactly the same delimiters that after a pattern segmented in a data stream. The head block can be affected by the prefix and tail block can be affected by suffix, due to this both blocks contains patterns which are having extra delimiters that they appear in a data stream. The core blocks are independent of context and totally confined to the pattern. These unchanged core blocks are very helpful to perform pattern narrow search.

**D. Buffered Winnowing Strides:**

The Winnowing algorithm is used for detecting copied content in a document fingerprinting scheme. This algorithm does not depends on position i.e., it is position independent. For the preserved two files it matches for the original matching content if any file contains some new text also.

The algorithm works like below. Initially, for every consecutive \(k\) characters it calculates hash value. \(l - k + 1\) hash values are generated from a stream of \(l\) characters. Next, to select the minimum hash value in the window a sliding window of \(w\) is used. By selecting the rightmost minimum value, the tie is broken. In our algorithm the selected values are used as delimiters to segment the window.

For example, consider the block with size \(s\) which is bounded by \(w\) and the hash values are taken in the range 0 to 255. The Winnowing algorithm is modified in several ways to use in pattern matching. Due to the short size of patterns, the block size \(s\) is kept relatively small to get meaningful number of blocks from a pattern. In the basic algorithm for our convenience the block size is segmented into three blocks. To get a better throughput the size of the block is kept as long as possible and the value of \(w\) will also be large. To verify pattern \(8\) bit size is less than sufficient for pattern matching in a winnowing algorithm. So the \(k\) value can be taken between 1 or 2 and the \(w\) value is chosen between 3 to 8.

![Winnowing with k=2 and w=3](image)

As shown in Figure, the delimiters are placed after the minimum hash value that is selected. The hash values of the hash window are align as the first characters. This states that \(w\) is always greater than the size of any block, when \(k \leq w + 1\).

**E. Optimal Detection Analysis:**

An optimization framework for pattern matching process analyzes the pattern within a minimum time compared to other processes. It gives best throughput for intrusion detection in most of the intrusion detection systems because we are constructing HEXA with the implementation of block strides for the comparison. So it gives optimal result for the intrusion detection in pattern matching process.

**III. ALGORITHMOPTIMIZATIONS**

**A. Reducing Single-Byte Blocks**

In every DFA traversal step the average large stride is used by VSDFA. This is main advantage of VSDFA to increase the throughput. The minimum block size of \((W+1)/2\) is used for every random data stream. But there is a possibility to generate single byte blocks at the time of segmentation of a data stream which increases the memory usage based on the hash value and window parameter.

For example, if one single packet may contain more single byte blocks cause reduction in systems throughput. There is also a possibility of Denial-of-Service (DoS) attack to the system.

The problem with data streams in a winnowing algorithm is the tie pattern and a single byte block. The tie break problem for the same hash value is not suitable for our application. So combination rule is implemented for a data stream to overcome these problems.

Combination Rule1 (applied on data streams): This rule generates new data streams for the previous once which contain more than one consecutive single byte blocks. It verifies from the first block and combines every \(w\) single byte block. It proceeds this processes for all blocks in a VSDFA. For example, consider a single byte consecutive block with \(w=4\), \(a1\), \(a2\), \(a3\), \(a4\), \(a5\), \(a6\). The result after applying combination rule1 to the example data stream is \(a1\), \(a2\), \(a3\), \(a4\), \(a5\), \(a6\).

All consecutive single byte blocks are eliminated by a Combination Rule1 by leaving some blocks which are placed in between the multi byte blocks. It also speedup's the system some extent without decreasing if the data stream contains a sequence single and double byte blocks.

Combination Rule1 (applied on pattern): This rule combines all the consecutive single byte blocks presented in a core block. It combines all core block patterns to form a new head block from the
original head block pattern. After the completion of the core block pattern the remaining data which is shorter than w bytes is added to tail block and apply the Combine Rule1 (applied on data streams).

Consider an example to perform operation on Combination Rule1 (applied on pattern) aaaaacbbbbddabc is originally segmented as aaaa|a|c|b|d|d|d|d|d|d|a|b|c. The segmentation contains two single byte blocks aa, those are present in the core block and combine them with head block. The core block also contains five dd's which are also single byte blocks and combine as three and two ddd pattern presented in core block using Combination Rule1. But the last two dd's length is shorter than the window size. So add these blocks to the tail block data. Hence the new segmentation after applying the rule becomes aaaa|a|c|b|d|d|d|d|d|d|a|b|c. If, in another case, the same pattern is originally segmented as aaaa|a|c|b|d|d|d|d|d|d|a|b|c. The new segmentation after applying the combination rule is aaaa|a|c|b|d|d|d|d|a|b|c. But the tail block remains same because the last core block [ab] is not a result of combination.

For example, consider a pattern a1a2a3a4a5a6a7a8a9a10a11 which is initially segmented as a1a2a3a4a5a6a7a8a9a10a11 with w=3. Applying combination rule1 to the pattern it combines the data patterns based on window size as a1a2a3a4a5a6a7a8a9a10a11. If we observe the data stream the head block pattern size is two times the window size. So to obtain correct process replicate the head block three times to get a new segmentation pattern on head block. The new segmented pattern is a1a2a3a4a5a6a7a8a9a10a11, a1a2a3a4a5a6a7a8a9a10a11, and a1a2a3a4a5a6a7a8a9a10a11. The head block segmentation pattern size increases then replications are also increases. These replications are not only appeared in head block data it also applicable and appeared on tail block data after segmentation.

B. Eliminating Single-Byte Blocks

To eliminate all the consecutive single byte blocks Combination Rule1 is used. Because these single byte blocks can slow down the process. But there is isolated single byte blocks are present. However, the remaining isolated single byte blocks may still slow down the processing. So another combination rule can be introduced to eliminate all single byte blocks. Due this Combination Rule2 the throughput also trying to improved.

Combination Rule2: After applying Combination Rule1, the remaining single byte blocks are combined to its preceding once. This rule is applied only to the single byte core block patterns. But if the tail block contains the single byte blocks then replicate the block to arrange the data stream into new segmentation. This mechanism somewhat decreases the single tail block patterns.

For example, consider a pattern a1a2a3|a4a5|a6|a7a8|a9a10a11. After applying combination rule2 it becomes a1a2a3|a4a5a6|a7a8|a9a10a11. This data stream contains a tail block which can be segmented as a9a10a11. After segmentation if we apply combination rule2, a8 in the tail block is added to the preceding segment pattern in a data stream. Then the combined data stream is placed in a VSDFA. In this case we replicate the tail block pattern using combination rule2. But in every case this replication is not necessary.

If the above pattern contains only c9 and c10 then the winnowing hash value is calculated between these two values. Based on that value we can avoid attacks. So no replication is required and the DOS attack is prevented generally by combination rule2 and increases the system throughput. There are other problems also with single byte blocks. The patterns in the head block increases the length of the head block to w+1 bytes. So they are replicated w+1 bytes time for every pattern in a data stream after applying combination rule1. These combination rules perform different operations and replications but do not increases the memory consumption for patterns of VSDFA. So mainly trying to avoid the single byte patterns in a data stream to increase throughput by minimizing memory usage.

IV. RESULT ANALYSIS

An experimental evaluation of the HEXA representations is performed in this result analysis. HEXA is verified in the pattern matching process because it is used as a compressed finite automata in pattern matching algorithm. Here we are considering two flavors of high performance string matcher, the classic Aho-Corasick automaton, and the recently proposed bit-split version. In both the cases, HEXA reduces memory without affecting the parsing performance. In this section, the results are obtained from the experiments in which we use HEXA to implement string based pattern matchers. The string sets are obtained from collection of different pattern sources. We are also normal strings which are generated from randomly comparable signatures. These strings were implemented with pointers and without pointers which gives different results. Without free pointers the memory used by the automata in transition increases compared to without pointers. The memory reduction gives fan-out results.

In order to verify HEXA in a deep manner, recently proposed bit-split version of Aho-Corasick is considered. In this multiple state machines are used, each handling a subset of the 8-bits in each input
symbol. For example, consider eight bit binary state machines, which takes the 8-bit input symbols in a single bit by each machine and it reduces the total node transactions to 16. First, consider the report on normal strings with zero spills without discriminator. It is clear from the experiment that without using discriminators to get zero spill is very difficult. Then single bit discriminator is used in HEXA which takes extra memory which is less than 10% and acts as a zero spill because the memory is unrecognizable. It also gives the same result when the HEXA identifier length increases. In this experiment the HEXA takes three bits to represent a node: one for its discriminator and other two is for HEXA identifier. The standard implementation requires 16 bits to represent a node but the spill pattern gives more than five fold reduction in the memory.

If we consider four state machines which takes 2 bits to represent a node from a 8 bit input stream. The bit split version surprisingly increases the memory requirement for discriminator and HEXA identifier for some machines. So variable strides are taken to perform this experiment. Due to these variable strides the length of the HEXA identifier does not take more memory because the variable stride are referred as block data without considering the bit oriented approach. On summarization the HEXA based representation with using variable strides get a result of two to five folds reduction in memory. So we include variable strides to HEXA will not only gives reductions in the on-chip memory but also yield higher throughput at lower power dissipation levels.

V. CONCLUSION

To implement high speed complex pattern matching operations fast content inspection systems are required. The system throughput can be increased many folds by using variable stride pattern matching algorithm. This algorithm is very memory efficient algorithm because it uses self synchronized segments of patterns to recognize patterns in a data stream. In this paper, for string matching purpose in the place of DFA we develop HEXA, which is a Compact DFA. In general normal DFA's uses pointers to identify the next state in traversing and it requires more memory. To overcome this HEXA uses a unique method to locate the state while transition in memory. Compared to standard representation the memory requirement for HEXA us less and significantly more compact. Due to these two implementations as a combination (depends on winnowing window size) the memory usage is decreased by increasing the throughput of a data packet multiple times in a network by pattern matching process Intrusion Detection System.

VI. REFERENCES


[19] Will Eatherton, John Williams, “An encoded version of reg-ex database from cisco systems provided for research purposes”.


