# Multi-Stride NFA-Split Architecture for Regular Expression Matching Using FPGA

Vlastimil Košař and Jan Kořenek

IT4Innovations Centre of Excellence
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, Brno, Czech Republic
{ikosar, korenek}@fit.vutbr.cz

**Abstract.** Regular expression matching is a time critical operation for any network security system. The NFA-Split is an efficient hardware architecture to match a large set of regular expressions at multigigabit speed with efficient FPGA logic utilization. Unfortunately, the matching speed is limited by processing only single byte in one clock cycle. Therefore, we propose new multi-stride NFA-Split architecture, which increases achievable throughput by processing multiple bytes per clock cycle. Moreover, we investigate efficiency of mapping DU to the FPGA logic and propose new optimizations of mapping NFA-Split architecture to the FPGA. These optimizations are able to reduce up to 71.85 % of FPGA LUTs and up to 94.18 % of BlockRAMs.

## 1 Introduction

Intrusion Detection Systems (IDS) [1–3] use Regular Expressions (RE) to describe worms, viruses and network attacks. Usually, thousands of REs have to be matched in the network traffic. Current processors don't provide enough processing power for wire-speed RE matching at multigigabit speed [4]. Therefore, many hardware architectures have been designed to accelerate this time critical operation [5–7]. Usually, hardware architectures are able to achieve high speed only for small sets of REs due to the limited FPGA resources or capacity of available memory. Hardware architectures based on Deterministic Finite Automata (DFA) [4, 8, 5] are limited by the size and speed of the memory, because the determinisation of automaton significantly increases the number of states and size of the transition table. Architectures based on Nondeterministic Finite Automata (NFA) [9, 6, 10] are limited by the size and capacity of FPGA chips since the transition table is mapped directly into the FPGA logic.

With the growing amount of attacks, worms and viruses, security systems have to match more and more REs. It means that the amount of required FPGA logic increases not only due to the increasing speed of network links, but also due to the growth of RE sets. Therefore, it is important to reduce the amount of consumed FPGA resources to support more REs. A lot of work has been done in this direction. FPGA resources have been decreased by a shared character

decoder [6], infix and suffix sharing [7], by better representation of the counting constraint [10] and by the NFA reduction techniques [11, 12]. High reduction has been achieved by NFA-Split architecture [13, 14], which splits NFA to deterministic and nondeterministic parts in order to optimize mapping to the FPGA. The NFA-Split architecture reduces FPGA logic at the cost of on-chip memory (BlockRAMs). As some kinds of REs can increase the size of transition table and require a lot of on-chip memory, we have recently introduced optimization [15], which uses a $k$-inner alphabet to reduce on-chip memory requirements.

NFA-Split is designed to process only one byte of input stream in every clock cycle. The matching speed can be increased by increasing the operating frequency, but for the FPGA the frequency is limited to hundreds of megahertz. Consequently, the NFA-Split architecture cannot scale the throughput to tens of gigabits. To achieve higher matching speed, it is necessary to improve the architecture to support multi-stride automaton and to accept multiple bytes per clock cycle. Therefore, we propose an NFA-Split architecture for multi-stride automata, which requires significantly less FPGA resources in comparison to other multi-stride architectures. For the largest Snort backdoor module, the proposed architecture was able to reduce the amount of FPGA lookup tables (LUTs) by 58 %. Moreover, we investigate the efficiency of mapping the DU to the FPGA logic and propose new optimizations of mapping deterministic and nondeterministic parts of a NFA to FPGA. Both optimizations are able to reduce up to 71.85 % of FPGA LUTs and up to 94.18 % of FPGA BlockRAMs.

The paper consists of six sections. Brief summary of the related work is described after the introduction. Then NFA-Split architecture for multi-stride automata is introduced in the third section. Optimizations of the NFA-Split architecture are described in the section four, experimental results are presented in the section five and conclusions in section six.

## 2 Related Work

One of the first methods of mapping the NFA to the FPGA was published by Sidhu and Prasanna [9]. A dedicated character decoder was assigned to each transition. Clark and Schimmel improved the architecture by shared decoders of input characters and sharing of prefixes [16]. Lin et al. created an architecture for sharing infixes and suffixes, but did not specify a particular algorithm to find them [7]. Sourdis et al. published [10] an architecture that allows sharing of character classes, static subpatterns and introduced components for efficient mapping of constrained repetitions to the FPGA.

Current efficient solutions for regular expression matching on common processors (CPUs), graphics processing units (GPUs) and application-specific integrated circuits (ASICs) are based on NFAs. An NFA based architecture for ASICs was recently introduced in [17]. It is capable processing input data at 1 Gbps. A solution for GPUs capable of processing rule-sets of arbitrary complexity and size is based on NFA. [18]. However, this architecture has unpredictable performance (950 Mbps - 3.5 Gbps for 8-stride NFA). A NFA based solution for

CPUs was introduced in [19]. It provides considerable best-case performance on high-end CPUs (2 - 9.3 Gbps on two Intel Xeon X5680 CPUs with total of 12 cores running on 3.33 GHz).

Algorithms based on DFA seek various ways to limit the impact of state explosion of the memory needed to store the transition table. Delay DFA introduced in [20] extended the DFA by default transitions. The default transitions limited a redundancy caused by similarity of output transitions of different states. Content Addressed Delayed Input DFA [5] improved the throughput of the previous methodology by content addressing. The concept of Delay DFA is further refined in [8]. Extended Finite Automaton [21] extends the DFA by a finite set of variables and instructions for their manipulations.

Hybrid methods combine DFA and NFA parts to use the best of their respective properties. Becchi introduced hybrid architecture [22] that splits the automaton to a head-DFA and tail-NFAs. The head-DFA contains frequently used states, while the tail-NFA contains the others. NFA-Split architecture [13, 14] is designed for FPGA technology. It utilizes properties of REs in IDS systems and significantly reduces FPGA resources in comparison to other NFA based architectures. As the NFA created from REs has usually only a small subset of states that can be active at the same time, the architecture splits the NFA into several DFA parts and one NFA part. The DFA parts contain only states that cannot be active at once. Therefore, these parts can be efficiently implemented as a standard DFA in a Deterministic Unit (DU) with binary encoded states. States in the NFA part are mapped to Nondeterministic Unit (NU), where every state is represented by a dedicated logic (register and next state logic). Therefore, new state value can be computed in parallel in every clock cycle.

We have improved the NFA-Split architecture by $k$-inner alphabet in [15], which decreases the on-chip memory requirements for matching RE with character classes. Character classes can be specified in REs to define set of characters. In the automaton, the transition on character class has to be represented by a set of transitions on individual characters. This can significantly increase the number of transitions and thus the size of a memory to store the automaton. The $k$-inner alphabet allows representing a character class by only one internal symbol and only one transition. Thus, less memory is needed to store the automaton.

## 3 NFA-Split Architecture for Multi-Stride Automata

Even while the NFA-Split architecture is highly optimized, requires only reasonable memory and provides high matching speed, it still doesn't support multi-stride automata to match multiple characters in a single clock cycle. Consequently, it cannot scale its matching speed well. Therefore, we propose an extension of the NFA-Split architecture to support multi-stride automata. Moreover, we provide optimizations of mapping the DFA and NFA parts to further reduce the FPGA logic in order to map larger set of RE to the FPGA.

We propose the necessary modifications of the NFA-Split architecture to support multi-stride automata (SNFA-Split) and to make the matching speed scalable to tens of gigabits. The method of creating the multi-stride automaton is based on performing all consecutive transitions from one state to all states reachable in the number of steps equal to the desired stride. Symbols along these consecutive transitions are merged into one multi-stride symbol. Multi-stride automata have usually more transitions due to the larger number of symbols [23, 11].

**Algorithm 1**: Compute all pairs of simultaneously active states. The algorithm uses intersection operation $\cap_k$.

**Input**: NFA $M = (Q, \Sigma, \delta, s, F)$
**Output**: Set of pairs of simultaneously active states
$concurrent = \{(p, q) | p, q \in Q, p \neq q)\}$

```
1  normalize(q1, q2) = (q1 < q2) ? ((q1, q2) : ((q2, q1);
2  concurrent = {(s, s)};
3  workplace = {(s, s)};
4  while ∃(q1, q2) ∈ workplace do
5      workplace = workplace\{(q1, q2)};
6      foreach q3 ∈ δ(q1, a) do
7          foreach q4 ∈ δ(q2, b) do
8              if a ∩k b ≠ (∅, ∅, ..., ∅) then
9                  if ((q5, q6) = normalize(q3, q4)) ∉ concurrent then
10                     concurrent = concurrent ∪ {(q5, q6)};
11                     workplace = workplace ∪ {(q5, q6)};
12 return concurrent\{(p, p)|p ∈ Q}
```

To accept multiple characters at once, we have to change the construction of the NFA-Split architecture. First, it is necessary to modify algorithm [15], which is able to identify the simultaneously active states in the NFA. The algorithm tests whether two symbols are equal. To perform this operation, character classes have to be expanded to individual characters. Then the number of transitions can be increased up to $2^n$ times, where $n$ is data width of input characters (usually $n = 8$). The situation is even worse for the multi-stride automaton. The amount of transitions can be increased up to $2^{kn}$ times, where $n$ is data width of one input character and $k$ is the number of input characters accepted at once.

To avoid this transition growth, we can preserve character classes in symbols and replace the exact comparison by intersection operation $\cap_k$. The inputs of the operation $\cap_k$ are two multi-stride symbols defined as $k$-tuples $(A_1, A_2, ..., A_k)$ and $(B_1, B_2, ..., B_k)$, where items $A_i, B_i$ are subsets of input alphabet $A_i, B_i \subseteq \Sigma$. The subsets $A_i, B_i$ can represent individual character or a character class. The result of the operation is $k$-tuple $C = (C_1, C_2, ..., C_k)$, which is defined by the Eq. 1.

$$(C_1, C_2, ..., C_k) = (A_1 \cap B_1, A_2 \cap B_2, ..., A_k \cap B_k) \quad (1)$$

The $k$-tuple $(C_1, C_2, ..., C_k)$ contains set of input symbols that are included in both input $k$-tuples. If any item $C_i$ is equal to $\emptyset$ (empty set), then both input symbols $A$ and $B$ cannot be expanded to the same $k$-tuple of characters. This means that any pair of expanded symbols from $k$-tuples $A$ and $B$ is not equal. We denote this situation as $\cap_k = (\emptyset, \emptyset, ..., \emptyset)$ in the Algorithm 1, which is a modified algorithm to detect the simultaneously active states in multi-stride automaton. Then identification of deterministic and nondeterministic parts in NFA can be the same as in the original NFA-Split architecture.

The DU architecture must be modified to support multi-stride automaton. In this paper, we consider the architecture of the DU introduced in [15] which utilizes $k$-inner alphabets in order to reduce memory requirements. This architecture can be easily extended to support multi-stride automata. As can be seen in Fig. 1, the architecture remains the same except for the first component, which transforms input symbols to $k$ inner alphabets. The component is marked by dotted line and is able to join input symbols. For the automaton $A = (Q, \Sigma, \delta, q_0, F)$, two symbols $a, b \in Q$ can be joined only if $\forall q \in Q : \delta(q, a) = \delta(q, b)$. The proposed architecture uses BlockRAMs [23] as tables that provide efficient transformation of input symbols to $k$-inner alphabet.

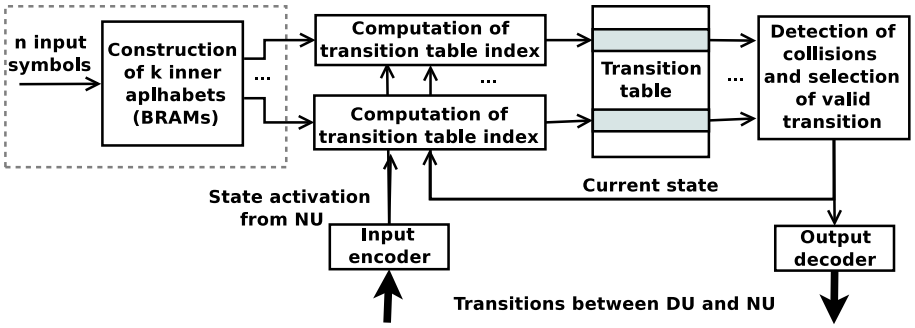


**Fig. 1.** Overview of DU architecture for SNFA-Split with $k$-inner alphabets and $n$ input symbols accepted at once. Dotted line is used to mark the new component to support multi-stride automata.

The nondeterministic part of the NFA-Split architecture for multi-stride automata is based on shared decoder architecture for multi-stride NFA as has been introduced in [6].

## 4 Optimizations of the NFA-Split Architecture

In the previous paper [15], we have presented reduction of memory and time complexity of the NFA-Split architecture. In this section, we consider optimizations of DUs and NUs in terms of efficiency of FPGA resource utilization. First, we analyze the efficiency of state representation in DU and NU. Then we propose an optimization of mapping the states to DU and several optimizations of mapping NU to the FPGA.

### 4.1 Optimization of Deterministic Parts of NFA

In this paper, we investigate the efficiency of mapping the DU to FPGA logic. The main factor influencing the resource utilization of the DU is the size of input encoding and output decoding logic. The size of the logic depends on the number of transitions to/from the DU. To analyze the number of transitions to/from the DU, we define *continuous parts* of the automaton as sets of states if:

1. All states are represented by the DU.
2. All states are reachable from some input state of the DU.
3. All states together with input/output transitions form a continuous graph of transitions.

As NFA-Split doesn't use continuous parts to derive mapping of states to DU and NU, we have to define a procedure how to identify continuous parts. First, we have to select an input state of the DU and then traverse along the transitions until a final state or a transition to NU is reached. If some state has more than one input transition, we have to traverse backwards until input transition to the DU is reached. Similarly, if some state has more than one output transition, then we have to traverse forward through all output transition until final state or transition to NU is reached. This procedure is finished if no new state can be added. The input states in already recognized continuous parts are not used for detection of next continuous parts.

We have analyzed the size of continuous parts for Snort backdoor rule set. The result is a histogram in Fig. 2. The x-axis represents the size of continuous parts and the y-axis represents the number of parts of that size. It can be seen that many continuous parts are very small (usually 1 to 3 states). Resource utilization for input encoders and output decoders associated with those very small continuous parts can be larger than the amount of logic resources needed for implementation of those parts in NU. This holds also for continuous part of any size with large number of inputs and outputs. Therefore, we define the Eq. 2 to have a simple condition when it is better to include continuous part $p_i$ in the DU.

$$cost_{inputs}(p_i) + cost_{outputs}(p_i) < cost_{NU}(p_i) \tag{2}$$

This means that the cost of input/output encoding has to be lower than the cost of implementation in the NU. The cost function $cost_{inputs}(p_i)$ computes

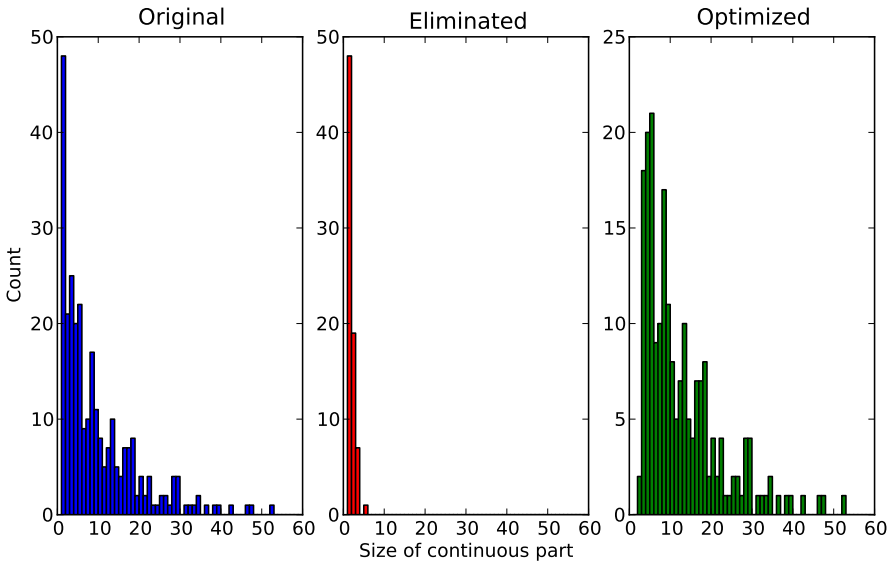


**Fig. 2.** Histogram of continuous parts in DU for Snort backdoor rule set. Distribution according to the size is provided for all parts in DU (Original), parts in DU after optimization (Optimized) and parts removed from DU because of inefficiency (Eliminated).

number of LUTs necessary to implement one-hot to binary encoding for input transitions. The cost function $cost_{outputs}(p_i)$ computes number of LUTs necessary to implement binary to one-hot encoding for output transitions and the cost function $cost_{NU}(p_i)$ computes number of LUTs needed to map the continuous part $p_i$ into NU.

Application of the Eq. 2 on the DU has direct impact to efficiency of the DU. Therefore, continuous parts violating Eq. 2 are better to be kept in NFA and represented by NU. Sizes of eliminated and optimized continuous parts are shown on histograms in Fig. 2. Eliminated parts are removed from the DU to the NU. Optimized continuous parts remain in the DU.

Characteristics of continuous parts of DU for various sets of RE before and after the DU optimization as well as characteristics of eliminated parts are shown in Table 1. Column Original contains characteristics before the optimization, column Optimized contains characteristics after the optimization and column Eliminated contains characteristics of continuous parts removed by the optimization. Three characteristics were measured: Number of continuous parts (Parts), Average size of continuous part (AS) and Average ratio between inputs/outputs and size of continuous part (AIOS). The sets of REs come from the Snort IDS [1] modules and from the L7 decoder [24]. Optimized DU has larger average size of continuous part and smaller average ratio between inputs/outputs and size of continuous part.

**Table 1.** Characteristics of continuous parts of DU for various sets of REs before and after the DU optimization.

| RE set | Original | | | Optimized | | | Eliminated | | |
|---|---|---|---|---|---|---|---|---|---|
| | Parts | AS | AIOS | Parts | AS | AIOS | Parts | AS | AIOS |
| | [-] | [-] | [-] | [-] | [-] | [-] | [-] | [-] | [-] |
| L7 selected | 53 | 10.66 | 0.28 | 37 | 14.51 | 0.21 | 16 | 1.75 | 1.57 |
| L7 all | 369 | 9.09 | 0.85 | 192 | 13.05 | 0.11 | 177 | 4.79 | 3.04 |
| backdoor | 284 | 9.96 | 0.32 | 208 | 12.37 | 0.13 | 76 | 3.39 | 2.30 |
| web-php | 21 | 10.67 | 0.30 | 19 | 11.68 | 0.14 | 2 | 1.00 | 18.00 |
| ftp | 42 | 3.55 | 1.26 | 19 | 4.89 | 0.39 | 23 | 2.43 | 2.70 |
| netbios | 40 | 7.72 | 0.28 | 13 | 20.46 | 0.08 | 27 | 1.59 | 1.53 |
| voip | 61 | 14.23 | 0.29 | 37 | 21.68 | 0.11 | 24 | 2.75 | 2.53 |
| web-cgi | 16 | 35.25 | 0.09 | 9 | 61.67 | 0.03 | 7 | 1.29 | 3.89 |

### 4.2 Efficient Encoding of the Nondeterministic Part

The encoding of the nondeterministic part into the NU utilizes the shared decoder architecture [25]. As we have shown in the previous chapter, some specific continuous parts of the DU can be moved into the NU to improve efficiency of mapping. Therefore, it is possible to utilize the properties of eliminated continuous parts and use more efficient encoding of the NU. We propose to use the At-most two-hot encoding (AMTH) introduced in [26], because it implements small 3 states parts efficiently with two LUTs and two flip-flops (FFs).

The mapping of constrained repetitions in the FPGA architecture also requires optimization. In the current NFA-Split architecture the constrained repetitions can be encoded either in the DUs or in the NU, depending on the type of the repetition and the NFA structure. The encoding of the Perl compatible RE (PCRE) into the DU is inefficient, because the counting constraint is represented by many states and transitions and size of the automaton is increased significantly. For example, PCRE `/^[abc]{100}/` in DU needs about 300 rows of the transition table. The NU architecture with a shared decoder is also not efficient. It consumes 100 FFs and 100 LUTs for the same example. The usage of the AMTH improves the efficiency (67 FFs and 67 LUTs). However, special subcomponents for constrained repetitions introduced in [10] are more efficient in logic utilization. Therefore, we propose to represent constrained repetition by this dedicated component in NU.

## 5 Evaluation

We performed the evaluation of proposed SNFA-Split architecture and optimizations on a selected set of Snort IDS [1] modules and set of REs from L7 decoder [24]. All used sets of REs come from Netbench framework [27]. The Netbench framework has been used to implement the proposed architecture to-

gether with optimizations and to make a comparison with other FPGA based multi-stride architectures.

**Table 2.** FPGA logic utilization of SNFA-Split and Clark multi-stride architectures. The results are for multi-stride automata with two and four input characters accepted at once.

| | **Statistics** | **Stride** | **Clark** | | **SNFA-Split** | | | |
|---|---|---|---|---|---|---|---|---|
| | **REs** | **Symbols** | **LUT** | **FF** | **LUT** | **FF** | **BRAM** | **Inner Alphabets** |
| **Rules** | [-] | [-] | [-] | [-] | [-] | [-] | [-] | [-] |
| L7 selected | 29 | 2 | 2744 | 673 | 1884 | 182 | 8 | 4 |
| backdoor | 154 | 2 | 7178 | 4383 | 3004 | 815 | 20 | 6 |
| web-cgi | 10 | 2 | 3456 | 1332 | 2688 | 738 | 9 | 2 |
| misc | 17 | 2 | 3200 | 1294 | 2551 | 944 | 10 | 4 |
| ftp | 35 | 2 | 3774 | 1944 | 3104 | 1595 | 10 | 4 |
| L7 selected | 29 | 4 | 4776 | 678 | 3631 | 192 | 24 | 8 |
| backdoor | 154 | 4 | 13509 | 4881 | 6137 | 1007 | 44 | 11 |
| web-cgi | 10 | 4 | 5608 | 1360 | 4598 | 738 | 12 | 4 |
| misc | 17 | 4 | 5322 | 1331 | 4219 | 951 | 20 | 6 |
| ftp | 35 | 4 | 6060 | 2081 | 4166 | 1601 | 20 | 6 |

First, we have evaluated FPGA logic utilization of SNFA-Split architecture. The results for two and four input characters accepted at once are compared in Table 2 to the multi-stride architecture with shared decoder of input characters. The amount of utilized LUTs, FFs and 18 Kb BlockRAMs was estimated for the Xilinx Virtex-5 architecture. However, the SNFA-Split architecture is suitable for any FPGA. Column Statistics presents the number of REs in particular set of REs. Column Clark shows the estimated utilization for the multi-stride architecture with shared decoder of input characters. Column SNFA-Split indicates the estimated utilization for the SNFA-Split architecture. It can be seen that the SNFA-Split architecture is able to reduce the amount of utilized LUTs by 58 % for the largest backdoor module. The table also indicates how many inner alphabets were used, because the utilization of FPGA resources depends on the number of inner alphabets.

We have also evaluated both proposed optimizations of NFA-Split architecture. Table 3 shows results of DU and NU optimizations. The amount of utilized FPGA resources is estimated. Column Statistics presents the number of REs in particular set of REs. Column Original shows the estimated utilization for the original NFA-Split architecture. Column Reduction indicates the reduction of FPGA resources by the proposed optimizations. It can be seen that the optimizations were able to achieve significant reduction of BlockRAMs and FPGA logic: 71.85 % LUTs for the nntp module and 94.18 % BlockRAMs for the voip module. This reduction is caused primarily by relocation of constrained repe-

titions from the DU into the optimized NU. The dedicated subcomponents for the constrained repetitions are more efficient. It can be seen in the results that the reduction mainly depends on the presence of counting constraints (e.g., L7 does not contain any, while voip does and the big ones were placed in the DU) and structure of the automaton. The last row of the Table 3 presents average reduction of utilized resources for 22 sets of REs from both Snort IDS and L7 project.

**Table 3.** Reduction of FPGA logic utilization by optimized DU and NU for in the NFA-Split Architecture.

| | **Statistics** | **Original** | | | **Reduction** | | |
|---|---|---|---|---|---|---|---|
| | **REs** | **LUT** | **FF** | **BlockRAM** | **LUT** | **FF** | **BlockRAM** |
| **Rules** | [-] | [-] | [-] | [-] | [%] | [%] | [%] |
| L7 selected | 29 | 1003 | 182 | 4 | 1.10 | -2.74 | 50 |
| L7 all | 143 | 8035 | 2945 | 8 | 15.08 | 2.11 | 25 |
| backdoor | 154 | 1696 | 727 | 10 | 7.05 | 1.15 | 20 |
| dos | 3 | 803 | 119 | 2 | -4.36 | 12.61 | 0 |
| ftp | 35 | 2284 | 1590 | 2 | 51.16 | 89.62 | 0 |
| misc | 17 | 1651 | 941 | 2 | 37.72 | 88.42 | 0 |
| nntp | 12 | 3133 | 2483 | 2 | 71.85 | 96.69 | 0 |
| web-cgi | 10 | 1651 | 736 | 4 | 39.83 | 88.59 | 50 |
| voip | 38 | 1936 | 834 | 34 | 35.18 | 77.46 | 94.18 |
| 22 RE Sets | 548 | 33421 | 12726 | 94 | 21.93 | 56.67 | 42.55 |

Four-stride SNFA-Split architecture running at 150 MHz has worst-case (Malicious network traffic) throughput of 4.8 Gbps. It outperforms GPU based solution presented in [18]. Even single stride architecture with throughput of 1.2 Gbps outperforms the GPU solution for rule-set L7 all. The efficient CPU solution [19] outperforms four-stride SNFA-Split architecture when running on high-end CPUs. However, the results in [19] are measured for best-case situation (Regular network traffic).

## 6 Conclusion

The paper has introduced the NFA-Split architecture optimization for multi-stride automata. The proposed architecture is able to process multiple bytes in one clock cycle. Therefore, RE matching speed can be increased despite frequency limits of current FPGAs. As can be seen in the Results section, the proposed multi-stride architecture utilizes up to 58 % less LUTs than multi-stride FPGA architectures with shared decoder. Consequently, additional REs can be supported.

Moreover, we have proposed several optimizations of the NFA-Split architecture in order to further reduce FPGA resources. First optimization is focused on

the overhead of encoding logic in DU. The optimization moves states from DU to NU, if the cost of encoding logic is higher than the cost of logic in the NU. The second proposed optimization is focused on NU mapping to the FPGA. At-most two-hot encoding and specific subcomponents for constrained repetitions are used to represent states and transitions relocated from DU to NU. Both optimizations are able to reduce up to 71.85 % of LUTs and up to 94.18 % of BlockRAMs.

As future work, we want to investigate efficient pattern matching on 100-Gigabit Ethernet.

## Acknowledgment

This work was supported by the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070 and the BUT project FIT-S-14-2297.

## References

1. Snort: Project WWW Page. `http://www.snort.org/` (2014)
2. The Bro Network Security Monitor: Project WWW Page. `http://www.bro.org/` (2014)
3. Koziol, J.: Intrusion Detection with Snort. Sams, Indianapolis, IN, USA (2003)
4. Becchi, M., Crowley, P.: Efficient Regular Expression Evaluation: Theory to Practice. In: ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ACM (2008) 50–59
5. Kumar, S., Turner, J., Williams, J.: Advanced Algorithms for Fast and Scalable Deep Packet Inspection. In: ANCS '06: Proceedings of the 2006 ACM/IEEE Symposium on Architecture for Networking and Communications Systems, ACM (2006) 81–92
6. Clark, C.R., Schimmel, D.E.: Scalable Pattern Matching for High Speed Networks. In: FCCM '04: Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, IEEE Computer Society (2004) 249–257
7. Lin, C.H., Huang, C.T., Jiang, C.P., Chang, S.C.: Optimization of Pattern Matching Circuits for Regular Expression on FPGA. IEEE Trans. Very Large Scale Integr. Syst. **15**(12) (2007) 1303–1310
8. Becchi, M., Crowley, P.: A-DFA: A Time- and Space-Efficient DFA Compression Algorithm for Fast Regular Expression Evaluation. ACM Transactions on Architecture and Code Optimization **10**(1) (2013) 4:1–4:26
9. Sidhu, R., Prasanna, V.K.: Fast Regular Expression Matching Using FPGAs. In: FCCM '01: Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, IEEE Computer Society (2001) 227–238
10. Sourdis, I., Bispo, J., Cardoso, J.M.P., Vassiliadis, S.: Regular Expression Matching in Reconfigurable Hardware. Journal of Signal Processing Systems **51**(1) (2008) 99–121
11. Becchi, M., Crowley, P.: Efficient Regular Expression Evaluation: Theory to Practice. In: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. ANCS '08, New York, NY, USA, ACM (2008) 50–59

12. Košař, V., Žádník, M., Kořenek, J.: NFA Reduction for Regular Expressions Matching Using FPGA. In: Proceedings of the 2013 International Conference on Field Programmable Technology, IEEE Computer Society (2013) 338–341
13. Kořenek, J., Košař, V.: Efficient Mapping of Nondeterministic Automata to FPGA for Fast Regular Expression Matching. In: Proceedings of the 13th IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems DDECS 2010, IEEE Computer Society (2010) 6
14. Kořenek, J., Košař, V.: NFA Split Architecture for Fast Regular Expression Matching. In: Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, Association for Computing Machinery (2010) 2
15. Košař, V., Kořenek, J.: On NFA-Split Architecture Optimizations. In: 2014 IEEE 17th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS), IEEE Computer Society (2014) 274–277
16. Clark, C., Schimmel, D.: Efficient Reconfigurable Logic Circuits for Matching Complex Network Intrusion Detection Patterns. In: Field Programmable Logic and Application, 13th International Conference, Lisbon, Portugal (2003) 956–959
17. Dlugosch, P., Brown, D., Glendenning, P., Leventhal, M., Noyes, H.: An Efficient and Scalable Semiconductor Architecture for Parallel Automata Processing. IEEE Transactions on Parallel and Distributed Systems **PP**(99) (2014)
18. Cascarano, N., Rolando, P., Risso, F., Sisto, R.: iNFAnt: NFA Pattern Matching on GPGPU Devices. SIGCOMM Comput. Commun. Rev. **40**(5) (2010) 20–26
19. Valgenti, V.C., Chhugani, J., Sun, Y., Satish, N., Kim, M.S., Kim, C., Dubey, P.: GPP-Grep: High-speed Regular Expression Processing Engine on General Purpose Processors. In: Proceedings of the 15th International Conference on Research in Attacks, Intrusions, and Defenses. RAID'12, Berlin, Heidelberg, Springer-Verlag (2012) 334–353
20. Kumar, S., Dharmapurikar, S., Yu, F., Crowley, P., Turner, J.: Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection. In: SIGCOMM '06: Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ACM (2006) 339–350
21. Smith, R., Estan, C., Jha, S., Kong, S.: Deflating the Big Bang: Fast and Scalable Deep Packet Inspection With Extended Finite Automata. SIGCOMM Comput. Commun. Rev. **38**(4) (2008) 207–218
22. Becchi, M., Crowley, P.: A Hybrid Finite Automaton for Practical Deep Packet Inspection. In: Proceedings of the 2007 ACM CoNEXT Conference. CoNEXT '07, New York, NY, USA, ACM (2007)
23. Brodie, B.C., Taylor, D.E., Cytron, R.K.: A Scalable Architecture For High-Throughput Regular Expression Pattern Matching. SIGARCH Computer Architecture News **34**(2) (2006) 191–202
24. L7 Filter: Project WWW Page. `http://l7-filter.sourceforge.net/` (2014)
25. Kořenek, J.: Fast Regular Expression Matching Using FPGA. Information Sciences and Technologies Bulletin of the ACM Slovakia **2**(2) (2010) 103–111
26. Yun, S., Lee, K.: Optimization of Regular Expression Pattern Matching Circuit Using At-Most Two-Hot Encoding on FPGA. International Conference on Field Programmable Logic and Applications **0** (2010) 40–43
27. Pus, V., Tobola, J., Kosar, V., Kastil, J., Korenek, J.: Netbench: Framework for Evaluation of Packet Processing Algorithms. Symposium On Architecture For Networking And Communications Systems (2011) 95–96