UDC 004.056
DOI: 10.32626/2308-5916.2021-22.67-76

Madjit Karimov*, D-r of Tech. Science, Professor,
Mirkhusan Sagatov**

* State Test Center under the Cabinet of Ministers of
the Republic of Uzbekistan, Tashkent, Republic of Uzbekistan,
**Tashkent State Technical University
named after Islam Karimov, Tashkent, Republic of Uzbekistan

APPLICATION THE AHO-CORASICK ALGORITHM FOR IMPROVING A INTRUSION DETECTION SYSTEM

One of the main goals of studying pattern matching techniques is their significant role in real-world applications, such as the intrusion detection systems branch. The purpose of the network attack detection systems NIDS is to protect the infocommunication network from unauthorized access. This article provides an analysis of the exact match and fuzzy matching methods, and discusses a new implementation of the classic Aho-Korasik pattern matching algorithm at the hardware level. The proposed approach to the implementation of the Aho-Korasik algorithm can make it possible to ensure the efficient use of resources, such as memory and energy.

Key words: *NIDS (Network Intrusion Detection System), precise matching, imprecise matching, FPGA (field-programmable gate array), algorithm Aho-Corasick.*

Introduction. In recent years, there has been rapid growth in both Internet penetration and bandwidth, driven by huge improvements in telecommunications infrastructure, the proliferation of competitively priced computers and mobile Internet-capable devices, and the declining cost of Internet access as a result of increased competition. The number of individuals using the Internet has increased by several billion over the past 10 years, mainly due to the increased use of high quality video communications.

In 2019, Positive Technologies specialists recorded more than 1,500 attacks; this is 19% more than in 2018. In 81% of cyber attacks, the victims were legal entities. At the end of the year, the five most frequently attacked industries included government agencies, industry, medicine, science and education, and the financial industry.

In September 2020, the AVTest Institute detected about 1.1 billion unique malicious programs, of which 12 million are new malware. In other words, a new malware was created every 2 seconds.

In 2019, the number of malware infections increased by 38% compared to 2018. In 41% of cases, malware infections were combined with social engineering techniques.

The growth in the success of malicious campaigns throughout the year was facilitated by the continuous modernization of both the malicious software itself and the delivery methods. First, in 2019, attackers were good at masking malware. For example, they hid them in files with extensions that are included in the whitelists and therefore are not detected by antiviruses, used legitimate processes and built-in mechanisms to avoid detection, signed malware with legitimate certificates, and actively developed fileless infection techniques. Trend Micro researchers published a report in September 2019, according to which the number of fileless attacks in the first half of the year increased by 265% compared to the first half of 2018. At the end of the year, Bitdefender specialists talked about a new technique for infiltration by miners, ransomware and spyware through the features of the RDP service. Second, cybercriminals added new exploits to malware vulnerabilities, including in widely used software. For example, the notorious WinRAR vulnerability in 2019, which affected half a billion users, was used both for infections by the JNEC.a ransomware and in complex targeted attacks. Finally, attackers tried to make the malware feature rich, which increased its chances of gaining benefit if it became infected. For example, the new Scranos rootkit steals credentials and billing information, installs adware, and subscribes to YouTube channels.

Network packet inspection is the examination of a packet's payload for patterns known as signatures, listed in a rule database called a rule set. Signatures are usually in the form of fixed strings or regular expressions, or a combination of both. In recent years, regular expressions have become more commonly used to describe increasingly complex attacks.

The topic of fixed string matching is well understood because of its importance in many applications such as Internet search engines, parsers, word processors, and digital libraries. This is important in signature-based NIDPS because most rules contain at least one fixed string pattern to match. Although fixed string matching is beyond the scope of this paper, a brief overview is provided below to give a complete understanding of the functionality of NIDPS.

Related works. Precise Matching. The string matching problem can be simply formulated — for two strings T and P of length m and n, respectively, determine if P occurs in T. Naive or brute force search involves trying to match a pattern using a window size of length n and iterating over each position in T from left to right, resulting in the worst-case complexity $O(mn)$. Boyer-Moore and KMP are two classic single-string matching algorithms. Both of these algorithms also use a window of size n, but they use a skip or shift table to determine where to look next after each mismatch. The shifts used by the Boyer-Moore algorithm are based on two rules known as the bad character shift rule and the good suffix shift

rule. The first rule eliminates the need to repeat unsuccessful comparisons with the target character, and the second ensures that the match only matches target characters already successfully matched. The KMP algorithm similarly uses information derived from partial matches to skip alignments that are guaranteed not to result in a match. The Boyer-Moore algorithm was later simplified by Horspool, resulting in an algorithm that is easier to implement. The Boyer-Moore algorithm has a worst-case search time of $O(m + n)$ if the pattern does not appear in the text, and $O(mn)$ if it does. The average seek time is sublinear and improves with increasing pattern length. KMP is $O(m + n)$ in both the average and worst case. Baeza-Yates and Gonnett found that the average performance of the Boyer-Moore-Horspool algorithm improves with increasing pattern length, and better than KMP for $n > 3$. These algorithms are not suitable for matching multiple patterns because the search time increases linearly with increasing template length, number of patterns.

Imprecise Matching. Dharmapurikar et al. describe a hardware technique using Bloom filters to detect fixed strings in streaming data. A Bloom filter is a randomized data structure that is «programmed» with strings using multiple hash functions and «queried» for a string based on a few bits. The request may result in a false positive, but never a false negative. (A false positive is when a match result incorrectly indicates that a match exists, while a false negative is when a match result incorrectly indicates that a match does not exist). The main advantage of this method is that it will probably only require a relatively small amount of memory, even for a very large set of templates. The disadvantages are that multiple Bloom filters are required, one for each pattern length found in the rule set, and that all possible matches must be fully checked for false positives. Song and Lockwood propose a more efficient data structure, called an extended Bloom filter, in an architecture that makes the most of FPGA block RAM. Zhou and Wang propose an FPGA implementation of multi-pattern string matching using parallel mechanisms based on the Bloom counting filter.

Markatos et al. propose an algorithm based on the use of exclusion matching. It basically splits patterns into multiple fixed size bit strings and searches for them without checking if they are in the correct sequence. If any of the subpatterns do not match, then the entire pattern does not match. When a matching subpattern is found, the system reverts to a standard algorithm, such as Boyer-Moore, to validate the complete pattern.

Algorithmic background-The Aho-Corassic and Commenzt-Walter algorithms. Two well-known multi-pattern matching algorithms are Aho-Corasick and Commenzt-Walter. The Aho-Corasick algorithm is an extension of the KMP algorithm for a set of templates.

This paragraph is devoted to providing a brief explanation of the pattern matching problem. In short, in the problem of the exact set matching algorithm, the main goal is to find occurrences of all patterns from a given set $P = \{p_1 \ldots p_k\}$ in the text $T[1 \ldots m]$. Let n be the length of all patterns in the set P. Obviously, the brute-force algorithm is probably the first algorithm one could think of when solving an exact match of a set in $O(n + km)$ time, applying any linear time an exact match k times e.g. Knuth-Morris-Pratt algorithm, Boyer-Moore algorithm, etc.). The Aho-Corasic algorithm (AC for short) is a natural extension of the famous Knuth-Morris-Pratt algorithm, which is a classic approach to solving a single pattern matching problem. AC works in $O(n + m + z)$ time, where z is the number of occurrences of patterns in the text T. AC is based on the refinement of the keyword tree.

Consider the following set of lines: P = {ATTACK, ASSET, CAT}. The keyword tree for P is shown in the figure.

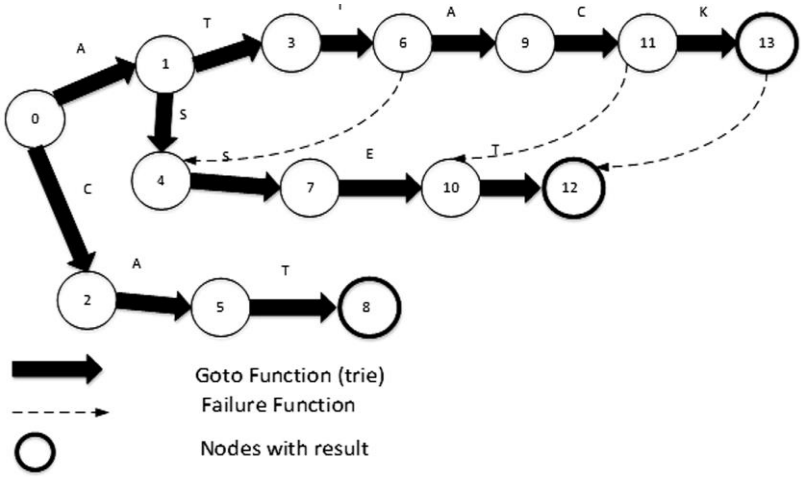

Fig. 1. *Aho-Corasick — automaton for set of patterns P = {ATTACK, ASSET, CAT}*

Time complexity of tree construction is $O(n)$. The next step is to convert the keyword tree to an automaton to support linear time matching. The action of the automaton is determined by three functions defined for states [8]:

$g(s, a)$ — a goto function: gives the states entered from current state s by matching text char a, if edge (u, v) is labeled by a, then $g(u, v) = v$; and $g(0, a) = 0$ for each a that does not label an edge out of the root (denoted 0),

$f(s)$ — a failure function: gives the state entered at a mismatch, when w is the longest proper suffix of $L(v)$ such that w is a prefix of some pattern, $f(s)$ is the node labeled by w,

out(*s*) — an output function: gives the set of patterns recognized when entering state.

If n is the number of states in the automaton, and *nocc* is the number of occurrences of the pattern in the string, then the time complexity of finding the algorithm is $O(n + nocc)$ when the transitions of the automaton are stored in the transition table and, $O(nlog|\Sigma| + nocc)$, when transitions are stored in a balanced tree [2, 5].

Theorem. Searching for a set of patterns P in text string $T[1 \ldots m]$ with an AC automaton takes time $O(m + z)$, where z is the number of pattern occurrences.

A transition table of a FSM automaton contains transitions from each state for every symbol of a input alphabet, so the size of this transition table is equal #S · #A (where S is the set of automaton states, A is the input alphabet and #X is a cardinality of set X). The transition table for an AC automaton contains only two transitions for each state (2·#S). Number of transitions is not dependant on size of the input alphabet.

Algorithm1. Searching patterns P in text T

```
Input:   string T[1,…,m]
         set of k-patterns P;
output:  starting indices of substrings of T
         matching patterns from set P;
  q=0;
  for i=1 to m do
         while g(q, T[i])=0 do q=f(q);
         q=g(q,T[i]);
         if out(q) ≠ 0 then print (i);
```

The Commentz-Walter algorithm combines the ideas of the Boyer-Moore and Aho-Corasick algorithms. For a string of length m and maximum template length *lmax*, its worst-case time complexity is $O(mlmax)$. In practice, it is faster than Aho-Corasick only for a small number of search patterns.

Both Aho-Corasick and Commentz-Walter algorithms suffer from the fact that memory requirements can grow exponentially with the number of patterns. This degrades the performance of the software as the entire machine cannot be stored in the cache. A number of solutions have been proposed for this memory explosion problem, most of which involve the use of hash tables. The Wu-Manber algorithm is a variant of the Boyer-Moore algorithm with multiple patterns that considers text in B blocks instead of individual characters, that is, it is a multi-step algorithm. It uses three tables: SHIFT, PREFIX and HASH. The SHIFT table stores the shift or gap values for each of the characters in the block, indexed by hashing

their value. If a potential match is found, the HASH and PREFIX tables are accessed to verify the actual match. Navarro and Raffino provide a detailed description accompanied by examples. The algorithm only requires $O(k)$ memory space, where k is the number of templates, and is very fast on average. It was previously used by Snort, but has been removed because its inferior performance makes it vulnerable to DoS attacks. Snort now uses the standard Aho-Corasick algorithm by default, but it can be configured to use other versions of the algorithm [3] that trade off memory versus speed. It also includes a binary tree based algorithm known as SFK lookup for very low memory systems.

A lot of research has been done to find improved variants of the Aho-Corasick algorithm, in particular for hardware implementation. The algorithms proposed by Tuck et al. reduce memory consumption through the use of bitmap nodes and path compression. Bitmaps reduce the number of state transition pointers, and path compression combines a number of sequential states. Tan and Sherwood use bit splitting to split the Aho-Corasick automaton into eight separate automata, each operating on one bit of each input character, thereby reducing the maximum number of transitions from each state from 256 to just 2.

Kennedy et al. proposed an FPGA architecture based on the Aho-Corasick algorithm, which uses multiple string matching mechanisms operating in parallel.

A well-known FPGA approach to string matching is to treat a string as a simple regular expression that can be represented by an NFA, which translates into FPGA logic. The main disadvantage of this method is the need to reprogram the FPGA when the rowset changes. Moreover, it does not scale well as recent rulesets generate too much logic.

TCAMs can perform concurrent searches at high speed, but they present two problems for matching multiple patterns: (i) TCAM entries are fixed length, as opposed to string patterns found in NIDS rulesets, and (ii) TCAMs return the first matching entry, and not all matches. Yu et al. propose a solution that overcomes these two difficulties. The number of TCAM lookups is in the order of $O(n)$, where n is the number of characters entered. Sung et al. [1, 4, 6, 7] present a jump window scheme that reduces the number of TCAM lookups to $O(n / m)$, where m is the window size. Although it gives very good matching performance, TCAM suffers from the problems of relatively high cost and energy inefficiency.

Hardware schemes for state machines. Finite machine. A block diagram of a state machine that finds character patterns is shown in Figure 3. The module consists of a character shift register, a state register, and a look-up table. The state register contains the code of the current state of the state machine.

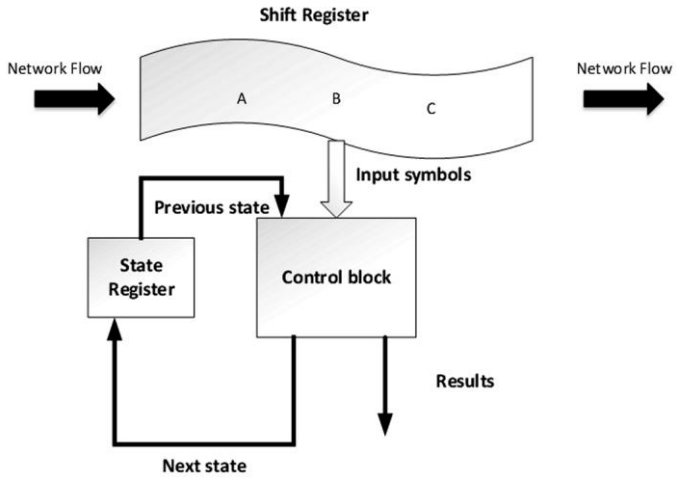

Fig. 2. *Block scheme of finite state machine*

The shift register shifts a new character into the look-up table at each clock cycle. The lookup table determines the next state from the current state and the input character. It also detects output patterns found, which tells whether the state machine has found a pattern. Much of the circuit's latency comes from a look-up table, which can be implemented in FPGA memory or logic cells.

Aho Corasick Machine. The module that implements the Aho-Korasik algorithm (shown in Figure 4) is similar to the basic module of the state machine. The most important extension is the brake light. It is set when the state machine returns through a failure transition and tells the shift register not to shift a new character. The conversion table has been converted to a more complex block. The table is split into two tables: a transition function and a failure function. The goto function determines the next state based on the input character and the current state in a situation where the Aho-Corasick machine matches any pattern. It is implemented in Addressable Memory (CAM). The number of CAM lines is equal to the number of goto branches in the keyword tree. The stop memory output signals whether the next state was executed in the CAM or not.

The failure function finds the next state from the current state when the input character does not match any pattern. Each state has one transition of the failure function, so it is easy to implement in memory, the size of which is proportional to the number of states.

The stop signal activates the next state multiplexer function. If the CAM finds a matching row, the next state is taken from the CAM pin, otherwise it is taken from the failure function. The templates function reports from the next status output which templates were found.

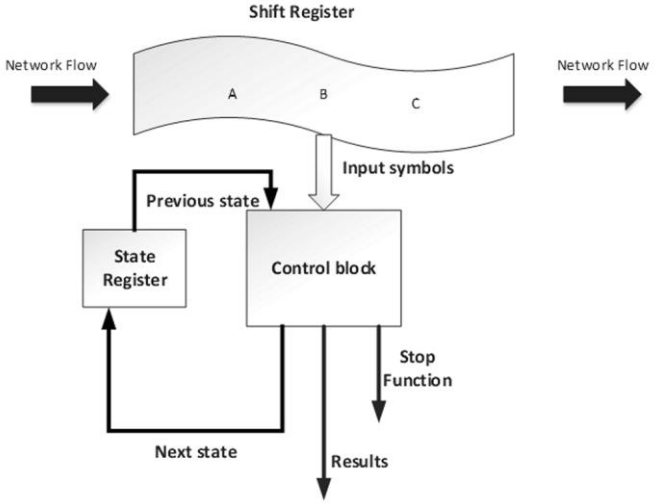

Fig. 3. *Block diagram of AC machine*

Parallel Aho-Corasick Machine. Taking into account the fact that the size of the Aho-Corasick jump table is independent of the size of the input alphabet, it is possible to parallelize and pipe it by taking multiple characters at each step and multiplying the number of Aho-Corasick Machine. This increases the size of the input alphabet and changes the number of states of the machine.

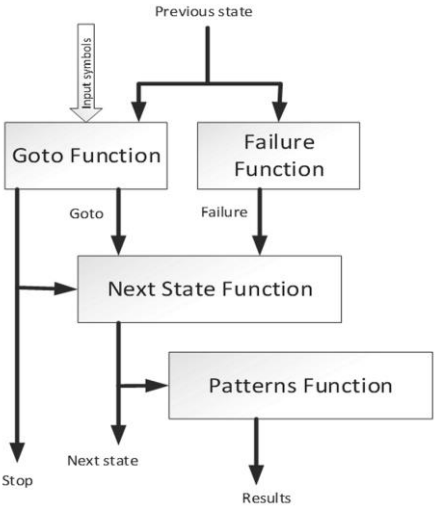

Fig. 4. *Block diagram of control unit of AC Machine*

Each n Aho-Corasick machine takes n input symbols from the shift register every n clock cycles. Each machine receives symbols in its own clock cycle, so every n adjacent symbols in the shift register are processed by one of the machines. The Aho-Corasick automaton is modified to process n symbols at each step. It can handle n steps of the goto function, but only one step of the failure function of the underlying Aho-Corasick machine.

The advantage of the parallel version is that machines can share the resources of an expensive CAM for the transition function and memory for the failure function. Thus, the memory is pipelined, resulting in a shorter clock cycle than the basic version of the AhoCorasick machine. The disadvantage is that the length of the templates must be divisible by n.

Conclusion. This article proposes a new implementation of the classic Aho-Corasik pattern matching algorithm at the hardware level. The presented results show that the Aho-Corasick algorithm provides efficient use of resources (memory and logical cells) at the expense of lower throughput. Using this algorithm, the same patterns can be accommodated as in a smaller FPGA. Further improvements are expected in the area of efficient hardware implementation of pattern matching algorithms.

References:

1. Bakhodir Y., Nurbek, N., Odiljon Z. Methods for applying of scheme of packet filtering rules. *International Journal of Innovative Technology and Exploring Engineering.* 2019. Vol. 8. Is. 11. P. 1014-1019.
2. Karimovich G. S., Turakulovich K. Z., Ubaydullayevna H. I. Computer's source based (Pseudo) random number generation. *International Conference on Information Science and Communications Technologies, ICISCT 2017*. Tashkent, 2017. CFP17H74-ART;133832;
3. Sherzod G., Dilmurod A., Nodira M., Husniya A. Construction of schemes, models and algorithm for detection network attacks in computer networks. *International Journal of Innovative Technology and Exploring Engineering*. 2019. Vol. 8. Is. 12. P. 2234-2241.
4. Karimov M., Tashev K., Yoriqulov M. Problems of increasing efficiency of NIDS by using implementing methods packet classifications on FPGA. *International Conference on Information Science and Communications Technologies, ICISCT 2019*. Tashkent, 2019. CFP19H74-ART; 158120.
5. Akhmatovich T. K., Turakulovich K. Z., Tileubayevna A. J. Improvement of a security enhanced one-time mutual authentication and key agreement scheme. *International Journal of Innovative Technology and Exploring Engineering.* 2019. Vol. 8. Is. 12. P. 5031-5036.
6. Karimov M., Gulomov Sh., Yusupov B. Method of constructing packet filtering rules. *International conference on information science and communications technologies applications, trends and opportunities (ICISCT)*.Tashkent, 2019.
7. Abdulatteef S. W. An Implementation of Firewall System Using MikroTik Router OS. *Journal of University of Anbar for Pure Science*. 2012. Vol. 6 (2). P. 65-69.

8. Zhang L., Wang Y., Jin R., Gao K. Approaches for a Stand-alone Network Attack and Defense Platform Using Yersinia Toolkits. *International Journal of All Research Education and Scientific Methods (IJARESM).* 2017. Vol. 5. Is. 3. P. 2455-6211.

ЗАСТОСУВАННЯ АЛГОРИТМА АГВ-КОРАСІКА ДЛЯ ВДОСКОНАЛЕННЯ СИСТЕМА ВИЯВЛЕННЯ ВТОРГНЕНЬ

Одна з основних цілей вивчення методів зіставлення зі зразком — їх значна роль в реальних додатках, таких як гілка систем виявлення вторгнень. Метою систем виявлення мережевих атак NIDS є захист інфокомунікаційної мережі від несанкціонованого доступу. У цій статті представлений аналіз методів точного і нечіткого зіставлення, а також обговорюється нова реалізація класичного алгоритму зіставлення зі зразком Ахо-Корасіка на апаратному рівні. Пропонований підхід до реалізації алгоритму Ахо-Корасіка може дозволити забезпечити ефективне використання ресурсів, таких як пам'ять і енергія.

Ключові слова: *NIDS, точний збіг, неточне збіг, FPGA, алгоритм Ахо-Корасік.*

Отримано: 12.10.2021

УДК 681.513
DOI: 10.32626/2308-5916.2021-22.76-87

В. М. Мельник, д-р техн. наук, професор,
В. П. Косова,
К. А. Бурсаков

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського», м. Київ

ОКРЕСЛЕННЯ ГРАНИЧНИХ ТА ФІЗИЧНИХ УМОВ МАТЕМАТИЧНОЇ МОДЕЛІ МАСООБМІННУ В АПАРАТІ ПІД ЧАС РОЗДІЛЕННЯ ВОДНО-ОРГАНІЧНИХ СУМІШЕЙ

Авторами розроблена та удосконалена математична модель, яка описує масообмінну обстановку в апараті під час розділення водно-органічних сумішей та показує процеси первапорації, що відбуваються з процесу десорбції компонентів з мембранного елемента з урахуванням взаємних впливів характеристик процесу на характеристики середовища. Враховано вплив зовнішніх факторів для температурного розрахунку та знайдено розв'язок відповідної модельної задачі з використанням рівняння руху для рідини в середині мембрани в умовах ламінарного руху. Наведені результати розрахунків розподілу концентрації органічної домішки у суміші та матеріалі мембрани. Досліджено залежність кількості проходів через мембрану