High Speed Pattern Matching Algorithm Based on Deterministic Finite Automata with Faulty Transition Table

Jan Kastil, Jan Korenek
Brno University of Technology
Faculty of Information Technology
Bozetechova 2, Brno 612 66, Czech Republic
(ikastil, korenek) @ fit.vutbr.cz

ABSTRACT

Regular expression matching is the time-critical operation of many modern intrusion detection systems (IDS). This paper proposes pattern matching algorithm to match regular expression against multigigabit data stream. As usually used regular expressions are only subjectively tested and often generates many false positives/negatives, proposed algorithm support the possibility to reduce memory requirements by introducing small amount of faults into the pattern matching. Algorithm is based on the perfect hashing and is suitable for hardware implementation.

Categories and Subject Descriptors

C.2.0 [**Network Communication Networks**]: General –Security and protection (e.g., firewalls)

General Terms

Algorithms,Design,Security

Keywords

Intrusion Detection, Protocol recognition, pattern matching, Perfect hashing

1. INTRODUCTION

Regular expression matching is usually based on deterministic or nondeterministic finite automata. The nondeterministic automaton is often used in FPGA implementations but it requires several concurrently active states or backtracking implementation. Deterministic Finite Automaton (DFA) is memory centric and is mostly used in processor implementations. Fast lookup in transitional table is the basic operation in most implementations of the DFA.

To obtain higher throughput of the pattern matching the number of characters accepted per symbol or the frequency has to be increased. Maximal frequency is bounded by the physical limits of the FPGA. The expansion of symbol's dimension rises the alphabet's size and is directly responsible for large memory requirements. Algorithms for increasing symbol's size were already presented in [4].

Analysis in [1] shown that transition tables for regular expressions used in IDS are sparse. Hash functions are very efficient for the look-up in the sparse data structures. If the perfect hash function is used for transition look-up, next state can be obtained in one memory access with a very small overhead. Algorithm in [2] requires less then two bits per key. Disadvantage of the PHF is it's inability to distinguish between the existing and nonexisting transition. In [3], the whole transition was stored in the transition table to check the existence of the transition, which lead to high memory consumptions. The part of the architecture responsible for confirmation of the existence of the transition is called validation block.

2. PERFECT HASHING WITH FAULTS

Modern IDS often drop packets if the network load is too high. If the packet is dropped due to high load, it goes into the protected network without any check. We propose to allow small probability of the failure into the automaton to decrease memory requirements.

Proposed algorithm address trade off between speed and correctness of the matching algorithms. Introducing faults into the DFA results in the ability to test every byte of the data stream with the probability, that some of the pattern will be missed. The probability of the faults during recognition is the parameter of the proposed solution.

The memory reduction can be introduce into validation block by storing the hash fingerprint instead of the complete representation of the transition. The probability of the failure depends on the size of the fingerprint.

The principle of the whole pattern matching unit can be seen at the Figure 1. First step is the implementa-

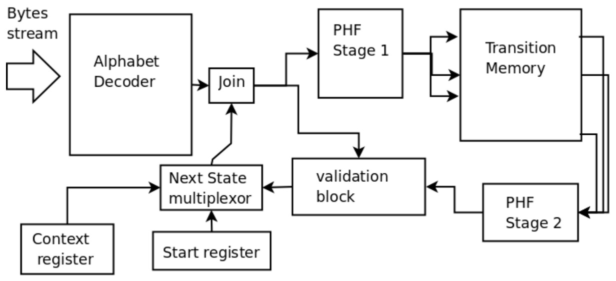

Figure 1: Principle of the matching unit

tion of the alphabet decoder. The decoded symbol is concatenated with active state information in the join block. The result of the concatenation is used as an input into the three uniform hash functions for the computation of the PHF. Computed hash values are used as pointers into the transition table implemented in the BlockRam of the FPGA. Therefore, the actual look up in the memory consumes one clock cycle. Three lines are obtained by the transition queries. Every line contains two bit information for the computation of the perfect hash function followed by the n-bit information of for the validation block and the lastest part is the next state of the automaton.

The computation of the PHF is finished and the result is used for the choosing of the correct line. The hash key stored in the line is checked in the validation block. If the validation is successful next state is connected to the input of the join block for the acceptation of the next symbol. If validation fail, than pattern matching is stopped or reset to the starting state.

Small state information allows context switching to support concurrent matching in several flows.

3. EXPERIMENTAL RESULTS

The experiments were done on subset of rules of L7 filters [5] used for protocol recognition in the Liberouter project [6]. Several pattern in this set created blow-up in number of transitions when joined together. Therefore whole group was divided into several subgroups by the algorithm described in [7]. The automaton was extended to accept 3 characters per transition which correspond to throughput 6 Gbps per unit.

Memory consumption for the implementation based on the perfect hash function was computed together with the memory consumption of the faulty unit. The memory consumption of the faulty unit depends strongly on the probability that the fault occur. Typical result of these experiments are shown in the graph in Figure 2.

4. CONCLUSION

The main contribution of this work is the decrease of the memory requirements by introducing a small portion of faults into the pattern matching in IDS. Proposed algorithm work at the flow level and achieves multigigabit throughput per flow. The algorithm is based on the Deterministic finite automata and therefore the state of the matching unit can be stored in small amount of bits.

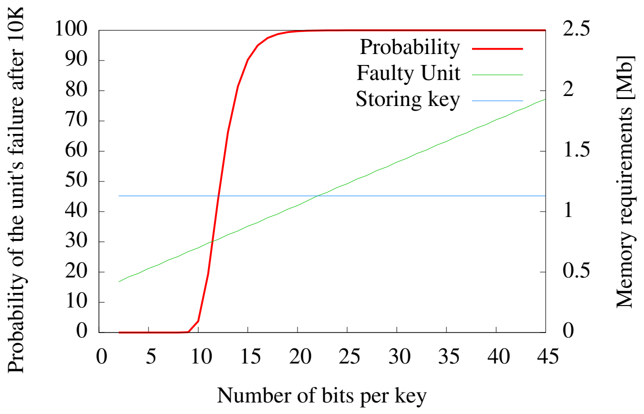

Figure 2: Typical memory savings

Algorithm uses perfect hash function to determine the next state and uniform hash function to determine validity of the transition. Early experiments shown memory savings more than 20 percent depending on the regular expressions complexity.

The future work should focus on evaluation how the quality of the IDS depends on the probability of the fault during the matching process.

Acknowledgements

This research was supported by the Research Plan No. MSM, 0021630528 —Security-Oriented Research in Information Technology, Research Plan MSM, 6383917201 and the grant BUT FIT-S-10-1.

5. REFERENCES

[1] J. Kastil and J. Korenek, "Hardware accelerated pattern matching based on deterministic finite automata with perfect hashing," in *DDECS 2010*, 2010, pp. 149–152.

[2] F. C. Botelho, R. Pagh, and N. Ziviani, "Simple and Space-efficient Minimal Perfect Hash Functions," in *In Proc. of the 10th Intl. Workshop on Data Structures and Algorithms.* Springer LNCS, 2007, pp. 139–150.

[3] J. Kastil, J. Korenek, and O. Lengal, "Methodology for fast pattern matching by deterministic finite automaton with perfect hashing," in *DSD 2009.* Washington, DC, USA: IEEE Computer Society, 2009, pp. 823–829.

[4] M. Becchi and P. Crowley, "Efficient regular expression evaluation: theory to practice," in *ANCS 2008.* New York, NY, USA: ACM, 2008, pp. 50–59.

[5] "Application layer packet classifier for linux." [Online]. Available: http://l7-filter.sourceforge.net/

[6] "Liberouter project." [Online]. Available: www.liberouter.org

[7] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz, "Fast and Memory-efficient Regular Expression Matching for Deep Packet Inspection," in *ANCS 2006.* New York, NY, USA: ACM, 2006, pp. 93–102.