Implementation of a DNA-based Anomaly Identification System Utilizing Associative String Processor (ASP)

Riham Mahdy, Magdy Saeb
Arab Academy for Science, Technology & Maritime Transport
School of Engineering, Computer Department
Alexandria, Egypt
E-mail: mail@magdysaeb.net

Abstract: The genetic material that encodes the unique characteristics of each individual, such as gender, eye color, and other human features is the well-known DNA. In this work, we introduce an anomaly intrusion detection system, built on the notion of a DNA sequence or gene, which is responsible for the normal network traffic patterns. Subsequently, the system detects suspicious activities by searching the "normal behavior DNA sequence" through string matching. Conversely, string matching is a computationally intensive task and can be converted into a potential bottleneck without high-speed processing. Furthermore, conventional software-implemented string matching algorithms have not kept pace with the ever increasing network speeds. As a result, we adopt a monitoring phase that is hardware-implemented with the intention that DNA pattern matching is performed at wire-speed. Finally, we provide the details of our FPGA implementation of the bioinformatics-based string matching technique. The associative string processor (ASP) is an associative memory-based micro-architecture with long fixed-length words that can be partially searched. We show that the proposed micro-architecture can handle fixed-length patterns at a rate of more than one character per cycle.

Keywords: FPGA, anomaly identification, Network Intrusion Detection, DNA computing, pattern matching, bioinformatics, CAM.

I. Introduction

The majority of Network Intrusion Detection Systems (NIDS) inspect the network packet payload to check for predefined signature strings starting at an arbitrary location. This process is often referred to as "signature-based deep packet inspection" NID. Other systems construct a model for the normal behavior of the Network and then flag any activity that deviates from this model as a suspicious behavior or anomaly. This process is called "anomaly-based" NID. The proposed associative processing system employs the notion of combining partial information to suggest that the unknown traffic pattern is probably an attack.

Some of the previous work in anomaly identification was network-based, such as in [MAH01], which proposed a system which learns the normal range of values for 33 fields of the Ethernet, IP, TCP, UDP, and ICMP protocols. Afterwards, the system checks incoming packets for deviations from this range. Other approaches, called host-based such as in [COU03], build a user signature by encoding user's command sequences or normal behavior in a DNA strand. Then the system uses a unique variation of Smith-Waterman pair-wise alignment algorithm to compare user's current session with the pre-produced signature. Others, such as in [GAO05], profile processes according to system calls emitted by processes in response to a certain input. Subsequently, it proposes a behavioral distance as a means to detect an attack on one process that causes its behavior to deviate from that of another. The most computationally extensive part in anomaly identification is pattern-matching. Therefore, previous work in hardware exact pattern-matching architecture was proposed such as in [BU04]. Nilsen et al. [NIL04] introduced another CAM-based architecture in which the length of each word is independent from the others, in contrast to common CAMs where all words have the same length. In this work, we propose and implement a Content Addressable Memory-based micro-architecture, constructed using bioinformatics-based string matching techniques for fixed-length words. We call this micro-architecture "associative string processor" or ASP. This ASP is able to match fixed-length patterns with "Snort rules" at the rate of one character per clock cycle.

In the following discussion, we present a framework for a bioinformatics-based network anomaly identification system. The system operation is divided into two phases. In the first phase, we build the DNA sequence responsible for the normal network traffic rates. A program is developed to capture incoming packets for a predetermined time to construct the basic building structure or codon of the DNA sequence. As soon as the computer DNA sequence has been created, the second phase sends the new base structures generated by real time network activity to the FPGA pattern matching module. In this module, the arriving patterns are compared to the existing structures encoded utilizing the DNA sequence. The ASP module is a CAM-based architecture with long fixed-length words that can be partially searched. We will show that the proposed system can handle fixed-length patterns at a rate of more than one character per cycle. Moreover, we will show that the matching time depends on the number of matching characters in the pattern, only one mismatch skips loading the rest of the codon and an intrusion alarm is flagged.

In the next few sections we discuss the algorithm, the implementation details of the software application, and the building blocks of the proposed hardware module along with details of its operation. The following sections include the simulation and implementation results. Finally, we present some statistics and results after using the 1999 DARPA intrusion detection dataset for training and testing the system [DARPA99].

II. BACKGROUND

The process that was summarized in the introduction section is shown below in Figure 1. In the next few lines, we provide the necessary background in order to explain our approach.

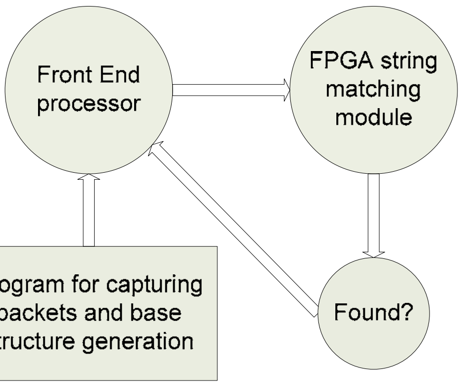

Front End processor
FPGA string matching module
Program for capturing packets and base structure generation
Found?

Figure 1: The block diagram of the system

Deoxyribonucleic Acid (DNA) is made two strands of complementary pairs of nucleotides. Each strand is made of different sequences of four bases (nucleotides) Adenine, Guanine, Cytosine, and Thymine. DNA has tremendous information storage capacity. For example, only 1 gram of DNA contains as much information as 1 trillion CD's [GEH00]. In addition, any DNA sequence can be synthesized in any desirable length. Genes contains variable-length combinations of fixed-length combinations or codons of these bases. Different combinations of codons generate different genes. Therefore, unique physical characteristics such as gender, hair color, and height are all encoded in the human gene. Recent research, *The Human Genome Project*, has shown that certain genes and mutations to other genes are responsible for undesired characteristics and certain conditions negatively affecting human health. These genes are considered DNA anomalies. It has been proven that some external factors can lead to mutations of human genes resulting in diseases such as X-ray and radioactive materials [JON04].

Yu et al., [YU01] have proposed that every computer system can be given a DNA characterization which contains all sequences or genes responsible for the important characteristics of a specific monitored system. These characteristics include network traffic, modification of system files, sequence of system calls, and user behavior. In this work, we propose and build a DNA gene responsible for the system's normal network traffic.

Prior to generating the DNA sequence for the network traffic, the structure of the fundamental base must first be determined.

As shown in Figure 2, the proposed base structure is constructed of five different categories of information that are collected for a predetermined time.

AVERAGE PACKET LENGTH	NUMBER OF TCP PACKETS	NUMBER OF UDP PACKETS	NUMBER OF ICMP PACKETS	NUMBER OF IGMP PACKETS

Figure 2: The Fundamental base structure

Similar to humans, the initial DNA sequences are predetermined on inception and ideally do not cause the system any harm. However, mutations to the DNA are possible, resulting in abnormal behavior [KIM99]. Yu proposed [YU01] that malicious software or users intruding upon a computer system can have the capability of "mutating a computer's normal characteristics" thereby causing various degrees of damage. However, by recognizing intrusions early enough, the damage can be limited and more rapid recovery is possible. Consequently, the development of an effective intrusion detection system can serve as a deterrent to intrusions as well as prevent hard-to-repair or extensive permanent damage.

III. METHODOLOGY

In the following few lines, we provide a description of the proposed system operating steps. The proposed system has two parts:

Part I: Network traffic DNA sequence generation or the offline training phase.

Part II: Network monitoring or the online monitoring phase.

Part I (TRAINING PHASE)

The aim of this phase is to collect raw data for a certain predefined time that is called the Training Time (T1) in order to produce a number of observations or base structures. The time of each observation is (T2). It is imperative that close structures are not repeated. Therefore, each field in the base structure is rounded to a certain threshold (Th).

Input: T1[training time], T2 [time of one observation], Th[threshold value]

Output: Txt file containing DNA sequence of Network traffic behavior

Algorithm Body:

```
Num of packets=0
Packet len =0
TCP num =0
UDP num =0
ICMP num =0
IGMP num=0
DNA file= file.clear()
for(T1)
    for (T2)
        for each(packet with destination address = to host m/c)
```

```
            Num of packets++
            ip= ip header
            packet len = packet len+ ip.total_len
            switch(ip.protocol)
                case 'tcp': TCP num++
                case 'udp': UDP num++
                case 'icmp': ICMP num++
                case 'igmp': IGMP num++
            end switch;
     end for;
    avg packet len = packet len / num of packets
    Round(avg packet,TCP num,UDP num,ICMP num,IGMP num, Th)
    Codon(avg packet,TCP num,UDP num,ICMP num,IGMP num)

        if (DNA file does not contain codon) then
            Add codon to DNA file
        else
            Codon freq++
end for;
Part II (MONITORING PHASE)
The aim of this phase is to monitor the rate of traffic, produce codons
(observations) and search the DNA file for it, if it does not exist then this
behavior is considered an anomaly (malicious behavior)
 Input: T2 [time of one observation] same as in phase I, Th[threshold value]
same as in phase I,
Output: codon found or not
for (T2)
        for each(packet with destination address = to host m/c)
            Num of packets++
            ip= ip header
            packet len = packet len+ ip.total_len
            switch(ip.protocol)
                case 'tcp': TCP num++
                case 'udp': UDP num++
                case 'icmp': ICMP num++
                case 'igmp': IGMP num++
            end switch;
     end for;
     avg packet len = packet len / num of packets
     Round(avg packet,TCP num,UDP num,ICMP num,IGMP num, Th)
    Codon(avg packet,TCP num,UDP num,ICMP num,IGMP num)
              Send codon to the hardware module for wire speed DNA search
                  if (found) then
                         Repeat monitoring
                   else
                        It is an attack
                      end if
End algorithm.
```

IV. IMPLEMENTATION

In this section we describe the software implementation and the hardware search module.

4.1 DNA sequence-producing Software

At the beginning, the base structures are to be defined, and then the sequences of these bases must be generated to form a "Computer Performance DNA". The base structure generation is illustrated by the example shown in Figure 3. In this example, each digit in the generated structure field is mapped into two nucleotides or base. Based on the training data, a five-digit limit was placed for each field in the base structure in order to produce a fixed size codon.

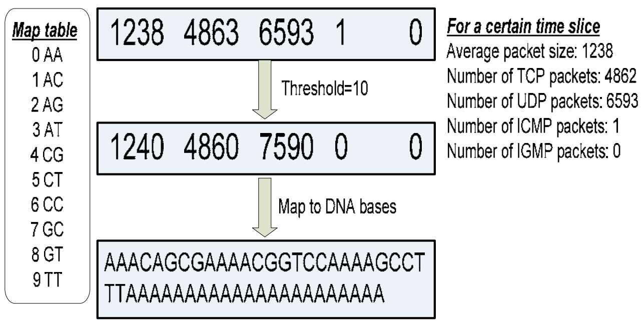


Figure 3: base structure generation steps

The software module was developed using Visual Studio .NET EDK and encoded in C# language. It is composed of an observation part that takes the time of one observation and the threshold value and produces the DNA sequence. Afterward, this sequence is downloaded onto the ASP-based hardware module for the monitoring phase. The program was developed and tested on a 1.8-GHz Pentium-based machine with 1024 MB memory.

4.2 The hardware monitoring module

The hardware module is used for performing the DNA pattern matching at the monitoring phase. Emulating physical DNA pattern matching that exploits a great level of parallelism O (1); associative string processor (ASP) modules are used to buffer the generated DNA. Each base is represented by two bits as follows: A→00, T→01, C→10 and G→11. Therefore, the resulting codon is 100-bit long.
The micro-architecture, along with its operation details, is described using a finite state machine approach (FSM) as shown in Figure 4. The machine operation takes place through seven basic states. These are summarized as follows:

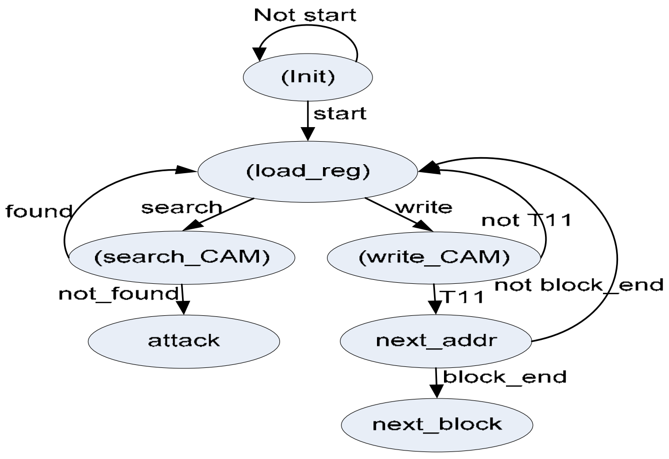


Figure 4: The finite state machine of the micro-architecture (The FPGA-based string matching module shown in Fig. 1)

The initial state “Init” holds back the execution of the successive states until the “start” signal is triggered and furthermore it resets all hardware modules. In the following state “load_reg”, a 10-bit part of a codon is latched in a 10-bit register that accepts 10 bits per each clock cycle. A write signal transfers execution to “write_CAM” state during which the 10-bit part of the normal DNA sequence is buffered into one of the ten blocks. This takes place in one of the associative string processor modules based on the module addressing signal. These two states are repeated for ten cycles to buffer one 100-bit codon. The timing signal “T11” transfers execution to “next_addr” state in which the address counter is incremented pending reaching the end of a module. As a result, the “next_block” state is activated by changing the module addressing signal. A “search” signal activates “search_CAM” state during which a new observation must be checked if it is in the normal DNA. Accordingly all of the ASP blocks are searched in-parallel. A “found” signal repeats the previous two states until the last block gives a found signal. On the other hand, a one not “found” signal alarms an attack. Figure 5 shows the top level schematic diagram of this micro-architecture.

To handle variable-length patterns, this ASP module can be further modified by adding extra base at the end of each word indicating if it is the last base or not. This technique is discussed in detail in future work.

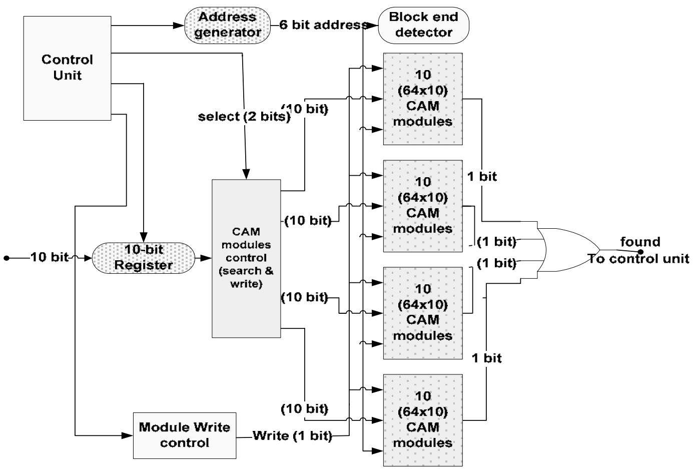


Figure 5: The block diagram of the micro-architecture

V. SOFTWARE TESTING

We performed the system evaluation on the “1999 DARPA off-line IDS evaluation data set” [LIP00]. The set consists of network traffic profiles (*tcpdump* files) that are collected at two points from a simulated local network of an imaginary air force base over a five week period. The simulated system consists of four real "victim" machines running SunOS, Solaris, Linux, and Windows NT, a Cisco router, and a simulation of a local network with hundreds of other hosts and thousands of users and an Internet connection without a firewall. We trained the system on five days of attack-free network traffic (week 1) collected from a sniffer between the router and victim machines (inside.tcpdump), and tested it on five days of traffic from the same point (week 4) during which time there were 183 attacks. The test set includes a list of all attacks, labeled with the victim IP address, time, type, and a brief description. we chose a single victim machine (172.16.112.50) and filtered all the packets destined for this machine to build its DNA sequence as it has experienced most of the denial of service attacks simulated in (week 4).

5.1 Testing results

The dataset contains three types of attacks; remote-to-local, user-to root, and denial-of-service [KEN99]. Our system builds a model for the normal network traffic pattern. Accordingly, we can focus mainly on the denial-of-service attacks. Figure 5 shows the DNA sequence graph for the chosen machine that contains 592 codons (X-axis) and the values for each field in each codon (Y-axis). Codons in this graph are generated with observation time that is equal to ten seconds. The threshold value is equal to ten and repetitions were omitted.

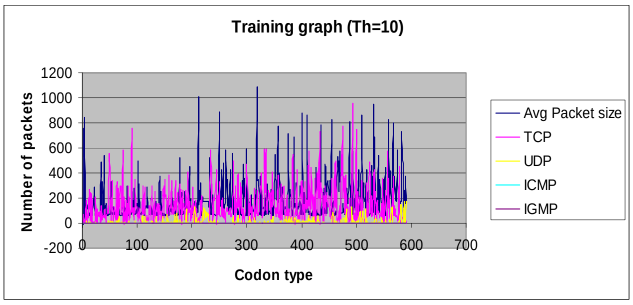


Figure 6: DNA sequence generated from week 1 training (Th=10)

Figure 6 shows codons generated with observation time equals 10 seconds, threshold value equals 1 and repetitions omitted, as the obvious number of codons is nearly six times more because similar sequences are not treated as equal.

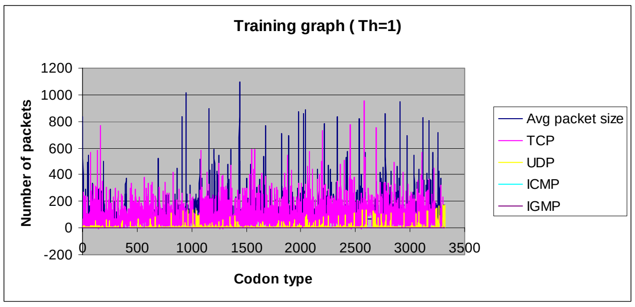


Figure 7: DNA sequence generated from week 1 training (Th=1)

We identified 370 attacking codons at different times on Monday, week 4 from the supplied attack list destined for the selected machine. There were some attacks that lasted for 5 minuets (generates up to 30 codons) and the system discovered them from the first codon. When a threshold value of 10 was used only 126 attacking codons were identified with false positive rate equal 7 per day but all attacking codons were identified with false positive equal to 453 when the threshold value is set to 1. The system can be adaptable to the level of security needed by assigning different threshold values. Figure 7 shows observations generated from Monday (inside.tcpdump) of week 4, a smurf attack shows an abnormal ICMP behavior (codons 324 and 325) and thus they are marked as attacking codons.

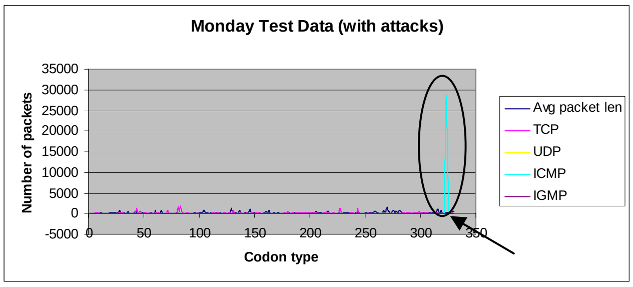
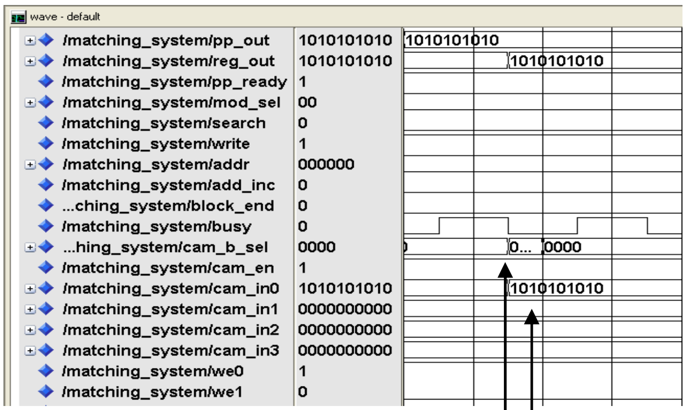


Figure 8: Codons generated from week 4 (Monday) testing data

The simulation of the designed micro-architecture is performed on the Logic Simulator of Mentor Graphics Modelsim SE 6.1. The first operation performed is latching the 10-bit register with the first 10 bits of the 100-bit codons at a rate of 10 bits per clock cycle during the "load_reg" state. The second operation is writing the register contents to the destined ASP block in the "write_CAM" state. Figure 9 depicts the operation of writing the input to the first CAM block in the first ASP module. The input 10 bits are in this case "1010101010". The "write" signal is active indicating a write operation.

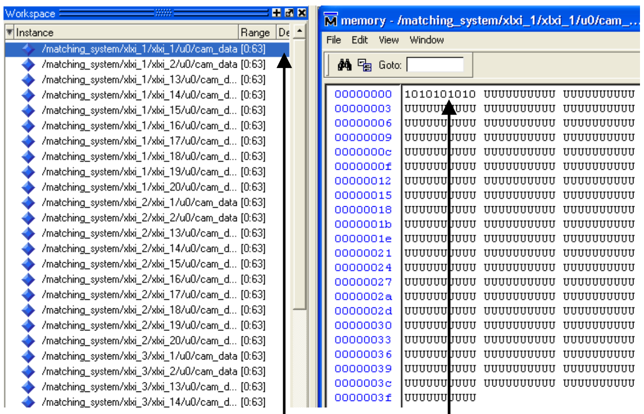

```
wave - default
/matching_system/pp_out          1010101010
/matching_system/reg_out         1010101010
/matching_system/pp_ready        1
/matching_system/mod_sel         00
/matching_system/search          0
/matching_system/write           1
/matching_system/addr            000000
/matching_system/add_inc         0
...ching_system/block_end        0
/matching_system/busy            0
...hing_system/cam_b_sel         0000
/matching_system/cam_en          1
/matching_system/cam_in0         1010101010
/matching_system/cam_in1         0000000000
/matching_system/cam_in2         0000000000
/matching_system/cam_in3         0000000000
/matching_system/we0             1
/matching_system/we1             0
```

ASP block select signals

First CAM module input

Figure 9: Simulation of performing a 10-bit write operation.

Figure 10 shows the contents of the first block in the first module.

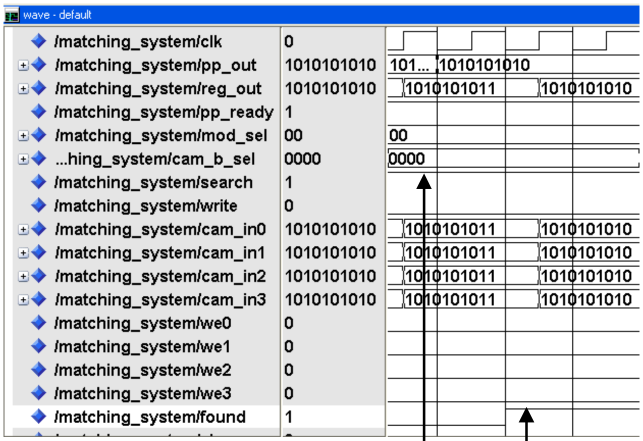

```
Workspace
Instance                                          Range
/matching_system/xlxi_1/xlxi_1/u0/cam_data [0:63]
/matching_system/xlxi_1/xlxi_2/u0/cam_data [0:63]
/matching_system/xlxi_1/xlxi_13/u0/cam_d... [0:63]
/matching_system/xlxi_1/xlxi_14/u0/cam_d... [0:63]
/matching_system/xlxi_1/xlxi_15/u0/cam_d... [0:63]
/matching_system/xlxi_1/xlxi_16/u0/cam_d... [0:63]
/matching_system/xlxi_1/xlxi_17/u0/cam_d... [0:63]
/matching_system/xlxi_1/xlxi_18/u0/cam_d... [0:63]
/matching_system/xlxi_1/xlxi_19/u0/cam_d... [0:63]
/matching_system/xlxi_1/xlxi_20/u0/cam_d... [0:63]
/matching_system/xlxi_2/xlxi_1/u0/cam_data [0:63]
/matching_system/xlxi_2/xlxi_2/u0/cam_data [0:63]
/matching_system/xlxi_2/xlxi_13/u0/cam_d... [0:63]
/matching_system/xlxi_2/xlxi_14/u0/cam_d... [0:63]
/matching_system/xlxi_2/xlxi_15/u0/cam_d... [0:63]
/matching_system/xlxi_2/xlxi_16/u0/cam_d... [0:63]
/matching_system/xlxi_2/xlxi_17/u0/cam_d... [0:63]
/matching_system/xlxi_2/xlxi_18/u0/cam_d... [0:63]
/matching_system/xlxi_2/xlxi_19/u0/cam_d... [0:63]
/matching_system/xlxi_2/xlxi_20/u0/cam_d... [0:63]
/matching_system/xlxi_3/xlxi_1/u0/cam_data [0:63]
/matching_system/xlxi_3/xlxi_2/u0/cam_data [0:63]
/matching_system/xlxi_3/xlxi_13/u0/cam_d... [0:63]
/matching_system/xlxi_3/xlxi_14/u0/cam_d... [0:63]
```
```
memory - /matching_system/xlxi_1/xlxi_1/u0/cam_...
00000000  1010101010 UUUUUUUUUU UUUUUUUUUU
00000003  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
00000006  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
00000009  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
0000000c  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
0000000f  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
00000012  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
00000015  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
00000018  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
0000001b  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
0000001e  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
00000021  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
00000024  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
00000027  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
0000002a  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
0000002d  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
00000030  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
00000033  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
00000036  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
00000039  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
0000003c  UUUUUUUUUU UUUUUUUUUU UUUUUUUUUU
0000003f  UUUUUUUUUU
```

First ASP block

Contents of address "000000"

Figure 10: Contents of the first ASP block in the first ASP module

Figure 11 provides the simulation results for the search operation. The "search" signal provides the first 10-bit part of a codon to all the first blocks in all the ASP modules to be searched in parallel, CAM0 "found" signal is high indicating that this pattern is found and so enabling the second blocks in the ASP modules with the high "found" signal till the last found which indicates a match.


```
wave - default
/matching_system/clk             0
/matching_system/pp_out          1010101010
/matching_system/reg_out         1010101010
/matching_system/pp_ready        1
/matching_system/mod_sel         00
...hing_system/cam_b_sel         0000
/matching_system/search          1
/matching_system/write           0
/matching_system/cam_in0         1010101010
/matching_system/cam_in1         1010101010
/matching_system/cam_in2         1010101010
/matching_system/cam_in3         1010101010
/matching_system/we0             0
/matching_system/we1             0
/matching_system/we2             0
/matching_system/we3             0
/matching_system/found           1
```

search signal

found signal

Figure 11: Simulation of searching for 10 bits in the four CAM modules

VII. Implementation Results

We have used the Virtex IV FPGA family to implement our design [VIRIV07]. Most of the hardware implemented string matching algorithms are variations of sequential algorithms e.g. KMP [BAK04] or ASP architectures developed for snort (IDS) rules [NIL04] and [Bu04] but our model is an emulation to DNA single-strand pattern matching process, it searches for fixed-length patterns or "codons" in a variable-length combinations of this fixed-length codons or "Gene". Therefore, comparison with other architecture is not achievable. In appendix A, we provide the implementation results using two different target devices based on the device utilization and maximum path delay. Moreover, the design floor plan is also provided. In appendix B we provide the schematics of the important micro-architecture modules.

VIII. Summary and Conclusion

Network behavior anomaly identification is a new alternative in the existing IDS technology of "anomaly identification". Anomaly identification improves on the classical "signature detection" by modeling the space of "use" instead of "misuse." It also eliminates the need for frequent signature updates and provides enhanced performance. In this work, we have introduced an anomaly identification system that encodes the network behavior in a DNA strand. The generated DNA sequence specifies the unique behavior of this monitored system. In order to perform packet inspection at wire-speed, we have proposed and implemented an FPGA-based micro-architecture employing associative string processor modules. The special features of this (IDS) can be summarized as follows:

- The system has a software interface that performs the offline training which is utilized for generating the DNA sequence.
- The system can be tailored to the degree of protection needed through changing the threshold value.
- All generated codons are of fixed size of fifty bases equivalent to 100 bits versus organic codons of three bases.
- DNA pattern matching is implemented on FPGA-based ASP to speed up online processing.
- Searching takes only one clock cycle per 10 bits using associative or content-addressable memory modules emulating organic DNA pattern matching.
- When compared to normal fixed-time ASP searches, higher expected throughput is obtained since the matching time depends on the number of matching characters in the pattern. That is, once a mismatch is detected, there will be no need for loading the rest of the codon. The proposed technique is suitable for relatively large word searches.

Based on this work, we believe that the proposed DNA-based methodology is a promising new area for IDS research. The implementation of the developed ASP micro-architecture, as demonstrated in the timing reports of Appendix A, provides a maximum path delay of slightly more than 4.7 nano seconds. This value is equivalent to about a 47-bit delay in a 10-Gigabit Ethernet. Certainly, we are not considering the buffering delays encountered with each sniffed packet. However, considering time slices between consecutive observations, we reason that such a delay is quite acceptable for most of today's fast networks.

References

[BAK04] Z. Baker, V. Prasanna, "Time and Area Efficient Pattern Matching on FPGAs," Proceedings of the 12th ACM International Symposium on Field Programmable Gate Arrays, 2004.

[BU04] Long Bu, John A. Chandy, "FPGA-Based Network Intrusion Detection using Content Addressable Memories," Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2004.

[COU03] Scott Coull, Joel Branch, Joel Branch, Eric Breimer, "Intrusion Detection: A Bioinformatics Approach," Proceedings of the 19th Annual Computer Security Applications Conference, 2003.

[DARPA99] "DARPA Intrusion Detection Evaluation Dataset", [Online source], available at: http://www.ll.mit.edu/IST/ideval/data/data_index.html.

[GAO05] Debin Gao, Michael K. Reiter, Dawn Song, "Behavioral Distance for Intrusion Detection," Proceedings of the 8th International Symposium On Recent Advances in Intrusion Detection, 2005.

[GEH00] Ashish Gehani, Thomas LaBean, John Reif, "DNA-Based Cryptography," IMACS DNA-Based Computers V, American Mathematical Society, USA, 2000.

[JON04] N. C. Jones, P. Pevzner, An Introduction to Bioinformatics Algorithms (Computational Molecular Biology), MIT Press, 2004.

[KIM99] Jungwon Kim, Peter Bentley, "The Human Immune System and Network Intrusion Detection," University College London, 1999.

[KEN99] Kristopher Kendall, "A Database of Computer attacks for the Evaluation of Intrusion Detection Systems," Masters Thesis, MIT, 1999.

[LIP00] Lippmann, R., et al., "The 1999 DARPA Off-Line Intrusion Detection Evaluation," Computer Networks, vol. 34(4), pp. 579-595, 2000.

[MAH01] Matthew V. Mahoney, Philip K. Chan, "PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic," Florida Institute of Technology Technical Report CS-2001.

[NIL04] Geir Nilsen, Jim Torresen, Oddvar Sorasen, “A Variable Word-Width Content Addressable Memory for Fast String Matching,” Proceedings of Norchip conference, 2004.
[XISE6.2i] Xilinx, Inc. Synthesis and Verification Guide, ISE 6.2i. Xilinx Inc.
[VIRIV07] Xilinx, Inc. “Virtex 4 Family Overview,” DS112 (v2.0), 2007.
[YU01] Yu, B., E. Byres, C. Howey, “Monitoring Controller’s “DNA Sequence for System Security,” BCIT, Burnaby, BC, 2001.

APPENDIX A: IMPLEMENTATION REPORTS

Implementation reports

In this appendix, we provide the details of the implementation reports as they were made available by the Xilinx CAD software [XISE6.2i].

Table 1 shows device utilization and timing report details for two implementations of our system on devices XC4VSX55 and XC4VfX100, both with speed grade equal to -12.

	XC4VSX55	*XC4vFX100*
Number of Slices:	9810 out of 24576 (39%)	9810 out of 42176 (23%)
Number of Slice Flip Flops:	3426 out of 49152 (6%)	3426 out of 84352 (4%)
Number of 4 input LUTs:	16668 out of 49152 (33%)	16668 out of 84352 (19%)
Number of bonded IOBs:	16 out of 642 (2%)	16 out of 860 (1%)
Number of FIFO16/RAMB16s:	160 out of 320 (50%)	160 out of 376 (42%)

Maximum Frequency:	89.892MHz
Maximum combinational path delay:	4.732ns

The Floor Plan (XC4VSX55)

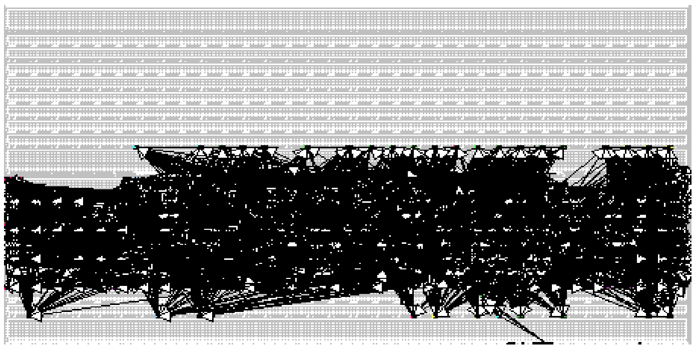


Figure 12: The floor plan

APPENDIX B: (SCHEMATICS)

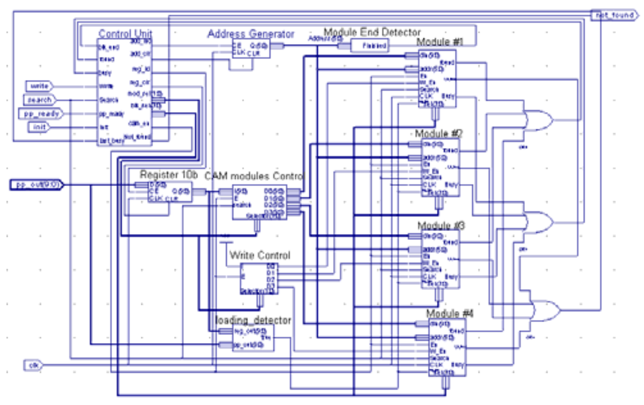


Figure 13: The circuit diagram of the entire design

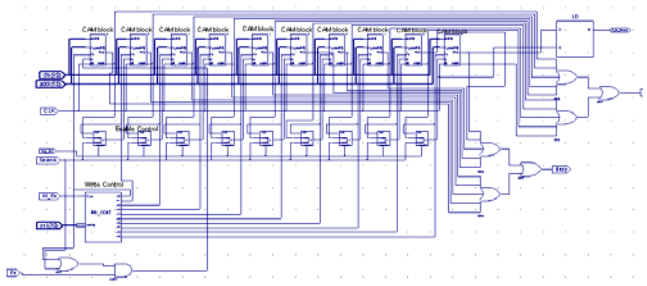


Figure 14: The circuit diagram of the ASP module.

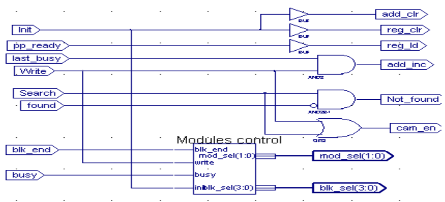


Figure 15: The circuit diagram of the control unit.

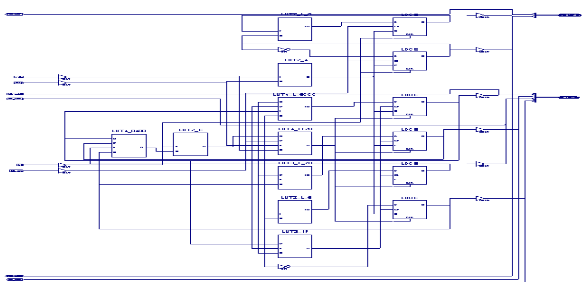


Figure 16: The circuit diagram of the modules control (Control unit)