Design of 8-bit Dedicated Microprocessor for Content Matching in NIDPS

Dejan Georgiev*

* Faculty of Electrical Engineering and Information Technologies - Skopje, Macedonia

e-mail: dejan@inbox.com

Abstract- Content or string matching is the core process of deep package inspection and pattern recognition used by the Network Intrusion Detection and Prevention Systems (NIDPS). Although there are many sophisticated algorithms in software it is an exhaustive process and still beneath the requirements of the high-speed network traffic. In this paper is presented a flexible hardware solution i.e. microprocessor able to recognize known attack patterns and its variants to overcome the software NIDPS outage caused by 1 Gbps (and beyond) throughputs on a single CPU core. Since many modified network attacks use the so called evasion techniques, the presented approach is an 8-bit dedicated microprocessor for exact and approximate string matching. To construct the design itself and to perform the simulation environment the Xilinx ISE Web Pack simulator is used.

Keywords- Data path, control unit, register, NIDPS, pattern

1. Introduction

Enormous number of malicious bits and bytes are spreading worldwide via Internet in form of worms, viruses and hacking attacks with a speed of beyond gigabit capacity networks. Identifying those potential threats requires inspection of all the single bits and bytes in the network packets, thus performing an exhaustive process of the so-called deep packet inspection. Deep package inspection (DPI) is the foundation of packet sniffing software, firewalls and network intrusion detection and prevention (NIDPS) systems. Most of the modern NIDPS as the open source Snort [1] is, accomplish DPI method. Snort maintaince DPI with *content* keyword in its rule option [2] Software solutions and general-purpose processor implementations have poor performances in term of speed for DPI, therefore many commercial and academic researches moved to hardware based systems sacrificing the flexibility in trade of speed. In this paper, an attempt to retain the speed and flexibility is made as trade off the logical circuit area. In the next chapter some existing hardware solutions and related work is briefly observed. In chapter 3 a dedicated microprocessor concept and its basic elements is proposed and in the next sub-chapters, a design of dedicated processor for exact matching is presented. Some sophisticated network attacks use slight modification in the pattern by *insertion*, *deletion* or *deletion* of one or more characters thus causing an evasion in the intrusion detection systems, therefore a design with expanded performances is explained in the next sub-chapter 3.2. The process of load and reload new patterns is explained in section 4. In chapter 5, some results and simulation of the design are presented. The goal of this paper is outlined in the conclusion section 6.

2. Background

Next generation network intrusion detection systems perform DPI by default. Taking in consideration the processing limitations of software applications, in the recent years many studies propose pattern matching in hardware. The three main designs approaches are brute-force, non- deterministic finite states automata (NFA) and deterministic finite state automata (DFA) realized on FPGA. The brute-force method produces circuits that perform a full comparison of every bit in the pattern against all the input bits.

Cho, et. al.[3] [2] presented simple design for deep packet inspection i.e. simple string searching engine and parallel and scalable design for 4-byte brute-force algorithm. In similar manner extending the brute force methods that includes matching with offsets, the authors of [4] achieved 10 Gbps throughput using parallelism. One of the main issues of the brute-force approach is its rigidness to detect patterns that are more complex, like for example the regular expressions. . Efficient logic NFA circuits for DPI with regular expressions that are able to detect zero, one or more repetitions of a character are observed by [5] . Furthermore they are the first to introduce the 8-to-256 decoder, yielding an increase of the area efficiency. Extra detection flexibility in hardware compared to the brute-force algorithm was implemented as NFA and DFA in [6] . Although FPGA's provides reconfiguration abilities, it still involves reprogramming of the circuit board. In this paper is presented significantly more flexible approach to load and reload new pattern. In particular, the dedicated processor design is combination of easy implementation of the brute-force algorithm, the extension capabilities of the NFA's and the flexibility of the general-purpose processors and software applications

3. Dedicated processor design

The simplest pattern matcher for deep packet inspection in hardware is presented by [3] . Taking in consideration the 8-bit ACSII presentation of string characters it consists of several 8-bit registers connected in pipeline or serial chain of registers as shown in Figure 1, a comparators and simple AND logical port. Since the simplest implementation of a comparator is the 8-to-1 decoder it is straightforward to conclude that an 8-input AND port is the most appropriate and area efficient solution. This is true, but not if the flexibility is the priority aspiration. Once hardwired, there is no ability to change the content of the comparator. A capable solution to load and reload new value i.e. new character could be implemented with 8-bit content addressable memories (CAM) or simple 8-bit registers.

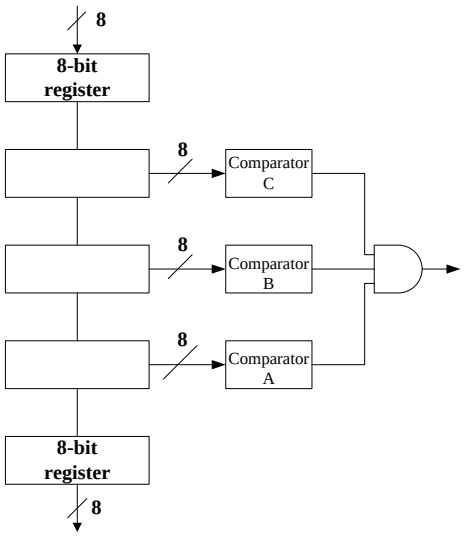

Fig 1. Simple pattern "ABC" matching hardware

In those cases, the matching process is accomplished with bit-by-bit comparison of the incoming value shifted in the registry chain and the value stored in the comparator i.e. the register. Figure 2 represents hardware comparator of two 8-bit characters stored in registers.

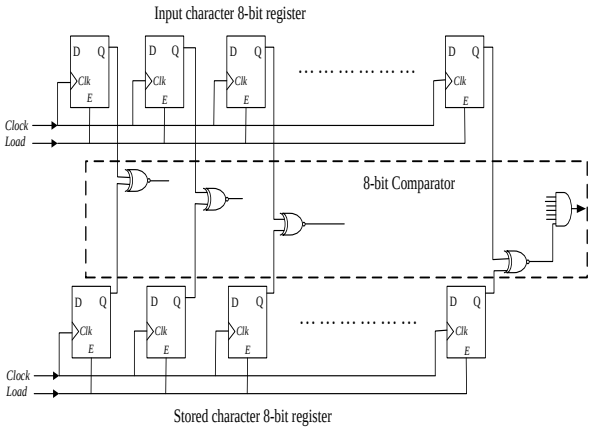

Fig 2 . Comparator of two 8-bit characters implemented with registers

It could be seen on Figure 2. that each bit comparison is performed by XNOR logical gate where the value equals to logical "1" for two same values bits and logical "0" is the bits are different in the same position of the register. With every clock cycle the value in the input registry changes according the 8-bit representation of the input character. The stored value waiting to be compared

remains the same. However, there is a possibility to reload the same value with every clock cycle and even more to load a new value with every clock cycle. Thus, instead of designing a chain of registers for the input sequence of character and a chain of registers for the pattern that should be recognized we can design an 8-bit comparator composed of eight XNOR gates and one AND gate synchronously comparing two input values i.e. characters. The values in the comparator might be loaded from real time input sequence or a pattern stored in memories or register files. This concept is the basic idea behind our dedicated microprocessor design. The design of microprocessor whether it is a general-purpose microprocessor or dedicated microprocessor, can be divided in two main parts: the *Datapath* and a *Control unit* as shown at the block diagram in Figure 3. [7] . In this paper the FSM+D (FSM plus data path) abstraction level approach for designing microprocessor is used, meaning that the datapath is manually constructed , and for constructing the control unit , in particular the FSM we use VHDL code to describe its operation. Afterword, the datapath and control unit are connected to form enclosed functional unit.

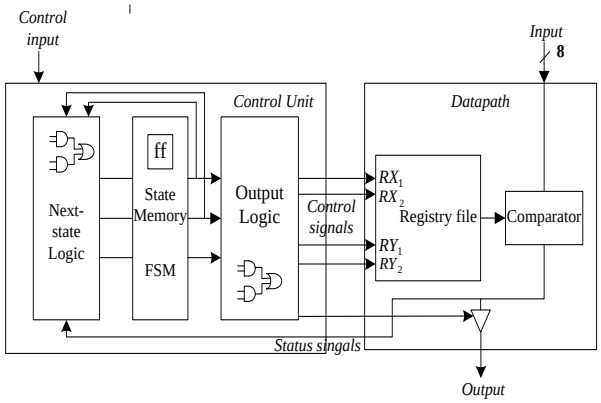

Fig 3. Schematic of microprocessor (adopted from [7])

The datapath part is composed of a registry file and one or more 8-bit comparators. The pattern to be recognized is stored in the L x 8 registry file as a chain of characters represented in binary values, where L is given by $L \geq N+1$ and N is the length of the pattern. The registry file gives flexibility to the hardware design to load and reload new values and modifications of the contents. With every clock cycle i.e. at the rising edge of the clock cycle a new character from the incoming stream is loaded at the *Input* of the datapath. The process is accomplished in the comparator by comparing the synchronically loaded values from the *Datapath_Input* and the character stored in the first address location of the registry file. The outcome result of the comparison is sent as a *status signal* to the control unit. If match is registered, the status signals named *Next* and *Reset* sends information to the control unit if match was found or it sends partial match or mismatch, as we will see later. If a full match of the content is recognized the *Output* of the datapath is set to a binary value '1' indicating that the pattern was identified.

In general, the control units inside microprocessors are finite state machines (FSM). By moving in sequences of states, control units sends signal named *control signal* to the datapath thus controlling the operations performed by the datapath. With every state in the control unit there is a control signal associated. Since the control unit is a FSM , its states could be described with a simple state diagrams, where every state in FSM represents the node of the deterministic graph. The input signals to the FSM are status signals from the datapath and the output signals are equivalent to the control signals distributed to the datapath. Figure 4. shows the concept of FSM state diagram of the control unit of proposed microprocessor.

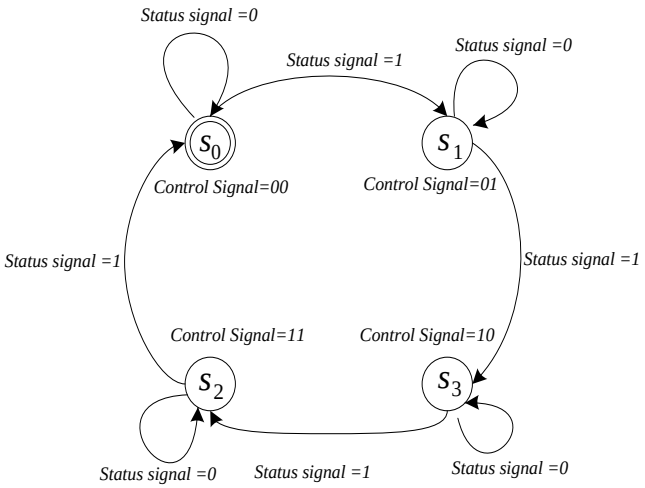

Fig 4. State diagram of simple control unit

As it is presented in Figure 4. the control unit has four states labeled as: s_0, s_1, s_2, s_3, one-bit *status signal* and two bits *control signal* associated with an appropriate state. It should be noticed that the value of the control signal is increased by 1 for every next state. It is normal behavior since the control unit should trigger loading successive characters stored in the registry file. Thus, the simplest control unit with the specified function could be designed as binary up counter where the count up signal is represented by the status signal and the control signals are equivalent to the address line of character in the registry file that will be compared.

3.1 Dedicated microprocessor for exact content matching

Content matching consists in finding one, or more generally, all the occurrences of a pattern in an input stream. Let the pattern be denoted by *x*=x[0 .. *n*-1] where its length is equal to *n*. The stream is denoted by *y*=y[0 .. *m*-1] and its length is equal to *m*. Both strings are built over a finite set of characters called an *alphabet* denoted by Σ. The simplest algorithm is a single pattern algorithm known as a naive or brute-force algorithm where the comparison is performed character-by-character at each position of the input stream. The exact matching dedicated processor

uses the above method. Character $x[i]$ and the character $y[i]$, where $i = 0...n$ are successively compared until mismatch is found. If mismatch occurs the process restarts from the first position i.e. *i=0*. The main issue of brute-force exact matchig appears if a mistmatch occures when the input stream character is identical with the first character of the pattern i.e. $x[i] \neq y[i]$ but $x[0] = y[i]$. In such a case the search process restarts from the second position i.e. *i=1*.

The process of exact content matching could be described with the following algorithm:

Repeat wtih every clock cycle:

1: i:=0; *Load x[i], Load y[i]*
2: *Next* = $\overline{x[i] \oplus y[i]}$, *Restart* = $\overline{x[0] \oplus y[i]}$
3: if *Next=1* **then** *i:=i+1;*
elsif *Restart=1* **then** *i:=1;*
else *i:=0;*
4: if *x[i]=end* **then** *Output:=1; i:=0*

where x[i] is the (*i+1*)-th character of the pattern stored at *i*-th address line of the registry file and its content is presented on *ReadA* port of the registry file whenever the *i* is equal to its address line. The input character from the stream is presented with *y*. The result of Comparator A is named as *Next* and the result of the Comparator B is named as *Restart* or *Jump* in case of approximate string matching as it will be seen later. In fact these are the *status signals* from datapath to the control unit. It is worth noting that ReadB port always presents the first character of the pattern x[0]. According to the above explaination we can design the logical circuit of the dedicated microprocessor. It is shown in Figure 5.

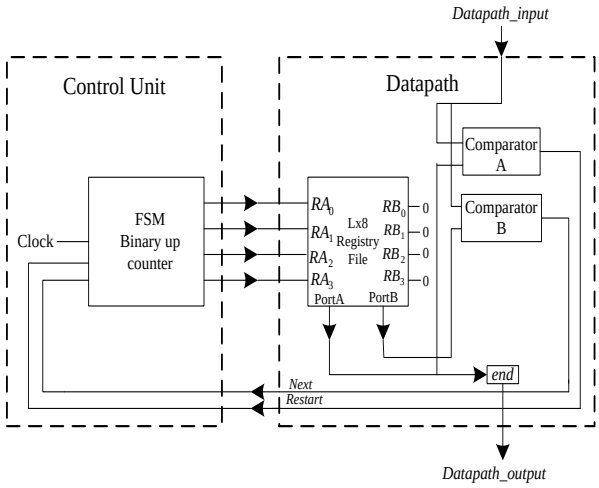

Fig 5. Dedicated microprocessor for exact content matching

.

3.2 Dedicated microprocessor for approximate content matching

Some of the sofisticated network attacks are using the so called "evasion" techniquies to avoid firewalls and NIDP systems. Deleting or replacing an existing character in the signature of the attack or inserting a new one may cause a baypass of the defending system. Enhancement of the software system to detect approximate variants of signature in the attack is possible but still exhausting process. Fisible solution in hardware was presented in [6] , but with limited flexibilty though. In this chapter a fully flexible hardware solution is presented able to detect approximate content matching.

The problem of approximate string matching can be described as finding a pattern $y = y_1 y_2 ... y_m$ of length *m* that with pattern $x = x_1 x_2 ... x_n$ of lenght *n* differs by at most *k* characters or the distance between *x* and *y* , D(x,y) is equal or less then *k* characters. In context of approximate string matching also known as k-differentiates problem, there are three basic operators: *deletion,insertion* and *substitution*. Since typical registry file has one write and two read ports (PortA and PortB) the processor design performs comparation of the two neighboring characters *x*[*i*] and *x*[*i+1*] agaianst the character *y*[*i*] from the input stream as it is described with the algorithm :

Repeat wtih every clock cycle:
1: *i=0; j=i+1, k=0, Load x, Load y*
2: *Next=* $\overline{x[i] \oplus y[i]}$*; Jump=* $\overline{x[j] \oplus y[i]}$
3: **If** *Next* =1 **then** *i=i+1, j=i+1;k=0;*
else
if *Jump=1* **then** *i=i+2; j=i+1;k=1;*
elsif *(Jump =0 and k=0)* **then** *i=i; j=i+1;k=1;*
elsif *(Jump=0 and k=1)* **then** *i=0;k=0*
4: **if** *x[i]=end* **then** *Output:=1; i:=0*

Similar with the exact matching the *x*[*i*] character, where *i* is the address of the stored character in the registry file is presented on PortA. Now on PortB we have the successive character stored in the memory i.e. the character located at j=i+1 address line. *Ne*xt and *Jump* are the status signals coming from comparator A and comprator B respectively. If a regular match was detected, the *Ne*xt signal is set to 1, thus increasing the address lines *i* and *j*, otherwise the *Jump* signal is set to '1' indicating that "jump" for one character will be considered or delete operation occurred. The whole pattern that will be matched will differ in one position i.e. k=1. For simplicity, the designed microprocessor will detect only for $k \le 1$. Insertion operation is performed by repeating the address line of the character where the mismatch occurs and in the next cycle the comparison is accomplished over the next coming character from the input stream. In similar manner, the substitution function is executed by combination of repetition so-called deletion function. The logical circuit for approximate processor is similar to circuit shown in Figure 5.

The two new components are the FSM for k and the adder at control unit as presented in Figure 6. In fact, the k FSM has three states: s_0, s_{0_1}, s_2. The state of s_{0_1} is intermediate state when only

repetition is executed. It is important to note the omission of the *Restart* signal for simplicity reasons. In reality our microprocessor design will contain third comparator where the current incoming character from the stream will be compared with the very first character from the pattern as it was described with the processor for exact matching in the previous chapter. The full functional design will include this component as well.

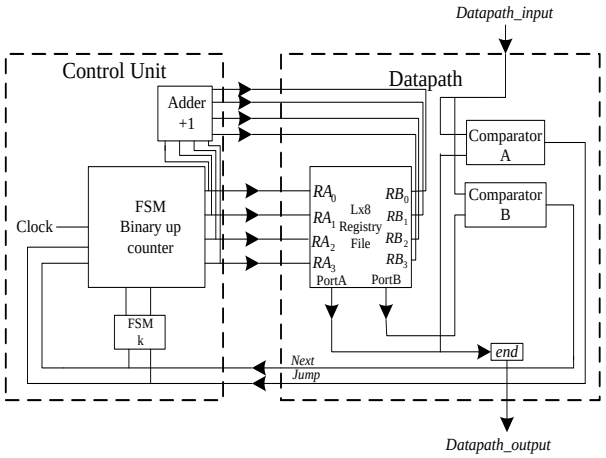

Fig 6. Dedicated micprocessor for approximate content matching with k=1

4. The flexibility of the design

Hardware solutions of NIDIPS are embedded on circular board or implemented on FPGA's. While the first are non-flexible designs the second offer reconfiguration by downloading the bit stream. The total time required by FPGA to execute the desired task of the detection , according [8] is:

$$T = T_M + T_R + T_E$$

where T_M represents the time required to map the design into FPGA, T_R is the reconfiguration time if some changes in the patterns occur and the matching process time is represent by T_E. Whenever a change in the content of the pattern is on demand or the number of the patterns increases a full or partial board reconfiguration will take T_E time interval of interruption.

The dedicated processor presented here is aimed as a module for content matching of an individual pattern. By setting *write enable* input of the registry file a pattern can be loaded, changed or delete obsolete patterns without affecting the matching process of the others patterns. Despite the FPGA reconfiguration, reloading of the new pattern in the registry file will be done in real time without interruption, except for that particular pattern. The number of empty and ready to load matching units is predicted by the update frequency of the order of days.

5. Simulation and results

Dedicated microprocessor designs observed in this paper are able to process one inputcharacter at clock cycle. In order to present the accurance of the designs, the Xilinx ISE WebPack software was used. For experimental enviorement the registry file is of dimesion 16×8 thus the address space is 2^n , n=4. The pattern to be recognized is "*abcd*" and it is stored on address lines "0000", "0001","0010" and "0011" respectively. Address line "0100" stores a NULL value or customly chosen to indicate the end sequenece, in our case it is “11111111” Every time when content match will be detected, the Datapath_output value will be set to logical '1' for one period of time of the clock cycle. The performances of the exact matchig are trivial. In case of approximate content matching, the processor should detect the exact value of the pattern i.e. "*abcd*" and all the variations achieved by insertion, deletion or substitution of a character. For example the sequences "acd" and "abd" are derivates of “*abcd*” with character deletion. In the same manner, the sequences “axcd”, “abxd”, “aybcd”, “abycd” etc. are modifications when supstitution or insertion operation is performed. Simulation results of approximate matcing are shown in Figure 7 .

Although the microprocessor is configured to detect the exact pattern i.e. “*abcd*” it will also recognize its k-differentiate variants. Figure 7 show detection of “*axcd*” meaning that the character “*b*” was replaced.

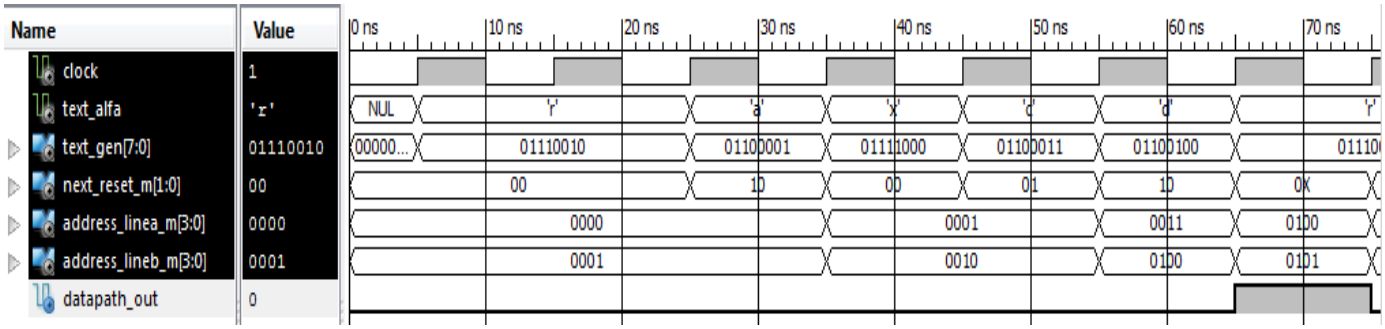

Fig .7 Simulation results for approximate content matching

6. Conclusion

Hardware components for string matching are an actual topic in the recent years. Many of the authors of the research papers use NFA/DFA or CAM's for their designs. NFA/DFA is efficient designs with low logical circuit area used, but unable to load new values of the pattern set.

If once is hardwired it needs to redesign the FPGA circular board. On the other hand, content matching hardware designs with CAM's are very flexible and most similar to the microprocessor designs presented here. Since the comparison is performed over the all memory elements, CAM's designs are poor in term of power dissipation.

As the author of this paper is aware, there is no similar approach as the one presented here. The proposed design , compared to the others hardware designs is not as efficient in term of the number of the logical circuits used or the speed performances, but it is far more configurable and flexible to load and reload new patterns. Although it is functional design, there is possibility for future enhancement of its features, for example an approximate matching with $k>1$ or including the *transposition* as an operation.

References

[1] SNORT official web site: (http://www.snort.org)

[2] Roesch, M.: Snort - lightweight intrusion detection for networks. *In: Proceedings of LISA'99: 13th Administration Conference. (1999) Seattle Washington, USA.*

[3] Cho, Young H., and William H. Mangione-Smith. "Deep network packet filters design for reconfigurable devices." *ACM Transactions on Embedded Computing Systems (TECS)* 7.2 (2008): 21.

[4] Sourdis, Ioannis, and Dionisios Pnevmatikatos. "Fast, large-scale string match for a 10Gbps FPGA-based network intrusion detection system." *Field Programmable Logic and Application*. Springer Berlin Heidelberg, 2003. 880-889.

[5] Clark, Christopher R., and David E. Schimmel. "Efficient reconfigurable logic circuits for matching complex network intrusion detection patterns." *Field Programmable Logic and Application*. Springer Berlin Heidelberg, 2003. 956-959.

[6] Georgiev, Dejan, and Aristotel Tentov. "FSM Circuits Design for Approximate String Matching in Hardware Based Network Intrusion Detection Systems."*International Journal of Information Technology & Computer Science* 6.1 (2013).

[7] Hwang, Enoch O. "Digital Logic and Microprocessor Design." *La Sierra University, Riverside* (2005).

[8] Christopher R. Clark "Design of Efficient FPGA Circuits for Matching Complex Patterns in Network Intrusion Detection Systems", *Georgia Institute of Technologies , December 2003*