Bulletin of TUIT: Management and Communication Technologies

Volume 4 *2021-fourth number* Article 6

9-20-2021

Application of the Aho-Corasick algorithm to create a network intrusion detection system

Komil Tashev
"Bulletin of TUIT: Management and Communication Technologies", k.akhmatovich@gmail.com

Mokhinabonu Agzamova
Bulletin of TUIT: Management and Communication Technologies, mshagzamova@gmail.com

Nozima Axmedova
"Bulletin of TUIT: Management and Communication Technologies, rrsanobar18@mail.ru

Recommended Citation

Tashev, Komil; Agzamova, Mokhinabonu; and Axmedova, Nozima (2021) "Application of the Aho-Corasick algorithm to create a network intrusion detection system," *Bulletin of TUIT: Management and Communication Technologies*: Vol. 4 , Article 6.
DOI: https://doi.org/10.51348/tuitmct446
Available at: https://uzjournals.edu.uz/tuitmct/vol4/iss4/6

Application of the Aho-Corasick algorithm to create a network intrusion detection system

Komil Tashev
Cryptology
Tashkent university of information technologies named after Muhammad al-Khwarizmi
Tashkent, Uzbekistan
k.akhmatovich@gmail.com

Agzamova Mokhinabonu
Providing Information security
Tashkent university of information technologies named after Muhammad al-Khwarizmi
Tashkent, Uzbekistan
mshagzamova@gmail.com

Axmedova Nozima
Cryptology
Tashkent university of information technologies named after Muhammad al-Khwarizmi
Tashkent, Uzbekistan
rrsanobar18@mail.ru

***Abstract*— Nowadays the Internet is very massive and the need for a system to protect the networks from being attacked becomes very important. 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-Corasick pattern matching algorithm at the hardware level. The proposed approach to the implementation of the Aho-Corasick algorithm can make it possible to ensure the efficient use of resources, such as memory and energy.**

Keywords—NIDS, precise matching, imprecise matching, FPGA, algorithm Aho-Corasick.

I. Introduction

The computer networks are growing exponentially and the use of personal computers are spreading widely in homes, companies and almost everywhere. This leads to increase the subscibers of the Internet services. The number of individuals using the Internet has increased from 1 billion in 2005 to over 4,6 billion in 2021 which is 58% of the world population, according to Cisco`s forecast (Cisco Visual Networking Index Complete Forecast, Cisco VNI).In particular, the number of Internet users increased three times in the Republic of Uzbekistan from 6 million in 2013 to more than 22.5 million in 2020, which is 67% of the total population of the country. Generally, the computer systems contain important data, such as users` information and business activities. At the same time, there are a huge number of data available and accessible from the Internet by any user.

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. [1]

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. [1]

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 NIDS 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 NIDS.

II. 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 [2] and KMP [3] 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

[4], 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 [5] 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. [6] describe a hardware technique using Bloom filters [7] 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 [8] 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 [9] propose an FPGA implementation of multi-pattern string matching using parallel mechanisms based on the Bloom counting filter.

Markatos et al. [10] 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.

Various algorithms and methods of pattern matching are classified according to specific categories, for example:

1. General, - There can be several architectures where the packet capturing, packet decoding, and event detection modules are simultaneously discussed in a single solution. For example, the solutions in [11,12] and [16] discuss the packet capturing, decoding and event detection/engine modules without describing the packet pre-processing module. Therefore, all the aforementioned architectures are incorporated in "General" category.

2. DFA, - The finite automaton (FA) is one of the prominent techniques in event detection module of pattern matching. Generally, it works like a finite state machine (FSM) to recognize the stored patterns within the incoming streams of input characters. The computation of FA mainly depends upon its five tuples: (1) total number of finite states, (2) set of alphabets, (3) transition functions, (4) start state and (5) final state. It constitutes two different types: deterministic finite automaton (DFA) and non-deterministic finite automaton (NFA).

3. NFA, - The NFA based pattern matching algorithms and techniques are memory efficient, but on the other hand, they provide limited throughput due to the involvement of multiple states per input character. Consequently, various architectures have been proposed to optimize the throughput of NFA based pattern matching. For example, the pipelined architecture to process multiple state transitions at the same time is described in [28]. Similarly, an architecture for translating the input pattern into an NFA, and then encoding using bit masking technique, is presented in [29]. Moreover, the splitting of larger patterns into smaller sets is shown in [42].

4. Hash - Mapping the variable length of input data onto a fixed length is known as hashing. There are a number of architectures, where the hashing technique has been utilized, either on the incoming input packets from the Ethernet [31] or on the stored variable pattern lengths in memory [33]

5. Tree, - A tree refers to the data structures with a capability to store the incoming streams of characters. Several architectures have been developed to store the fixed length patterns into the data structures [35]. Similarly, various architectures have been designed where the larger pattern sets are divided into groups of smaller sets and the smaller grouped sets are stored in the data structures [36]

Algorithms and methods for matching strings.

For string matching algorithms and methods, 27 studies were selected, as shown in Table 4. Of these 27 studies, 23 studies use pattern matching algorithms and the remaining 4 studies use different pattern matching methods. Hence, the identified string matching algorithms are: Likewise, the identified methods are diode-resistor logic using CMOL [9], a combined state transition approach [18], auto mapping [20], and a range tree reference list [24].

For the general category, the Aho - Corasick algorithm is used in [1] and [8], the KMP (Knuth - Morris - Pratt) algorithm is considered in [2], the Booyer - Moore algorithm is used in [3], the State - Traversal algorithm is used in [5], and the Shift - And algorithm is implemented in [4,6] and [7]. A high bandwidth architecture for line matching using diode-resistor logic with molecular FPGA (CMOL) technology is presented in [9]. The run-time configurable rule architecture in [1] reduces system downtime whenever a ruleset needs to be updated. A multistage pipelined architecture for executing the Aho-Korasic algorithm using fast sampling using the on-demand validation approach is achieved in [8]. The scalable accelerator in [2] is more focused on optimizing the power of string matching. For Transmission Control Protocol (TCP) packets, real-time string matching is performed using the SoC platform described in [3]. As far as memory optimization is concerned, the solution shown in [4] uses splitting long pattern sets into smaller sets. Classification and grouping of homogeneous traffic with its subsequent forwarding to the appropriate hardware unit for processing is considered in [6]. A solution for matching multi-character strings using line alignment detection and correction is presented in [7], and a memory-optimized solution is presented in [5].

For the DFA category, the Aho - Corasick algorithm is implemented in [10,13] and [14], and the Bit - Split algorithm is used in [11] and [12]. To optimize the memory of the Aho - Korasik algorithm, a fail-safe pipelined DFA was built in [14], the reduction of the state graph to a symbolic tree was implemented in [10], and the efficient use of FPGA LUTs is aimed at [13]. The memory optimization of the Bit - Split algorithm is shown in [11] and [12]. The grouping of longer template sets into multiple fixed-length sub-templates is presented in [12], while the group of unique and non-unique sub-templates is aimed at [11].

For the NFA category, the Aho - Corasick algorithm is used in [15] and [16], the Shift - And algorithm is considered in [17], and the Bit - Parallel algorithm is considered in [19]. A hybrid solution using both NFA and DFA in [16] is presented by checking multiple characters in parallel, which causes alignment problems. Hence, a security solution in terms of helper transitions for the alignment problem is presented in [15]. For the Shift - And algorithm, a pipelined solution to optimize pattern matching throughput is shown in [17]. [19] discusses a dynamic reconfigurable architecture for precise and extended pattern sets using the Bit-Parallel algorithm. A pooled state transition approach using a combination of both common common prefixes and no-failure transitions is developed in [18]. For applications of network security and bioinformatics, a finite state machine for pattern matching was constructed in [20].

For the hash category, the algorithms Bloom - Filter [21], Pattern_Matching [22] and Cuckoo [23] are considered. The reconfigurable solution in [21] leans more towards area and power optimization. The Pattern_Matching algorithm in [22] performs content matching using a variable length pattern. A highly scalable and energy efficient string matching solution using the Cuckoo algorithm is presented in [23].

The tree-based category in [25,26] and [27] uses only the Binary Search Tree (BST) algorithm. In addition, [24] expanded the list of range tree references and value-encoded tree structures to address backtracking and memory issues. The solutions shown in [26] and [27] provide memory optimization by constructing data structures. For memory optimization, pipelined throughput is also discussed in [25].

Table 1. Implementation details for string matching algorithms and techniques

Ref#	**Algorithm/technique**	**Rule set**	**Total rules**	**Targeted board**
General Category				
[11]	Aho-Corasick	SNORT	16K	Xilinx ZC706
[12]	KMP	-	-	Xillinx Zedboard
[13]	Boyer-Moore	-	-	Zynq 7000
[14]	Shift-And	-	524,287	-
[15]	State-Traversal	SNORT	2217	-
[16]	Shift-And	SNORT	5567	Invea Combo_Ixt
[17]	Shift-And	SNORT	3095	-
[18]	Aho-Corasick	ClamAV	82,000	-
[19]	Diode-Resistor logic using CMOL	-	10 million	-
DFA Category				
[20]	Aho-Corasick	SNORT	6166	Altera
[21]	Bit-Split	-	-	Altera Quartus II
[22]	Bit-Split	SNORT	7784	-
[23]	Aho-Corasick	SNORT	2200	-
[24]	Aho-Corasick	-	-	-
NFA Category				
[25]	Aho-Corasick	SNORT	1000	-
[26]	Aho-Corasick	-	-	-
[27]	Shift-And	SNORT	<1000	-
[28]	Merged state transition approach	Web	-	-
[29]	Bit-Split	-	-	-
[30]	Automation mapping	SNORT	4312	-
Hash Category				
[31]	Bloom-Filter	TNTG	16,028	-
[32]	Pattern_ Matching	-	-	-
[33]	Cuckoo	-	-	-
Tree Category				
[34]	Range tree link list	ACL	10K	-
[35]	BST	SNORT	16,816	-
[36]	BST	SNORT	11,895	-
[37]	BST	Rogets SNORT	21,667 8368	-

Algorithms and methods for matching strings. 27 studies were selected for string matching algorithms and methods as shown in Table 1. 23 studies use pattern matching algorithms from these 27 studies, and the remaining 4 studies use different pattern matching methods. Hence, the identified string matching algorithms are: Aho - Corasick, Knuth Morris Pratt, Booyer - Moore, State - Traversal, Shift - And, Bit - Split, Bit - Parallel, Bloom - filter, Pattern_Matching, Cuckoo, and Binary Search Tree

III. NFA based Aho-Corasick algorithm

A string P of length $|P| = m$ over a given finite alphabet Σ is any sequence of m characters of Σ. For $m = 0$, we obtain the empty string ε. Σ^* is the collection of all finite strings over Σ. We denote by $P[i]$ the $(i+1)$-st character of P, for $0 \le i < 1$. Likewise, the substring of P contained beween $(i+1)$-st and the $(j+1)$-st characters of P is denoted by $P[i..j]$, for $0 \le i \le j < m$. We also put $P_i =_{Def} P[0..i]$, for $0 \le i < m$, and make convention that P_{-1} denotes the empty string ε. We write P^r to denote the reverse of the string P, i.e., $P^r = P[m-1]P[m-2]\dots P[0]$. Given a finite set of patterns $\mathcal{P}$, we put $\mathcal{P}^r =_{Def} \{P^r | P \in \mathcal{P}\}$ and $\mathcal{P}_l =_{Def} \{P[0..l-1] | P \in \mathcal{P}\}$.

Also we put $size(\mathcal{P}) =_{Def} \sum_{P\in\mathcal{P}} |P|$ and extend the maps $Fact(\cdot)$ to $\mathcal{P}$ by putting $Fact(\mathcal{P}) =_{Def} \bigcup_{P\in\mathcal{P}} Fact(P)$.

We recall the notation of some bitwise infix operators on computer words, namely the bitwise and «&», the bitwise or «|», the $left\ shift$ "$\ll$" operator (which shifts to the left its first argument by a number of bits equal to its second argument), and the unary bitwise *not* operator "~". The functions that compute the first and the last bit set to 1 of a word x are $\lfloor \log_2(x\&(\sim x+1))\rfloor$ and $\lfloor \log_2(x)\rfloor$, respectively [2].

A nondeterministic finite automation (NFA) with ε – transitions is a 5-tuple $N = (Q, \Sigma, \delta, q_0, F)$, where Q is a set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the collection of final states, Σ is an alphabet, and $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ is the transition function ($\mathcal{P}(\cdot)$ is the powerset operator).In the case of an NFA without ε –transitions, we have $\delta: Q \times \Sigma \to \mathcal{P}(Q)$ [6].

Both the transition function δ and the extended transition function δ^* can be naturally generalized to handle set of states, by putting $\delta(D,c) =_{Def} \bigcup_{q\in D}\delta(q,c)$ and $\delta^*(D,u) =_{Def} \bigcup_{q\in D}\delta^*(q,u)$, respectively, for $D \subseteq Q$, $c \in \Sigma$ и $u \in \Sigma^*$. The extended transition function satisfies the following property:

$\delta^*(q,u,v) = \delta^*(\delta^*(q,u),v)$, for all $u, v \in \Sigma^*$. (1)

Given a set $\mathcal{P}$ of patterns over a finite alphabet Σ, the trie $\mathcal{T}_{\mathcal{P}}$ associated with $\mathcal{P}$ is a rooted directed tree, whose edges are labeled by single characters of Σ, such that

(i) distinct edges out of the same node are labeled by distinct characters,

(ii) all paths in $\mathcal{T}_{\mathcal{P}}$ from the root are labeled by prefixes of the strings in $\mathcal{P}$,

(iii) for each string P in $\mathcal{P}$ there exists a path in $\mathcal{T}_{\mathcal{P}}$ from the root which is labeled by P.

For any node p in the trie $\mathcal{T}_{\mathcal{P}}$, we denote by $lbl(p)$ the string which labels the path from the root of $\mathcal{T}_{\mathcal{P}}$ to p and put $len(p) =_{Def} |lbl(p)|$. Plainly, the map lbl is injective. Additionally, for any edge (p,q) in $\mathcal{T}_{\mathcal{P}}$, the label of (p,q) is denoted by $lbl(p,q)$.

For a set of patterns $\mathcal{P} = \{P_1, P_2, \dots, P_r\}$ over an alphabet Σ, the maximal trie of $\mathcal{P}$ is the trie $\mathcal{T}_{\mathcal{P}}^{max}$ obtained by merging into a single node the roots of the linear tries $\mathcal{T}_{\mathcal{P}_1}, \mathcal{T}_{\mathcal{P}_2}, \dots, \mathcal{T}_{\mathcal{P}_r}$ relative to the patterns $P_1, P_2, \dots, P_r$ respectively. Strictly speaking, the maximal trie is a nondeterministic trie, as property (i) above may not hold at the root.

The directed acyclic word graph (DAWG) for a finite set of patterns $\mathcal{P}$ is a data structure representing the set $Fact(\mathcal{P})$. To describe it precisely, we need the following definitions. Let us denote by $end-pos(u)$ the set of all positions in $\mathcal{P}$ where an occurrence of u ends, for $u \in \Sigma^*$; more formally, we put $end-pos(u) =_{Def} \{(P,j) \,|\, u \sqsupseteq P_j,\ \text{с}\ P \in \mathcal{P}\ \text{и} |u| - 1 \le j < |P|\}$.

For instance, we have $end-pos(\varepsilon) = \{(P,j) | P \in \mathcal{P}$ and $-1 \le j < |P|\}$, since $\varepsilon \sqsupseteq P_j$, for each $P \in \mathcal{P}$ and $-1 \le j < |P|$ (we recall that $P_{-1} = \varepsilon$, by convention).

We also define an equivalence relation $R_{\mathcal{P}}$ over Σ^*, by putting

$u\ R_{\mathcal{P}}\ v \quad \Leftrightarrow_{Def} \quad end-pos(u) = end-pos(v)$, 2)

for, $v \in \Sigma^*$, and denote by $R_{\mathcal{P}}(u)$ the equivalence class of $R_{\mathcal{P}}$ containing the string u. Also, we put

$val(R_{\mathcal{P}}(u)) =_{Def}$ the longest string in the equivalence class $R_{\mathcal{P}}(u)$. (3)

Then the DAWG for a finite set $\mathcal{P}$ of patterns is a directed acyclic graph (V, E) with an edge labeling function $lbl()$, where

$V = \{R_{\mathcal{P}}(u) | u \in Fact(\mathcal{P})\}$, $E = \{(R_{\mathcal{P}}(u), R_{\mathcal{P}}(uc)) | u \in \Sigma^*, c \in \Sigma, uc \in Fact(\mathcal{P})\}$

and $lbl((R_{\mathcal{P}}(u), (R_{\mathcal{P}}(uc)) = c$, for $u \in \Sigma^*, c \in \Sigma$ such that $uc \in Fact(\mathcal{P})$ [3].

We define below the Aho-Corasick NFA for a set $\mathcal{P}$ of patterns.

Aho–Corasick NFA

The Aho-Corasick NFA for a set $\mathcal{P}$ of patterns over an alphabet Σ is induced directly by the trie $\mathcal{T}_{\mathcal{P}}$ for $\mathcal{P}$. More precisely, it is the NFA $A_{\mathcal{P}} = (Q, \Sigma, \delta_A, q_0, F)$, where [7]:

- Q is the set of nodes of $\mathcal{T}_{\mathcal{P}}$ (the set of states);
- $q_0 \in Q$ is the root of $\mathcal{T}_{\mathcal{P}}$ (the initial state);
- $\delta_A: Q \times \Sigma \to \mathcal{P}(Q)$ is the transition function, with

$$\delta_A(q,c) =_{Def} \begin{cases} \{p \in Q | lbl(p) = c\} \cup \{q_0\} & if\ q \neq q_0, \\ \{p \in Q | lbl(p) = lbl(q).c\} & if\ q \neq q_0, \end{cases}$$

for $\in Q$, $c \in \Sigma$, and where we recall that $\mathcal{P}(\cdot)$ denotes the powerset operator;

- $F =_{Def} \{q \in Q | lbl(q) \in \mathcal{P}\}$ is the set of final states.

Plainly we have $|Q| \le \sum_{P\in\mathcal{P}} |P|$.

We also associate with the NFA $A_{\mathcal{P}}$ a failure function $fail: Q\setminus\{q_0\} \to Q$ such that

- $lbl(fail(q)) \sqsupset lbl(q)$, and
- $len(fail(q)) \ge len(p)$, for each $p \in Q$ such that $lbl(p) \sqsupset lbl(q)$

(in other words, $lbl(fail(q))$ is the longest proper suffix of $lbl(q)$, which is also a prefix of a string $\mathcal{P}$).

The automaton $A_{\mathcal{P}}$ can be seen as the nondeterministic version of the Aho-Corasick automaton: this is a trie $\mathcal{T}_{\mathcal{P}}$ for a set of patterns $\mathcal{P}$ augmented with failure links, which are followed when no transition is possible on a text character [1].

An immediate, yet useful, property of the Aho-Corasick NFA, which can be readily proved by induction, is the following

$q_0 \in \delta_A{}^*(q_0, u)$, for every $u \in \Sigma^*$. (4)

The Aho – Corasick NFA $A_{\mathcal{P}} = (Q, \Sigma, \delta_A, q_0, F)$ relative to a given set $\mathcal{P}$ of patterns can be used to find the occurrences of the patterns of $\mathcal{P}$ in a given text T, by observing that a pattern $P \in \mathcal{P}$ has an occurrence in T ending at position i, i.e., $P \sqsupseteq T_i$, if and only if $\delta_A{}^*(q_0, T[0..i])$ contains a final state $q_0 \in F$ such that $lbl(q) = P$. Thus, to find all the occurrences in T of the patterns of $\mathcal{P}$, it is enough to compute the set $\delta_A{}^*(q_0, T_i) \cap F$, for $= 0, 1, \dots, |T| - 1$. As an immediate consequence of (1) and the definitions of δ_A and $\delta_A{}^*$ on $\mathcal{P}(Q)$, we have $\delta_A{}^*(q_0, T_i) = \delta_A(\delta_A{}^*(q_0, T_{i-1}), T[i])$, for $i = 1, 2, \dots, |T| - 1$. Hence, the problem of computing efficiently the sets $\delta_A{}^*(q_0, T_i)$ can be reduced to the problem of evaluating efficiently transition actions of the form $\delta_A(D, c)$, for any $c \in \Sigma$ and any reachable configuration $D \subseteq Q$ of $A_{\mathcal{P}}$, namely any subset $D \subseteq Q$ such that $D = \delta_A{}^*(q_0, u)$, for some $u \in \Sigma^*$.

The following property is an immediate consequence of the definition of the failure function.

Lemma 1. Given the Aho–Corasick NFA $A_{\mathcal{P}} = (Q, \Sigma, \delta_A, q_0, F)$ for a set $\mathcal{P}$ of patterns and its associated failure function fail: $Q\setminus\{q_0\} \to Q$, we have *[7]*

$lbl(p) \sqsupset lbl(q) \to lbl(p) \sqsupseteq lbl(fail(q))$, For all $p \in Q$ and $q \in Q \setminus \{q_0\}$.

Bit-parallel simulation of NFAs for the multiple string matching problem.

To simulate efficiently the NFAs $A_{\mathcal{P}}$ P with the bit-parallel technique, a suitable representation of the transition function δ is needed, in order that $\delta(D,c)$ can be computed by $O\lceil |Q|/w \rceil$ computer operations, for any reachable configuration $D \subseteq Q$ and character $c \in \Sigma$ (as before, w is the number of bits in a computer word).

Our construction is based on a result for the Glushkov automaton that can be immediately generalized to NFAs like $A_{\mathcal{P}}$, as follows [5].

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA with ε- transitions such that up to the ε- transitions, for each state $c \in \Sigma$, either

(i) all the incoming transitions in q are labeled by the same character, or

(ii) all the incoming transitions in q originate from a unique state.

Let $B(c)$, for $c \in \Sigma$, be the set of states of N with an incoming transition labeled by c, i.e.,

$B(c) =_{Def} \{q \in Q | q \in \delta(p,c)$, for some $p \in Q\}$.

Likewise, let $Follow(q)$ for $q \in Q$, be the set of states reachable from state q with one transition over a character in Σ, i.e.

$$Follow(q) =_{Def} \bigcup_{c \in \Sigma} \delta(q,c).$$

Also, let

$$\Phi(D) =_{Def} \bigcup_{q \in D} Follow(q),$$

for $D \subseteq Q$. Then the following result holds

Lemma 2. For every $\in Q$, $D \subseteq Q$ and $c \in \Sigma$, we have *[5]*

(a) $\delta(q,c) = Follow(q) \cap B(c)$;

(b) $\delta(D,c) = \Phi(D) \cap B(c)$.

Proof. Concerning (a), we notice that $\delta(q,c) \subseteq Follow(q) \cap B(c)$ holds painly. To prove the converse inclusion, let $p \in Follow(q) \cap B(c)$. Then $p \in \delta(q,c') \cap \delta(q',c)$ for some $c' \in \Sigma$ and $q' \in Q$. If p satisfies condition (i), then $c' = c$, and therefore $p \in \delta(q,c)$. On the other hand, if p satisfies condition (ii), then $q = q'$ and therefore we have again $p \in \delta(q,c)$.

From (a), we obtain immediately (b), since [7]

$$\delta(D,c) = \bigcup_{q \in D} \delta(q,c) = \bigcup_{q \in D} (Follow(q) \cap B(c)) = \bigcup_{q \in D} Follow(q) \cap B(c) = \Phi(D) \cap B(c)$$

Provided that one finds an efficient way of storing and accessing the maps $\Phi(\cdot)$ and $B(\cdot)$, equation (b) of Lemma 2 is particularly suitable for bit-parallelism, as set intersection can be readily implemented by the bitwise and operation.

We observe at once that the immediate solution of storing the maps $\Phi(\cdot)$ and $B(\cdot)$ as tables of bit words, respectively indexed by set of states and by characters in Σ, requires $(2^m + \sigma) \cdot m$ bits, which is exponential in the number m of states $A_{\mathcal{P}}$ (σ is the size of the alphabet Σ). Thus we have to find a better way to store the map $\Phi(\cdot)$, exploiting the fact that $\Phi(D)$ needs to be evaluated over reachable configurations D for $A_{\mathcal{P}}$.

Bit-parallel simulation of the Aho–Corasick NFA for a set of patterns

We first show that each reachable configuration of $A_{\mathcal{P}}$ is uniquely identified by a single state. Then, we devise the map $\dot{\Phi}(\cdot)$, by using the relation between reachable configurations of the automaton and the associated failure function, and prove its correctness. Finally, we show that the map $lead(\cdot)$admits an efficient implementation.

We begin by showing in the following elementary lemma that, for any string u, the configuration of the automaton after reading u consists of all the states whose labels are a suffix of u.

Lemma 3. Let $A_{\mathcal{P}} = (Q, \Sigma, \delta, q_0, F)$ be the Aho-Corasick *NFA* for a finite set $\mathcal{P}$ of patterns over the alphabet Σ , and let $p \in \Sigma^*$. *Тогда* $\delta^*(q_0,u) = \{q \in Q | lbl(q) \sqsupseteq u\}$.

Proof. For $u = \varepsilon$, the lemma holds plainly. Thus, let $u = u'.c$, with $u' \in \Sigma^*$ and $c \in \Sigma$. We first show by induction on u that $\delta^*(q_0,u) \subseteq \{q \in Q | lbl(q) \sqsupseteq u\}$. Let $p \in \delta^*(q_0,u)$. Since, by (1)

$$\delta^*(q_0,u'.c) = \delta^*(\delta^*(q_0,u'),c) = \delta(\delta^*(q_0,u'),c) = \bigcup_{q \in \delta^*(q_0,u')} \delta(q,c),$$

we have $p \in \delta(\bar{q},c)$, for some $\bar{q} \in \delta^*(q_0,u')$, so that, by inductive hypothesis, $lbl(\bar{q}) \sqsupseteq u'$ and, therefore, $lbl(p) = lbl(\bar{q}).c \sqsupseteq u'.c = u$.

To show the converse inclusion relationship, let $p \in Q$ be such that $lbl(p) \sqsupseteq u$. We prove by induction on $lbl(p)$, that $p \in \delta^*(q_0,u)$. In view of (4), we may dismiss at once the case in which $lbl(p) = \varepsilon$, i.e., $p = q_0$, and therefore assume that $lbl(p) = lbl(p').c$, for some $p' \in Q$ and $c \in \Sigma$. Hence $u = u'.c$ for some $u \in \Sigma^*$ such that $lbl(p') \sqsupseteq u'$, so that, by inductive hypothesis, we have $p' \in \delta^*(q_0,u')$. Thus, by (1)

$$p \in \delta(p',c) \subseteq \delta(\delta^*(q_0,u'),c) = \delta^*(\delta^*(q_0,u'),c) = \delta^*(q_0,u'.c) = \delta^*(q_0,u)$$

.

Given a reachable configuration D, the previous lemma implies that for any two distinct states $p, p' \in D$ we have $|lbl(p)| \neq: |lbl(p')|$, since either $lbl(p) \sqsupset lbl(p')$ or $lbl(p') \sqsupset lbl(p)$. Thus there must exist a unique state $\bar{q} \in D$ such that $|lbl(p)| \leq |lbl(\bar{q})|$, for every $p \in D$. Let us denote such a state by $lead(D)$. Then we have:

Corrolary 1. Let $A_{\mathcal{P}} = (Q, \Sigma, \delta, q_0, F)$ be the Aho-Corasick NFA for a finite set $\mathcal{P}$ of patterns Σ, and let D be a reachable configuration of $A_{\mathcal{P}}$. Then $D = \{q \in Q | lbl(q) \sqsupseteq lbl(lead(D))\}$.

Proof. Let $u \in \Sigma^*$ be such that $D = \delta^*(q_0, u)$. In view of Lemma 3, it is enough to observe that $lbl(p) \sqsupseteq u$ if and only if $lbl(q) \sqsupseteq lbl(lead(D))$, for every q∈Q $q \in Q$.

From the preceding corollary, it follows at once that the reachable configurations of the Aho-Corasick NFA $A_{\mathcal{P}} = (Q, \Sigma, \delta, q_0, F$, for a set $\mathcal{P}$ of patterns, are in 1-1 correspondence with its states, and therefore their number is $|Q|$.

A convenient way to represent Φ uses the map $\dot{\Phi}_A : Q \to \mathcal{P}(Q)$, recursively defined by

$$\dot{\Phi}_A(q) =_{Def} \begin{cases} Follow(q_0) & if\ q = q_0, \\ Follow(q) \cup \dot{\Phi}_A(fail(q)), & if\ q \neq q_0, \end{cases} \quad (5)$$

As shown in the following lemma.

Lemma 4. For any reachable configuration D of the Aho-Corasick NFA $A_{\mathcal{P}}$, we have $\Phi(D) = \dot{\Phi}_A(lead(D))$.

Proof. We proceed by induction on $|lbl(lead(D))|$. If $|lbl(lead(D))| = 0$, then $lead(D) = q_0$ and $D = \{q_0\}$, so that $\Phi(D) = Follow(q_0) = \dot{\Phi}_A(q_0) = \dot{\Phi}_A(lead(D))$. For the inductive step, we have D in

$$\begin{aligned}\Phi(D) &= \bigcup_{q \in D} Follow(q) = \bigcup_{\substack{q \in D \\ lbl(q) \sqsupseteq lbl(lead(D))}} Follow(q) \\ &= Follow(lead(D)) \cup \bigcup_{\substack{q \in Q \\ lbl(q) \sqsupset lbl(lead(D))}} Follow(q) \\ &= Follow(lead(D)) \cup \bigcup_{\substack{q \in Q \\ lbl(q) \sqsupseteq lbl(fail(lead(D)))}} Follow(q) \\ &= Follow(lead(D)) \cup \Phi\left(\left\{q \in Q \middle| lbl(q) \sqsupseteq lbl\left(fail(lead(D))\right)\right\}\right) \\ &= Follow(lead(D)) \cup \dot{\Phi}_A(fail(lead(D))) = \dot{\Phi}_A(lead(D))\end{aligned}$$

Plainly, the map $\dot{\Phi}_A(\cdot)$ requires only $|Q|^2$ bits. Additionally, the map $lead(\cdot)$ can be computed very efficiently at run-time, provided that the states of $A_{\mathcal{P}}$ are ordered in such a way that a state p precedes a state q whenever $|lbl(p)| < |lbl(q)|$ (say, by a breadth-first visit of $A_{\mathcal{P}}$ from q_0). Indeed, in such a case, if we assume that D is encoded as a bit mask, then $lead(D)$ is the index of the highest bit of D set to 1, and therefore is equal to $\lfloor \log_2 D \rfloor$.

The Log-And algorithm

Based on the previous considerations, we present an efficient bit-parallel algorithm, which we call Log-And, for solving the multiple string matching problem.

In the Log-And algorithm, reported in Fig.1, the sets D, B and the map $\dot{\Phi}_A$ are encoded as bit tables.

As opposed to the Shift-And algorithm, bit 0 is reserved for the initial state, so that $lead(D)$ is never computed for an empty set (0 value) as the initial state is always active.

In the preprocessing phase, the Log-And algorithm iterates over the nodes of $A_{\mathcal{P}}$, which are assumed to be sorted by a breadth-first search; for each node, the corresponding Φ mask is computed using (4), and the B masks associated to the labels of its outgoing edges are augmented accordingly. The algorithm precomputes also a final state bit mask, L, where a bit is set to 1 if and only if it corresponds to a final state of the automation. The maps $Folow\ (\cdot)$, $lbl(\cdot)$ and $fail(\cdot)$ can be constructed using the algorithm introduced in [1], while the loop iterating over the states of the automation in breadth-first order can be easily implemented using a queue.

Log-And $(T, \mathcal{P} = \{P_1, P_2, \dots, P_r\})$

/*Preprocessing */

1. Let $A_{\mathcal{P}} = (Q, \Sigma, \delta, q_0, F)$ be the Aho-Corasick NFA relative to the set of patterns $\mathcal{P}$ and let the maps $Follow(), lbl()$, and $fail()$ be defined as before, relative to $A_{\mathcal{P}}$. We also assume that $Q = \{0, 1, \dots, \ell - 1\}$, where

$\ell = |Q|$, and that if $|lbl(p)| < |lbl(q)|$ then $p < q$, for any $p, q \in Q$.

```
2.  $L \leftarrow 0^{\ell}$
3.  **for** $c \in \Sigma$ **do** $B[c] \leftarrow 0^{\ell-1}1$
4.  **for** $p \leftarrow 0$ **to** $\ell - 1$ **do**
5.          $\Phi_A[p] \leftarrow 0^{\ell-1}1$
6.          **for** $q \in Follow(p)\backslash\{0\}$ **do**
7.                  $H \leftarrow 0^{\ell-1}1 \ll q$
8.                  $c \leftarrow lbl(p,q)$
9.                  $B[c] \leftarrow B[c]|H$
10.                 if $q \in F$ then $L \leftarrow L|H$
11.                 $\Phi_A[p] \leftarrow \Phi_A[p]|H$
12.         **if** $p \neq 0$ **then**
13.                 $\Phi_A[p] \leftarrow \Phi_A[p]|\Phi_A[fail(p)]$
/* Searching */
14. $D \leftarrow 0^{\ell-1}1$
15. **for** $j \leftarrow 0$ **to** $|T| - 1$ **do**
16.         $lead \leftarrow \lfloor \log_2(D) \rfloor$
17.         $D \leftarrow \Phi_A[lead]\&B[T[j]]$
18.         **if** $D\&L \neq 0^{\ell}$ **then** $Output(j)$
```

Fig.1. The Log-And algorithm for the multiple string matching problem

Then, during the searching phase, the Log-And algorithm scans the text T, character by character, using the following basic transition, based on Lemma 2 (b),

$$D \leftarrow \dot{\Phi}_A[\lfloor \log_2(D) \rfloor]\&B[c].$$

The resulting algorithm has $O((m+\sigma)\lceil m/w \rceil)$ - space and $O(n\lceil m/w \rceil)$ - searching time complexity, where $n = |T|$, m is the number of nodes of $A_{\mathcal{P}}$, σ is the alphabet size, and w is the word size in bits. When $m \in O(w)$, the Log-And algorithm turns out to have an $O(m+\sigma)$ -space and $O(n)$ -searching time complexity.

If one is interested also in retrieving the patterns that match (if any) at each final state of $A_{\mathcal{P}}$ to the corresponding pattern index. Then, in the searching phase, for each position j, the algorithm iterates over the bits of $(D \,\&\, L)$ by computing the index of the highest bit set and querying the corresponding pattern number. The whole sequence is repeated, after having cleared the highest bit, until there are no more bits set.

IV. 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] Current Cyber Threats: Results of 2019 - https://www.ptsecurity.com/ru-ru/research/analytics/cybersecurity-threatscape-2019/

[2] Boyer R.S. and Moore, J.S. (1977). A Fast String Searching Algorithm. Communications of ACM, 20(10), pp.762-772.

[3] Knuth, D.E., Morris, J. and Pratt, V.R. (1977). Fast Pattern Matching in Strings. SIAM Journal on Computing. 6(2), pp.323-350

[4] Horspool, R.N. (1980). Practical fast searching in strings. Software: Practice and Experience. 10(6), pp.501-506.

[5] Baeza-Yates, R. and Gonnet, G.H. (1992). A new approach to text searching. Communications of the ACM , 35(10), pp.74-82.

[6] Dharmapurikar, S., Krishnamurthy, P., Sproull, T. and Lockwood, J.W. (2003). Deep Packet Inspection Using Parallel Bloom Filters. Proceedings 11th Symposium on High Performance Interconnects (HotInterconnects). pp.44-51.

[7] Bloom, B.H. (1970). Space/time trade-offs in hash coding with allowable errors. Commun. ACM. 13(7), pp.422-426.

[8] Song, H. and Lockwood, J.W. (2005b). Multi-pattern Signature Matching for Hardware Network Intrusion Detection Systems. IEEE Global Telecommunications Conference GLOBECOM'05. vol.3, 5 pages.

[9] Zhou, Y. and Wang, X. 2010. Efficient Pattern Matching with Counting Bloom Filter. CIICT2010.

[10] Markatos, E., Antonatos, S., Polychronakis, M. and Anagnostakis, K. (2002). Exclusion-based Signature Matching for Intrusion Detection. In Proceedings of IASTED International Conference on Communications and Computer Networks (CCN 2002);

[11] Tharaka PMK, Wijerathne DMD, Perera N, Vishwajith D, Pasqual A. Runtime rule-reconfigurable high throughput NIPS on FPGA. In: International Conference on Field Programmable Technology (ICFPT); 2017. p. 251–4.

[12] Lei S, Wang C, Fang H, Li X, Zhou X. SCADIS: a scalable accelerator for data-intensive string set matching on FPGAs. In: IEEE Trustcom/BigDataSE/ISPA; 2017. p. 1190–7.

[13] Domínguez A, Carballo PP, Nunez A. Programmable SoC platform for deep packet inspection using enhanced boyer-moore algorithm. In: 12th international symposium on reconfigurable communication-centric systems-on-chip (ReCoSoC); 2017. p. 1–8.

[14] Sarbishei I, Vakili S, Langlois JMP, Savaria Y. Scalable memory-less architecture for string matching with FPGAs. In: IEEE International Symposium on Circuits and Systems (ISCAS); 2017. p. 1–4.

[15] Lin CH, Chang SC. Efficient pattern matching algorithm for memory architecture. In: IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 19; 2011. p. 33–40.

[16] Pontarelli S, Bianchi G, Teofili S. Traffic-aware design of a high-speed FPGA network intrusion detection system. In: IEEE transactions on computers. 62; 2013. p. 2322–33.

[17] Bande JM, Palancar JH, Cumplido R. Multi-character cost-effective and high throughput architecture for content scanning. Microprocess Microsyst 2013;37: 1200–7.

[18] Pao D, Wang X. Multi-stride string searching for high-speed content inspection. Comput J 2012;55:1216–31.

[19] Madhavan A, Sherwood T, Strukov DB. High-throughput pattern matching with CMOL FPGA circuits: case for logic-in-memory computing. In: IEEE transactions on very large scale integration (VLSI) systems. 26; 2018. p. 2759–72.

[20] PAO D, LIN W, LIU B. A memory-efficient pipelined implementation of the Aho-Corasick string-matching algorithm. ACM Trans Architect Code Optimiz 2010;7: 1–27.

[21] Kim H, Choi K, Choi S. A memory-efficient deterministic finite automaton-based bit-split string matching scheme using pattern uniqueness in deep packet inspection. PLoSONE 2015;10:1–24.

[22] Kim HJ, Kim HS, Kang S. A memory-efficient bit-split parallel string matching using pattern dividing for intrusion detection systems. In: IEEE transactions on parallel and distributed systems. 22; 2011. p. 1904–11.

[23] Wang X, Pao D. Memory-based architecture for multicharacter Aho–Corasick string matching. In: IEEE transactions on very large scale integration (VLSI) systems. 26; 2017. p. 143–54.

[24] Kim H. A failureless pipelined Aho-Corasick algorithm for FPGA-based parallel string matching engine. Lect Notes Electric Eng (LNCS) 2015;339:157–64.

[25] Chen CC, Wang S. An efficient multi-character transition string-matching engine based on the Aho-Corasick algorithm. ACM Trans Architect Code Optimiz 2013; 10:1–22.

[26] Chen CC, Wang S. A hybrid multiple-character transition finite-automaton for string matching engine. Microprocess Microsyst 2015;39:1–13.

[27] Serrano JMB, Palancar JH. String alignment pre-detection using unique subsequences for FPGA-based network intrusion detection. Comput Commun 2012;35: 720–8.

[28] Kim H, Choi KI. A pipelined non-deterministic finite automaton-based string matching scheme using merged state transitions in an FPGA. PLoSONE 2016;11: 1–24.

[29] Kaneta Y, Yoshizawa S, Minato SI, Arimura H, Miyanaga Y. Dynamic reconfigurable bit-parallel architecture for large-scale regular expression matching. In: International Conference on Field-Programmable Technology; 2010. p. 21–8.

[30] Roy I, Srivastava A, Nourian M, Becchi M, Aluru S. High performance pattern matching using the automata processor. In: IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2016. p. 1123–32.

[31] Arun M, Krishnan A. Functional verification of signature detection architectures for high speed network applications. Int J Autom Comput 2012;9:395–402.

[32] Xue CJ, Liu M, Zhuge Q, Sha EHM. Variable length pattern matching for hardware network intrusion detection system. J Signal Process Syst 2010;59:85–93.

[33] Thinh TN, Kittitornkun S. Massively parallel cuckoo pattern matching applied for NIDS/NIPS. In: 5th IEEE International Symposium on Electronic Design, Test & Applications, Ho Chi Minh; 2010. p. 217–21.

[34] Erdem G, Carus A. Multi-pipelined and memory-efficient packet classification engines on FPGAs. Comput Commun 2015;67:75–91.

[35] Erdem O. Tree-based string pattern matching on FPGAs. Comput Electric Eng 2016;49:117–33.

[36] Hajiabadi MH, Saidi H, Behdadfar M. Scalable high-throughput and modular hardware based string matching algorithm. In: 11th International ISC Conference on Information Security and Cryptology; 2014. p. 192–8.

[37] Le H, Prasanna VK. A memory-efficient and modular approach for large-scale string pattern matching. IEEE Trans Comput 2013;62:844–57

[38] Aho, A. and M. Corasick (1975). Fast pattern matching: an aid to bibliographic search. In: Communications of the ACM. Vol. 18. pp. 333–340.

[39] Commentz-Walter, B. (1979). A String Matching Algorithm Fast on the Average. Lecture Notes in Computer Science. 71, pp.118-132.

[40] Charras, C. and T. Lecroq (2004). Exact String Matching Algorithms. King’s College London Publications. London, UK.

[41] Navarro, G. and Raffinot, M. (2002). Flexible Pattern Matching in Strings: Practical On-Line Search Algorithms for Texts and Biological Sequences. Cambridge University Press, New York, NY, USA.

[42] Wu, S. and Manber, U. (1994). A fast algorithm for multi-pattern searching. Technical Report TR94-17, Department of Computer Science, University of Arizona, 1994.

[43] Norton, M. (2004). Optimizing Pattern Matching for Intrusion Detection, Sourcefire Inc.;

[44] Tuck, N., Sherwood, T., Calder, B. and Varghese, G. (2004). Deterministic memoryefficient string matching algorithms for intrusion detection. INFOCOM 2004. 4, pp.2628-2639. March 2004.

[45] Tan, L. and Sherwood, T. (2005). A High Throughput String Matching Architecture for Intrusion Detection and Prevention. In Proceedings of the 32nd annual international symposium on Computer Architecture (ISCA '05). pp.112-122.

[46] Sidhu, R. and Prasanna, V.K. (2001). Fast Regular Expression Matching Using FPGAs. In Proceedings of the the 9th Annual IEEE Symposium on FieldProgrammable Custom Computing Machines (FCCM '01). pp.227-238.

[47] Yu, F. and Katz, R.H. (2004). Efficient Multi-Match Packet Classification with TCAM. Proceedings 12th IEEE Symposium on Hot Interconnects (HOTI’04). pp.28-34.

[48] Sung, J., Kang, S., Y. Lee, T. Kwon, and B. Kim (2005). A Multi-gigabit Rate Deep Packet Inspection Algorithm using TCAM. IEEE Global Telecommunications Conference GLOBECOM’05. vol.1, 5 pages.

[49] Bakhodir, Y., Nurbek, N., Odiljon, Z. Methods for applying of scheme of packet filtering rules. International Journal of Innovative Technology and Exploring Engineering. Volume 8, Issue 11, September 2019, Pages 1014-1019

[50] Ganiev, S.K., Irgasheva, D.Y. About of One Methods Synthesis the Structural Protected Computer Network. International Conference on Information Science and Communications Technologies, ICISCT 2019; Tashkent; Uzbekistan; November 2019 6; CFP19H74-ART; 158120;

[51] Rajaboevich, G.S., Saidaminovna, X.C., Irkinovna, G.T., Tagirovna, D.S. Analysis of methods for measuring available bandwidth and classification of network traffic. International Journal of Emerging Trends in Engineering Research. Volume 8, Issue 6, June 2020, Pages 2753-2759;

[52] 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 University of Information Technologies (TUIT)Tashkent; Uzbekistan; November 2017; CFP17H74-ART;133832;

[53] 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. Volume 8, Issue 12, October 2019, Pages 2234-2241;

[54] 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; Uzbekistan; November; CFP19H74-ART; 158120;

[55] 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. Volume 8, Issue 12, October 2019, Pages 5031-5036;

[56] Jakub Botwicz ∗ Piotr Buciak ∗ Tadeusz Łuba ∗ Henry Selvaraj ∗∗ Piotr Sapiecha ∗. Aho-Corasick algorithm implementation in hardware for network intrusion detection. IFAC Proceedings Volumes. Volume 37, Issue 20, November 2004, Pages 203-207.

[57] Karimov Madjid, Gulomov Sherzod, Yusupov Bokhodir. Method of constructing packet filtering rules. International conference on information science and communications technologies applications, trends and opportunities (ICISCT). 4-6 November, 2019, Tashkent Uzbekistan.