Regular Expression Matching with Pipelined Delayed Input DFAs for High-speed Networks

Denis Matoušek
Juraj Kubiš
imatousekd@fit.vutbr.cz
xkubis15@stud.fit.vutbr.cz
Brno University of Technology
Faculty of Information Technology
Brno, Czech Republic

Jiří Matoušek
Jan Kořenek
imatousek@fit.vutbr.cz
korenek@fit.vutbr.cz
Brno University of Technology
Faculty of Information Technology
Centre of Excellence IT4Innovations
Brno, Czech Republic

ABSTRACT

Regular expression matching (RE matching) is a widely used operation in network security monitoring applications. With the speed of network links increasing to 100 Gbps and 400 Gbps, it is necessary to speed up packet processing and provide RE matching at such high speeds. Although many RE matching algorithms and architectures have been designed, none of them supports 100 Gbps throughput together with fast updates of an RE set. Therefore, this paper focuses on the design of a new hardware architecture that addresses both these requirements. The proposed architecture uses multiple highly memory-efficient Delayed Input DFAs (D^2FAs), which are organized to a processing pipeline. As all D^2FAs in the pipeline have only local communication, the proposed architecture is able to operate at high frequency even for a large number of parallel engines, which allows scaling throughput to hundreds of gigabits per second. The paper also analyses how to scale the number of engines and the capacity of buffers to achieve desired throughput. Using the parameters obtained while matching a sample RE set represented by a D^2FA in a real network traffic, the architecture can be tuned for wire-speed throughput of 400 Gbps.

KEYWORDS

Regular expression matching, 100 Gbps, 400 Gbps, Delayed Input DFA, Pipelined automata

ANCS '18, July 23–24, 2018, Ithaca, NY, USA

ACM ISBN 978-1-4503-5902-3/18/07…$15.00
https://doi.org/10.1145/3230718.3230730

ACM Reference Format:
Denis Matoušek, Juraj Kubiš, Jiří Matoušek, and Jan Kořenek. 2018. Regular Expression Matching with Pipelined Delayed Input DFAs for High-speed Networks. In *ANCS '18: Symposium on Architectures for Networking and Communications Systems, July 23–24, 2018, Ithaca, NY, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3230718.3230730

1 INTRODUCTION

The speed of network links is constantly increasing with the amount of network devices and the size of data centers. Large data centers started to use 100 Gb links to connect top-of-rack switches and call for 400 Gb and 1 Tb links. With the increasing speed of network links, a network security and monitoring analysis have to be accelerated to achieve wire-speed throughput. One of the most computationally intensive task is regular expression matching (RE matching). REs are widely used to identify application protocols, detect network security incidents, or diagnose specific network issues.

Current processors are not powerful enough to achieve 100 Gbps throughput. Throughput of one processor core is limited to less than one Gbps [2]. Matching speed can be increased to hundreds of Gbps only at the cost of a large number of processor cores. To achieve 100 Gbps throughput, Bro IDS [10] had to use five servers with 10 Gbps input lines and one switch. Although network processors have dedicated hardware units for RE matching, these units have usually throughput [2] significantly lower than the capacity of network links. To achieve 100 Gbps, 400 Gbps, or 1 Tbps throughput, it is more efficient to use hardware acceleration using FPGA technology.

Many hardware architectures utilize a non-deterministic finite automaton (NFA) to map an RE matching task to an FPGA [4, 5, 9]. A lot of researchers tried to optimize hardware architectures in order to reduce FPGA logic utilization and support more REs, because RE matching has been primarily

motivated by intrusion detection systems with a large set of REs. Throughput of the NFA architectures has been increased by multi-striding [1, 3] and spatial-stacking [11, 12]. Both these techniques increase the throughput by processing multiple input symbols at a single step.

The scaling of processing speed using multi-striding is limited because the number of transitions grows exponentially with the increasing number of simultaneously processed input symbols. The circuit frequency drops because of the increasing complexity of transition logic that limits the throughput in the order of tens of Gbps [1, 3, 7]. Spatial stacking features the frequency drop and limitation of throughput in the order of tens of Gbps as well [11, 12]. Both techniques are able to scale the throughput only to tens of Gbps. Therefore, pipelined automata architecture has been introduced to scale the processing speed to 100 Gbps or 400 Gbps throughput [7].

Mapping an NFA to an FPGA is highly efficient because of massively parallel processing provided by FPGA logic. For network attack mitigation, a fast change of an RE set is more important than the number of matched REs, because REs are determined by a network traffic analysis (e.g., DNS attack detection) and they have to be uploaded to a matching engine as soon as possible. The NFA-based architectures need FPGA reconfiguration, which is strongly technology dependent and slow in updating the RE set. Therefore, we have focused on hardware architectures which use a deterministic finite automaton (DFA) and utilize an on-chip memory to store DFA's transition table. The new RE set can then be uploaded only by changing the content of the memory. The problem of a DFA is a large size of the transition table. Therefore, Kumar et al. have introduced Delayed Input DFA (D^2FA) [6], which has significantly fewer transitions and needs less memory resources.

Delayed Input DFA (D^2FA) [6] reduces the size of the transition table by replacing specific groups of transitions of the original automaton with default transitions. The set of states does not change. A default transition is triggered if any other transition cannot be triggered for the current input symbol and the current state. It means that D^2FA may need to trigger multiple transitions to accept single input symbol, and thus accessing the transition table multiple times. As a result, the throughput of input data processing can drop.

Therefore, the paper deals with the design of a high-speed RE matching architecture, which (i) utilizes a memory-efficient D^2FA to allow a fast change of an RE set and (ii) is able to achieve 100 Gbps throughput. We have designed the parallel D^2FA architecture, which consists of multiple D^2FAs organized to a processing pipeline. As D^2FAs use only local communication, the architecture is able to operate at high frequency and to scale throughput to hundreds of Gbps.

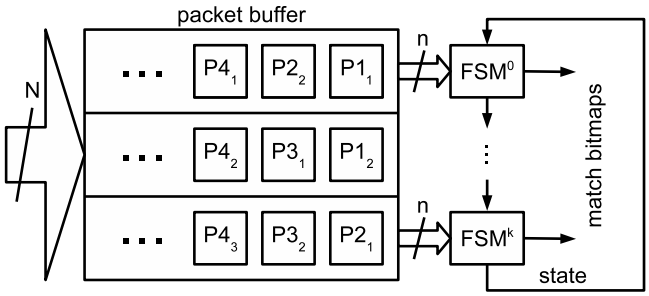

Figure 1: The architecture with k pipelined automata sharing a packet buffer, which stores three N-bit data words (columns) comprising packets P1-P4. Each word is divided into k independent n-bit data blocks (rows).

To achieve wire-speed throughput, the architecture introduces additional processing engines connected to the second port of a transition table memory, which deal with default transitions. The paper provides a detailed analysis of the architecture presenting information about buffers' sizes, the relation between the number of D^2FAs and target throughput, and FPGA logic utilization for 100 Gbps and 400 Gbps throughput.

The paper is structured as follows. After the brief introduction of the previously proposed architecture with pipelined automata in Section 2, the paper presents the proposed architecture and analyses the properties of utilized buffers (Section 3). In order to evaluate the architecture, Section 4 focuses on the properties of D^2FAs representing a real set of REs and on the process of RE matching in a real network traffic. The analyzed properties of the buffers as well as FPGA resource utilization of the proposed architecture are evaluated in Section 5. The paper is concluded in Section 6.

2 ARCHITECTURE WITH PIPELINED AUTOMATA

The architecture with pipelined automata was proposed by Matoušek et al. [7] in order to allow RE matching with throughput in the order of hundreds of Gbps. As shown in Figure 1, the architecture consists of k parallel FSMs organized to a processing pipeline and a shared packet buffer. N-bit input data words are stored in the buffer as k independent n-bit data blocks and each FSM can directly read the blocks stored on a corresponding buffer's row. Using such block of data and the current state of RE matching (i.e., a state from the previous FSM), the FSM can determine the next state of matching and pass it to the next FSM. If the next state is a final state, the FSM outputs a match bitmap encoding the matched RE(s).

RE matching of a packet starts in an FSM corresponding to the first block of its data and continues in successive FSMs

until the last block of packet's data is processed. Therefore, a pipeline comprising k FSMs is able to perform matching in k packets in parallel. However, to achieve the full utilization of the pipeline, the buffer has to contain at least k packets and the FSMs corresponding to the first block of packets' data must not be utilized for RE matching in another packet.

As demonstrated in [7], both throughput and resource utilization of the architecture scale linearly with the number of FSMs. Because each FSM communicates only locally (with the buffer and neighboring FSMs), increasing their number does not have a negative effect on operating frequency. The throughput can thus be scaled to hundreds of Gbps.

Even though the authors of [7] used NFAs, the architecture itself does not preclude the use of DFAs. Nevertheless, used automata have to have a constant matching rate of one input block of data per clock cycle.

3 PROPOSED ARCHITECTURE

When designing an RE matching architecture proposed in this paper, we addressed two main requirements: (i) ability to achieve 100 Gbps and 400 Gbps throughput and (ii) possibility to perform fast updates of an RE set. Although the architecture with pipelined automata [7] is able to achieve desired throughput and architectures based on D^{2}FA [6] are able to perform fast updates, none of these approaches is able to fulfill both requirements at the same time. Therefore, we propose a new RE matching architecture that is inspired by the pipelined-automata approach and utilizes D^{2}FAs. We have already briefly sketched this architecture in [8].

3.1 Pipelined D^{2}FAs

The main issue that an architecture with pipelined D^{2}FAs has to address is triggering default transitions without delaying the whole pipeline, which would negatively affect architecture's throughput. To deal with this issue, instead of k pipelined FSMs, we propose to use k pipelined matching engines (MEs), each of which comprises two automata—D^{2}FA$_0$ for standard transitions and D^{2}FA$_1$ for default transitions. The architecture of a single ME is shown in Figure 2. Its core consists of two D^{2}FAs sharing a BlockRAM (BRAM) that stores their common transition table. While the basic function implemented by these automata corresponds to original D^{2}FA, each of them implements some modifications that allow to use it in the proposed architecture.

D^{2}FA$_0$ triggers just a single transition defined by a current state and input symbol (i.e., an n-bit data block from the buffer) and then decides on further processing in the current ME. If D^{2}FA$_0$ triggered a standard transition, the new state is sent towards the ME's output. But in case of a default transition, a (*new state*, *input symbol*) tuple is submitted for further processing by D^{2}FA$_1$ via FIFO$_{\text{def}}$, which has the

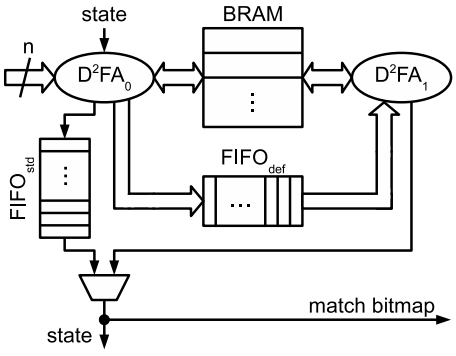

Figure 2: Matching engine architecture

capacity of s tuples. Regardless of the transition, D^{2}FA$_0$ will process the next input symbol in the next clock cycle. D^{2}FA$_1$ is responsible for triggering sequences of default transitions. It reads the *new state* and *input symbol* from FIFO$_{\text{def}}$ and repeatedly triggers a corresponding transition until the *input symbol* is accepted (i.e., until the transition is not default). Then D^{2}FA$_1$ sends the new state to the ME's output and starts processing the next tuple from FIFO$_{\text{def}}$.

Since D^{2}FA$_1$ may trigger multiple default transitions per input symbol, its output is always sent directly to the ME's output. To solve possible output collisions, the output of D^{2}FA$_0$ is buffered in FIFO$_{\text{std}}$. The worst-case requirement on its capacity applies when FIFO$_{\text{def}}$ contains s tuples and there is a long sequence of symbols triggering a standard transition on the ME's input. In such situation, each of s tuples will cause a collision. Thus, to prevent the overflow of FIFO$_{\text{std}}$, its capacity has to be at least s items.

3.2 Probability of FIFO$_{\text{def}}$ Overflow

The ME's architecture guarantees no delays when D^{2}FAs with radius (i.e., the maximum length of a default path) $r \leq 1$ are used. This is because accepting a symbol along a default path of length r takes $1 + r$ clock cycles (1 in D^{2}FA$_0$ and r in D^{2}FA$_1$). Therefore, processing in D^{2}FA$_1$ is as fast or faster than in D^{2}FA$_0$ when $r \leq 1$, but it might be slower when $r > 1$. An unlimited sequence of input symbols triggering a default path longer than 1 will thus require FIFO$_{\text{def}}$ of infinite capacity. However, if only d out of a sequence of $l \geq d \cdot r$ input symbols trigger a default transition, the capacity of FIFO$_{\text{def}}$ can be bounded. Clearly, if accepting d symbols along a default path of length r requires $d \cdot r$ clock cycles in D^{2}FA$_1$, then all d tuples written into FIFO$_{\text{def}}$ during $l \geq d \cdot r$ clock cycles will also be read from it during this interval.

Since the actual value of parameter d depends on an input traffic and the structure of used D^{2}FAs, we will now attempt to determine its maximum allowed value d_{max} for given $r > 1$ and constant s. Each of d_{max} symbols has to pass

through FIFO_{def}. In the worst case, D^2FA_0 writes a new tuple into FIFO_{def} every clock cycle and D^2FA_1 reads a tuple every r-th clock cycle. Thus, the number of written tuples form an infinite geometric sequence starting from s with common ratio $1/r$ (while writing s tuples into FIFO_{def}, s/r tuples are read, making space for new tuples). The sum of the corresponding series expressed in (1) is equal to d_{max}. The use of the floor function corresponds to our aim of finding the maximum value of d for which FIFO_{def} does not overflow.

$$d_{max} = \left\lfloor s \cdot \sum_{i=0}^{\infty} (1/r)^i \right\rfloor = \lfloor (s \cdot r) / (r-1) \rfloor \qquad (1)$$

Once we are able to compute the value of d_{max}, we can determine the probability P_{ovf} of FIFO_{def} overflow. We will again analyze the worst case, which is represented by the shortest sequence of symbols that allows a bounded capacity of FIFO_{def} (i.e., $l = d_{max} \cdot r$). If it contains at most d_{max} arbitrarily ordered symbols triggering a default transition, FIFO_{def} will not overflow. P_{ovf} can thus be computed as the complement of probability P_{not_ovf} that FIFO_{def} will not overflow, as shown in (2) where P_{def} and P_{std} are the probability of default and standard transition's occurrence, respectively. Using (2) we can determine how likely it is that FIFO_{def} of size s will overflow when the ME utilizes D^2FAs with radius r and input symbols trigger a default transition with probability P_{def} ($P_{std} = 1 - P_{def}$). This can facilitate the selection of appropriate FIFO_{def} capacity because its overflow would result in a decrease of architecture's throughput.

$$P_{ovf} = 1 - P_{not\_ovf} = 1 - \sum_{d=0}^{d_{max}} \binom{l}{d} \cdot P_{def}^{d} \cdot P_{std}^{l-d} \qquad (2)$$

3.3 Probability of Packet Buffer Overflow

Since each ME can work with $s + 1$ symbols, in the whole architecture there might be up to $k \cdot (s + 1)$ different packets, whose symbols are either waiting for processing in FIFO_{def} or are currently under processing in D^2FA_0. Each of these packets has to be stored in the packet buffer, thus its capacity could be set to $S = k \cdot (s + 1)$. However, FIFO_{def} stores only symbols triggering a default transition. Therefore, even smaller packet buffer might usually be sufficient.

To determine the probability of packet buffer overflow, we will transform this problem into the FIFO_{def} overflow problem. First of all, we will replace k independent instances of FIFO_{def} by FIFO_{def} shared by all k MEs. It can be shown that this replacement is transparent if shared FIFO_{def} supports k-times faster writing and reading of tuples. Shared FIFO_{def} represents all parts of the MEs that can store symbols triggering a default transition, thus its capacity s determines the number of packets that have to be stored in the buffer for these parts of the architecture.

Because shared FIFO_{def} supports k-time faster reading and writing, the number of written and read tuples is the same as in case of non-shared FIFO_{def}, which allows us to compute both d_{max} and P_{ovf} for shared FIFO_{def} using (1) and (2), respectively. The only difference lies in the computation of the length l of symbols' sequence. Its value is defined as $l = k \cdot \lceil d_{max}/k \rceil \cdot r$, where $\lceil d_{max}/k \rceil$ represents the number of r-clock-cycle steps required for reading all d_{max} tuples from shared FIFO_{def}. During this time, the whole architecture can process a total of $k \cdot \lceil d_{max}/k \rceil \cdot r$ symbols.

Now we can finally determine the probability of packet buffer overflow. Total capacity S of the buffer required for the architecture with k MEs consists of (i) k packets, whose symbols are currently under processing in D^2FA_0 of MEs and (ii) s packets, whose symbols are currently waiting for processing by D^2FA_1 in shared FIFO_{def}. Since the buffer has to be able to store at least k packets, we can constrain its capacity to $S \geq k$. This allows us to focus only on the remaining s packets and define probability that the buffer with capacity $S = k + s$ will overflow as probability that shared FIFO_{def} with capacity s will overflow, which can be computed using (2). As for non-shared FIFO_{def}, we can use (2) to facilitate the selection of appropriate buffer's capacity.

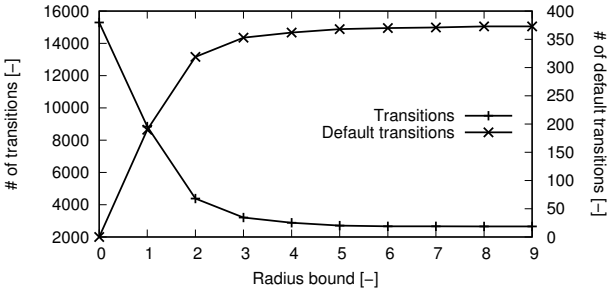

Figure 3: The number of transitions and default transitions for L7 selected

4 ANALYSIS OF D^2FA

In order to evaluate the effectiveness of using D^2FA technique, two analyses were carried out. The first one inspects the influence of D^2FA radius on the size of the transition table. The other analysis determines the frequency of triggering a default transition while processing real data. It reveals how often the transition table needs to be accessed multiple times to accept a single input symbol.

Both analyses are based on automaton derived from the set of REs used to classify network traffic. It is denoted "L7 selected" and contains six REs for identification of application protocols.

The graph in Figure 3 shows the results of the first analysis. Each data point represents the number of transitions on the left y axis and the number of default transitions on the right y axis for D^2FA with corresponding radius on the x axis.

In the second analysis, a sample of network traffic, which contains 2 699 525 packets, captured on a real network of a big ISP was used. The histogram in Figure 4 shows the results of the second analysis. It represents the probability distribution of triggering a default transition over the whole data set for all inspected D^{2}FA radii. Each box represents the packets that triggered a default transition with probability on the x axis (boxes are centered).

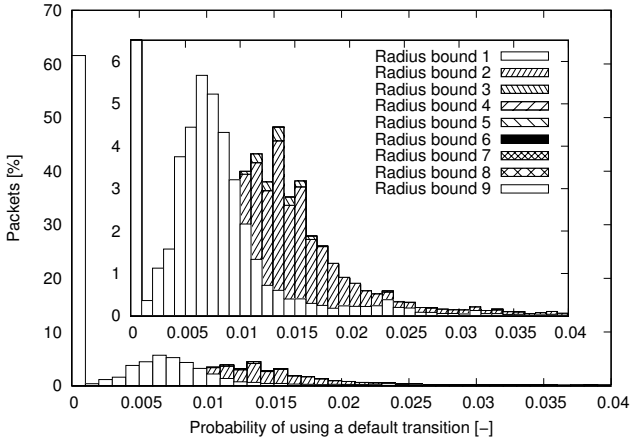

Figure 4: Probability distribution of triggering a default transition with detail for L7 selected

The histogram in Figure 4 shows that the majority of the packets (about 60%) do not trigger any default transition. In the majority of such cases, it is caused by absence of application layer in TCP or ICMP packets.

In order to determine an optimal D^{2}FA radius for the proposed architecture, it is necessary to take into account the results of both analyses. The first analysis revealed logarithmic drop of the number of transitions with increasing D^{2}FA radius. The most distinct drop is seen at the radii one (57.8% of the size of the original DFA) and two (28.7%). The second analysis revealed that with rising D^{2}FA radius, the number of packets with higher probability of triggering a default transition increases, which can be seen from the distribution shift to the right-hand side of the histogram. Thus, the smaller D^{2}FA diameter is, the smaller the probability is. We chose D^{2}FA radius of two for the design of the architecture to reduce the size of the transition table and assure reasonably small probability of triggering a default transition in comparison to bigger D^{2}FA radii. In the following text, we will use the probability of triggering a default transition of 1.5334%. It is computed based on the L7 selected D^{2}FA with radius of two, which appeared to be the worst case.

Table 1: The probability of FIFO$_{\text{def}}$ overflow and the theoretical throughput for various FIFO$_{\text{def}}$ capacity

s	1	2	3	∞
P_{ovf}^{64} [%]	0.0912	0.0003	0.0000	0.0000
P_{ovf}^{256} [%]	0.3650	0.0012	0.0000	0.0000
T^{64} [Gbps]	99.9088	99.9997	100.0000	100.0000
T^{256} [Gbps]	398.5402	399.9953	400.0000	400.0000

5 RESULTS

We will now evaluate the proposed architecture comprising $k = 64$ and $k = 256$ MEs, each of which accepts input symbols of width $n = 8$ bits and runs at $f = 195.3125$ MHz. Such settings should allow to achieve desired throughput $T = 100$ Gbps and $T = 400$ Gbps, respectively. In the evaluation, we will consider D^{2}FAs with radius $r = 2$ and the probability of a symbol triggering a default transition $P_{def} = 1.5334\%$.

5.1 FIFO$_{\text{def}}$

First, we quantify the probability of FIFO$_{\text{def}}$ overflow, whose analytical solution was presented in Section 3.2. The results for 64 (P_{ovf}^{64}) and 256 (P_{ovf}^{256}) MEs are presented in Table 1. The values in the table represent probability that FIFO$_{\text{def}}$ will overflow in at least one of k MEs. Unsurprisingly, the probability is the highest when the architecture comprises $k = 256$ MEs, in which FIFO$_{\text{def}}$ has a capacity of $s = 1$. However, even in this worst-case scenario, the probability is lower than 0.4%. Moreover, for all evaluated values of k, the probability of FIFO$_{\text{def}}$ overflow when $s = 3$ is practically the same as of FIFO$_{\text{def}}$ with infinite capacity.

Even though the probability of FIFO$_{\text{def}}$ overflow is very low, it might happen. Nevertheless, it affects only throughput and has no effect on RE matching result's correctness (it just stalls the whole pipeline). Therefore, Table 1 also shows theoretical throughput for 64 (T^{64}) and 256 (T^{256}) MEs and various FIFO$_{\text{def}}$ capacity computed using (3). Similarly to the upper part of the table, theoretical throughput T^k is very close to desired throughput T even in the worst case and FIFO$_{\text{def}}$ with a capacity of $s = 3$ ensures practically the same theoretical throughput as FIFO$_{\text{def}}$ of infinite capacity.

$$T^k = T \cdot P_{ovf}^k \tag{3}$$

5.2 Packet Buffer

Using the analytical solution presented in Section 3.3 and Equation (3), we now quantify the probability of packet buffer

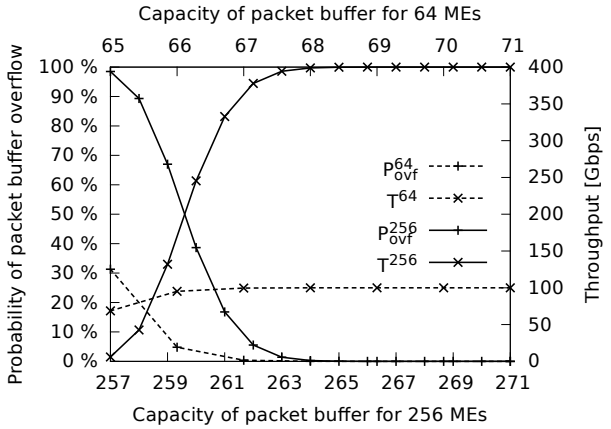

Figure 5: The probability of packet buffer overflow and theoretical throughput of the architecture with 64 and 256 MEs for various packet buffer capacity.

overflow and theoretical architecture's throughput for various capacity of the buffer. The results are presented in Figure 5, which shows the probability of buffer overflow (P^k_{ovf}) and theoretical throughput (T^k) for $k = 64$ and $k = 256$ MEs.

The figure confirms expected behavior. A decrease of P^k_{ovf} with an increasing capacity of the buffer, which is natural, has a positive effect on T^k. It is also not surprising that the actual value of P^k_{ovf} and T^k as well as the rate of their decrease and increase are not the same for architectures comprising 64 and 256 MEs. However, the main finding illustrated in Figure 5 is similar for both: to retain desired throughput, it is sufficient to store only a few more than k packets in the buffer. Specifically, a capacity of $S = 71$ and $S = 271$ is sufficient for retaining the throughput of 100 Gbps and 400 Gbps, respectively. This means that due to default transitions, the capacity of the buffer has to be increased by 10.94% to support 100 Gbps throughput and only by 5.86% to support 400 Gbps throughput.

5.3 FPGA Resource Utilization

In order to evaluate the architecture from the point of view of resource utilization, the architecture of an ME depicted in Figure 2 and L7 selected RE set with D^{2}FA radius two are considered. For the sake of brevity, the description of the process of transformation of an RE set into a transition table is omitted. The transformation resulted in 4089 items, each 12 bits wide since $2^{12} \geq 4089$. Each item of the transition table and FIFO_{def} contains a next state number (12 bits) and an input symbol (8 bits) that gives 20 bits in total. Each item of default transition table and FIFO_{std} contains only a 12-bit next state number. The capacity of both FIFOs is set to three items based on the analysis carried out in Section 5.1.

Table 2: Resource utilization (L7 selected, D^{2}FA radius two)

Component	BRAMs	LUTs	FFs
D^{2}FA pair	3	142	66
$\text{FIFO}_{\text{def}}/\text{FIFO}_{\text{std}}$/MUX	0/0/0	28/20/12	14/14/0
Single ME	3	202	94
64 MEs	**192**	**12 928**	**6016**
256 MEs	**768**	**51 712**	**24 064**

Synthesis results summed up in Table 2 show resource utilization of individual components and of the whole parallel pipeline with 64 MEs (to achieve 100 Gbps throughput) and 256 MEs (to achieve 400 Gbps throughput). Note that 64 parallel MEs fit into Xilinx Virtex-7 VH580T chip (20.4 % of block memories are used) and 256 MEs fit into Xilinx Virtex UltraScale+ VU13P chip (28.6 % of block memories are used). Since the circuit is deeply pipelined using synchronous block memories and FIFOs in all stages, the required frequency of 200 MHz is met.

6 CONCLUSION

This paper introduces a new RE matching architecture that supports throughput in the order of hundreds of Gbps and fast updates of an RE set. The architecture utilizes multiple parallel D^{2}FAs organized to a processing pipeline, which allows to scale its throughput linearly with the number of parallel pipeline stages. Moreover, because of only local communication between neighboring stages, scaling does not affect architecture's operating frequency.

Apart from describing the architecture, the paper also analyses the properties of utilized buffers and the properties of D^{2}FAs, which the architecture is based on. The results of these analyses are used in the evaluation of the architecture with respect to its throughput and utilized FPGA resources.

The architecture able to achieve wire-speed throughput of 100 Gbps and 400 Gbps comprises 64 and 256 parallel pipeline stages, respectively. While the 400 Gbps architecture fits into modern Virtex UltraScale+ FPGAs (it utilizes 51 712 LUTs, 24 064 FFs, and 768 BRAMs), the 100 Gbps architecture can be implemented even in older Virtex-7 chips (it utilizes 12 928 LUTs, 6016 FFs, and 192 BRAMs).

ACKNOWLEDGMENT

This work was supported by The Ministry of Education, Youth and Sports of the Czech Republic from the National Programme of Sustainability (NPU II); project IT4Innovations excellence in science - LQ1602. It was also supported by Brno University of Technology from grant no. FIT-S17-3994.

REFERENCES

[1] Michela Becchi and Patrick Crowley. 2008. Efficient Regular Expression Evaluation: Theory to Practice. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '08)*. ACM, New York, NY, USA, 50–59. https://doi.org/10.1145/1477942.1477950

[2] Michela Becchi, Charlie Wiseman, and Patrick Crowley. 2009. Evaluating Regular Expression Matching Engines on Network and General Purpose Processors. In *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '09)*. ACM, New York, NY, USA, 30–39. https://doi.org/10.1145/1882486.1882495

[3] Benjamin C. Brodie, David Edward Taylor, and Ron K. Cytron. 2006. A Scalable Architecture For High-Throughput Regular-Expression Pattern Matching. *SIGARCH Comput. Archit. News* 34, 2 (May 2006), 191–202. https://doi.org/10.1145/1150019.1136500

[4] Christopher R. Clark and David E. Schimmel. 2003. Efficient Reconfigurable Logic Circuits for Matching Complex Network Intrusion Detection Patterns. In *Field Programmable Logic and Application (FPL '03)*, Peter Y. K. Cheung and George A. Constantinides (Eds.). Springer, Berlin, Heidelberg, Germany, 956–959. https://doi.org/10.1007/978-3-540-45234-8_94

[5] Christopher R. Clark and David E. Schimmel. 2004. Scalable Pattern Matching for High Speed Networks. In *Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04)*. IEEE Computer Society, Washington, DC, USA, 249–257. https://doi.org/10.1109/FCCM.2004.50

[6] Sailesh Kumar, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, and Jonathan Turner. 2006. Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection. *SIGCOMM Comput. Commun. Rev.* 36, 4 (Aug. 2006), 339–350. https://doi.org/10.1145/1151659.1159952

[7] Denis Matoušek, Jan Kořenek, and Viktor Puš. 2016. High-speed Regular Expression Matching with Pipelined Automata. In *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 93–100. https://doi.org/10.1109/FPT.2016.7929431

[8] Denis Matoušek, Jiří Matoušek, and Jan Kořenek. 2018. High-speed Regular Expression Matching with Pipelined Memory-based Automata. In *Proceedings of the 26th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM '18)*. IEEE Computer Society, Washington, DC, USA, 214–214. https://doi.org/10.1109/FCCM.2018.00048

[9] Reetinder Sidhu and Viktor K. Prasanna. 2001. Fast Regular Expression Matching Using FPGAs. In *Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '01)*. IEEE Computer Society, Washington, DC, USA, 227–238. https://doi.org/10.1109/FCCM.2001.22

[10] Vincent Stoffer, Aashish Sharma, and Jay Krous. 2015. *100G Intrusion Detection*. Technical Report. https://commons.lbl.gov/display/cpp/100G+Intrusion+Detection

[11] Yi-Hua Edward Yang, Weirong Jiang, and Viktor K. Prasanna. 2008. Compact Architecture for High-throughput Regular Expression Matching on FPGA. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '08)*. ACM, New York, NY, USA, 30–39. https://doi.org/10.1145/1477942.1477948

[12] Yi-Hua Edward Yang and Viktor K. Prasanna. 2012. High-Performance and Compact Architecture for Regular Expression Matching on FPGA. *IEEE Trans. Comput.* 61, 7 (July 2012), 1013–1025. https://doi.org/10.1109/TC.2011.129