High-speed Regular Expression Matching with Pipelined Memory-based Automata

BRNO UNIVERSITY OF TECHNOLOGY

Denis Matoušek, Jiří Matoušek, Jan Kořenek
{imatousekd, imatousek, korenek}@fit.vutbr.cz

FACULTY OF INFORMATION TECHNOLOGY

Introduction

Pattern matching based on regular expressions (REs) is a widely used operation in the domain of computer networks. It is a computationally intensive operation that is not suitable for sequential processing. It is used for DPI (deep packet inspection) techniques, which are used for network security monitoring and threat detection in IDS/IPS (intrusion detection/prevention systems) applications. Typical example is open-source software like Snort, Suricata, and Bro. Another use case is application-level load balancing. Required throughput for current computer networks is 10 Gbps, moving on to 40 Gbps and 100 Gbps and it will be 400 Gbps soon. FPGAs (field-programmable gate array) are used to leverage their massive parallelism and programmability in order to accelerate the operation of pattern matching.

Throughput	Data bus width	Symbols
10 Gbps	64 b	8
40 Gbps	256 b	32
100 Gbps	512 b	64
400 Gbps	2048 b	256

Table 1: Data bus width for parallel processing @ 200 MHz

Problem

Current pattern matching architectures do not scale above 10 Gbps. The techniques *multi-striding* [1, 2] and *spatial stacking* [3, 4] modify the original automaton to process multiple input symbols with a single transition. The frequency of corresponding circuit decreases with increasing number of simultaneously processed input symbols as shown in the following table.

Architecture	Clock cycle	Frequency	Throughput
Multi-striding [1] (FPGA)	4 symbols	133 MHz	4 Gbps
Multi-striding [1] (ASIC)	4 symbols	500 MHz	16 Gbps
Multi-striding [2]	2 symbols	436.7 MHz	7 Gbps
Spatial stacking [3]	8 symbols	160.9 MHz	10.3 Gbps
Spatial stacking [4]	4 symbols	N/A	7.5 Gbps

Table 2: Parameters of multi-striding and spatial stacking techniques

Delayed Input DFAs

Delayed Input DFA (D2FA) technique was introduced in [5]. It modifies an original DFA (deterministic finite automaton) by replacing multiple standard transitions with a *default transition*. The algorithm can be applied iteratively to further reduce overall number of transitions. The number of iterations is called the *radius* of a D^{2}FA. As a result, more than one transition can be used in order to accept a single input symbol. Effectively, the throughput of processing input characters is reduced. We carried out a research on how the throughput is reduced in real cases.

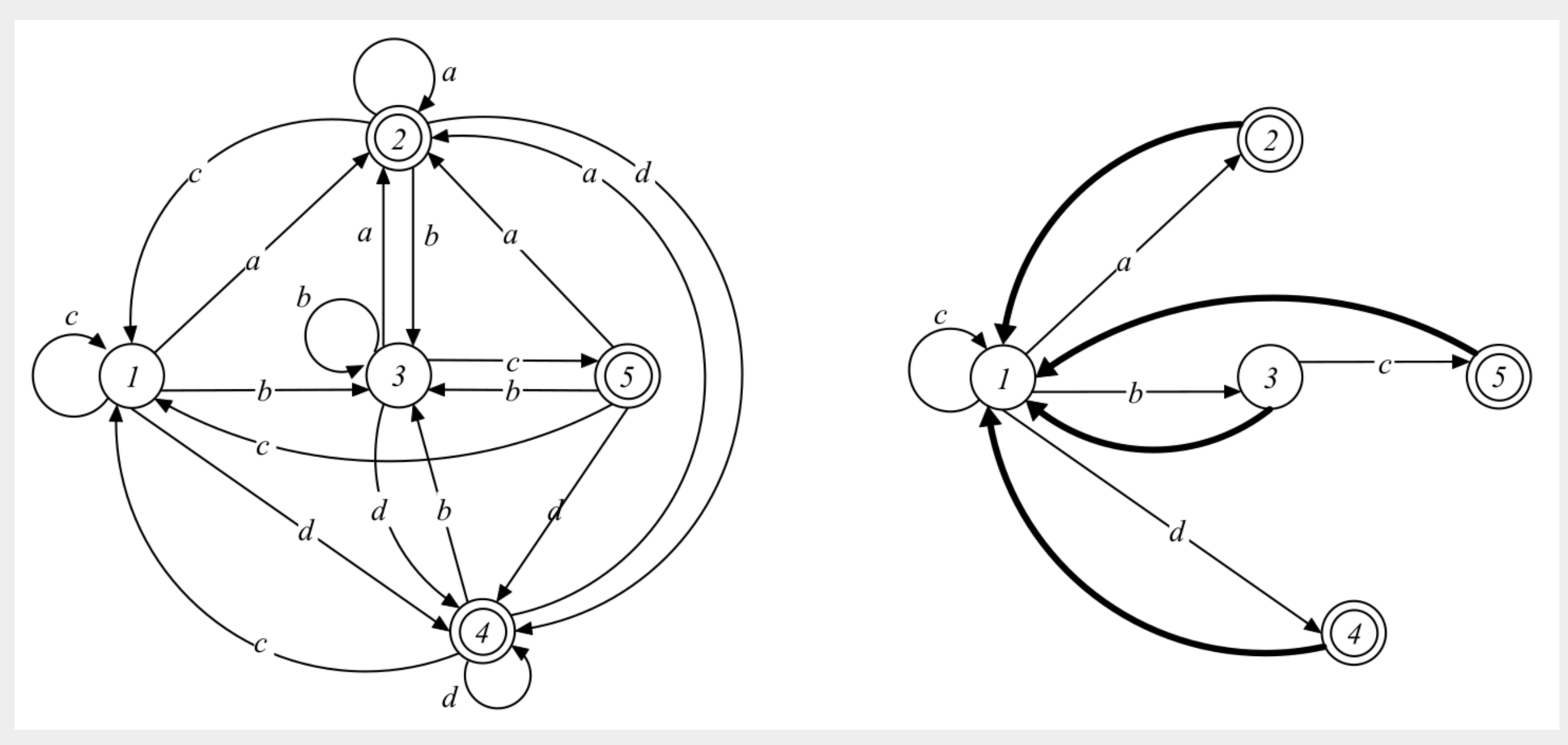

Figure 1: Illustration of D^{2}FA technique [5]

Analysis of D^{2}FA

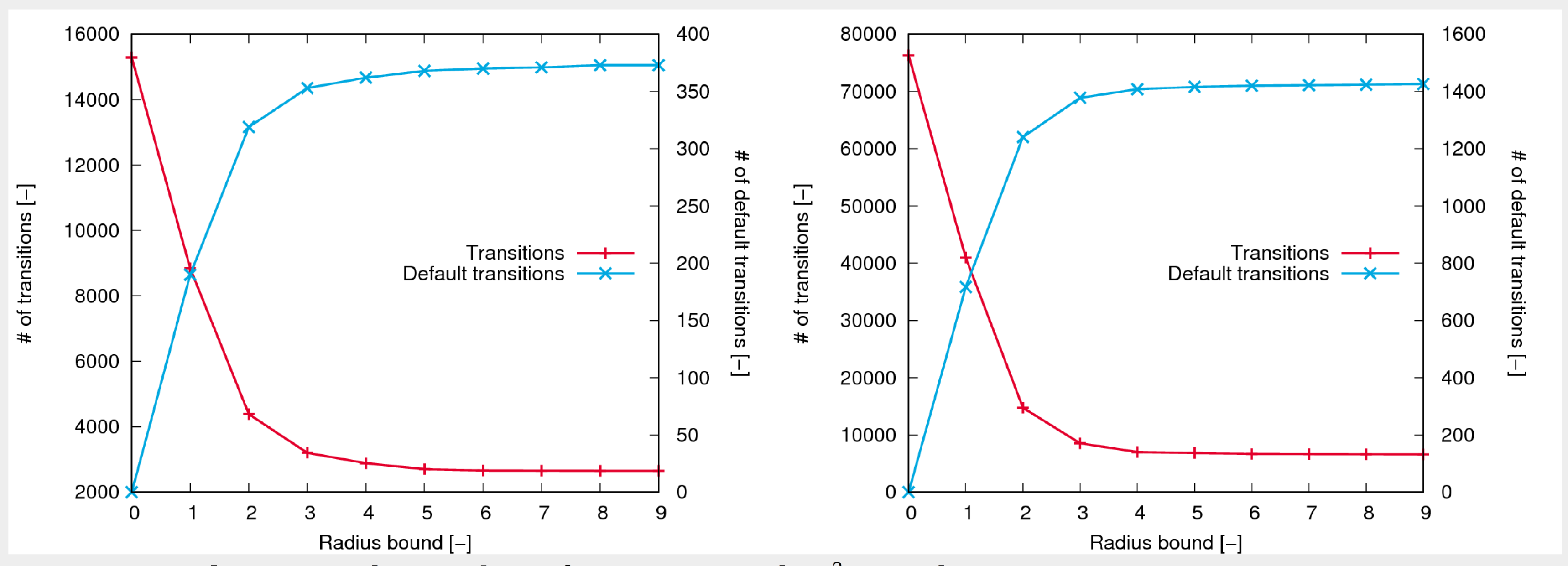

Figure 2: Reduction in the number of transition with D^{2}FA technique

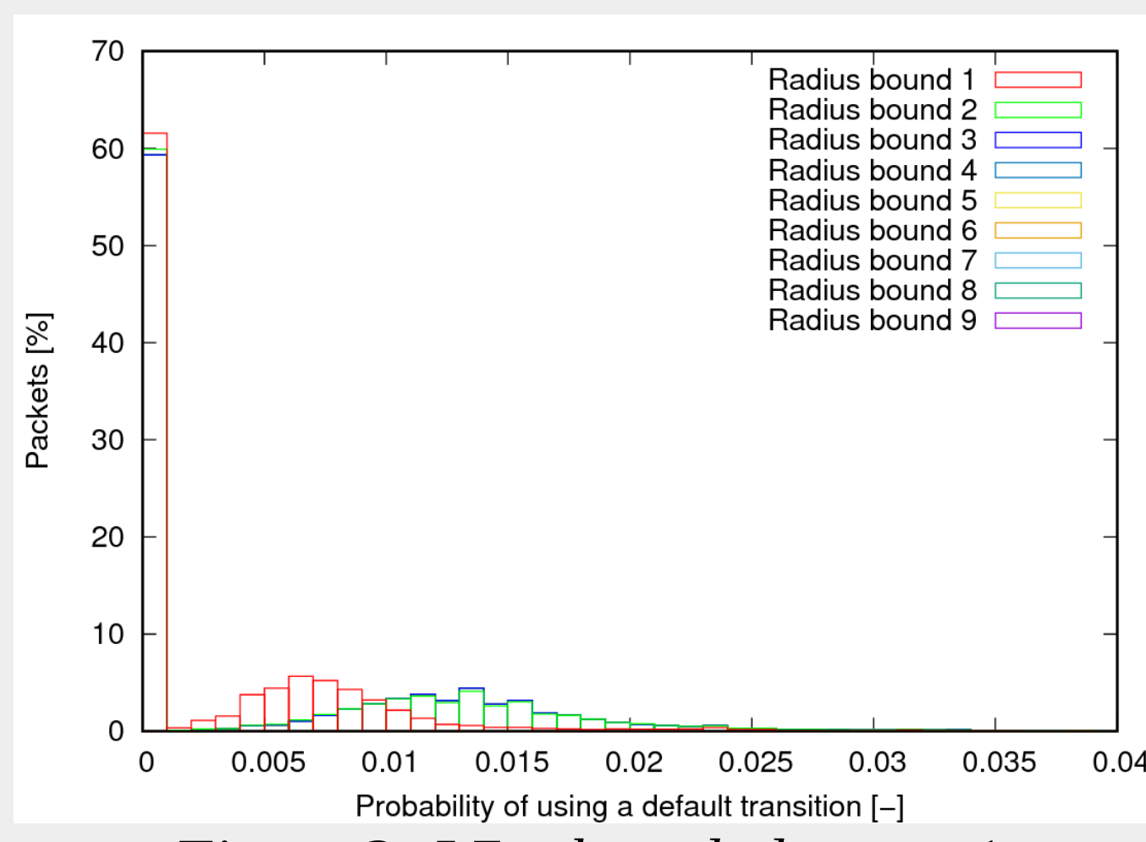

Figure 3: L7 selected, data set 1

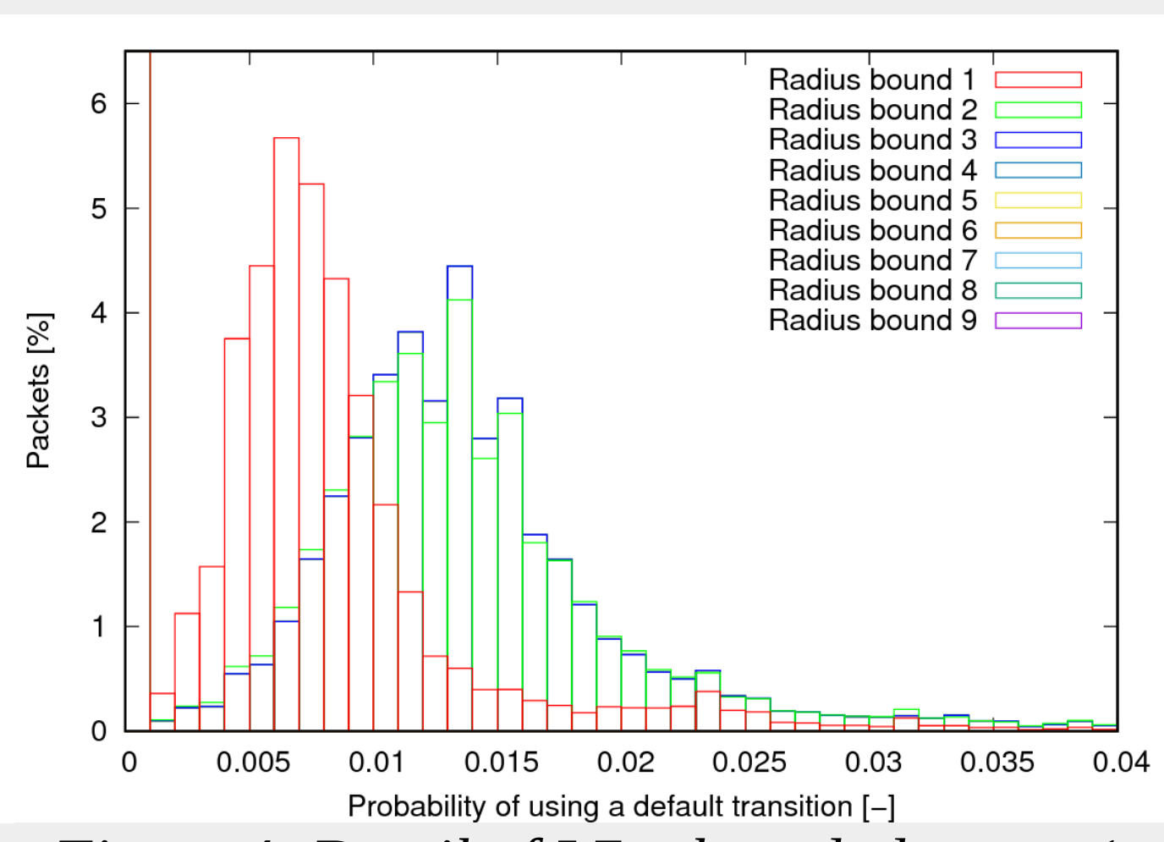

Figure 4: Detail of L7 selected, data set 1

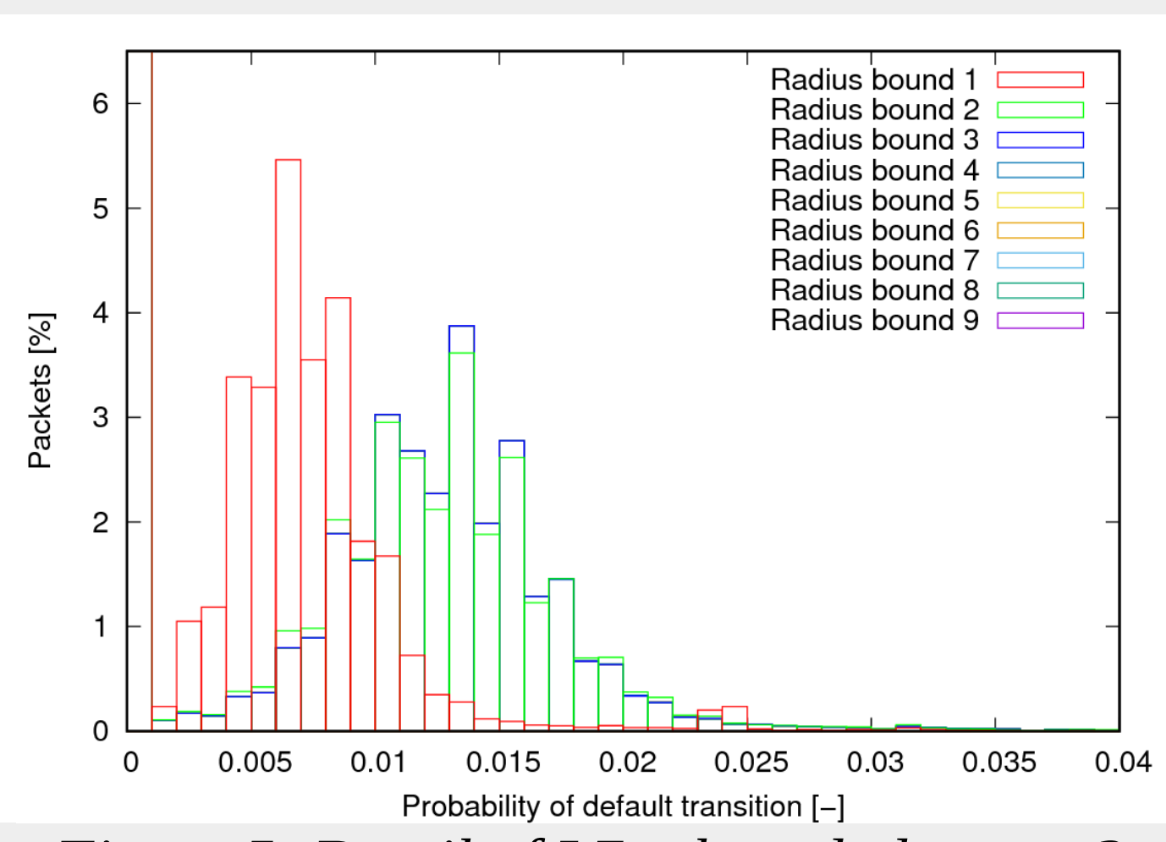

Figure 5: Detail of L7 selected, data set 2

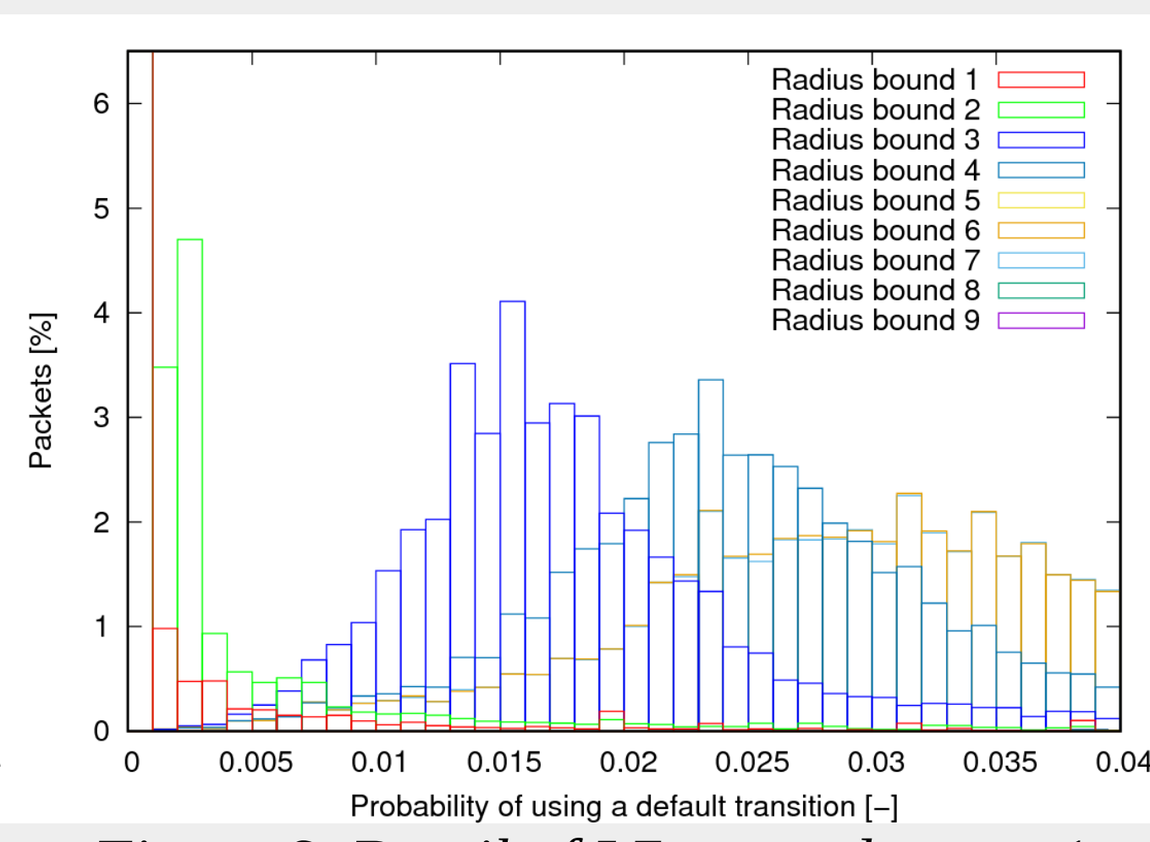

Figure 6: Detail of L7 great, data set 1

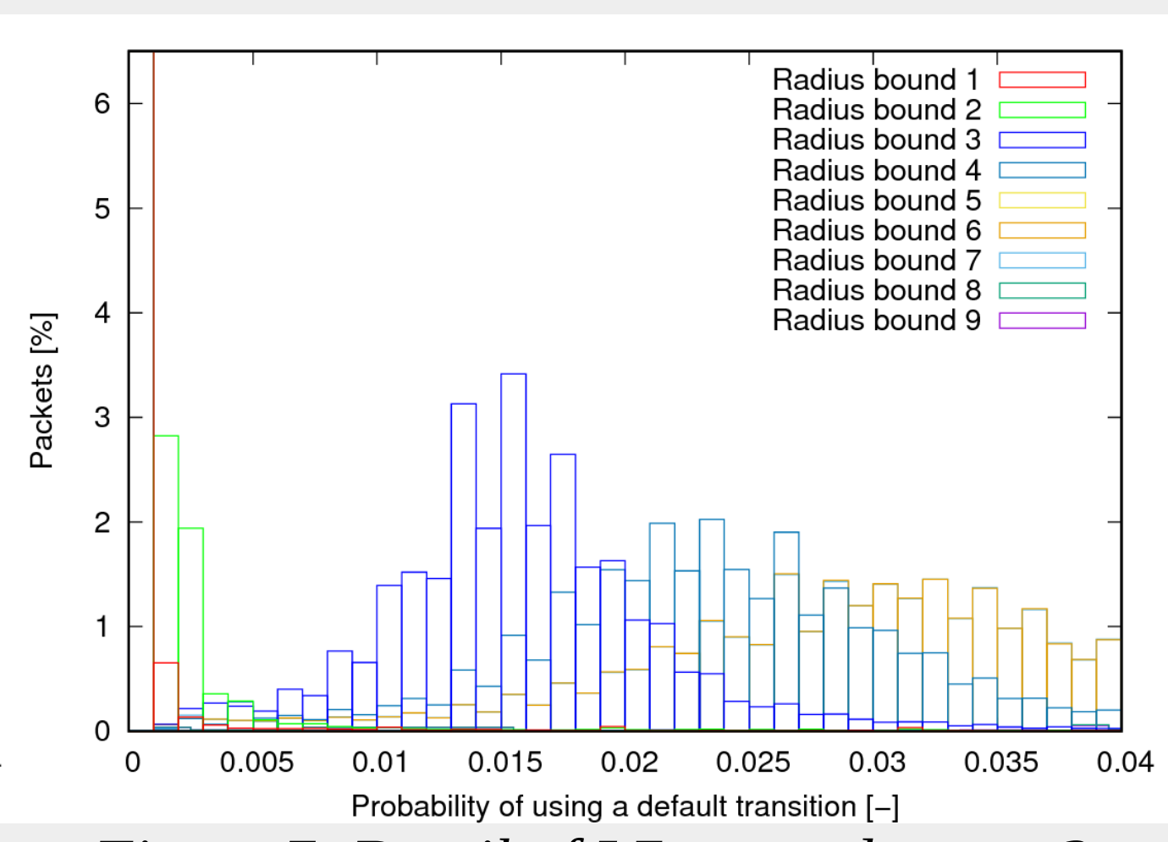

Figure 7: Detail of L7 great, data set 2

Architecture and Input Data Processing

The architecture uses parallel *pipelined automata* concept [6]. Instead of distributing data among multiple independent matching engines (MEs), which would require complex distribution logic and utilize a lot of chip's resources, state is passed between adjacent matching engines. An example of packet processing is shown on the figures below. The state from the matching engine ME^0, which processes the first part of the packet P1, is passed to the adjacent matching engine ME^1, which processes the second part of the packet P1. The processing of the packet P2 begins immediately with the processing of the packet P1 since corresponding matching engine ME^2 is not occupied.

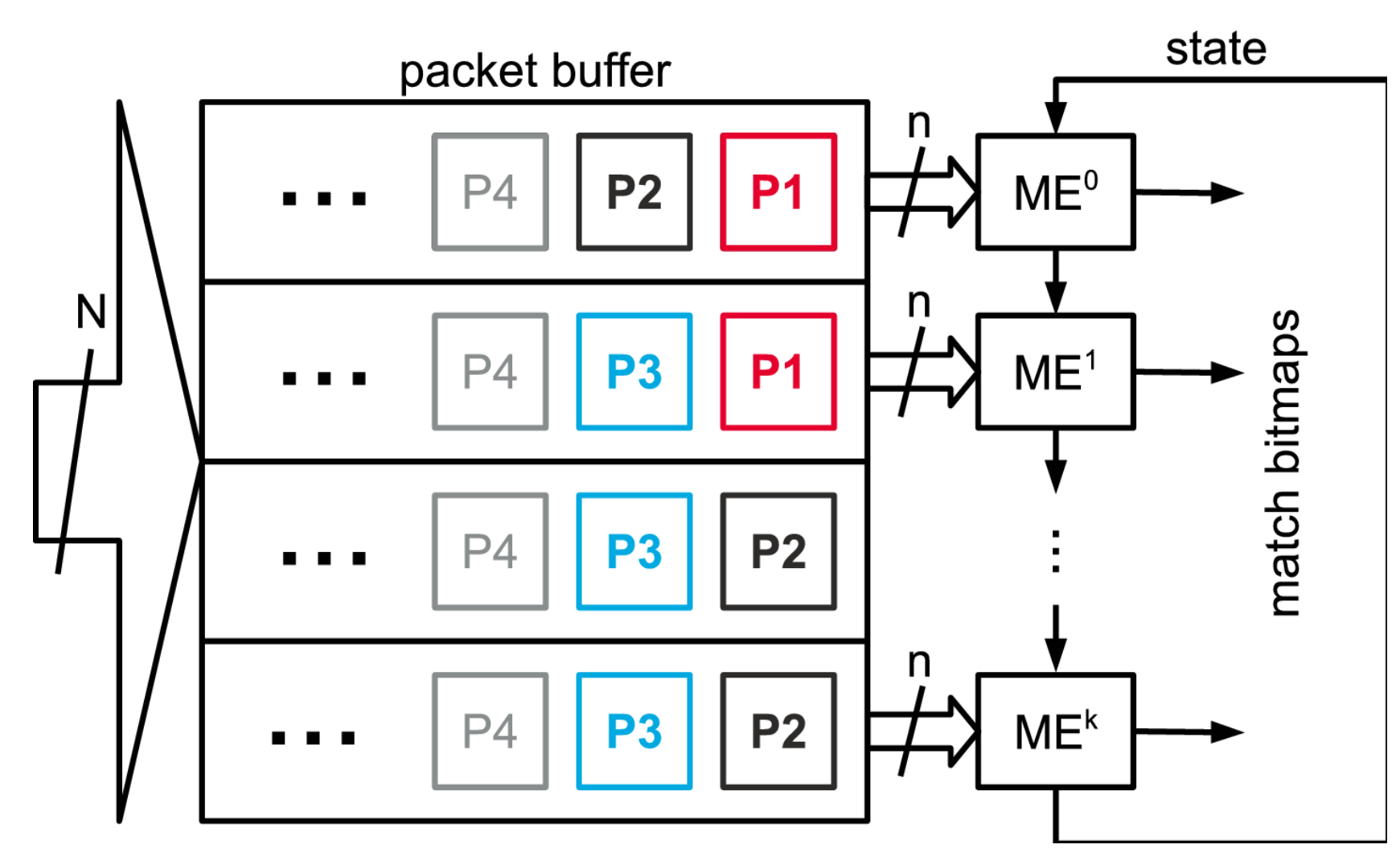

Figure 8: High-level architecture

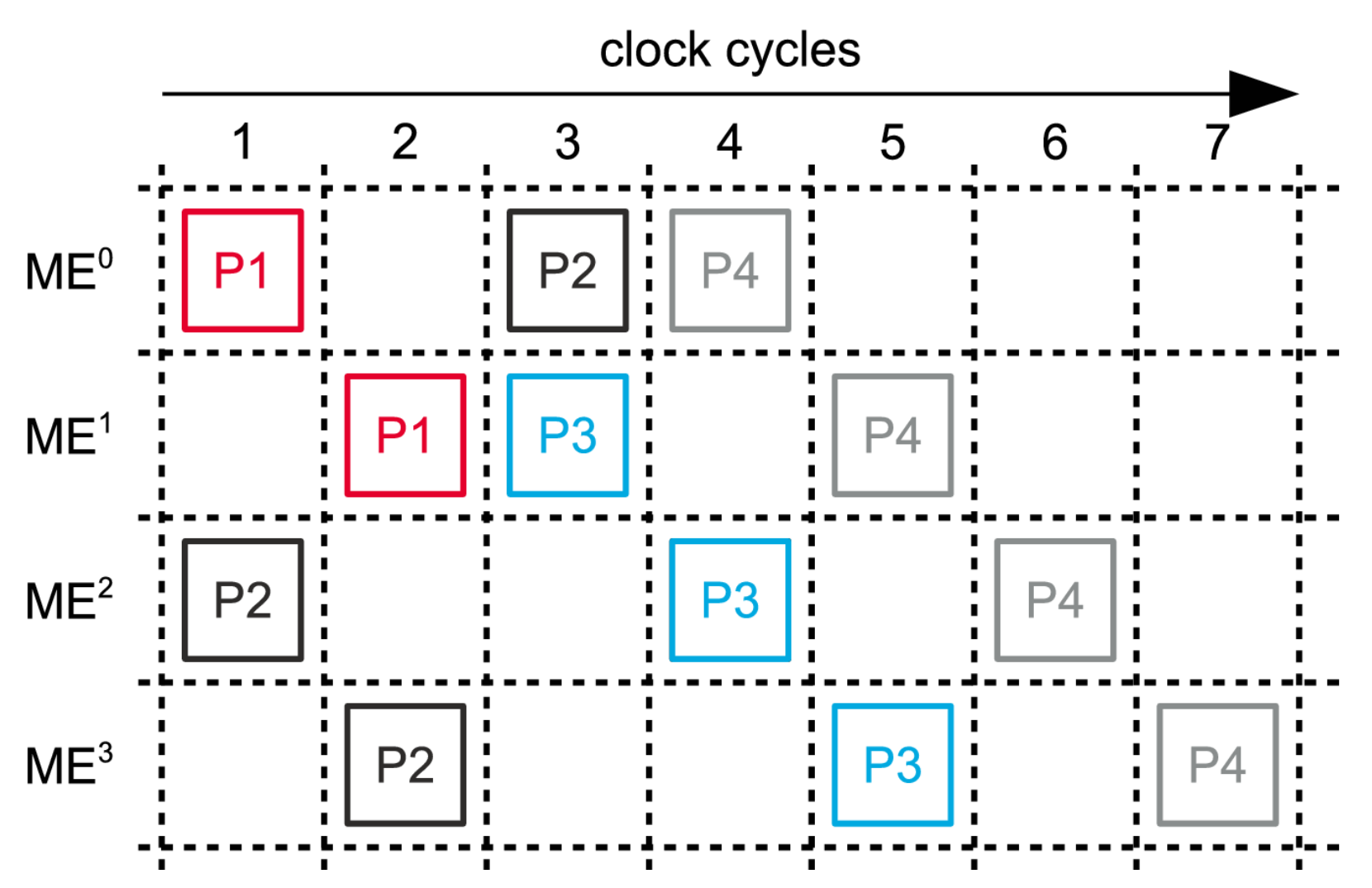

Figure 9: Processing of packets

Matching Engine

Matching engines use a pair of D^{2}FAs to process input symbols. D^{2}FA$_0$ is used to process standard transitions and D^{2}FA$_1$ to process default transitions. A single transition table is used for both automata since dual-port feature of FPGA block memories is used. FIFO$_{standard}$ is used to buffer standard transitions in case the following matching engine is not ready to process another input symbol. FIFO$_{default}$ is used to buffer default transitions in order to accept input symbols.

Since there are two D^{2}FAs in a ME, performing one default transition for each input symbol does not cause any delay in processing. In such case, neither FIFO$_{standard}$ nor FIFO$_{default}$ would be necessary. The FIFOs are present for cases when D^{2}FA radius is greater than one. The fullness of FIFOs depends on the data that are being processed. The capacity of the FIFOs depends on the frequency of using default transitions. The greater the D^{2}FA radius is, the greater the probability of using a default transition is. It is necessary to find a trade-off between the degree of reduction of the transition table and frequency of using default transitions.

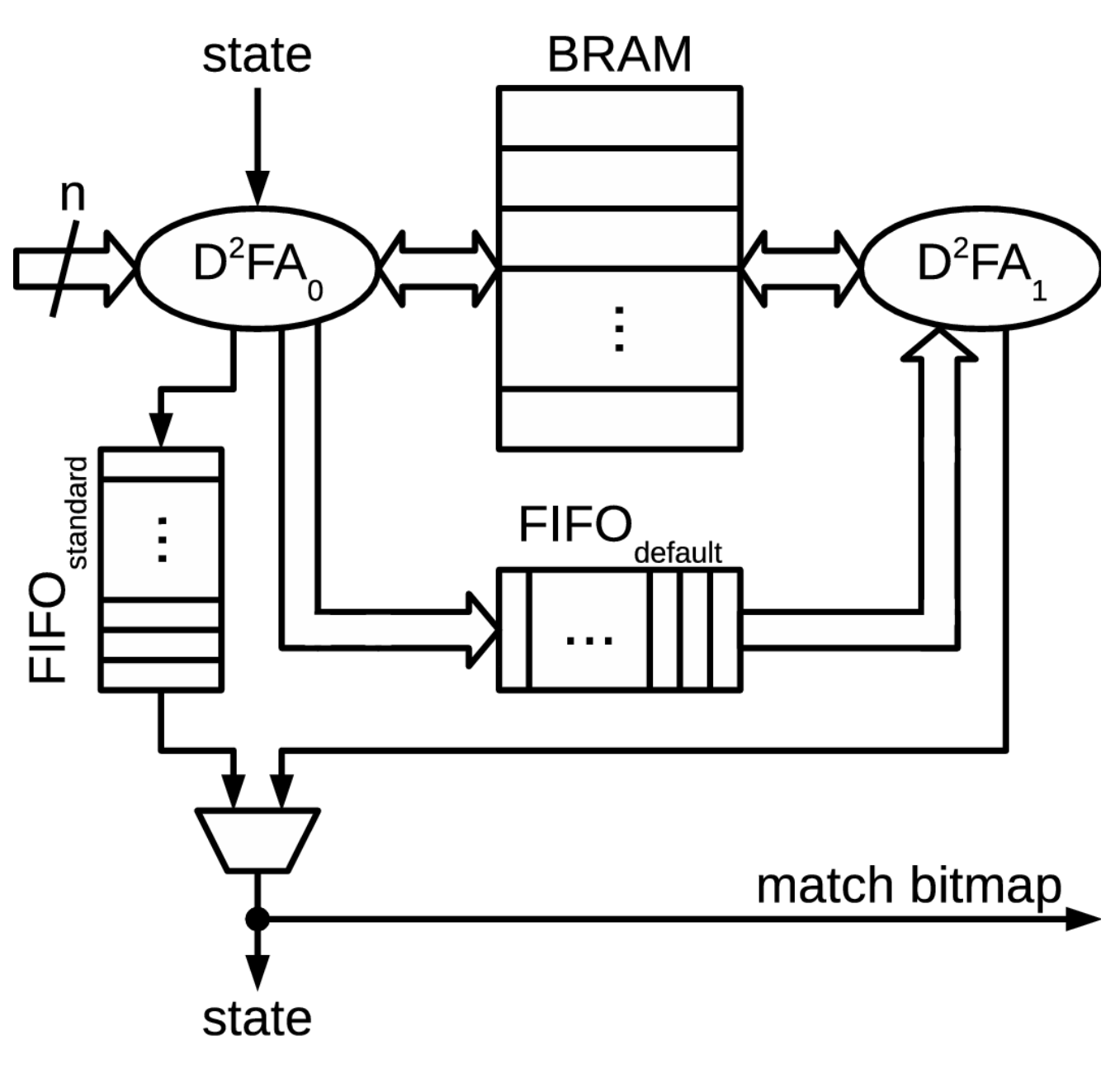

Figure 10: Architecture of a matching engine

Conclusion

The research addresses insufficient processing performance and flexibility of current architectures for pattern matching based on regular expressions. Resource utilization of the proposed architecture scales linearly with the width of input data bus and frequency remains the same. It achieves the throughput of 100 Gbps and 400 Gbps on current FPGA chips, Xilinx Virtex-7 and Xilinx Virtex UltraSCALE+, respectively. The use of memory-based automata brings the possibility to change the set of regular expressions without chip reconfiguration. Delayed Input DFAs significantly reduce the size of transition tables.

Resource utilization for two configurations is shown in the table below. They use a transition table with 8192 items. The width of items stored in FIFO$_{standard}$ is 13 bits (state) and in FIFO$_{default}$ 13 bits (state) + 8 bits (input symbol). The capacity of both FIFOs is three items.

Component	Throughput	BRAMs	LUTs	FFs
Single ME	N/A	9	201	124
64 MEs	**100 Gbps**	**576**	**12 864**	**7936**
256 MEs	**400 Gbps**	**2304**	**51 456**	**31 744**

Table 3: Resource utilization for sample configurations

References

[1] B. C. Brodie, D. E. Taylor, and R. K. Cytron, "A scalable architecture for high-throughput regular-expression pattern matching," SIGARCH Comput. Archit. News, vol. 34, no. 2, pp. 191–202, May 2006, ISSN 0163-5964.

[2] M. Becchi and P. Crowley, "Efficient regular expression evaluation: Theory to practice," in Proc. of ANCS'08, Nov. 2008, pp. 50–59.

[3] Y.-H. E. Yang and V. K. Prasanna, "High-performance and compact architecture for regular expression matching on fpga," IEEE Trans. Comput., vol. 61, no. 7, pp. 1013–1025, Jul. 2012, ISSN 0018-9340.

[4] Y.-H. E. Yang, W. Jiang, and V. K. Prasanna, "Compact architecture for high-throughput regular expression matching on fpga," in Proc. of ANCS'08, Nov. 2008, pp. 30–39.

[5] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," ACM SIGCOMM Comput. Commun. Rev., vol. 36, no. 4, pp. 339–350, Aug. 2006, ISSN 0146-4833.

[6] D. Matoušek, J. Kořenek, and V. Puš, "High-speed regular expression matching with pipelined automata," in Proc. of FPT'16, Dec. 2016, pp. 93–100.