Hierarchical NFA-Based Pattern Matching for Deep Packet Inspection

Yan Sun, Victor C. Valgenti, and Min Sik Kim
School of Electrical and Computer Engineering
Washington State University
Pullman, Washington, U.S.A.
Email: {ysun,vvalgent,msk}@eecs.wsu.edu

***Abstract*—Many security applications in today's networks are based on deep packet inspection, examining not only the headers but also the payloads of data packets. Traffic monitoring, layer-7 filtering, and network intrusion detection classify traffic by identifying predefined patterns within packet payloads that are specific to certain classes of attacks. Pattern matching is the primary task in deep packet inspection. The most common and efficient implementations of pattern matching are based on Non-deterministic Finite Automata (NFA) and Deterministic Finite Automata (DFA). In this paper, we propose an efficient NFA-based pattern matching in Binary Content Addressable Memory (BCAM) which uses multi-bit, binary search words. Our approach can process multiple characters at a time using limited BCAM entries, providing for greater parallelism and potential scalability through examining larger numbers of characters per cycle. Furthermore, we build a hierarchical pattern matching architecture which filters most of the packets from full evaluation using a small number of BCAMs and leaving only a minor percentage of packets to be checked in the full pattern matching process. Such filtering greatly improves throughput for expected traffic as demonstrated in our simulations. We evaluate our algorithm using patterns provided by Snort, a popular open-source intrusion detection system. The simulation results show that our approach outperforms existing TCAM-based and software-based approaches.**

I. Introduction

Many security applications in today's networks are based on deep packet inspection. These applications, such as traffic monitoring, layer-7, and network intrusion detection compare the headers and payloads of data packets to predefined databases describing potential attack traffic. These predefined databases are populated by numerous string patterns that, when found, imply that the packet under question is possibly malicious. The inspection is performed using exact matching algorithms. However, the number of patterns has continued to grow in order to describe more and more payloads. For example, Snort [1], an open-source Network Intrusion Detection System (NIDS), has seen its rule set double in size in the last six years from roughly 3,000 rules to more than 5,000. On a similar note, the signature database for the open source ClamAV anti-virus software had about 27,000 patterns in 2010 [2]. Further, the volume of network traffic is continuing to increase such that open-source NIDSs like Bro [22] and Snort [1] expend all the resources, both CPU time and memory, and halt immediately when they are deployed under high-speed network environments [23]. Thus, efficiency in matching, especially for large pattern sets, has become a major concern.

The most popular method to implement pattern matching is to use finite automata [3]–[6]. A finite automaton is built as a composite of all the patterns in the rule set and is run with the packet payload as input. The exact automaton used is dependent on implementation considerations and may either be Deterministic Finite Automata (DFA) or Non-deterministic Finite Automata (NFA). NFA offer the best efficiency in terms of memory required to store the automaton, however NFA can have multiple transitions per state and thus may need to maintain multiple states as the automaton is traversed resulting in less efficiency in throughput. This makes NFA more suitable for Application-Specific Integrated Circuit (ASIC) or Field-Programmable Gate Array (FPGA) implementations which can provide wide bandwidth but small amount of on-chip memory. Conversely, DFA tend to be quite large in terms of memory, but only have a single transition between states. Therefore, DFA are more suitable for general-purpose processors and network processors.

Ternary Content Addressable Memories (TCAM) have been widely adopted by network applications, such as routers, to improve the speed of the longest prefix matching. Deep packet inspection systems have begun to use TCAM to perform pattern matching. TCAM matching is extremely fast because it allows the input key to compare against all patterns in parallel and return a result in a single clock cycle. Unfortunately, TCAM suffer from high circuit complexity which results in high cost and increased power consumption. The increase in power consumption has become a major concern as the cost of electricity continues to rise. Thus, Binary Content Addressable Memories (BCAM) are offered as an alternative to TCAM solutions. BCAM require fewer transistors and exhibit less complex circuitry which translates into reduced power consumption. This reduction, however, comes at the cost of the "don't care" state that is present in TCAM that allows a comparison to match either a 0 or a 1. The absence of this state is problematic in pattern matching as the "don't care" state is common to many patterns.

Our approach is threefold. First, we reduce the complexity of each storage cell so that BCAM may be employed rather than TCAM. Secondly, we utilize NFA rather than DFA in order to reduce the number of required entries stored in the

BCAM. Finally, we implement a scalable, parallel processing architecture in order to achieve high throughput. Under our approach, it becomes possible to implement high throughput pattern matching in BCAM and garner the benefits of BCAM's reduced cost and power consumption.

The remainder of the paper is organized as follows. In Section II, related work in pattern matching is presented. Section III describes our algorithm and its implementation. The proposed algorithm is evaluated in Section IV. Finally, we conclude in Section V.

II. Related Work

Pattern matching is a primary function of NIDS and Network Intrusion Prevention Systems (NIPS) in commercial products such as 3Com TippingPoint X505 [7] and Cisco IPS [8]. Pattern matchers are typically implemented using finite automata, either NFA or DFA. Typical pattern matching approaches fall into one of the following categories:

a) Software-based: The software-based approach, also called the general-purpose approach, are based on general-purpose processors or network processors [4], [9]–[13]. DFA are popular in software-based approaches because they only need one state transition per input character, which causes at most one memory access for each character input. Therefore, they are often desirable at high network link rates. However, the practical use of DFA is limited because of their excessive memory usage. In order to mitigate this issue, many methods have been proposed such as [10]–[14]. These methods have developed several compression techniques for DFAs, focusing on reducing the number of transitions between states, and in some cases, 99% of all transitions can be eliminated.

b) Application-Specific Integrated Circuit (ASIC) based: Several commercial network equipment vendors, including 3Com [7] and Cisco [8] have supplied their own NIDS, and a number of smaller players have introduced pattern matching ASICs which go inside these NIDS. Developing ASICs for NIDS, however, has several disadvantages. It requires a large investment, a long development cycle, and remains hard to upgrade.

c) Field Programmable Gate Array (FPGA) based: There is a body of literature advocating FPGA-based pattern matching [15]–[19]. FPGA can provide not only a fast matching cycle but also parallel matching operations. NFAs are well-suited to FPGA-based matching because of their wide bandwidth requirement and low memory consumption. FPGA however, do not have sufficient resources to accommodate very large rule sets.

d) Ternary Content Addressable Memory (TCAM) based: TCAM has become a popular approach as exhibited in recent research like [2], [20], [21]. Both [20] and [21] used TCAMs to store the transition rule table but even with transition compression approaches they still consumed a large amount of TCAM and SRAM resources. The research in [2] stored states instead of transitions in TCAMs to reduce the usage of both TCAM and SRAM, and their simulation results showed a significant improvement because there are usually many fewer states than transitions in a DFA. Unfortunately, the computational complexity of the approach is very high. We note that the high speed of TCAM is often offset by not only the cost of the TCAM, but also a slow clock speed and slow memory accesses. Thus, rather than using TCAM, we see BCAM as an attractive alternative to not only reduce cost, but to increase the clock speed. Finally, our approach eliminates the usage of SRAM and thus removes that as a factor in pattern matching.

III. Proposed Approach

A. Proposed Pattern Matching

As stated earlier, most pattern matching solutions utilize NFA or DFA. We observe that for any single pattern an input string must match every character in that pattern in order to register a match. From this, we conclude that at any point in matching the current match state is based on the previous match state and the current matching result. Thus, it is possible to chain matching decisions and that an input string will not match unless the chain is followed to the final state. To illustrate we employ the same example finite automata as in [2] utilizing a simple pattern set of: CF, BCD, BBA, BA, EBBC, and EBC. The corresponding NFA and DFA for this set are shown in Fig. 1. We note that NFA use ε transitions to represent the possibility of moving to a new state without consuming any input character. This can result in multiple active states (concurrent traversals of the NFA) while processing an input string. It also means that if an input string fails to match, then that active state halts. Thus, if the chain of matching characters is broken, then traversal is restarted at the beginning of the NFA. We utilize this aspect, and the following two properties to simplify our matching circuitry:

- Only the start state of the NFA has incoming or outgoing ε transitions.
- Each state of the NFA has only one incoming transition.

Fig. 1(a) illustrates an NFA created under the above constraints. The DFA is shown to the right in Fig. 1(b), and most of the transitions are omitted for clarity here. We can see that both the NFA and DFA have 14 states but the DFA has many more transitions. Furthermore, every state in the NFA has only one incoming transition except for the initial state.

The final piece to our approach concerns BCAM memory. A single BCAM entry may store some number of BCAM cells each containing a 0 or 1. A TCAM entry, however, contains a third, "don't care," state that matches both 0 and 1. To support this a TCAM entry stores content as a (value, mask) pair, where value and mask are W-bit numbers, requiring W storage cells for the value and an additional W storage cells for the mask. Moreover, the matching circuitry is more complicated than that of a BCAM entry. A typical TCAM cell requires two SRAM cells plus circuitry for the matching logic. Each SRAM cell typically requires six transistors and the matching logic requires 4 transistors. Thus, a single TCAM cell is 2.7 times larger than the typical SRAM cell, though different techniques used by CAM manufactures can result in different sizes for

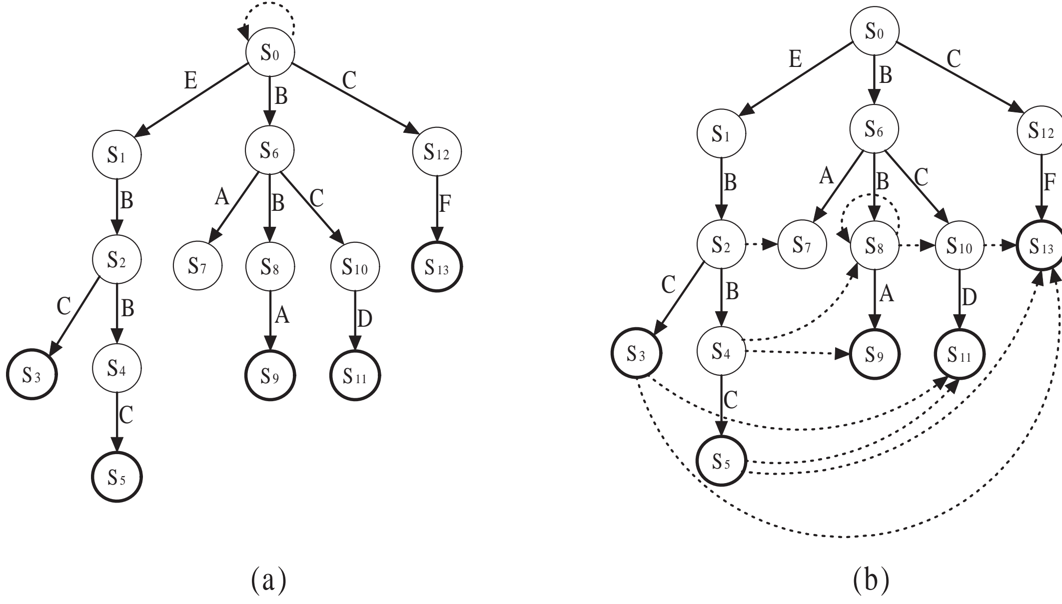

Fig. 1. NFA in (a) and DFA in (b) for the patterns {EBC,EBBC,BA,BBA,BCD,CF}. Failure and restartable transitions are omitted for clarity in (b)

SRAM cells [24]. Conversely, a single BCAM cell requires only a single SRAM cell and simple matching logic. As a result, we assume that the number of transistors and power consumption of a TCAM cell are roughly two times that of a BCAM cell.

The basic logic of the proposed approach is shown in Fig. 2. We assume there are n patterns and the sum of all their lengths, in characters, is N. Each BCAM entry stores the active state of the previous BCAM (labeled V_i in Fig. 2) as well as the next corresponding character from the composite of all patterns (labeled C_i Fig. 2). For the first BCAM entry, the active state will always match due to the epsilon transition in the NFA. From the second BCAM entry on, the active states depend on the matching result of the previous BCAM entry from the previous clock cycle combined with the matching result of the current BCAM entry in the current clock cycle. A complete match occurs after the final state for a pattern has successfully matched, implying that all previous states also matched as illustrated by r_1 in Fig. 2.

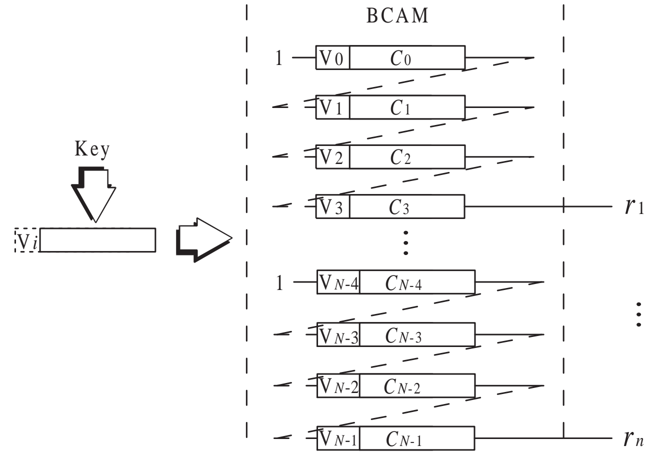

Fig. 2. The logic of proposed approach based on BCAM

The real architecture of the proposed approach is shown in Fig. 3. There are still n patterns and the sum of all their lengths, in characters, is N. Each BCAM entry stores a character from the patterns, in sequence, so that all the BCAM entries are 8 bits. Registers here are used to record the current matching result for usage in the next clock cycle, and the “AND” logics are used to determine when both previous and current matches happen. For example, imagine that an incoming packet matches the first pattern r_1. In this example, C_0 matches in the first clock cycle and a “1” is stored in the first register. In the next clock cycle, a logical “AND” is performed on the result from the first clock cycle, stored in the first register, with the output of C_1. That result is stored in the second register. This continues with the result of each stage stored in its respective register. At C_3, the final character for r_1 is matched, and since all successive characters have matched, the last register (the third in this case) contains a “1” so when a logical “AND” is applied to the match result for C_3 and the previous register a “1” is returned indicating a complete match to r_1. Further, since the last stage has matched a complete pattern there is no need to store the result in a register. Likewise, a failure to match at any stage will of course make matching the complete pattern impossible.

In order to increase the throughput, we designed our approach to process multiple characters at a time. In this paper we process four characters at a time though the architecture can easily be extended to process more characters at a time. The architecture of the example is shown in Fig. 4. In this example, we consider the pattern “ABCD” and the input string

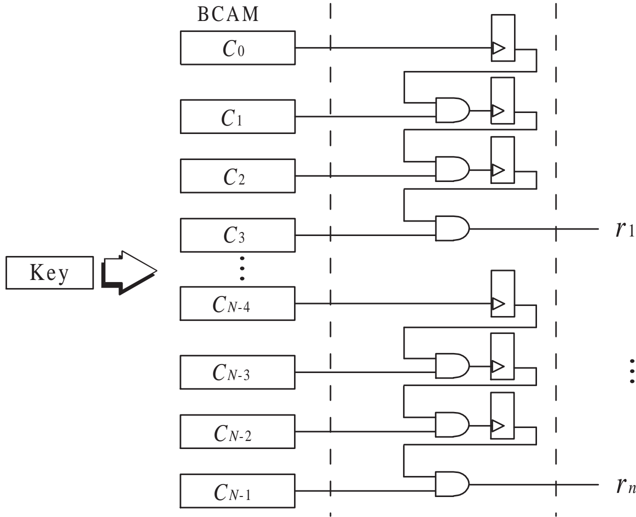

Fig. 3. The main architecture of proposed approach based on BCAM

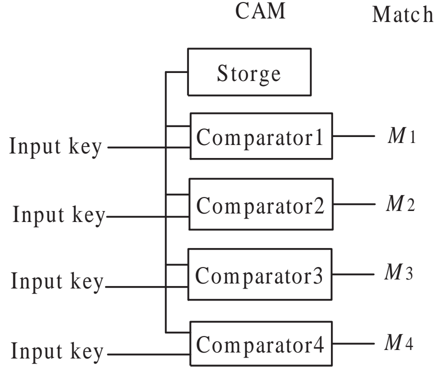

Fig. 5. Combine multiple BCAM entries with the same content

"CABCDFE". In order to process four characters in a single clock cycle, we need to consider four beginning points in the incoming string, so we need four sets of patterns for "ABCD" to be stored in the BCAM entries. The dashed lines in Fig. 4 show the comparison relationships between incoming characters and BCAM entries. The four "AND" logics mean the incoming string must match all characters in the pattern in a single clock cycle, and the "OR" logic means the pattern can appear anywhere in the incoming string. The incoming string must shift four characters every time, but all seven characters in the incoming string must be examined. So in the example, the second set of BCAMs store pattern "ABCD" and match the incoming string. Further, the four BCAM entries in the same column in Fig. 4 can be combined together to save hardware logic as they store the same content. The combined architecture is shown in Fig. 5 illustrating how the BCAM entries with the same content can share data storage and utilize their own comparator logic.

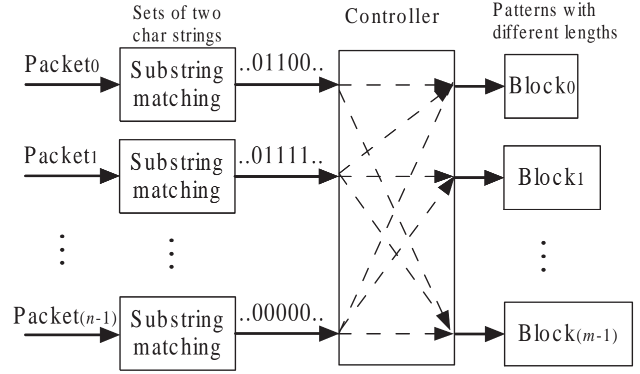

Fig. 6. Proposed hierarchical pattern matching architecture

B. Hierarchical Pattern Matching

In our simulations, we observe that most of the packets do not contain any matching string patterns nor do they contain any matching substrings of the rule set. Furthermore, many of the string patterns in the rule set contain common substrings. Based on these observations, we propose a hierarchical architecture and use a substring set of the rule set patterns as a pre-filter of the string pattern matching. We select all two-character substrings within all the patterns and store multiple sets of the whole substrings in BCAMs of 16-bit length. Multiple packets can be checked in parallel against the substrings, the output is "1" if current continuous two characters in the packet payload matches any substring and "0" if no match found, only matched substrings in the packet payload need to be checked against string patterns and the whole packet payload doesn't need to be checked against string patterns if there is no match in the first stage. Another benefit of this approach is that if the length of continuous "1" in the output is n, we only need to check the patterns of length no shorter than n, then we can divide patterns into multiple blocks to perform pattern matching in parallel. However, this could result in poor load balancing and the short pattern matching may become the bottleneck because the short patterns will be checked more often. Fortunately, we can build multiple short pattern blocks to work in parallel because they consume far fewer resources than long patterns. The hierarchical architecture is shown in Fig. 6. We use 32 sets of substrings in the first stage in this paper.

IV. Evaluation

To evaluate the proposed approach, we collected the pattern sets from Snort [1] and ClamAV [25] which are the same as used in [2] for comparison. We build a single NFA, for each set, with only prefix merging because we need to make sure each state in the NFA only has one incoming transition; excepting the beginning state. We implement the NFA in NetFPGA [26], which is a network hardware accelerator that augments network functions of a standard computer. We use the Xilinx Virtex-II Pro FPGA on the NetFPGA and implement our algorithm on it for the simulation. The NetFPGA

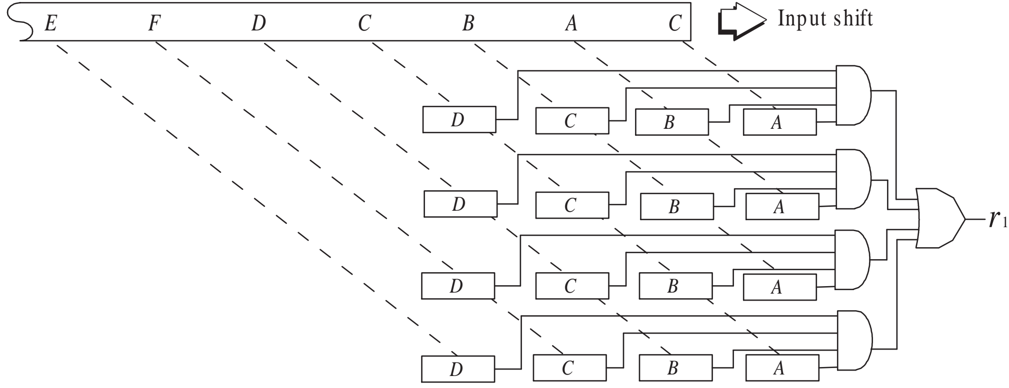

Fig. 4. Proposed parallel processing unit for pattern matching

TABLE I
STATISTICS OF THE PATTERNS COLLECTED FROM SNORT AND CLAMAV

Pattern Set	Approaches	Patterns	States in NFA	States in DFA
Snort	In [2]	6,423	-	75,256
Snort	Our approach	6,423	75,256	-
ClamAV	In [2]	26,987	-	1,565,874
ClamAV	Our approach	26,987	1,565,874	-

TABLE II
COMPARISON BETWEEN COMPACTDFA AND HIERARCHICAL NFA

Items	Pattern set	CompactDFA	Hierarchical NFA
CAM type	Both	TCAM	BCAM
CAM length	Both	36-bits	8-bit
CAM Size (MB)	Snort	0.36	0.08
CAM Size (MB)	ClamAV	8.18	1.57
SRAM Size (MB)	Snort	0.32	0.00
SRAM Size (MB)	ClamAV	7.37	0.00
Need memory access	Both	Yes	No
Construction speed	Both	Slow	Fast
Pattern update speed	Both	Slow	Fast
Platform	Both	TCAM chip	FPGA

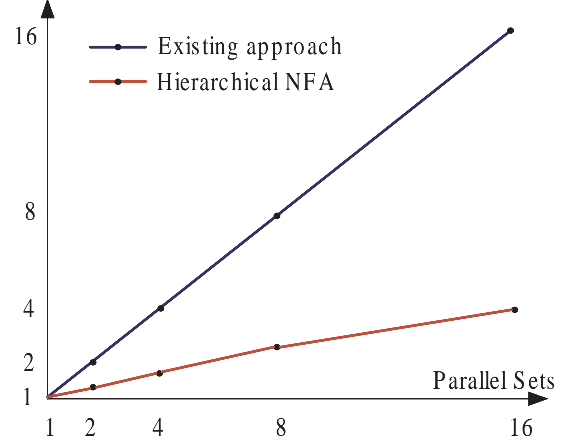

Fig. 7. CAM entry consumption increases as the number of parallel processing units increases.

card has four Gigabit Ethernet ports, SRAM and DRAM chips on board, and the NetFPGA communicates with the host PC through a Peripheral Communication Interconnect (PCI) bus. All the BCAM entries are 8-bit wide and each character is stored in a BCAM entry. So the number of BCAM entries used is the number of states in the NFA, as well as total number of characters in patterns when only counting common prefixes once.

We compare our approach with the latest and the most efficient TCAM-based pattern matching approach "CompactDFA" [2] to the best of our knowledge. The rule sets collected from Snort and ClamAV are shown in TABLE I, and simulation results are shown in TABLE II.

From the simulation results we can see our algorithm outperforms "CompactDFA" proposed in [2] in almost all aspects. For power consumption by CAM, as we mentioned the power consumption of a TCAM cell is two times as large as a BCAM cell, our approach consumes 11.1% and 9.6% power of "CompactDFA" based on the Snort and ClamAV pattern sets respectively. However, the disadvantage of our approach is that it needs to be implemented on a configurable hardware platform such as FPGA. Fortunately, our approach is small enough to be configured into FPGAs. Further, checking four characters in a single clock cycle, allows our approach to easily achieve a throughput of 16 Gbps.

Increasing the number of parallel processing units will increase the CAM entry consumption. We illustrate how our approach scales in this increase as opposed to other approaches in Fig. 7. As is evident, our approach scales better due to resource sharing by parallel processing units.

For the hierarchical approach, we focus on the first stage of substring matching. We selected 5,261 unique patterns with length of at least two characters from the Snort rule set as published by the Sourcefire Vulnerability Research team [27] for December 20, 2010. From these 5,261 patterns we identified only 333 unique substrings with length of two characters. Thus, a single substring set for this rule set would consume only 666-bytes of BCAM, (at two bytes per substring pattern) which is less than 1% of the total memory the BCAM consumes for the proposed pattern matching approach. We

can extrapolate from these results and estimate that 32 such sets would only consume about 21% more memory than the approach as outlined in this paper. This implies that such an approach is suitable for parallel processing and further implies that the substring matching is a valid pre-filter. In our simulation, we collected a total of 5,203 packets from attack exercises performed locally which exhibited several attacks against networks, primarily cross-site scripting attacks. In the substring pre-filtering stage, there are a total of 51,635 matched substrings and 1,887,078 unmatched substrings in all the packets' payload, which means only 2.7% of the aggregate packet payload matched any substring. More interestingly, the few matches that occurred typically happened in the same packets. In other words, only a few packets matched and those that did matched multiple times. About 90.3% of the packets contain no substring matches and do not need further checking. That means the majority of the traffic will not require further processing than the pre-filter. With the clock speed of 500 MHz and an optimal balance of output from the pre-filter to pattern matching then our approach can achieve up to 128 Gbps throughput.

V. Conclusion and Future Work

In this paper, we studied more effective techniques for pattern matching in deep packet inspection. We built an efficient NFA generator based on BCAMs, which consumes fewer transistors and which has reduced latency because of shorter BCAM entries used in our approach. As a result, our approach demonstrates improved matching speed for every character while reducing the amount of SRAM resources needed. Also, our approach can process multiple characters at a time while using only a limited number of BCAM entries, which improves potential scalability scalable. Furthermore, we built a hierarchical pattern matching architecture which serves to exclude most packets from full pattern matching leaving only a small percentage to be fully checked in the pattern matching process. This hierarchical pattern matching improves throughput for the average case. In all, our evaluation demonstrates that our approach outperforms existing similar approaches. In the future, we hope to apply our approach to regular expression matching to combat the increasing complexity of regular expressions.

References

[1] *Snort User Manual 2.8.6*, The Snort Project, Apr. 2010, http://www.snort.org/assets/140/snort_manual_2_8_6.pdf.

[2] A. Bremler-Barr, D. Hay, and Y. Koral, "CompactDFA: Generic state machine compression for scalable pattern matching," in *Proceedings of the 29th conference on Information Communications, INFOCOM'10*, 2010, pp. 659–667.

[3] M. Paolieri, I. Bonesana, and M. D. Santambrogio, "ReCPU: a parallel and pipelined architecture for regular expression matching," in *Proceedings of 15th Annual IFIP International Conference on Very Large Scale Integration*, Oct. 2007, pp. 19–24.

[4] M. Becchi and P. Crowley, "A hybrid finite automaton for practical deep packet inspection," in *Proceedings of ACM CoNEXT*, Dec. 2007.

[5] S. Kumar, B. Chandrasekaran, J. Turner, and G. Varghese, "Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia," in *Proceedings of the 2007 ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, Dec. 2007, pp. 155–164.

[6] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, Dec. 2006, pp. 339–350.

[7] "TippingPoint x505," http://www.tippingpoint.com/products_ips.html.

[8] "Cisco IOS IPS signature deployment guide," http://www.cisco.com/.

[9] Y. Sun, H. Liu, V. Valgenti, and M. S. Kim, "Hybrid regular expression matching for deep packet inspection on multi-core architecture," in *Proceedings of the 19th International Conference on Computer Communications and Networks, ICCCN'10*, Aug. 2010.

[10] D. Ficara, S. Giordano, G. Procissi, F. Vitucci, G. Antichi, and A. Di Pietro, "An improved DFA for fast regular expression matching," *Computer Communication Review*, vol. 38, no. 5, pp. 29–40, 2008.

[11] M. Becchi and P. Crowley, "An improved algorithm to accelerate regular expression evaluation," in *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, 2007, pp. 145–154.

[12] S. Kumar, J. Turner, and J. Williams, "Advanced algorithms for fast and scalable deep packet inspection," in *Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems*, 2006, pp. 81–92.

[13] M. Becchi and S. Cadambi, "Memory-efficient regular expression search using state merging," in *Proceedings of the 26th IEEE International Conference on Computer Communications*, May 2007, pp. 29–40.

[14] M. Becchi, C. Wiseman, and P. Crowley, "Evaluating regular expression matching engines on network and general purpose processors," in *Proceedings of the 2009 ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, Oct. 2009.

[15] W. Lin and B. Liu, "Pipelined parallel AC-based approach for multi-string matching," in *Proceedings of the 14th IEEE International Conference on Parallel and Distributed Systems, ICPADS'08*, Dec. 2008, pp. 665–672.

[16] I. Bonesana, M. Paolieri, and M. Santambrogio, "An adaptable FPGA-based system for regular expression matching," in *Proceedings of the conference on Design, Automation and Test in Europe*, Mar. 2008, pp. 1262–1267.

[17] N. Yamagaki, R. Sidhu, and S. Kamiya, "High-speed regular expression matching engine using multi-character NFA," in *International Conference on Field Programmable Logic and Applications*, Sep. 2008, pp. 131–136.

[18] C. R. Clark and D. E. Schimmel, "Efficient reconfigurable logic circuit for matching complex network intrusion detection patterns," in *Conference on field-programmable logic and applications*, Sep. 2003, pp. 956–959.

[19] A. Mitra, W. Najjar, and L. Bhuyan, "Compiling PCRE to FPGA for accelerating SNORT IDS," in *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems*, 2007, pp. 127–136.

[20] M. Alicherry, M. Muthuprasanna, and V. Kumar, "High speed pattern matching for network IDS/IPS," in *Proceedings of the 14th IEEE International Conference on Network Protocols, ICNP'06*, Oct. 2006, pp. 187–196.

[21] F. Yu, H. R. Katz, and T. V. Lakshman, "Gigabit rate packet pattern-matching using TCAM," in *Proceedings of the 12th IEEE International Conference on Network Protocols, ICNP'04*, Oct. 2004, pp. 174–183.

[22] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23–24, pp. 2435–2463, Dec. 1999.

[23] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Operational experiences with high-volume network intrusion detection," in *Proceedings of the 11th ACM Conference on Computer and Comm. Security*, Oct. 2004.

[24] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: a tutorial and survey," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006.

[25] *ClamAV*, Clam AntiVirus, http://www.clamav.net/lang/en/.

[26] G. Gibb, J. W. Lockwood, J. Naous, P. Hartke, and N. McKeown, "NetFPGA: an open platform for teaching how to build gigabit-rate network switches and routers," *IEEE Transactions on Education*, vol. 51, no. 3, pp. 364–369, Aug. 2008.

[27] *Sourcefire Vulnerability Research Team (VRT) Snort Rule-set*, 2nd ed., Sourcefire Vulnerability Research Team, Sep. 2010, available at http://www.snort.org/vrt.