IOSR Journal of Engineering (IOSR JEN) **www.iosrjen.org**
ISSN (e): 2250-3021, ISSN (p): 2278-8719
PP 36-40

A High Performance Scalable Pre-Processor for Network Security Systems

Savitha.K[1], Ms. S. Lakshmi, M.E., (Ph.D.) (Hod)[2]
Pg Student Thirumalai Engineering College
Supervisor & Asso.Prof. Thirumalai Engineering College

Abstract: *Network Intrusion Detection Systems (NIDS) and Anti-Denial-of-Service (DoS) employ Deep Packet Inspection (DPI) which provides visibility to the content of payload to detect network attacks. All DPI engines assume a pre-processing step that extracts the various protocol-specific fields. However, application layer (L7) field extraction is computationally expensive. We propose a novel Deep Packet Field Extraction Engine (DPFEE) for application layer field extraction to hardware. DPFEE is a content-aware, grammar-based, Layer 7 programmable field extraction engine for text-based protocols. Our prototype DPFEE implementation for the Session Initiation Protocol (SIP) and HTTP protocol on a single FPGA, achieves a bandwidth of 408.5 Gbps and this can be scaled beyond 500 Gbps. Single DPFEE exhibits a speedup of 24X-89X against widely used preprocessors. Even against 12 multi-instances of a preprocessor, single DPFEE demonstrated a speedup of 4.7-7.4X. Single DPFEE achieved 3.14X higher bandwidth, 1020X lower latency and 106X lower power consumption, when compared with 200 parallel streams of GPU accelerated preprocessor.*

I. Introduction

The proliferation of Internet and networking applications, coupled with the wide-spread availability of system hacks and viruses have increased the need for network security. Firewalls have been used extensively to prevent access to systems from all but a few, well defined access points (ports), but they cannot eliminate all security threats, nor can they detect attacks when they happen. Stateful inspection firewalls are able to understand details of the protocol that are inspecting by tracking the state of a connection. They actually establish and monitor connections for when it is terminated. However, current network security needs, require a much more efficient analysis and understanding of the application data. Content-based security threats and problems occur more frequently, in an everyday basis. Virus and worm inflections, SPAMs (unsolicited e-mails), email spoofing, and dangerous or undesirable data, get more and more annoying and cause innumerable problems. Therefore, next generation firewalls should provide deep packet Inspection capabilities, in order to provide protection from these attacks. Such systems check packet header, rely on pattern matching techniques to analyze packet payload, and make decisions on the significance of the packet body, based on the content of the payload.

MOTIVATION

Network Intrusion Detection Systems (NIDS) perform deep packet inspection. They scan packet's payload looking for patterns that would indicate security threats. Matching every incoming byte, though, against thousands of pattern characters at wire rates is a complicated task. Measurements on SNORT show that 31% of total processing is due to string matching; the percentage goes up to 80% in the case of Web-intensive traffic. So, string matching can be considered as one of the most computationally intensive parts of a NIDS and in this thesis we focus on payload matching. Many different algorithms or combination of algorithms have been introduced and implemented in general purpose processors (GPP) for fast string matching, using mostly SNORT open source NIDS rule-set. However, intrusion detection systems running in GPP can only serve up to a few hundred Mbps throughput. Therefore, seeking for hardware-based solutions is possibly the only way to increase performance for speeds higher than a few hundred Mbps. Until now several ASIC commercial products have been developed. These systems can support high throughput, but constitute a relatively expensive solution. On the other hand, FPGA-based systems provide higher flexibility and high throughput comparable to ASICs performance. FPGA-based platforms can exploit the fact that the NIDS rules change relatively infrequently, and use reconfiguration to reduce implementation cost. In addition, they can exploit parallelism in order to achieve satisfactory processing throughput. Several architectures have been proposed for FPGA-based NIDS, using regular expressions (NFAs/DFAs), CAM, discrete comparators, and approximate filtering techniques. Generally, the performance results of FPGA systems are promising, showing that FPGAs can be used to support the increasing needs for network security. FPGAs are flexible, reconfigurable, provide hardware speed, and therefore, are suitable for implementing such systems. On the other hand, there are several issues that should be faced. Large designs are complex and therefore hard to operate at high frequency. Additionally, matching a large number of patterns has high area cost, so sharing logic is critical, since it could save a significant amount of resources, and make designs smaller and faster.

SCOPE OF THIS THESIS

Since string matching is the most computationally intensive part of an NIDS, our proposed architectures exploit the benefits of FPGAs to design efficient string matching systems. The proposed architectures can support between 3 to 10 Gbps throughput, storing an entire NIDS set of patterns in a single device. In this thesis, we suggest solutions to maintain high performance and minimize area cost, show also how pattern matching designs can be updated and partially or entirely changed, and advocate that brute-force solutions can offer high performance, while require low area. Techniques such as fine-grain pipelining, parallelism, partitioning, and pre-decoding are described, analyzing how they affect performance and resource consumption.

This thesis proposes a pattern matching algorithm that reduces total memory requirements by sharing common infixes of target patterns. For the pattern identification, a state should contain its own match vector with a set of bits, where each bit represents a matched pattern in the state. Even though the information of shared common infixes was stored in match vectors, the number of shared common infixes was limited by the size of the match vectors. In addition, throughput could decrease due to the modified state transition mechanism. The memory requirements for match vectors were reduced by relabeling states and eliminating the match vectors of non-output states. By sharing common infixes of target patterns or relabeling states and eliminating the match vectors of non-output states, the memory usage in the match vectors could be efficient.

However, the variety of target pattern lengths is another serious problem in achieving regularity and scalability with low hardware cost. Each pattern consists of multiple character codes, where the number of character codes is defined as the pattern length. According to the rule sets, the distribution of pattern lengths could be different from each other. In addition, the variation of pattern lengths in each rule set is irregular. If target patterns are to be mapped onto multiple homogeneous string matchers, memory usage cannot be balanced without considering different pattern lengths.

In order to reduce the memory requirements of the DFA-based string matching engine, this proposes a memory-efficient parallel string matching scheme using the pattern dividing approach and its hardware architecture for the pattern identification. Long target patterns are divided into sub-patterns with a fixed length; therefore, the variety of target pattern lengths can be mitigated. By balancing memory usage between the string matchers, unused memory area in homogeneous string matchers decreases. Moreover, the number of shared common states increases due to both the reduced length and the increasing number of sub-patterns, compared with the cases of the string matching with long target patterns. For each string matching, DFAs are built with bit-level input symbols for the bit splitting in order to reduce the number of state transitions from each state. For identifying the original long target patterns, the successive matches with sub-patterns are detected using the proposed two-stage sequential string matching engine. Experimental results show that memory requirements decrease on average by 47.8 percent and 62.8 percent for selected rules Snort and ClamAV, compared with several existing bit-split string matching approaches.

WORKING ON THE PROJECT

In order to reduce the memory requirements of the DFA-based string matching engine, this proposes a memory-efficient parallel string matching scheme using the pattern dividing approach and its hardware architecture for the pattern identification. Long target patterns are divided into sub-patterns with a fixed length; therefore, the variety of target pattern lengths can be mitigated.

Most of the previous researches on hardware **regex** detection methods are based on ***pcre*** patterns taken from Snort [11]. A few major differences between the ClamAV format and the ***pcre*** format are listed below

For identifying the original long target patterns, the successive matches with sub-patterns are detected using the proposed two-stage sequential string matching engine. Experimental results show that memory requirements decrease on average by 47.8 percent and 62.8 percent for selected rules Snort and ClamAV, compared with several existing bit-split string matching approaches.

PROPOSED WORK:

In this work we proposed hardware efficient VLSI architectures to detect the complex NIDS patterns based on the information reduction approach. Here we preprocessed input bytes to transform the byte-oriented matching problem to a token-based matching problem. The input byte stream is converted into a token-stream using dedicated hardware units which can perform parallel computations for high throughput rate. A NIDS pattern contains one or more segments will be subdivided into multiple non-trivial tokens. Finally the token-stream is processed by a NFA-based aggregation unit to determine the virus to be found.

Here FEMEs are finite state machines (FSM) designed to extract field values within a header data of the application layer as shown in Figure 3.1. FEMEs search the header data sequentially. The extraction time is a function of the length of the header and number of bytes searched in a clock cycle

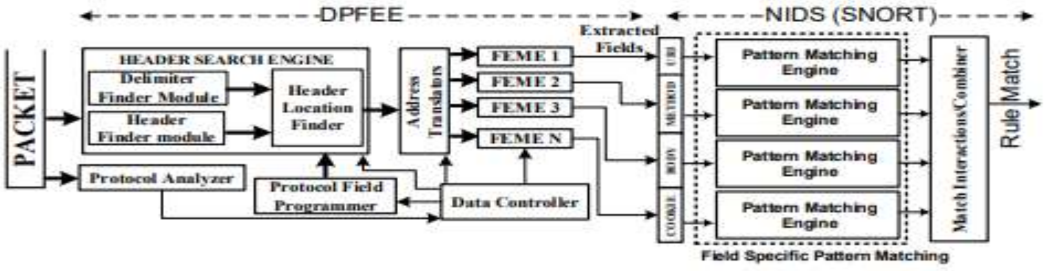
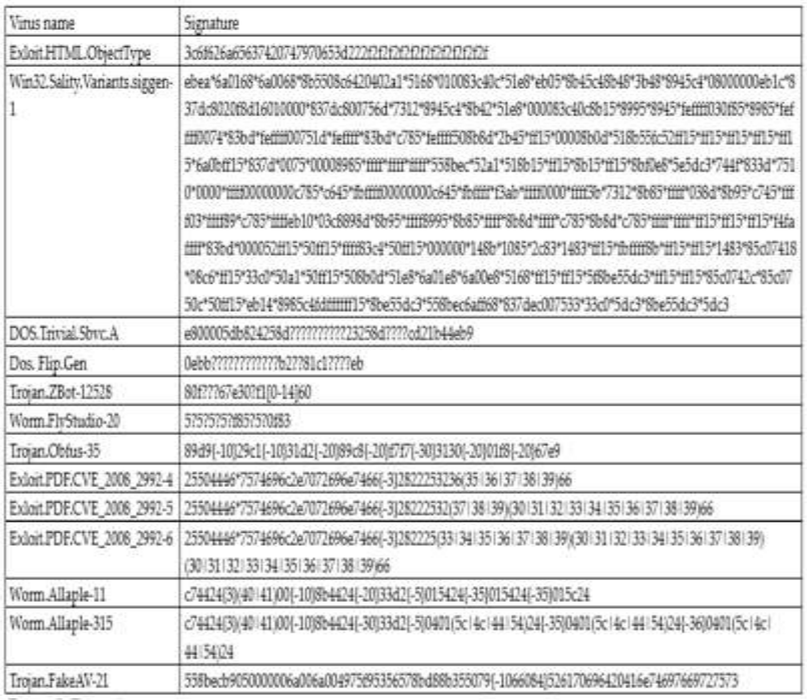


Figure 3.1. DPFEE with a hardware based NIDS

Divided Patterns

A target pattern and a set of its k sub-patterns, which are obtained after dividing the target pattern, are denoted as P_i and Q_i ¼ fSP_{i1}; SP_{i2}; . . . ; SP_{ik}g, respectively. The subscript i is the index of the target pattern. A set of k sub-patterns Q_i will be called the quotient vector of P_i. The fixed length of k sub-patterns is denoted as f. If the length of a target pattern is shorter than f, the target pattern does not need to be divided, so the pattern is defined as the short pattern. The remnant pattern R_i represents a suffix or residual sub-pattern of the target pattern P_i that succeeds the quotient vector of P_i. The match in a sub-pattern is encoded with a quotient index, which represents the unique index of the sub-pattern match. The remnant pattern R_i should be matched sequentially after the quotient vector Q_i is matched.



Virus name	Signature
Exloit.HTML.ObjectType	3c6f626a656374207479706533d222f2f2f2f2f2f2f2f2f2f2f2f2f
Win32.Sality.Variants.siggen-1	ebea*6a0168*6a0068*8b5508c6420402a1*5168*010083c40c*51e8*eb05*8b45c48b48*3b48*8945c4*08000000eb1c*837dc8020f8d16010000*837dc800756d*7312*8945c4*8b42*51e8*000083c40c8b15*8995*8945*fefffff030f85*8965*fefffff0074*83bd*fefffff00751d*fefffff*83bd*c785*fefffff508b6d*2b45*ff15*00008b0d*518b556c52ff15*ff15*ff15*ff15*ff15*6a0bff15*837d*0075*00008965*ffff*ffff*ffff*558bec*52a1*518b15*ff15*8b15*ff15*8bf0e8*5e5dc3*744f*833d*7510*0000*ffff00000000c785*c645*fbffff00000000c645*fbffff*f3ab*ffff0000*ffff3b*7312*8b65*ffff*038d*8b95*c745*ffff03*ffff89*c785*ffffeb10*03c8898d*8b95*ffff8995*8b85*ffff*8b8d*ffff*c785*8b8d*c785*ffff*ffff*ff15*ff15*ff15*f4faffff*83bd*000052ff15*50ff15*ffff83c4*50ff15*000000*148b*1085*2c83*1483*ff15*fbffff8b*ff15*ff15*1483*85c07418*08c6*ff15*33c0*50a1*50ff15*508b0d*51e8*6a01e8*6a00e8*5168*ff15*ff15*5f8be55dc3*ff15*ff15*85c0742c*85c0750c*50ff15*eb14*8965c4ddffffff15*8be55dc3*558bec6aff68*837dec007533*33c0*5dc3*8be55dc3*5dc3
DOS.Trivial.Sbvc.A	e800005db824258d??????????23258d????cd21b44eb9
Dos. Flip.Gen	0ebb????????????b2??81c1????eb
Trojan.ZBot-12528	80f??67e30?f1[0-14]60
Worm.FlyStudio-20	57?57?57?f857570f83
Trojan.Obfus-35	89d9[-10]29c1[-10]31d2[-20]89c8[-20]f7f7[-30]3130[-20]01f8[-20]67e9
Exloit.PDF.CVE_2008_2992-4	25504446*7574696c2e7072696e7466[-3]2822253236(35|36|37|38|39)66
Exloit.PDF.CVE_2008_2992-5	25504446*7574696c2e7072696e7466[-3]28222532(37|38|39)(30|31|32|33|34|35|36|37|38|39)66
Exloit.PDF.CVE_2008_2992-6	25504446*7574696c2e7072696e7466[-3]282225(33|34|35|36|37|38|39)(30|31|32|33|34|35|36|37|38|39)(30|31|32|33|34|35|36|37|38|39)66
Worm.Allaple-11	c74424(3)(40|41)00[-10]8b4424[-20]33d2[-5]015424[-35]015424[-35]015c24
Worm.Allaple-315	c74424(3)(40|41)00[-10]8b4424[-30]33d2[-5]0401(5c|4c|44|54)24[-35]0401(5c|4c|44|54)24[-36]0401(5c|4c|44|54)24
Trojan.FakeAV-21	558becb9050000006a006a004975f953565788d88b355079[-1066084]5261706964204166e74697669727573

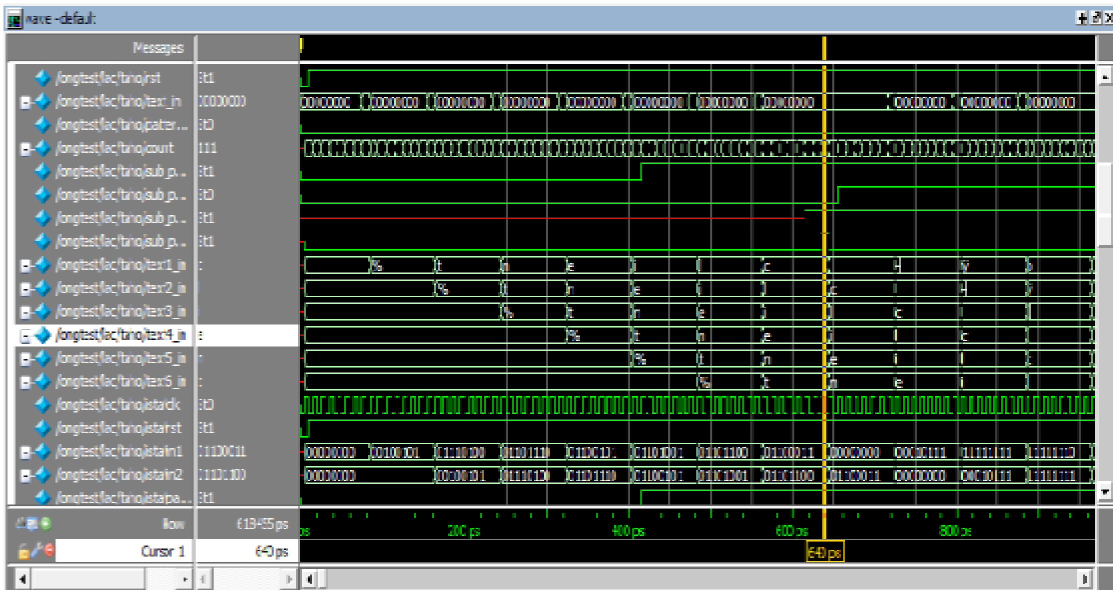
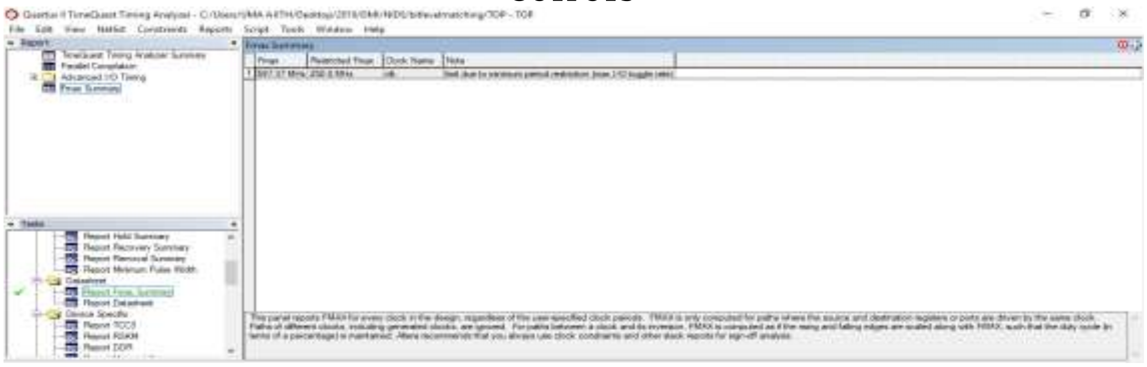


Figure 3.2. Example virus signatures.

II. Results And Outputs

Here showing some of the outputs taken using Model Sim software and Quartus II software. Further, these results are used to compare the proposed bit based algorithm with Aho-Corasick algorithm, Deterministic finite automaton and bit level pattern matching algorithm.

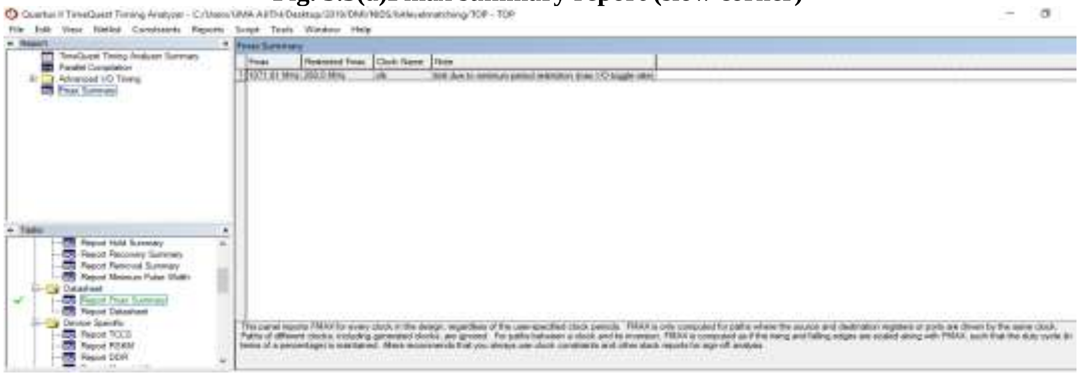
OUTPUTS



Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	597.37 MHz	250.0 MHz	clk	limit due to minimum period restriction (max I/O toggle rate)

Fig. 5.3(a)Fmax summary report (slow corner)



Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	1071.81 MHz	250.0 MHz	clk	limit due to minimum period restriction (max I/O toggle rate)

Fig.5.3(b)Fmax summary report (fast corner)

MODEL SIM

High Performance and Capacity Mixed HDL Simulation – Model Sim

Mentor Graphics was the first to combine single kernel simulator (SKS) technology with a unified debug environment for Verilog, VHDL, and SystemC. The combination of industry-leading, native SKS performance with the best integrated debug and analysis environment make ModelSim the simulator of choice for both ASIC and FPGA design. The best standards and platform support in the industry make it easy to adopt in the majority of process and tool flows.

4.6.1 ModelSim-Altera Edition

- Recommended for simulating all FPGA designs (Cyclone®, Arria®, and Stratix® series FPGA designs)
- 33 percent faster simulation performance than ModelSim®-Altera® Starter Edition.
- No line limitations
- Buy today for $945.

ModelSim-Altera Starter Edition:

Each SNORT rule can contain header and content fields. The header part checks the protocol, and source and destination IP address and port. The content part scans packets payload for one or more patterns. The matching pattern may be in ASCII, HEX or mixed format. HEX parts are between vertical bar symbols ” *j*”. An example of a SNORT rule is: alert tcp any any ->192.168.1.0/32 111(content: "idc*j*3a3b*j*"; msg: "mountd access";)
The above rule looks for a TCP packet, with any source IP and port, destination IP = 192.168.1.0, and port=111. To match this rule, packet payload must contain pattern ”idc *j*3a3b*j*”, which is ASCII characters ”i”, ”d”, and ”c” and also bytes ”3a”, and ”3b” in HEX format

- Boyer-Moore
- Knuth-Morris-Pratt (KMP)

  State Machine Matching
- Aho/Corasick

2. Hardware Solutions

- Bloom Filters and Extended Bloom Filters
- NFA/DFA implementation at hardware level

III. Conclusion

Here we verified the functionality proposed novel Deep Packet Field Extraction Engine-based parallel string matching scheme with minimized memory requirements for NIDS virus database. The problem of various pattern lengths with wild cards can be mitigated by dividing long target patterns into tokens with a fixed length. The memory-efficient architectures were proposed for both string matching and complex NIDS pattern matching which can reduce the total memory requirements. Considering the reduced memory requirements for the real rule sets, it is concluded that the proposed matching scheme is useful for reducing total memory requirements of parallel string matching engines.