MPFC: Massively Parallel Firewall Circuits

Sven Hager Frank Winkler Björn Scheuermann Klaus Reinhardt

Computer Engineering Group
Humboldt University of Berlin, Germany
Email: {hagersve, fwinkler, scheuermann, reinhakl}@informatik.hu-berlin.de

***Abstract*—The process of matching the header fields of network packets against a set of rules is a performance critical task of firewalls. Software-based solutions have no chance to keep pace with the ever-growing data rates in high-speed networks. However, specialized filtering hardware is costly because complex logic is required in order to be able to apply arbitrary rulesets to a packet stream. By adapting the implemented logic to the specific firewall ruleset, FPGAs allow for much more specifically tailored and thus more efficient processing than ruleset-independent circuits in an ASIC. We present MPFC, a method to generate customized firewall circuits in the form of synthesizable VHDL code for FPGA configuration. The highly parallel MPFC circuits achieve a deterministic throughput of one packet per clock cycle, can be operated at high clock frequencies, and provide orders of magnitudes shorter processing latencies than previous work in this direction.**

***Index Terms*—FPGA; Firewall; Circuit Generation**

I. Introduction

Many companies and institutions employ packet filter firewalls in order to effectively regulate network traffic. Packet filter firewalls match certain protocol header fields of incoming network packets against a list of policy rules defined by an administrator. In case of a rule match, an action associated to that rule is applied, e. g., the packet could be dropped or passed through the filter. Today, many operating systems already ship with an integrated software firewall, a popular example is GNU/Linux together with iptables/Netfilter [1]. However, the constant growth of network bandwidth makes the task of matching packet headers against potentially large rulesets more difficult, as more network packets have to be analyzed per second—and the gap between CPU speed and network speed keeps growing.

The throughput of today's high performance packet filters can hardly be achieved in pure software solutions running on commodity hardware [15], [21]. It is commonly known that link rates of 40 Gbps or higher demand specialized hardware, as each incoming minimum-size (i. e., 40 byte) packet must be processed in 8 ns or less in order to meet line rate requirements [6], [12]. Therefore, high-speed firewalls are typically implemented in hardware or as a hybrid of hardware and software. The use of dedicated hardware resources provides several advantages over a general purpose CPU: a high degree of parallelism, flexible opportunities for operation pipelining, and low-latency access to network data. Consequently, the task of hardware-based packet classification has been a research topic in recent years [8], [13]–[15], [20], [21]. Typical hardware approaches store filter rules in RAM or CAM (*Content-Addressable Memory*) blocks. In such a design, the circuit structure is static and independent from the specific ruleset, therefore it must be designed in a way that the rules can be accessed and applied efficiently. It can be realized as an ASIC (*Application Specific Integrated Circuit*), or—for prototypes or in more specialized products—using an FPGA (*Field-Programmable Gate Array*). Here, we note that loading static, generic rule processing logic into an FPGA does not even nearly make good use of the FPGA's key benefit: its reconfigurability. We therefore take a radically different approach here: we propose to automatically synthesize specialized firewall circuits that implement one single, specific ruleset. The circuit design generated from a ruleset can then be loaded into an FPGA. We call this approach *MPFC*: Massively Parallel Firewall Circuits.

Of course, one has to keep in mind that a rule update requires the generation of a new bitfile. Hence, the MPFC approach is best suited for environments where the ruleset does not change too frequently. However, in practical applications of firewalls, this is the typical case: often, rulesets do not change for days, weeks, or even months. The circuit update can happen through a complete fabric reconfiguration or a partial reconfiguration at runtime.

The benefits of such an approach are best illustrated by a simple example. Virtually all IP firewall rulesets will contain rules which check an IP address against a given, fixed subnet. For instance, a rule might check whether the packet's source address falls into the IPv4 address range 10.0.0.0/8. Now consider the logic that is required to perform such a check in a common, static firewall processor. In such a design, the logic would need to be able to perform checks against arbitrary subnets; the information that the subnet is 10.0.0.0/8 would be obtained from the configuration memory. Consequently, comparison logic for two arbitrary 32-bit vectors would be needed, plus additional logic to disregard bits that are not enabled in the network mask. At least 65 logic gates are necessary for this functionality: two gates per address bit for bit-wise comparison (under consideration of the subnet mask bit), plus one gate with 32 inputs to obtain the overall result.

Now, in contrast, consider the logic that is necessary to perform the same check in a firewall circuit which is on-demand synthesized for the individual ruleset. For our example

Fig. 1: Creating a customized circuit design.

rule, this circuit only needs to compare to one single, fixed address range: 10.0.0.0/8. Therefore, it only needs to verify that the first eight bits of the address equal 00001010. This corresponds to a single 8-input NOR gate plus two negations for the 1-bits. Moreover, if rules are hardcoded in the circuit, there is no need for any logic to access the configuration memory—and, in fact, no need for configuration memory. Instead, we achieve the implementation of a ruleset as a combination of per-rule circuits.

The resulting overall matching circuit can be further minimized by removing redundancies across two or more rules. However, the sole use of automated logic minimization does not immediately yield an efficient filtering circuit—the real filtering efficiency is achieved through rule check parallelism and a heavily pipelined design.

In this paper, we underline the viability of the MPFC approach by introducing a compiler which translates (stateless) firewall rulesets into a synthesizable VHDL (*Very High Speed Integrated Circuit Hardware Definition Language*) hardware description. This workflow is sketched in Figure 1. The main contributions of this work include:

1) a compilation scheme to generate ruleset-tailored firewall hardware descriptions with CAM-like deterministic processing latency,
2) techniques to maintain a high operating frequency and therefore a high throughput, even when compiling large rulesets, and
3) extensive evaluation results examining the influence of different ruleset characteristics on the generated circuitry demonstrating throughputs of more than 100 Gbps of worst case traffic on commodity FPGAs.

The remainder of this paper is structured as follows: Section II discusses related work in the area of packet classification. Section III defines the packet classification problem, and Section IV describes how efficient rule matching and decision making logic can automatically be generated from firewall rulesets. Next, we evaluate system performance in Section V, before finally this work is concluded by Section VI.

II. Related Work

Firewalls have long been a subject of research [11], and many systems both in software [1], [2] and in hardware [20] have been developed. Static packet filtering discriminates network packets based on header fields, and is a special case of the packet classification problem [9]. Its algorithmic difficulty arises from the potentially huge number of rules, so that a linear search is expensive.

One way to speed up packet classification in hardware as well as in software is to perform independent searches per header field (which can be done in parallel) and then to compute the final result by combining the partial results. Well-known approaches along these lines are the cross-producting [26] and bit vector algorithms [17], which are amenable to both software and hardware implementation. Both have high memory requirements, which can partially be traded off against more complex rule matching [5].

Other algorithms tackle packet classification as a geometric problem and regard header fields as points in a multi-dimensional space. Accordingly, rules correspond to multi-dimensional cuboids. Matching a packet then means finding the set of cuboids which contain the corresponding point. The HiCuts algorithm [10] recursively cuts the multi-dimensional space into smaller pieces until each piece intersects only with a sufficiently small number of rule-cuboids. This subdivision defines a decision tree, which is traversed for classification. The performance mainly depends on the tree shape, which is determined by the cutting process. Further decision-tree approaches build upon HiCuts and improve, in particular, the hierarchical subdivision [8], [23], [27].

All schemes discussed so far follow the idea of a dedicated ruleset memory (cross-producting table, decision tree,...), along with a generic algorithm or circuit which iteratively traverses this memory. This also holds for FPGA-based classification engine designs in the existing literature, which are based on this class of algorithms [8], [13], [15], [20], [21]. Our approach encodes rules directly into the logic circuit and therefore fundamentally differs in the way in which rules are represented and applied—and, as discussed before, allows for much more efficient hardware and highly parallelized processing.

A well-known hardware-based classification technique with a high degree of parallelism is based on (T)CAMs [4]. A CAM is an associative memory which can be searched fully in parallel. A TCAM, in addition, supports wildcarded entries via "don't care" bits in the keys. When combined by mapping rules to TCAM entries, these properties allow for powerful packet classification. However, ranges or negations of header field entries cannot be expressed in single TCAM entries in a straightforward way [18]. There exists significant work on TCAM range and negation encoding, which aims to mitigate this inherent problem at the expense of additional hardware; examples are [18], [28]. In contrast, the MPFC approach entirely avoids this representation problem because range and negation tests can be directly implemented as logical functions on the FPGA fabric.

There have been previous approaches for building specialized firewall circuits for FPGAs in the past. In [16], [24], the authors convert sets of firewall rules into boolean expressions in the form of reduced ordered binary decision diagrams (ROBDDs). In a second step, these ROBDDs are directly translated to multiplexer based hardware descriptions. However, while ROBDDs provide a canonical representation of a given boolean expression (here, the firewall ruleset), their size can grow exponentially in the number of used variables (here, the bits of all relevant header fields, which in our case is 104) [7]. In contrast, the size of the generated circuits in

our approach grows linearly in the number of rules, as our evaluation results clearly show.

The probably closest relative to our work is a policy-to-hardware compilation scheme published by Lee et. al. that translates a given ruleset into a classification pipeline, where each rule represents a stage in the pipeline [19]. Thus, the number of pipeline stages and the latency until a firewall decision is finally computed grows linearly with the number of rules, which may not be acceptable if the ruleset is large. But while our scheme also uses a heavily pipelined design and thus is able to process multiple packet headers in different pipeline stages in parallel, it checks the entire ruleset simultaneously in a small fixed number of clock cycles. The final result (i. e., the most prioritized matching rule) is then computed using a pipelined priority encoder of logarithmic height. As a result, the overall computation latency is drastically reduced, as our evaluation results underline.

Therefore, our results are complementary to [16], [19], [24] and go beyond them: (1) we propose a method to efficiently compute the final firewall decision in a heavily pipelined way, which first aggregates rules with equivalent actions and subsequently generates priority encoders of logarithmic depth, (2) we demonstrate that modern synthesis tools can perform a large amount of rule logic optimization without any ruleset preprocessing, and (3) we examine in detail how different ruleset characteristics influence the size and the operating clock speed of the generated circuits.

III. Problem Statement

In order to decide how to deal with an incoming network packet, a firewall has to match the packet's header fields against a database of rules, the *ruleset*. For the remainder of this work, we will regard the ruleset as an ordered list of rules

$$\mathcal{R} \ = < R_1,\ R_2,\ \ldots,\ R_n > . \tag{1}$$

A rule R_i specifies checks (that is, matching criteria) for one or more header fields, along with an action Act_i that shall be performed in case of a match. We will refer to the entire set of actions specified by the ruleset as $\mathcal{ACT}$. If the rule performs checks $C_{j,i}$ on M packet header fields, it can thus be written as

$$R_i \ = \ (< C_{1,i},\ \ldots,\ C_{M,i} >,\ Act_i). \tag{2}$$

The rules are prioritized, where the priority is given by the order of the rules in the ruleset. If a packet matches more than one rule, only the action of the most highly prioritized matching rule should be executed. Without loss of generality, we assume

$$\text{priority}(R_1) \ > \ \ldots \ > \ \text{priority}(R_n). \tag{3}$$

Let $\mathcal{H}_j$ be the set of possible values of header field j. Then, a check $C_{j,i}$ can be considered a boolean function

$$C_{j,i} : \mathcal{H}_j \rightarrow \{\text{true}, \text{false}\}. \tag{4}$$

A network packet P with header fields $P_j, 1 \leq j \leq M$, is said to match rule R_i if all of the packet's header fields obey the respective restrictions $C_{j,i}$ imposed by R_i, i. e., if and only if

$$\bigwedge_{j=1}^{M} C_{j,i}(P_j). \tag{5}$$

For the remainder of this work, we focus on checks of the classical header fields used for packet classification in IP networks: the protocol field, the source and destination IP addresses from the IP header, and the source and destination ports from the transport header. As is common for firewalls, checks can be (in)equality checks, range checks, or subnet membership tests. Generalizations to further header fields and other kinds of checks are straightforward.

Given a packet P, the central task of the firewall is to identify the most highly prioritized matching rule $R_{i*} \in \mathcal{R}$. The firewall will then apply action Act_{i*} to the packet. Our aim here is to automatically construct an efficient circuit for a given ruleset $\mathcal{R}$: our compiler generates synthesizable VHDL code from the ruleset specification. We first discuss the digital logic that is required for matching a set of packet header fields against one single rule. Subsequently, we turn towards comparing against a complete ruleset.

IV. The MPFC Compiler

Our MPFC compiler translates a given firewall ruleset to a single synthesizable VHDL entity. The translation of n rules is possible in time $\mathcal{O}(n \log n)$. First, independent VHDL processes for each rule are generated, whose local classification results are fed into subsequently created tree structures for global result computation. In the remainder of this section, we detail the generated structures and point out the most important optimizations.

A. Rule Matching

In order to make best use of the massive parallelism available on FPGAs, we perform a parallel search over the full ruleset. To this end, the entire ruleset is encoded in the generated circuit: all the checks can be applied simultaneously. Matching packet headers against a rule requires several logic and arithmetic checks to be performed. As implied by (5), the individual checks can be performed independently. Combining their results with an AND operation yields the overall result for a rule. This pattern can be mapped to a logic circuit in a straightforward way, as sketched in Figure 2.

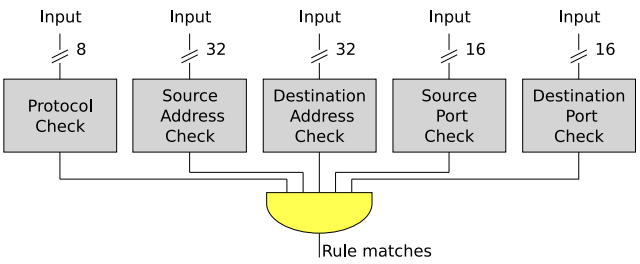


Fig. 2: Rule matching circuit.

The compiler translates each rule R_i to a separate VHDL process p_i, which implements the logic circuit for R_i. In order to test all rules in parallel, arriving header data are distributed to all these rule circuits. After one clock cycle, a result bit is computed for each rule R_i; a value of 1 means that R_i matched the packet's header fields. We call the entirety of these results—a binary vector of $|\mathcal{R}|$ bits—the *matching vector* and denote it by MV. Figure 3 shows the high-level structure.

For the individual checks, the MPFC compiler generates only the minimal amount of logic that is needed. For example if a rule R_i requires the source IP address to belong to a specific $/8$ subnet, the VHDL process p_i will check only the first eight bits of the address. Moreover, as we will discuss later on, the finally implemented circuit for the overall ruleset is further minimized during synthesis and may thus be even more compact.

B. Aggregating Rules to Blocks

The next important step in the classification process is to determine the action that should be applied to the network packet. Following (1) and (3), we assume without loss of generality that bits in MV have decreasing priority from left to right. Hence, the most prioritized matching rule is indicated by the leftmost set bit in MV, which can be found using a priority encoder. However, we make the observation that the input vector size for the priority encoder can be reduced when the result bits of consecutive rules with the same action are aggregated in a previous step. For this purpose, we introduce the concept of a *block*, which is a maximum set of consecutive rules with identical action. In firewall ruleset design, it is good practice to define a default policy and then to specify sequences of rules which override the default policy in application-dependent special cases. This leads to ruleset structures which do in fact often contain blocks of rules with identical action. Figure 4 shows a concrete example for a ruleset with five rules and the corresponding list of blocks. The first two pairs of rules can be aggregated to blocks $B_1 = \{R_1, R_2\}$ and $B_2 = \{R_3, R_4\}$. Rule R_5 forms a third block by itself.

We now observe that the relative priority of rules within a block is irrelevant, since all rules take the same action. In the above example, action Act_1 will be applied if any of the rules in the first block matches—regardless which of these rules matches, and regardless of the results of rules in subsequent blocks. Each block can therefore efficiently be summarized into one single logical test. Hence, instead of identifying the matching *rule* with the highest priority, it suffices to identify the most highly prioritized *block* which contains at least one matching rule.

We call this property *block-level equivalence*. By block-level aggregation we obtain an aggregated block-level matching vector AMV out of the per-rule matching vector MV. To this end, an OR operation is applied to all bits in MV that belong to the same block, yielding the respective bit in AMV. If $\mathrm{offset}(B_i)$ denotes the index in vector MV of the first rule in block B_i and $|B_i|$ denotes its length, then bit i of AMV is given by

$$\mathrm{AMV}_i = \bigvee_{j=0}^{|B_i|-1} \mathrm{MV}_{j+\mathrm{offset}(B_i)} \,. \tag{6}$$

C. Summarizing Match Results in Trees

When implementing the blocking concept, we found that simply computing the bitwise OR of a huge block of result bits in one clock cycle results in a low maximum clock frequency for the circuit. The reasons are likely to be high gate and wiring delays in case of gates with very high fan-in. Since FPGA circuit implementation is generally performed by proprietary, closed-source tools, this is difficult to track down in detail. The above hypothesis, however, suggests to reduce the fan-in by aggregating the result bits in a pipelined, hierarchical manner. We implemented such a strategy using a balanced tree, in which each node represents an OR operation between few inputs. The tree levels are independent and are computed in consecutive clock cycles. At the root of the tree, the overall aggregated block result is obtained. Figure 5 illustrates the structure for an eight-rule block and two hierarchy levels. The height of such an aggregation tree T_{B_i} is

$$\mathrm{height}(T_{B_i}) = \lceil \log_\alpha |B_i| \rceil \,, \tag{7}$$

where α is the maximum number of input pins to a tree node. $\mathrm{height}(T_{B_i})$ is also the number of clock cycles it takes to compute AMV_i, since each level of the aggregation tree is computed in a separate cycle. Accordingly, the total number of aggregation pipeline stages is $\max_i \ \mathrm{height}(T_{B_i})$, i. e., the length of the pipeline for the largest rule block. In order to keep all block results available until the result of the largest block is computed, we use delay queues.

Good values for the parameter α depend on the fan-in that can be well supported by the used FPGA platform. This is mainly determined by the size of the lookup tables and by the degrees of freedom for the place and route process. On

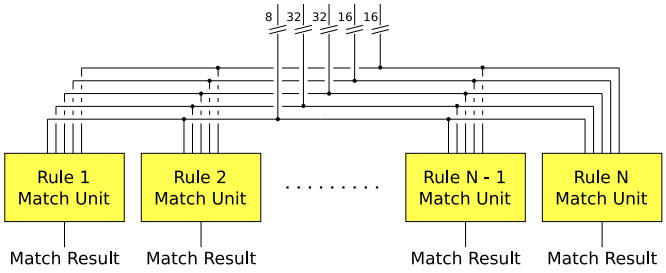


Fig. 3: Parallel matching units.

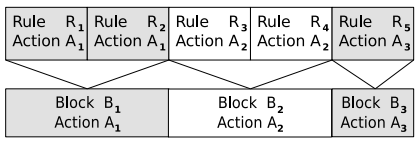


Fig. 4: Joining rules to blocks.

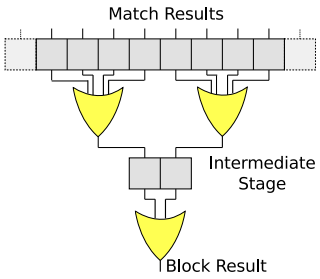


Fig. 5: Aggregating a block of match results.

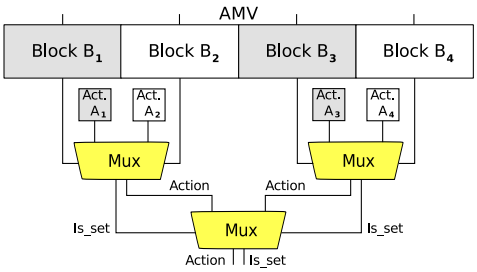


Fig. 6: The priority encoding tree.

a Xilinx Virtex 5, $\alpha = 4$ gave good results. The benefit of aggregation trees is particularly well visible when the compiled ruleset defines one or more huge blocks, as we will demonstrate in the evaluation section.

The aggregation tree structure increases the latency for a single packet classification. However, it also allows for pipelined processing at a higher clock rate, which increases the overall throughput. Consequently, there exists a trade-off between throughput and latency. Yet, as we will show, typical latencies are far below one microsecond; therefore, the pipelined design appears highly favorable.

D. Determining the Packet's Fate

When all block results are available, a generalized priority encoder determines the action of the most prioritized matching block. Note that the generalized priority encoder does not compute the index of the matching block, but instead a code that maps to the corresponding action. The encoder is a generated decision tree PT whose input is the vector of block results AMV. Each tree node TN_i consists of a multiplexer and provides two output signals `Is_set` and `Action`. The `Is_set` signal of node TN_i is active if any of TN_i's predecessors has an active `Is_set` output. If this is the case, TN_i's multiplexer propagates the `Action` signal of its most highly prioritized predecessor with `Is_set` enabled. Otherwise, TN_i will set its own `Is_set` output signal to zero, and its `Action` output signal should then be considered invalid. If a packet matches no rule, which can happen if the specified ruleset is non-total, then consequently also the root node will assign zero to its `Is_set` signal in order to inform the circuit which uses the firewall circuit. Each level of the priority encoding tree PT is a pipeline stage with registers to hold intermediate results. Since PT is a balanced tree, its height is $\lceil \log |\,\mathrm{AMV}\,| \rceil$. Figure 6 shows a priority encoding tree for a ruleset with four action blocks.

V. Evaluation

We evaluated the generated circuits by compiling various firewall rulesets and examining the clock rate at which they can be operated as well as their size in terms of used FPGA slices. We will also compare our approach to circuit structures as proposed by Lee et. al. [19] in a number of settings. In all our measurements, we used the Xilinx ISE 14.4 CAD tool to synthesize, map, place and route the generated VHDL code [3]. Allowable clock rate and circuit size were obtained from ISE after the build. In our measurements, we used a Xilinx Virtex 5 FPGA[1] as our build target. We evaluated both publicly available rulesets and custom synthetic rulesets. In the synthetic rulesets, each rule contains tests on all five considered header fields. This is a worst case setting: real-world rules often contain less checks and can be implemented with less hardware. Each rule yields one of the two fundamental actions implemented by all firewall systems: accept the packet or drop the packet.

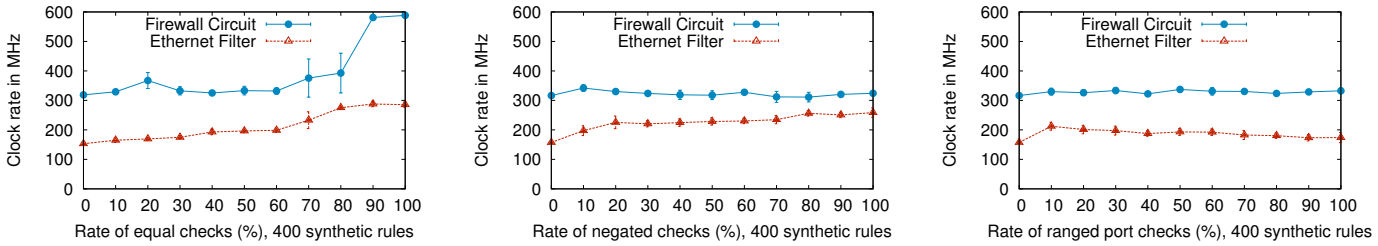
The rule processing circuits were evaluated in two different environments. First, we evaluated the performance of the pure rule checking logic (the "Firewall Circuit") in order to examine the generated filtering circuits in isolation. In order to also test the firewall circuit in a small application, we implemented a hardware filter for minimal (64 byte) Ethernet frames (the "Ethernet Filter") which wraps the generated circuit and adds I/O components: a system of interconnected FIFO memories, a frame parser, and a gateway that either forwards or drops frames depending on the forwarding decision of the firewall circuit. The evaluation of this filter application shows the impact of realistic overheads caused by memory elements and implementation details on the Virtex 5 FPGA architecture.

A. Self-Generated Synthetic Rulesets

In our first series of evaluation results, we examine the impact of overlapping checks, negated checks, ranged port checks, the number of blocks as well as the overall number of rules on the generated circuits. Here, we use synthesized, artificial rulesets in order to control and vary the abovementioned parameters. Each experiment was run ten times with different random synthetic rulesets. The corresponding figures show average results and 95% confidence intervals.

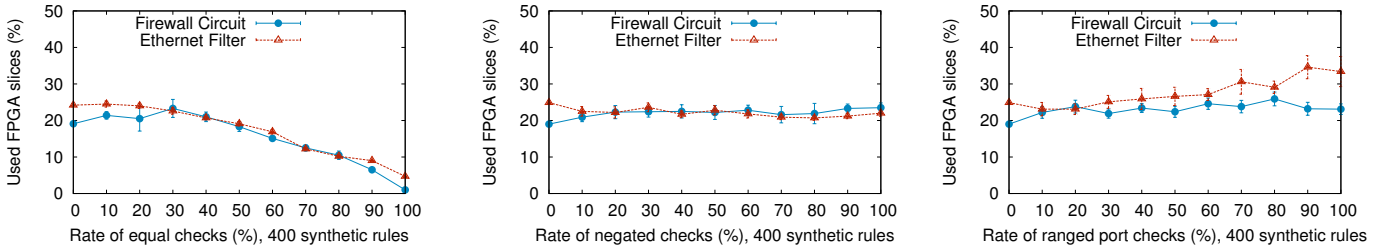
First, we examine the impact of check overlaps, i. e., the ratio of checks that are shared between multiple rules (e. g., two rules which both include a check whether the packet comes from source port 22), on the structure of the generated circuits. Due to the fact that each check is represented by combinational logic, redundant checks can, in principle at least, be removed by logic minimization over the full rule matching circuit description. The extent to which this is feasible also depends on routing and placement constraints. We generated eleven random rulesets and increased the ratio of overlapping checks from 0% to 100%. Here, "n % of overlapping checks" means

[1] device XC5VSX50T, package FF1136, speed grade -1



(a) Check overlaps vs. clock rate. (b) Negated checks vs. clock rate. (c) Range checks vs. clock frequency.

Fig. 7: Impact of check overlap, negated checks and range checks on clock frequency.



(a) Check overlaps vs. resource usage. (b) Negated checks vs. resource usage. (c) Range checks vs. resource usage.

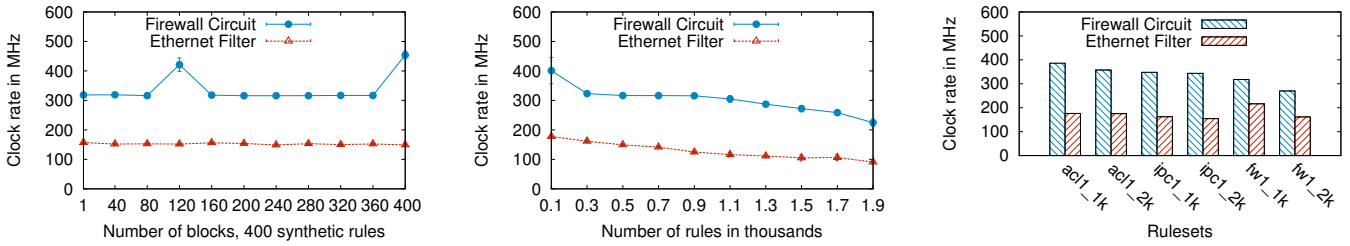
Fig. 8: Impact of check overlap, negated checks and range checks on resource usage.

that $n\,\%$ of the checks also appeared in at least one previous rule and could therefore be eliminated during minimization. Hence, a larger overlap should result in a smaller synthesized circuit. This is confirmed by Figures 7a and 8a, which show that circuits with a higher check overlap can be operated faster and can be implemented with less FPGA slices.

In Section II we claimed that negated checks (e. g., "source address *not* in 10.0.0.8/30") and range checks (e. g., "source port between 51 and 100") can be implemented efficiently in a canonical way on FPGAs using the MPFC approach, as opposed to TCAM-based approaches which suffer from range and negation check explosion [18], [28]. In order to validate this claim, we again generated synthetic rulesets, but now we varied the ratio of negated checks and ranged port checks, respectively. As expected, Figures 7b and 8b show that an increased number of negated checks does not deteriorate either the clock rate or the circuit size significantly. The ratio of ranged port checks does also not impact the circuit clock rate, see Figure 7c. The only noticeable impact is that a high number of port range checks slightly increases the circuit size, as can be seen in Figure 8c. This was to be expected, though: ranged port checks require two comparisons instead of one.

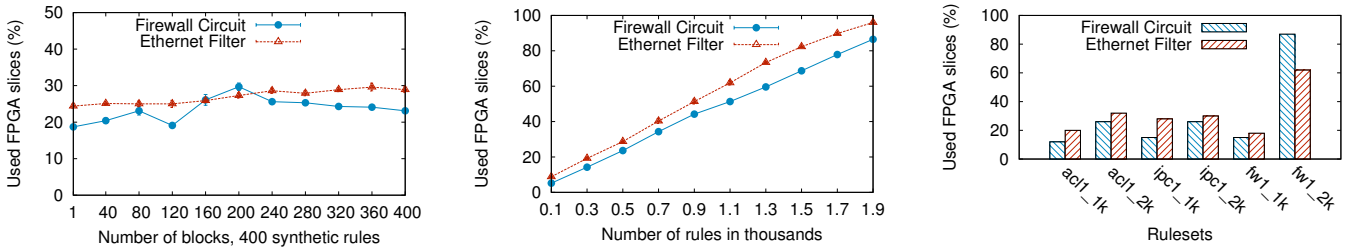
Next, we examined the impact of the number of blocks on circuit characteristics by generating rulesets with a varying number of blocks from 1 to 400. According to Section IV-B, a block is defined as a set of adjacent rules which define the same firewall action. In order to compute the firewall's forwarding decision, the match results of all blocks must be combined in a priority encoder, whose size depends on the number of blocks. We designed the layout of the priority encoder such that its size should not negatively affect the circuit's clock rate. Due to the staged computation of the firewall decision by the tree-shaped priority encoder PT, the clock rate can be kept nearly constant at an increased number of blocks (except for two spikes in the Firewall Circuit frequency, which we attribute to ISE synthesis details), as illustrated by Figure 9a. However, Figure 10a reveals that this comes at the cost of a slightly increased number of slices. The reason for this is that the staged computation requires additional circuitry and more memory elements.

Furthermore, we measured how the size of a ruleset influences the resulting circuit. To this end, we generated rulesets with mutually different checks (in order to avoid check overlap optimization) in a single block and gradually increased the number of rules. As one would expect, smaller rulesets require less circuitry and can be operated at higher clock rates, which is confirmed by Figures 9b and 10b. Note that the linear frequency decrease and linear resource increase shown in Figures 9b and 10b partially result from the fact that all field checks are different, which is unlikely to happen in practice. Often, firewall rulesets perform the same field checks in several rules. For example, the `acl1_1k` ruleset which is analyzed in Section V-B defines 911 source subnet checks, but only 102 of these checks are unique. As demonstrated above, this kind of check redundancy leads to smaller generated circuits and a higher clock frequency.



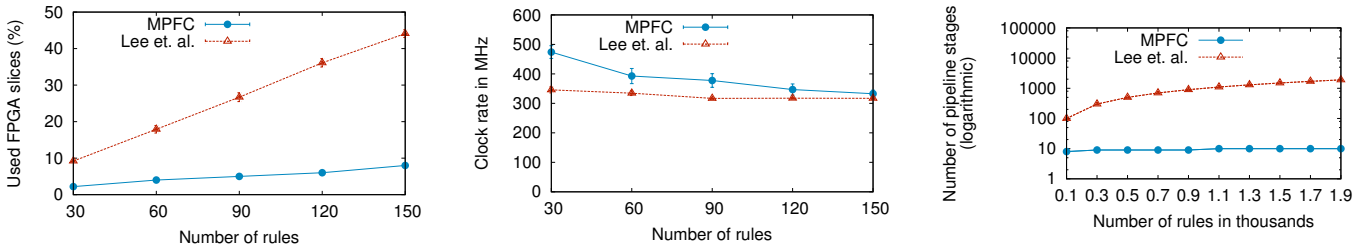
(a) Number of blocks vs. clock rate. (b) Number of rules vs. clock rate. (c) Public ruleset circuit clock rate.

Fig. 9: Number of blocks, ruleset size, and public rulesets vs. clock rate.



(a) Number of blocks vs. resource usage. (b) Number of rules vs. resource usage. (c) Public ruleset circuit resource usage.

Fig. 10: Number of blocks, ruleset size, and public rulesets vs. resource usage.



(a) Comparison of resource usage. (b) Comparison of circuit frequency. (c) Comparison of pipeline depths.

Fig. 11: Comparison of our approach with Lee et. al. [19].

As explained in Sections IV-C and IV-D, the number of pipeline stages grows logarithmically with the size of the biggest block (for the aggregation trees) and with the number of blocks (for the priority encoder PT). However, the longest pipeline length we observed in our experiments had 13 stages. The corresponding Ethernet Filter circuit could be operated at 149 MHz, so that the delay for a firewall decision is only 87 ns. We then conducted an experiment to outline the possible performance gains when using aggregation trees instead of a plain OR operation when computing the block results. To do so, we generated the Firewall Circuit from a ruleset of 1000 rules in a single block with and without tree aggregation. The circuit with tree aggregation could be operated at 315 MHz, which was nearly twice as fast as the variant with the plain OR operation (158 MHz).

Finally, we compared our work with the scheme proposed by Lee et. al. in [19]. Figures 11a to 11c show a comparison of circuit size, circuit frequency, and resulting pipeline depth of our circuit structure and the structure proposed in [19]. For this measurement we used the raw filtering circuits in order to achieve comparison results unbiased by I/O components. We generated rules without check overlaps (as defined above) which could potentially be shared in order to measure the worst case scenario. It can be seen that the circuits generated by our compiler are much smaller and can be operated at higher frequencies (Figures 11a and 11b). However, the most obvious advantage of the MPFC circuits is their low pipeline depth, which is several orders of magnitude smaller than those of the circuits proposed by Lee et. al. (depending on the number of rules) and therefore allows for far shorter processing latencies: note the logarithmic y-axis in Figure 11c.

B. Publicly Available Rulesets

In the previous subsection we evaluated the generated circuits using self-generated, artificial rulesets which were tailored to meet certain characteristics like a given check overlap or a certain fraction of range checks. For additional realism and comparability, we now complement this with publicly available packet classification rulesets provided by Haoyu Song [25]. We grouped all rules of each ruleset in alternating "DROP" and "PASS" action blocks of size 20.

Figures 9c and 10c show the respective circuit clock rates and the ratio of used FPGA slices for the regarded public rulesets. Similar to the synthetic rulesets, the clock rate and the ratio of used FPGA slices depend on the size of the source rulesets: FPGA utilization increases with the number of rules, while the clock rate decreases. Moreover, once again not only the size, but also the structure of the ruleset influences the performance and size of the generated circuit. This is particularly well visible for ruleset `fw1_2k`, which has more than twice as many unique subnet checks as the other rulesets of equal size (`acl1_2k`, `ipc1_2k`)[2]. For both the Firewall circuit and the Ethernet Filter, Table I shows the maximum number of packet headers that can be processed per second by these circuits. Due to the pipelined design, the generated classification circuit can emit one packet per clock cycle, so this directly corresponds to the achieved clock frequency. It can be seen that the Firewall Circuit can be operated at higher frequencies than the Ethernet Filter, which results from the additional circuit complexity and memory elements of the Ethernet Filter.

Because the achievable throughput is proportional to the clock rate, the results also allow for throughput estimates. For instance, the `fw1_1k` Ethernet Filter (i. e., including I/O components on the FPGA) can be operated at 216.2 MHz; thus, in the worst case of 64 byte Ethernet frames (the minimum size) it achieves 110.7 Gbps. Packets in practical networks are typically much larger, meaning even higher throughput at the same classification speed [22]. These numbers assume that one packet header per clock cycle can be provided to the FPGA logic, i. e., that the environment into which the FPGA is embedded is able to handle and process entire packets at the respective rate.

[2]The `*_2k` rulesets were obtained by taking the first 2000 rules from the corresponding `*_5k` rulesets available at [25].

TABLE I: Classified packet headers per second.

Ruleset	Firewall Circuit	Ethernet Filter
acl1_1k	386.1×10^6	176.0×10^6
acl1_2k	357.3×10^6	174.9×10^6
ipc1_1k	348.0×10^6	162.1×10^6
ipc1_2k	343.6×10^6	154.4×10^6
fw1_1k	317.8×10^6	216.2×10^6
fw1_2k	269.7×10^6	161.3×10^6

VI. Conclusion

This work presents MPFC, an approach for generating heavily pipelined firewall decision logic on FPGAs. We propose a number of techniques to automatically create logic circuitry from a given stateless firewall ruleset. We pay particular attention that the clock rate of the generated circuits is kept at a high level. An FPGA that is configured to process a ruleset provides deterministic classification time for given header data. In fact, the classification pipeline has stable throughput of one packet header per clock cycle, where each packet is matched against the full ruleset.

Due to the fact that we translate entire rulesets to synthesizable VHDL code, cross-rule logic minimization can reduce the size of the resulting firewall circuit. Furthermore, we show how to take advantage of block structures in firewall rulesets in order to reduce the size of the priority encoder which is needed to determine the classification result. Hence, the resulting designs are efficient in terms of clock rate and size.

We evaluated our system by generating firewalls from publicly available and synthetic rulesets. We compared the performance characteristics in terms of clock rate and logic resource utilization. The results indicate that the generated system can support thousands of rules on medium-sized commodity FPGAs, and that they are able to apply the ruleset to hundreds of millions of packets per second on such a platform.

Acknowledgments

The authors would like to thank Markus Appel and Martin Brückner for their invaluable help with taming our FPGA boards. We also want to express our gratitude for fruitful discussions towards Alexander von Gernler and the other folks at genua mbH, a manufacturer of specialized firewall devices.

References

[1] "The netfilter.org project," www.netfilter.org, last access: June 16, 2014.
[2] "OpenBSD packet filter," http://www.openbsd.org/faq/pf/, last access: June 16, 2014.
[3] "Xilinx homepage," http://www.xilinx.com/, last access: June 16, 2014.
[4] F. Baboescu, S. Singh, and G. Varghese, "Packet classification for core routers: Is there an alternative to CAMs?" in *INFOCOM '03: Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies*, Mar. 2003, pp. 53–63.
[5] F. Baboescu and G. Varghese, "Scalable packet classification," in *SIGCOMM '01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug. 2001, pp. 199–210.
[6] P. Bosshart, G. Gibb, H. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *SIGCOMM '13: Proceedings of the 2013 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug. 2013, pp. 99–110.
[7] R. E. Bryant, "Symbolic boolean manipulation with ordered binary-decision diagrams," *ACM Computing Surveys*, vol. 24, no. 3, pp. 293–318, Sep. 1992.
[8] J. Fong, X. Wang, Y. Qi, J. L. 0003, and W. Jiang, "ParaSplit: A scalable architecture on FPGA for terabit packet classification," in *HOTI '12: Proceedings of the 20th Symposium on High Performance Interconnects*, Aug. 2012, pp. 1–8.
[9] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network*, vol. 15, no. 2, pp. 24–32, Mar. 2001.

[10] ——, "Packet classification using hierarchical intelligent cuttings," in *HOTI '99: Proceedings of the 7th Symposium on High Performance Interconnects*, Aug. 1999, pp. 34–41.

[11] K. Ingham and S. Forrest, "A history and survey of network firewalls," University of New Mexico, Tech. Rep., 2002.

[12] W. Jiang and V. Prasanna, "Scalable packet classification on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 9, pp. 1668–1680, Sep. 2012.

[13] W. Jiang and V. K. Prasanna, "Large-scale wire-speed packet classification on FPGAs," in *FPGA '09: Proceedings of the ACM/SIGDA 17th International Symposium on Field Programmable Gate Arrays*, Feb. 2009.

[14] ——, "Large-scale wire-speed packet classification on FPGAs," in *FPGA '09: Proceedings of the ACM/SIGDA 17th International Symposium on Field Programmable Gate Arrays*, Feb. 2009, pp. 219–228.

[15] W. Jiang and V. Prasanna, "A FPGA-based parallel architecture for scalable high-speed packet classification," in *ASAP '09: Proceedings of the 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, Jul. 2009.

[16] A. Johnson and K. Mackenzie, "Pattern matching in reconfigurable logic for packet classification," in *CASES '01: Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded System*, Nov. 2001, pp. 126–130.

[17] T. V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," in *SIGCOMM '98: Proceedings of the 1998 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug. 1998, pp. 203–214.

[18] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, "Algorithms for advanced packet classification with ternary CAMs," in *SIGCOMM '05: Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug. 2005, pp. 193–204.

[19] T. Lee, S. Yusuf, M. Sloman, E. Lupu, and N. Dulay, "Compiling policy descriptions into reconfigurable firewall processors," in *FCCM '03: 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, Apr. 2003, pp. 39–48.

[20] A. A. McEwan and J. Saul, "A high speed reconfigurable firewall based on parameterizable FPGA-based content addressable memories," *The Journal of Supercomputing*, vol. 19, no. 1, May 2001.

[21] Y. Qi, J. Fong, W. Jiang, B. Xu, J. Li, and V. Prasanna, "Multi-dimensional packet classification on FPGA: 100 Gbps and beyond," in *FPT '10: The 2010 International Conference on Field-Programmable Technology*, Dec. 2010, pp. 241–248.

[22] H. Schulze and K. Mochalski, "Internet study 2008/2009," ipoque, Tech. Rep., 2009.

[23] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *SIGCOMM '03: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug. 2003, pp. 213–224.

[24] R. Sinnapan and S. Hazelhurst, "A reconfigurable approach to packet filtering," in *FPL '01: Proceedings of the 11th International Conference on Field Programmable Logic and Applications*, Aug. 2001, pp. 638–642.

[25] H. Song, "Evaluation of packet classification algorithms," website includes publicly available rulesets, last access: June 16, 2014. [Online]. Available: http://www.arl.wustl.edu/~hs1/PClassEval.html

[26] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fast and scalable layer four switching," in *SIGCOMM '98: Proceedings of the 1998 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug. 1998.

[27] B. Vamanan, G. Voskuilen, and T. N. Vijaykumar, "EffiCuts: Optimizing packet classification for memory and throughput," in *SIGCOMM '10: Proceedings of the 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug. 2010, pp. 207–218.

[28] F. Yu and R. H. Katz, "Efficient multi-match packet classification with TCAM," in *HOTI '04: Proceedings of the 12th Symposium on High Performance Interconnects*, Aug. 2004, pp. 28–34.