A Multi-Gb/s Parallel String Matching Engine for Intrusion Detection Systems

Vahid Rahmanzadeh[1] and Mohammad Bagher Ghaznavi-Ghoushchi[2]

[1] EE. Dept. School of Engineering, Tarbiat Modarres University, Tehran, Iran
[2] EE. Dept. School of Engineering, Shahed University, Tehran, Iran
rahmanzadeh@modares.ac.ir, ghaznavi@shahed.ac.ir

Abstract. This paper describes a Finite State Machine (FSM) approach on string matching for Intrusion Detection Systems (IDS). Search patterns are sliced into multiple interleaved substrings and feed into parallel FSMs. The final match results from combining the outputs of parallel individual FSMs. The proposed engine is primarily designed for ASCII codes and extended to support (16-bit) Unicode. The designed engine with 4-byte input words can reach search rates of over 30 Gb/s.

Keywords: String matching, Intrusion Detection Systems, Bit-splitting, Finite State Machine (FSM).

1 Introduction

String matching algorithms are crucial in text processing toolsets, data mining and especially in intrusion detection systems. In network intrusion detection systems, high throughput string matching engines are required to search in network traffic without degrading the speed of data transferring in network. Software string matching algorithms can reach the throughput of 250MHz at most [5]. In hardware solutions by using the natural properties of hardware such as parallelism, the Gb/s throughputs can be achieved [1].

Our work is based on two recent string matching techniques. First, Bit-Splitting for implementation of Finite State Machines [2] and second, parallel searching in multi-byte input words [1]. In our architecture, input sequence is partitioned to subsequences and each subsequence searched by one match module. In each clock cycle outputs of match modules are combined in an efficient way to determine whether a string pattern has been matched across all match modules or not.

The rest of this paper is laid out as follow. In section 2, the hierarchical structure of proposed string matching engine is described. Section 3 is about preprocessing algorithm. In section 4, simulation results are presented and conclusion is in section 5.

2 Architecture

Our architecture at the highest level is a full match machine. Full machine is composed of string matching engines that are structurally similar. All string patterns that

H. Sarbazi-Azad et al. (Eds.): CSICC 2008, CCIS 6, pp. 847 – 851, 2008.

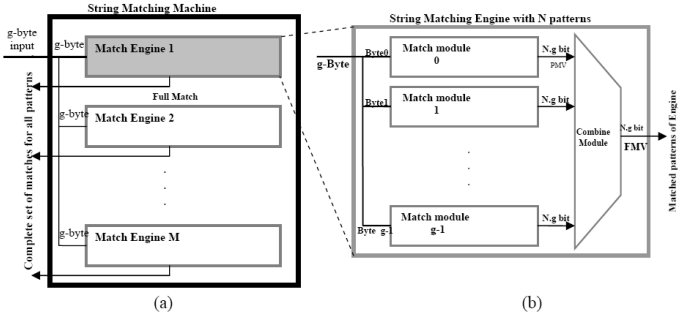

Fig. 1. Part (a) is a full string matching machine and part (b) is details of match engine

machine search them, classify to separated groups and each group is searched by one string matching engine. In each clock cycle, a g-byte wide input word is entered to machine and is sent to all parallel engines, where g is the length of input word in bytes and the number of match modules in a match engine.

Fig.1 (a) shows the full string matching machine, the part (b) of Fig.1 shows a match engine that is composed of g parallel match modules and a combine module.

Each match engine is composed of g-parallel match modules and a combine module. The set of g-byte input words are entering to engine each clock cycle. Bytes of input word distributed between g-match modules consecutively. Distribution is done based on index of each byte in input word. In this schema, input stream that is incoming to engine continuously, divided to g-individual subsequences in a g-interleaved manner.

For consistency, each string pattern also sliced to g-substrings in a way similar to input partitioning. It is supposed that a string pattern sliced to g-substrings from pat(0) to pat(g-1). At the first step for pattern splitting, pattern is divided to consecutive g-byte parts. Then by arranging the first byte of pattern parts consecutively, pat(0) is produced, pat(1) produced by second byte and pat(g-1) produced by g-th byte of pattern parts.

Substrings are entered to all match modules. Each match module is an implementation of Aho-Corasick string matching algorithm that searches all substrings of engine in its input subsequence and report matched substrings in output vector. Output vector of g-match modules is entered to combine module. In match modules Aho-Corasick algorithm implemented is based on bit-splitting technique by multiple parallel binary FSMs. Fig.2 shows structure of implemented match modules.

For the case of matching a string pattern, all substrings from pat(0) to pat(g-1) must match consecutively by match modules. Appropriate orders of substrings that are matched by match modules are checked in combine module. In our design, combine module is implemented by 'AND' and 'OR' gates to reduce delay of combine module. After implementation, structure of combine module is fixed and rule updating is done on the base of configuration of combine module of match engines.

Using ram-based technique for FSM implementation, allows non-interrupting rule update of match modules. Rules of each engine and total machine can be updated without stopping the operation of match machine. This rule updating process can complete in the order of seconds while FPGA based methods generally require days, to recompile rules. If rules are added one at a time, each match engine will take less than a total of g seconds to update [2].

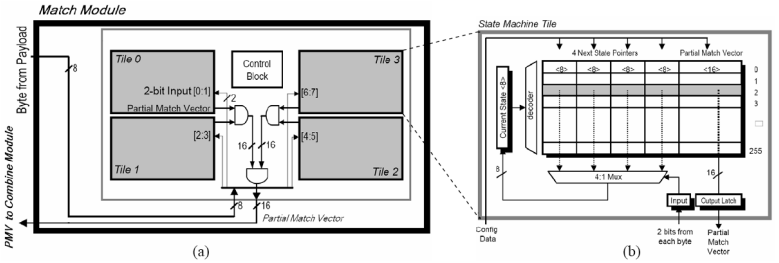

Fig. 2. Part (a) is the structure of match module and part (b) is a Tile implemented by table[ref.2]

3 Preprocessing Algorithm

In this section system software and data preprocessing algorithms implemented by C++ language are described.

At first step, in each match engine, strings sliced to g substrings in g-way interleaved manner. An Aho-Corasick FSM named D is built from substrings and each FSM is sliced apart into a new set of four FSMs, 2-bit groups of each byte of substrings are extracted to construct own Binary State Machine (BSM). The alphabet elements of this FSM are {0, 1, 2, 3}. Binary FSMs of B0, B1, B2, B3 are built by splitting D in 2-bit groups, by the following steps.

Splitting starts from the initial state of D and all next states from root node in D-FSM are traversed. The output edges of each state are partitioned to four groups; those that are set to 0, 1, 2 or 3. These edges go to four new states in B. The process is repeated for other states of D to complete construction of B_i forward tree. Forward tree is the primary structure of FSM that in each state only success edges inserted. In next stage, failure edges are added to forward tree and matched patterns are added in each state. This part of algorithm is based on [2].

4 Simulation Results

Match modules are large FSMs implemented by bit-splitting into 4-tiny FSMs. Each FSM for 1-byte input words can achieve worst case throughput of 10-Gb/s for ASIC implementation [2], and 1.6-Gb/s for FPGA implementation on virtex 4fx100 [4]. By parallelism, total time of searching divided in two parts. One part is parallelizable time T_p that divided between parallel modules and reduced by factor g. Another part is serial time T_s that after parallelism being fixed or may be increased. Total serial time is sum of the original serial time of algorithm and inter-module communication that increased by parallelism. Speed-up formula for g stages parallelism is (1).

$$S(g) = \frac{T_s + T_p}{T_s + T_p / g} \tag{1}$$

As declared in (1), if delay of T_s is negligible as compare to T_p, speed-up can reach around g by g-stages parallelism.

In our proposed architecture, critical part of serial time is the delay of combine module. Combine module is implemented by g, g-input 'AND' gates and one g-input 'OR' gate. The implemented structure for combine module is really simpler than the structure of match modules. Therefore, the delay of combine module is much less than the delay of match module but yet, the delay of combine module vitiates the number of parallel match modules. By using pipelining, combining module works during the next clock cycle. It adds one clock cycle latency in design but throughput becomes linear on the number of match modules in each match engine.

In FPGA implementation, by using 4-parallel match modules, we achieved the throughput of 6.75Gb/s on virtex5. With this structure, throughput of over 30Gb/s for ASIC implementation can be achieved.

5 Conclusion

A parallel string matching for searching in multi-byte input words is developed. Input string patterns are divided to substrings in an interleaved way. Each substring is feed into all parallel FSMs of engine. The parallel FSMs search in separated input subsequences for substrings. The results of parallel FSMs are combined with a simple logic and matched patterns are determined.

Specially, this paper makes the following research contributions.

- Our structure search in multi-byte input words by parallel matching blocks in an interleaved way. The throughput of over 6.5 Gb/s on FPGA and over 30Gb/s for ASIC implementation can be achieved by 4 matching blocks in parallel.
- Our algorithm simply can be extended for processing larger input words in each clock cycle. On the other hand, Throughput can be increased linearly by using more parallel match modules.
- Our searching method and architecture can be easily extended for processing non-ASCII strings and languages in 16-bit Unicode.
- Our efficient architecture allows the rules to be updated without interrupting when string matching machine is working. The updating process can complete in the order of seconds.

References

1. Tripp, G.: A parallel String Matching engine for use in high speed network intrusion detection. J. computer virol. (2006)
2. Tan, L., Sherwood, T.: A high Throughput string matching architecture for intrusion detection and prevention. In: Proceedings of the 32nd International Symposium on Computer Architecture, ISCA 2005, pp. 112–122 (June 2005)
3. Aho, A.V., Corasick, M.J.: Efficient string matching: An aid to bibliographic search. Communications of the ACM 18(6), 333–340 (1975)
4. Jung, H.J., Backer, Z.K., Prasannaa, V.K.: Performance of FPGA Implementation of Bit-Split Architecture for Intrusion Detection Systems. IEEE, Los Alamitos (2006)

5. Aldwairi, M., Conte, M., Franzon, P.: Configurable string matching hardware for speedup up intrusion detection. In: Workshop on Architectural Support for Security and Anti-virus (WASSA) Held in Cooperation with ASPLOS XI (October 2004)
6. Attig, M., Lockwood, J.W.: SIFT: snort intrusion filter for TCP. In: Proceedings of IEEE symposium on high performance interconnects (Hot Interconnects-13), Stanford, California (2005)
7. Baker, Z.K., Prasanna, V.K.: A methodology for synthesis of efficient intrusion detection systems on FPGAs. In: Proceedings of IEEE symposium on field-programmable custom computing machines FCCM 2004, Napa, California (2004)
8. Sidhu, R., Prasanna, V.K.: Fast regular expression matching using FPGAs. In: Proceedings of the 9th international IEEE symposium on FPGAs for custom computingmachines, FCCM 2001, Rohnert Park, California, USA (2001)
9. Rahmanzadeh, V.: Generalized Approaches on 1D/2D FSPR-FSM design and Implementation. Ms. Thesis, Tarbiat modarres University, 1386, Tehran, Iran (2007)