Hardware Architecture of a Parallel Pattern Matching Engine

Meeta Yadav
Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27606
Email: myadav@ncsu.edu

Ashwini Venkatachaliah
Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27606
Email: akvenkat@ncsu.edu

Paul D. Franzon
Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27606
Email: paulf@ncsu.edu

***Abstract*— Several network security and QoS applications require detecting multiple string matches in the packet payload by comparing it against predefined pattern set. This process of pattern matching at line speeds is a memory and computation intensive task. Hence, it requires dedicated hardware algorithms. In this paper we describe the hardware architecture of a parallel, pipelined pattern matching engine that uses trie based pattern matching algorithmic approach. The algorithm optimizes pattern matching process through two key innovations of parallel pattern matching using incoming content filter and multiple character matching using trie pruning. The hardware implementation is capable of performing at line-speeds and handle traffic rates upto OC-192, the underlying architecture allows for multiple patterns to be detected and for the system to gracefully recover from a failed partial match, the throughput of the system does not degrade with the increase in the number of patterns or the length of the patterns to be matched. The solution described outperforms most current implementations in terms of speed and memory requirement and outperforms TCAM based solutions in terms of power consumption, area, and cost while remaining competitive in terms of throughput and update times. We use the complete Snort rule set (2005 release) and VoIP RFC to validate our performance and achieve a throughput of 12Gbps with 6KBytes of content filter memory and 0.3 MBytes of total memory for Snort and 0.5KBytes of filter memory and 12KBytes of total memory for SIP.**

I. Introduction

Deep packet inspection can be broadly defined as the process of inspecting the header as well as the content of the packet. Several networking applications like Network Intrusion Detection Systems (NIDS), Network Intrusion Prevention Systems (NIPS) and Packet Classifiers (PC) perform deep packet inspection to analyze the packet payload by searching for predefined content. The performance of these application is is critically affected by pattern matching algorithms and their implementations. Software based pattern matching implementations cannot achieve line speeds. Hence there is an emphasis on hardware based NIDS and NIPS that can operate in multi Gbps range, are configurable and scalable, and have a stringent upper bound on their worst case performance.

In this paper we present a multi-way trie based algorithmic approach to pattern matching. The trie based technique was developed for IP forwarding by Mehrotra et al. and has been described in [9]. We present two key enhancements to this approach that adapt the technique to pattern matching and significantly improve the throughput of the system. The two enhancements are *Incoming Content Filter* and *Trie Pruning*. The incoming content filter splits the packet payload into multiple subs-streams that could lead to a possible match, these multiple streams are then passed to dedicated pattern matching engines that match the sub-streams in parallel. The ability of the algorithm to match multiple streams in parallel significantly improves the throughput of the system and enables the algorithm to recover gracefully from a failed partial search. Trie pruning aids in reducing the memory required for storing the patterns and allows for multiple characters to be matched in one clock cycle, thus removing the dependence of the throughput of the system on the number of patterns to be matched and the length of patterns to be matched.

The rest of the paper is organized into the following four sections. In Section II we present some of the related work in the field of hardware based pattern matching system. In Section III we describe trie based pattern matching algorithm. The hardware architecture is presented in section IV and we conclude with the results in section V.

II. Related Work

Hardware based pattern matching algorithms have been a point of interest for researches for some time, several innovative algorithms have been proposed so far that can process multi gigabits of data per second. However, most implementations have not handled the issue of scalability and the worst case performance issues which occur due to partial match failures.

Several systems use off the shelf TCAMs to perform pattern matching, TCAMS based solutions are easy to implement, have high throughput and are scalable. In [13], the authors present a TCAM based approach that performs at 2 Gbps but the performance of the system degrades for large patterns. TCAM based solutions also suffer from high cost and high power consumption.

Pattern matching algorithms have also been designed for specific FPGA implementations. In [12], [3], [4], [5] the authors present algorithms that exploit the embedded technology in the FPGAs to speed up the task of pattern matching. These designs achieve high throughput and are reconfigurable but lack portability since they are designed for specific FPGAs.

The state based pattern matching architectures are also extremely popular owing to their high throughput. The authors of [7] [11] propose deisgns that use Aho-Corasick algorithm and its variations. The systems proposed by however do not recover efficiently from partial match failure and perform poorly in detecting long strings and also suffer from scalability issues due to higher memory requirements.

Bloom Filters have been extensively used in many networking applications. The authors of [6] present a parallel bloom filter technique to pattern matching that performs extremely well for short string matches. The biggest drawback of this approach is performance degradation with increase in the size of incoming content and high false positive rate.

III. Pattern Matching Algorithm

Pattern matching problem is broadly defined as the task of finding multiple occurrences of signatures P[0], P[1],..P[m-1] in the incoming content *S* of length *i*. The incoming content should be matched against all possible signatures to detect a match. The signatures do not occur on predefined boundary and can occur anywhere from 0 to $i - 1^{th}$ characters of the incoming content.

A. Incoming Content Filter

The content filter breaks down the incoming stream S into multiple sub-streams S[0], S[1], S[2]..S[i-1] that can result in a possible match. The characters in the input stream S are combined into sets of two as shown in figure 1, and passed through the content filter. On detecting a match the stream S is split and all the characters preceding the matched character set are stored in a buffer. Each stream is then passed to dedicated parallel pattern matching units.

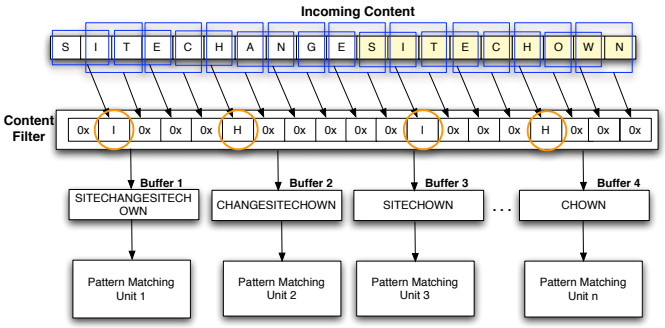

Fig. 1. Incoming Content Filter

B. Parallel Multi-Way Trie Algorithm

Tries are ordered tree data structure that use the "thumb indexing" method of dictionaries to store and retrieve information. Information is stored in tries as paths from root node to the leaf nodes and information is retrieved by traversing to the leaf nodes through the root node. Figure 2 shows a multi-way trie constructed using a few patterns from Snort's FTP rule set, the empty leaves of the trie are not shown. Trie

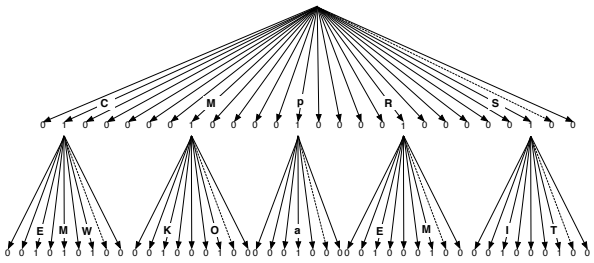

Fig. 2. Trie constructed using partial Snort FTP rule set content

TABLE I
SRAM Memory Structure

Sum of 1s	Bit Map	Shadow Bit
0	00001000100000000000...0	00000..0

based schemes perform search by looking at multiple bits at a time. This feature determines the degree of the trie. The number of leaves in a particular node are determined by the degree of the trie and the depth of the trie is given by the number of bits in the largest pattern.

$$No.\ Of\ Leaves\ Per\ Node = 2^{Degree\ Of\ The\ Trie} \quad (1)$$

All the leaves of the nodes in the trie are initially set to 0, presence of a character is marked by setting the corresponding indexed location to 1 and the absence of a character is represented by setting the corresponding indexed location to 0. Each 1 in the SRAM pattern gives rise to a child with 256 bits, where a 0 is not propagated while generating the bit pattern. The patterns *P* are stored in the on-chip memory in the trie form. Searching for a pattern requires partitioning address bits into sections of *m* bits and addressing different levels of *m-ary* trie. The Trie depth is given by equation 2, where X is the degree of the trie.

$$Trie\ Depth = \frac{No.\ Of\ Address\ Bits}{log_2 X} \quad (2)$$

C. Trie Pruning

The efficiency of the algorithm is further enhanced by pruning the tries and matching multiple characters in one cycle. The trie is pruned at a stage after which a particular node and its children stop branching out. Figure 3 shows sections of the trie that can be pruned and stored in memory for dedicated multi character match. The pruned sections of the tries are stored in off-chip mempry, rhe address of the pruned portion of the trie is calculated by adding the number of ones in all the previous levels with the *Sum of 1s* before the indexed value. The pruned trie sections are stored as characters in the off-chip memory and are matched against the incoming content using dedicated comparators.

We illustrate our pattern matching algorithm with an attempt to match the incoming content "SITECHANGESITECHOWN" against predefined patterns of snort FTP ruleset.

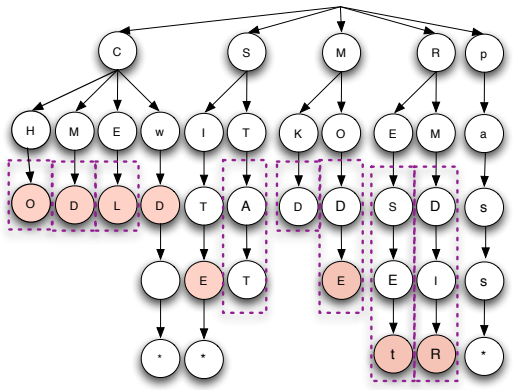

Fig. 3. Trie Pruning

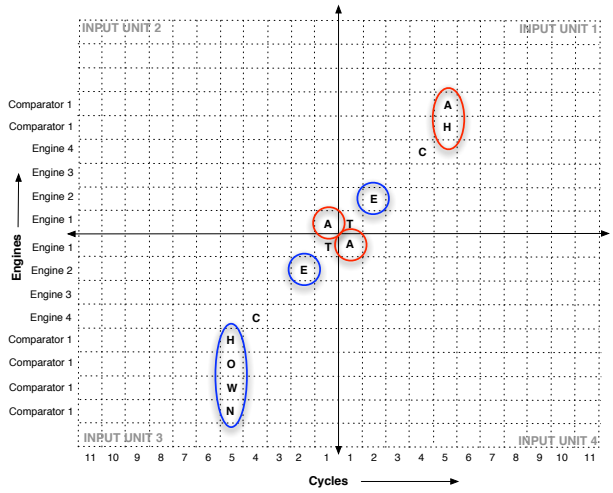

Fig. 4. Pattern Matching Graph using Filter and Trie Pruning

The signature SITECHOWN in the incoming content is embedded in other innocuous content. The content filter splits the incoming stream into four sub streams "SITECHANGE-SITECHOWN", "CHANGESITECHOWN", "SITECHOWN", "CHOWN". These sub streams are passed to separate pattern matching units as shown in figure 4, where red indicates failure and blue indicates success.

IV. Hardware Architecture

In this section we discuss the hardware design details and implementation details of the Pattern Matching Engine. The Pattern Matching Module has "n" stages. Each stage is associated with a separate input buffer.

A. Content Filter

The content filter consists of level 0 and level 1 of the tries. The incoming data is passed through the content filter and split into several sub-streams which are stored into separate buffers. We use two trie levels as content filters instead of three since the memory requirement for the filter goes up substantially in case of three levels, figure 5 illustrates that.

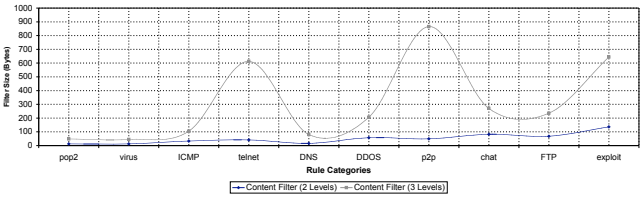

Fig. 5. **Content Filter Size**

B. Trie Lookup Engine

Figure 6 shows a dedicated pattern matching module. A pattern matching module has 3 input fields and 3 output fields; the three inputs are: *Input Character*: the input character to me matched, *Match Input*: signal from the previous trie lookup that indicates match of a character in the previous level, *Sum of 1s*: used to calculate the memory address of the node to be inspected.

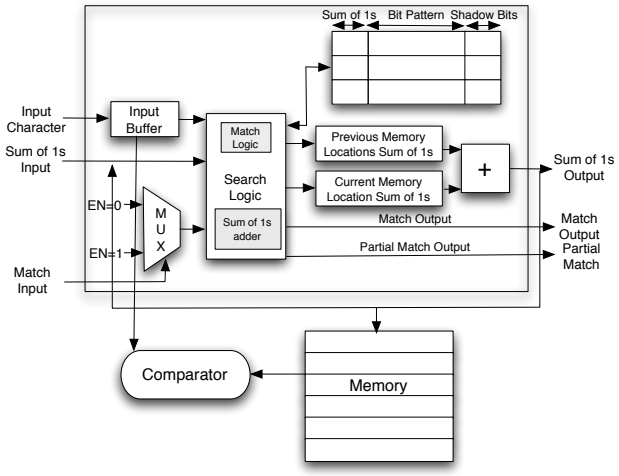

Fig. 6. **Sub Engine of the Pattern Matching Unit**

The *Match Input* enables or disables a particular pattern matching unit. The SRAM contains, Sum of Ones field, Trie Bitmap and Shadow Bit Vector. The *Sum of Ones* field is used to calculate the memory address of the next trie location. The *Bitmap* indicates presence or absence of a character and the *Shadow Bit Vector* indicates a partial match. The SRAM is indexed into using the *Sum of 1s* field and the corresponding trie bitmap is traversed using the character in the input buffer, if the corresponding bit in the bitmap register is set to 1 then the character is matched positively. The match output signal is then set to 1 and communicated to the input of the pattern matching unit to enable it to look at the next character. The sum of ones in the *Bit Map* before the indexed bit are calculated and added to the *Sum of 1s of Previous Nodes* and the *Sum of 1s Output Signal* is generated and communicated to the input of the pattern matching unit.

C. Comparators

The pattern matching trie lookup sub-engine matches a character at a time and, on hitting a pruned node calculates the memory address. The contents of the memory are read and compared against the unmatched content in the input buffer using dedicated comparators.

V. Results and Conclusions

In this paper we have presented a hardware architecture of a parallel pattern matching coprocessor using multiway tries which is capable of detecting multiple patterns at line-speeds. The design uses memory compaction and scales gracefully with the increase in the number of signatures without a significant increase in the memory. The throughput of the system does not degrade with the number and length of patterns to be matched. The graphs in figure 8,9 show the performance results, graph 8 shows the comparison between traditional state based approaches and trie based approach in terms of average number of cycles required for pattern matching. The technique using 'tries with pruning'" out performs "tries without pruning" technique and "state based" techniques. Graph 9 compares the memory required for tries with pruning and tries without pruning with shows significant increase in the memory required for tries without pruning with the increase in the number and length of signatures.

The pattern matching engine was implemented using the 0.25 micron library with an area of 2sq mm with 20 input buffers and 20 dedicated trie look up modules and comparators. We use the complete Snort rule set (2005 release) and VoIP RFC to validate our performance. We achieve a throughput of 12Gbps with 6KBytes of content filter memory and 0.3 MBytes of total memory for Snort and 0.5KBytes of filter memory and 12KBytes of total memory for SIP.

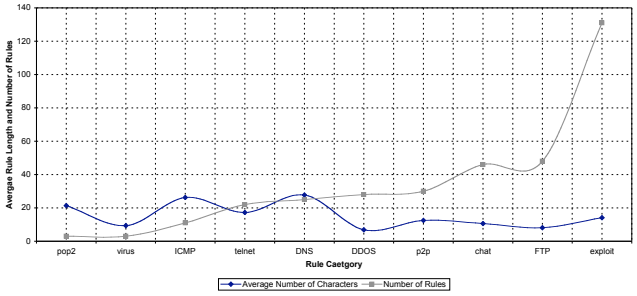

Fig. 7. Rule Summary for Snort Rules

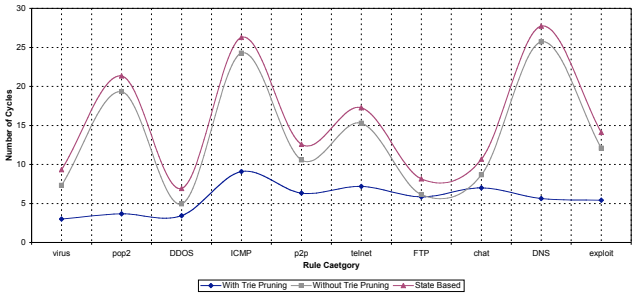

Fig. 8. Average Cycles for Pattern Matching

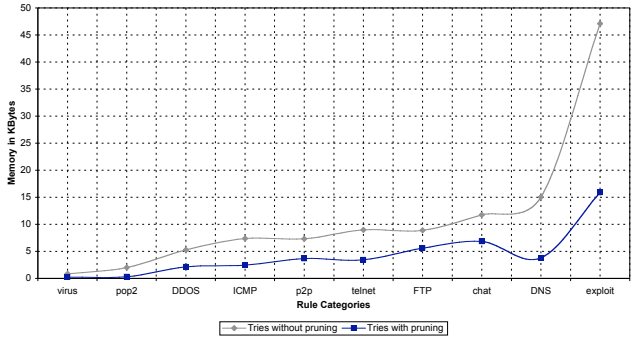

Fig. 9. Memory Usage

Acknowledgment

The authors would like to thank John Leon from Irvine Sensors Corporation.

References

[1] http://www.snort.org.
[2] A.V.Aho and M. J. Corasick. Efficient string matching: An aid to bibliographic search. volume 18(6), pages 333–340. Commun. ACM, 1975.
[3] Z. K. Baker and V. K. Prasanna. A computationally efficient engine for flexible intrusion detection. ACM/SIGDA, 2004.
[4] Z. K. Baker and V. K. Prasanna. Time and area efficient pattern matching on fpgas. pages 223–232. 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays, 2004.
[5] C.R.Clark and D.E.Schimmel. Efficient reconfigurable logic circuits for matching complex network intrusion detection systems. In 13th International Conference on Field-Programmable Logic and Applications, 2003.
[6] S. Dharmapurikar, P. Krishnamurthy, T.S.Sproull, and J.Lockwood. Deep packet inspection using parallel bloom filters. volume 24, pages 52–61. IEEE Micro, 2004.
[7] S. Dharmapurikar and J. Lockwood. Fast and scalable pattern matching for content filtering. volume 1-59593-082-5/05/0010. ANCS, 2005.
[8] P. Mehrotra. *Memory Intensive Architectures for DSP and Data Communication*. PhD thesis, North Carolina State University, 2002.
[9] P. Mehrotra and P. Franzon. Novel architecture for fast address lookups". pages pp. 66–71. IEEE Communications Magazine, 2002.
[10] RFC. http://www.ietf.org/rfc/rfc3261.txt.
[11] L. Tan and T. Sherwood. A high throughput string matching architecture for intrusion detection and prevention. volume 1063-6897/05. International Symposium on Computer Architecture, 2005.
[12] Y.Cho and W.Mangione-Smith. A pattern matching co-processor for network security. ACM, 2005.
[13] F. Yu, R.Katz, and T.V.Lakshman. Gigabit rate packet pattern matching using tcam. IEEE International Conference on Network Protocols, Oct. 2004.