VNFAccel: An FPGA-based Platform for Modular VNF Components Acceleration

Filipe B. Lopes, Gabriel Luca Nazar, Alberto E. Schaeffer-Filho
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
{fblopes, glnazar, alberto}@inf.ufrgs.br

***Abstract*—The expected benefits of Network Function Virtualization (NFV), i.e., flexibility and efficiency, require platforms able to instantiate Virtualized Network Functions (VNFs), replacing traditional dedicated middleboxes. However, to fulfill performance requirements, especially for functions operating at high-throughput line rates, dedicated hardware acceleration may be required and, to provide the expected flexibility, hardware accelerators must be able to support different VNFs. Thus, reconfigurable devices such as Field-Programmable Gate Arrays (FPGAs) have received attention due to their ability to provide both properties. Not all VNFs, however, require or are even suitable for hardware acceleration and, therefore, heterogeneous platforms composed of FPGAs and General Purpose Processors (GPPs) can be used. Even for a single network function, distinct VNF Components (VNFCs) may have different demands and restrictions, meaning that different hardware substrates may be more suitable for their implementation. In this paper, with the goal of promoting the efficient and seamless integration of accelerators in NFV, we propose VNFAccel, a platform to manage the execution of VNFCs in a heterogeneous infrastructure. We evaluate the effectiveness of FPGAs as accelerator devices for two VNF case studies and analyze the instantiation latency of FPGA-based VNFCs in the platform. We also provide guidelines to design VNFCs that are more flexible, promoting increased reuse and reducing configuration time, and identify properties that make them suitable for FPGA implementation.**

***Index Terms*—FPGA, Network Function Virtualization, hardware acceleration.**

I. Introduction

Given the demand for efficient, reliable, and economical network architectures, academia and industry are looking for new methods to fulfill these expectations. Traditional networks rely on middleboxes, e.g. Intrusion Detection System (IDS), Network Address Translators (NATs), which are a strict combination of software and application-specific hardware. This fixed configuration generates a lack of flexibility and portability, and maintaining a network environment using several specific equipment incurs, in addition to high capital expenditure, in complex integration between these devices [1].

Network Function Virtualization (NFV) is a novel paradigm that has gained attention in the past years, which takes network functions off the hands of Application-Specific Integrated Circuits (ASICs) and specific hardware to put them in virtualized infrastructures (e.g., virtual machines and containers). This effort results in lower-cost devices handling network functions, with more flexibility to changes and updates. In this context, Virtualized Network Functions (VNFs) are the virtualization of functions such as Deep Packet Inspection (DPI), firewall, load balancing. In turn, these functions are typically built in terms of building-block components (e.g., regex matching engine, alerts, header classifier) that are common across different VNFs. These fine-grained, modularized components are called Virtualized Network Function Components (VNFC) [2].

One of the main challenges in the context of network function virtualization is to provide the performance guarantees similar to those achieved with middleboxes on specific hardware [3]. For example, considering latency and throughput, it is expected that a VNF implemented on dedicated hardware will have better performance than a function based on pure software implementations [4]. Several efforts aimed at increasing throughput and decreasing delay on pure software approaches, either using minimalistic VMs, such as ClickOS [5], or using libraries and drivers, such as Intel's Data Plane Development Kit (DPDK) [6]. However, these approaches rely on optimizations for VMs running on General Purpose Processors (GPPs). The deployment of VMs generates high consumption of resources and possibly overheads caused by the operating system over the hypervisor, often not providing the required performance and throughput for VNFs. Therefore, hardware acceleration may still be required, in tandem with software solutions, to perform and accelerate network functions.

In this paper we propose VNFAccel, a platform for accelerating the execution of VNFs in heterogeneous hardware-software infrastructures. Our platform relies on the management and allocation of VNFCs upon heterogeneous architectures comprised of Field Programmable Gate Arrays (FPGAs) and GPPs, observing the potential provided by FPGAs for the feasible execution of VNFs (or, in a fine grain, VNFCs) [7]. Our main contributions are: *(i)* the presentation of guidelines and discussion of properties of VNFCs to help select the appropriate substrate for each VNFC, based on the advantages and disadvantages of FPGA and GPP for different processing conditions; *(ii)* a platform for the deployment and management

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, the Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS) and the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

of network functions, which allocates VNFCs on a hybrid GPP-FPGA infrastructure; *(iii)* the analysis of two use cases, firewall and DPI, demonstrating the modularization of these VNFs. Our implementation prototype runs on a NetFPGA device built upon Xilinx EDK [8], and experimental results comparing the performance of VNF Components executing on the FPGA and on pure software solutions are also presented.

This paper is organized as follows. Section II presents a brief background on NFV and FPGA acceleration. Section III defines guidelines for the development of VNFCs and discusses the main design aspects of the VNFAccel platform. Section IV presents our prototype implementation and experimental results, while Section V reports relevant related work. Finally, Section VI presents the concluding remarks.

II. Background

In this section we present the background related to the basic concepts of NFV, focusing specifically on VNFs and VNFCs. Next we discuss the characteristics of FPGAs that can be used to accelerate network functions.

A. Network Function Virtualization

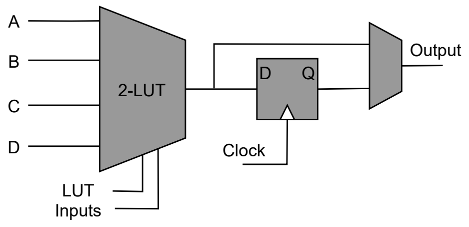
Network Function Virtualization (NFV) is a paradigm proposed and standardized by the European Telecommunications Standards Institute (ETSI) [1]. Its main objective is to offload network functions from physical devices developed and manufactured for only a single specific purpose (e.g., firewalls, DPI, NAT) and to execute them in a virtualized infrastructure. The proposed ETSI architecture is divided into Virtualized Network Functions (VNFs), NFV Infrastructure (NFVI), and NFV Management and Orchestration (NFV MANO).

In particular, VNFs consist in the virtualization of basic functions in Commercial Off-The-Shelf (COTS) servers, thus lowering costs and turning the infrastructure more amenable to improvements and updates. Network functions typically have different objectives inside the network, either monitoring incoming and outgoing traffic based on different rules or remapping IP addresses of packets while they are in transit [9]. These specific functions can be decomposed into generic and less complex components, enabling the sharing of modules between VNFs. These internal components of VNFs, responsible for a subset of a VNF functionality, are called Virtualized Network Function Components (VNFCs) [2].

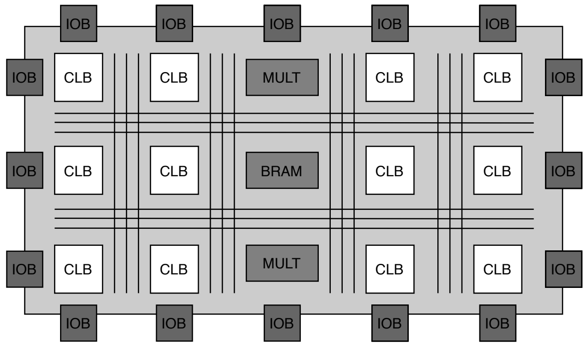
B. FPGA Acceleration

Development in FPGA starts with the design of the requested hardware and logic in a Hardware Description Language (HDL) that is synthesized by a proper tool, generating a specific bitstream to program the FPGA to perform the designed logic. The complete process is composed of synthesis, mapping, placement, routing, and finally generating the configuration bitstream that programs the device. This process can take from several minutes to hours, depending on the design complexity.

The technology behind an FPGA is a set of generic logic blocks interconnected in a programmable switch fabric. These blocks are programmed as boolean logic functions using lookup-tables (LUTs), which consist of multiplexers used to perform combinational functions. To compute one function with K inputs, a 2^K-to-1 multiplexer is necessary to select one bit out of 2^K. This arrangement results in a K-input LUT, the most common on standard FPGAs being 4 to 6-input LUTs. LUTs are grouped in Configurable Logic Blocks (CLBs) alongside with flip-flops to form the basic block of FPGAs, illustrated in Fig. 1a. Additionally to these blocks, FPGAs commonly have memory (e.g., Block RAM or BRAM), input/output blocks (IOBs), multipliers, and high-speed interfaces, as represented in Fig. 1b, but varying from device to device. We refer readers interested in further details on FPGA architecture to [10].



(a) CLB Organization

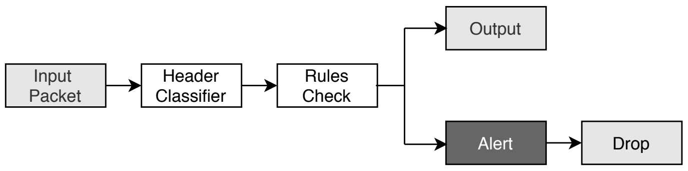


(b) Basic FPGA architecture

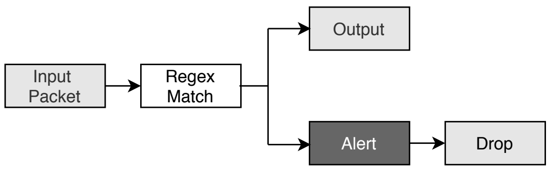
Fig. 1. FPGA Organization

With this programmable logic, an FPGA achieves one of the most important characteristics demanded by NFV, which is the reconfiguration on demand. This avoids the inflexibility present in ASICs with superior performance compared to GPPs for functions that can be efficiently mapped to this fabric, as will be discussed in Section III.

It is important to note that the reconfiguration of an FPGA with a new bitstream takes time, although much less than generating the bitstream itself, and makes the chip idle, thus adding complexity in the management and orchestration of a network. Another important metric is power efficiency, in which FPGAs may achieve better efficiency when compared to GPPs. However, this gain depends on the specific function and hardware. Moreover, these gains are smaller than those obtained with an ASIC, since the reconfigurability introduces costs. Since one of the NFV's goals is to reduce power usage [11], FPGA-based VNFs can consider this metric as well.



(a) Firewall modules.



(b) DPI modules.

Fig. 2. Examples of VNF modules

III. VNFAccel: A VNF Platform Based on FPGA Acceleration

A platform for the execution of VNFs needs to provide strict performance guarantees. Besides, it must support different combinations of network functions. We advocate that FPGA, a programmable hardware solution, can meet these performance and the flexibility requirements [12]. In this section, we first present use cases to demonstrate the modularization of VNFs and common functionalities across different functions. Next, we outline development guidelines and properties to help determine which modules should execute in hardware or software. We then discuss how the FPGA reconfiguration features can be used to provide VNFC flexibility. Finally, we present our FPGA-based platform, VNFAccel, to support the operation, management, and control of VNFCs in a heterogeneous infrastructure.

A. VNF Modularization Use Cases: Firewall and DPI

Typically, VNFs can be divided into basic modules called Virtualized Network Function Components (VNFCs). Fig. 2a shows the components that comprise a firewall function. The modules in light grey execute generic actions present in many functions, such as packet forwarding. Modules for communication and that do not necessarily process packets are represented in dark grey. Finally, some components are too specific and have a minor chance of being shared among VNFs, represented in white, and are classified as VNF-specific components. Further, Fig. 2b shows the modules of a regex-based DPI, using the same color conventions for the blocks. This exhibits the common components between different functions.

This modularization promotes flexibility and reusability since components can be shared between functions. Managing these components in a heterogeneous network can reduce resource and time consumption since functions tend to need few specific modules. These modules must be well confined to detach their service from generic modules, obtaining independent functionalities. Fig. 3 presents a possible combination of VNFCs to achieve a merged firewall and DPI functionality simultaneously.

The ability to modularize the implementation of VNFs using common building blocks allows instantiating new functions by combining independent, fine-grain modules. Furthermore, these modules may be possibly allocated in multiple Points of Presence (PoPs) in the network and reused by different functions. Ultimately, this promotes deployment flexibility, since functions are not dependent on fixed configurations.

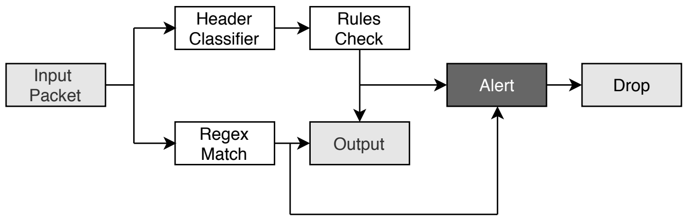


Fig. 3. Example of VNF combination through modules.

B. Guidelines for the Development of VNFCs

In this section, we discuss *guideline* criteria to decide if a VNFC is more appropriate for execution on an FPGA or on a GPP. Each module needs careful analysis before development to exploit the advantages of hardware acceleration, avoiding a mismatch between application and device properties. The list below summarizes the main criteria to be considered. These points are complementary to each other in the definition of the proper device for each VNFC. These guidelines consider experiences from previous work such as [7], [13]–[15].

1) Data-flow vs. control-flow intensity: Each module processes packets employing different operations to accomplish its specific service. Operations that involve control and conditional constructs, without data-intensive computation, are more suitable for GPPs. Data-flow intensive operations, especially with high fine-grained parallelism, are more suitable for FPGAs.

2) Priority: In a network function, modules may have priorities regarding the provided service. VNFCs with low priority and usage do not demand FPGA acceleration and could result in resource waste. Conversely, VNFCs with high priority and demand could not have enough throughput in GPPs.

3) Memory usage: Compared to GPPs, FPGAs typically have less internal memory, and it is not transparent as caches. Therefore, tasks with regular and predictable access patterns, preferably in a streaming-like behavior, can be more easily mapped to FPGA memory resources.

4) Development complexity: Although high-level synthesis has gained popularity in the past years to improve the design productivity [16], [17], FPGA development is still more cumbersome and less mature than software development. Therefore, development costs, especially for highly-complex functions, should be accounted for when choosing FPGAs over GPPs.

C. Reconfiguration Layers in FPGA-based VNFCs

Development of FPGA modules seeks performance and flexibility using the main feature of FPGAs: reconfigurability. However, there is a tradeoff between high performance and flexibility to changes. When developing an FPGA-based VNFC, different layers of abstraction and configuration can be placed over the FPGA hardware and these affect the adaptability of the VNFC to changing demands. For example, in a DPI or firewall function, rules may change over time, and a VNFC should accommodate these changes quickly and efficiently.

In this section, we discuss the two main paradigms to support VNFC adaptation, and how these affect development and management practices. To compare performance data, we analyzed implementations of a regex matching module, present in a typical DPI function. These modules commonly use a Nondeterministic Finite Automaton (NFA) or Deterministic Finite Automaton (DFA) to perform rule checking.

1) Fixed-configuration VNFC: These VNFCs completely embed the desired functionality in the FPGA bitstream. As shown in Fig. 4a, once the FPGA configuration is applied over the FPGA hardware, the VNFC is both instantiated and fully configured for operation. Modifications in fixed-configuration VNFCs, however, require resynthesizing a new bitstream. As the compilation of complex functions may demand significant time, in the order of several minutes or even hours, this can introduce undesirable reconfiguration latency. Nevertheless, since these modules are developed and configured directly into the FPGA, they usually provide high performance.

Fixed-configuration approaches of the regex module as [18] and [19] rely on one-input character matching. Using different techniques of compression, they can achieve throughputs up to 6 Gbps. Other proposals as [20] use multi-character inputs, which may cause state explosion as the state combinations increase, consuming more resources. However, they reach 11 Gbps of concurrent throughput. For these implementations, any required change in the ruleset requires synthesizing a new bitstream, which delays the function's adaptation.

2) Overlay VNFC: Overlays avoid resynthesis by decoupling instantiation and configuration. VNFCs are instantiated with the FPGA configuration, but their functionality is fully defined only when the overlay configuration is applied, as shown in Fig. 4b. This is typically done with user-accessible internal memories, such as BRAMs, to configure the VNFC operation. Thus, the VNFC can have its operation changed by rewriting the contents of these BRAMs, without synthesizing a new bitstream. The time required for this type of reconfiguration depends on the amount of transferred data, but it is typically in the order of dozens of milliseconds. This is orders of magnitude faster than synthesizing a new FPGA configuration.

In the case of DPI functions, the ruleset being checked depends exclusively on an interchangeable memory. However, the use of FPGA memory increases along with the complexity and size of rulesets. Based on [21], we implemented a module using this method, explained in Section IV, which achieves 0.8 Gbps of throughput. If higher performance is required, both approaches can be combined: it is possible to maintain fixed-configuration VNFCs previously synthesized for popular rulesets, and use the overlay version when unforeseen rulesets are requested, or until when synthesis is completed.

The throughput and the reconfiguration time of both approaches demonstrate the tradeoff between performance and flexibility. The fixed-configuration method presents a higher throughput. However, the downtime of overlays is much lower. Developers should analyze the project priorities in these terms. The need for post-deployment adaptation should also be considered. The generic and communication modules in Fig. 2, in light and dark grey, respectively, may not need constant modification. However, specific modules, represented in white, are likely to require frequent adjustments.

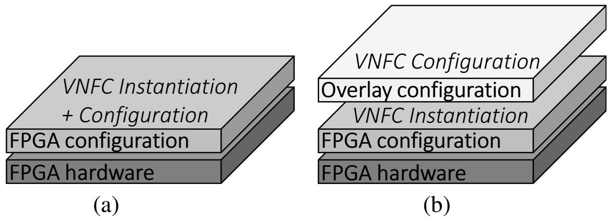


Fig. 4. Fixed-configuration (a) and overlay (b) VNFCs.

D. VNFAccel Platform

VNFAccel can be used to manage VNFCs in a modular and heterogeneous infrastructure. It is composed of, as seen in Fig. 5, VNFC Allocation Controller, VNFC repositories, and NFV Points of Presence (PoPs). FPGA and GPP devices, linked by a high-speed local interface such as PCIe, constitute an NFV PoP. Different VNFs can be instantiated into these PoPs, with VNFCs distributed in the infrastructure. Thus, multiple PoPs could allocate resources to provide one VNF service. VNFCs are allocated on GPP or FPGA depending on the designed modules. The controller entity manages the service provided by the PoPs, issuing instantiation and configuration commands. The system maintains a repository detached from the controller, where the developed VNFCs for GPP or FPGA are stored. In addition to performing VNFC allocation, the GPP has a management role, receiving controller commands, requesting VNFCs from repositories, updating FPGA memories, and operating FPGA reconfiguration as needed.

These PoPs are not necessarily equal in terms of availability and system configuration. Since the fragmentation of VNFs and management of VNFCs are the focus of VNFAccel, PoPs have to provide flexibility in terms of resources, possibly using different alternatives of GPPs and FPGAs, seeking enhancement of memory, resources, performance or throughput. Thus, the possibilities for the allocation of VNFCs are improved, creating a more comprehensive and adaptable system.

Once a VNF is requested to the VNFC Allocation Controller, a sequence of events perform the deployment of VNFCs at an NFV PoP. The diagram presented in Fig. 6 illustrates these events. The controller sends a deployment command to the GPP of a PoP, indicating the VNFCs required and the target device in which to deploy. The GPP requests the VNFC implementations and the VNFC configuration layer data (e.g. rulesets) from the repositories. The GPP allocates the downloaded images and instantiates the VNFCs in the corresponding devices, applying the required configurations. Finally, the GPP confirms the VNF deployment, and the network traffic can be processed.

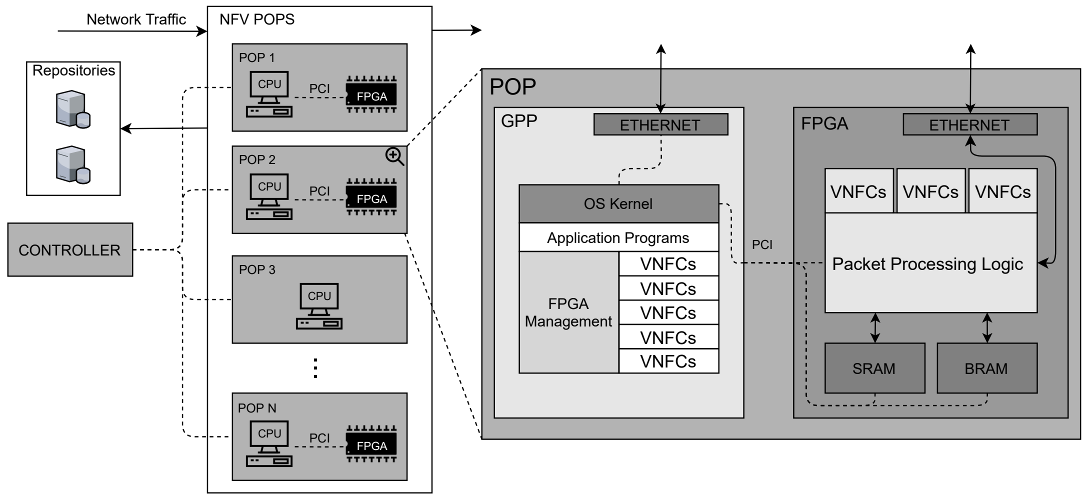


Fig. 5. VNFAccel platform for accelerating the execution of VNFs in heterogeneous hardware-software infrastructures.

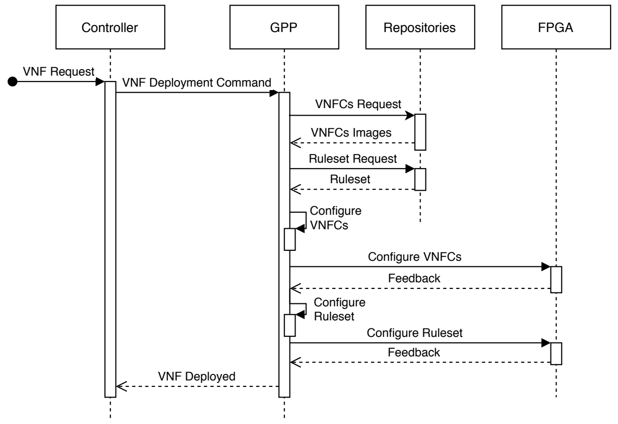


Fig. 6. Sequence diagram for VNFC allocation.

IV. Prototype Implementation and Experimental Results

In this section, we first detail the implementation of the use cases that illustrate the development of VNFCs for FPGA. We then evaluate the solutions in terms of: *(i)* throughput compared to software implementations; *(ii)* reconfiguration and downtime; *(iii)* instantiation latency; and *(iv)* FPGA resource usage.

A. Implementation and Experimental Setup

To host the FPGA components of VNFAccel, we relied on the NetFPGA project [22], a flexible open-source platform for research. This system operates as an abstraction layer over FPGA, shown in Fig. 7. The platform manages the Ethernet and PCIe communications, and provides the mapping of SRAM and BRAM addresses. These resources handle the I/O interface and memory access, facilitating the implementation of network functions. This platform is largely built upon Xilinx EDK [8], a development environment for embedded systems which enables the development of individual modules for FPGA. It provides a simple way of combining these components, enabling the straightforward implementation of function components.

In our proof-of-concept prototype[1] we developed the components of a firewall and a DPI function to be integrated with the NetFPGA platform, obtaining VNFCs for allocation on an FPGA. We implemented each component seeking module reusability and flexibility to changes, following the organization presented in Fig. 3. Moreover, the core matching VNFCs of each VNF were implemented as reconfigurable overlays, providing the decoupled configuration discussed in Section III-C.

The modules were implementated in Verilog and followed project standards of NetFPGA. The FPGA implementations were executed on a NetFPGA-1G-CML [23] featuring a Xilinx Kintex-7 XC7K325T FPGA. To synthesize the complete functions, we used Xilinx ISE 14.6 and Vivado Design Suite 15.6, the standard vendor tools for the employed FPGA.

To compare the results between FPGA and GPP, software functions were executed on an Intel Core i7-7500U@2.70GHz running Ubuntu 18.04.4 LTS directly over CPU, without any VMs or Containers, to emulate a minimum overhead scenario. The firewall implementation is compared to iptables [24], a utility program that configures packet filter rules of the Linux kernel firewall. The FPGA DPI is compared to Snort [25], an open-source intrusion detection and prevention system. The DPI rules used in the FPGA implementation were based on the Snort ruleset, and the Firewall ruleset used combinations of IP/Port of computer or router addresses to ensure stimulation of the functions.

B. Evaluation

1) Throughput: Fig. 8a and Fig. 8b present respectively the results for the DPI and firewall performance, comparing FPGA and software-based implementations with an increasing number of rules. For both, we evaluate the throughput with a network bandwidth monitor. Experiments with snort had a standard deviation from *4.8* to *15.6*, and with iptables from *6.1* to *14.3*.

[1]Source code of VNFAccel as well as the VNF implementation in HDL are publicly available in: https://github.com/FBachini/VNFAccel

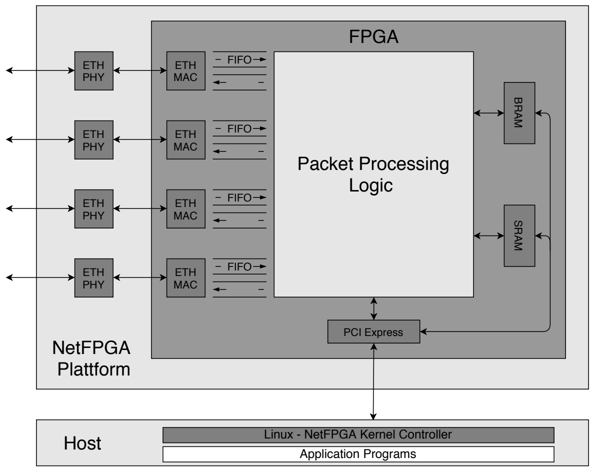
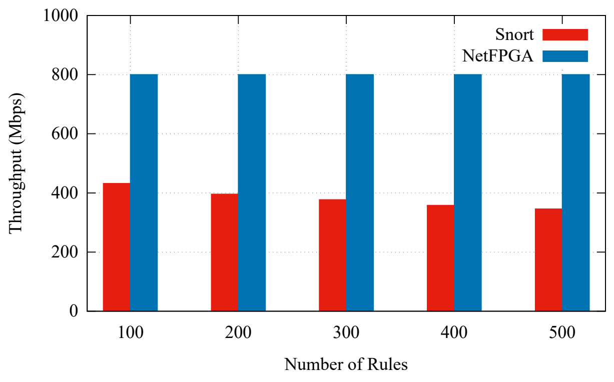
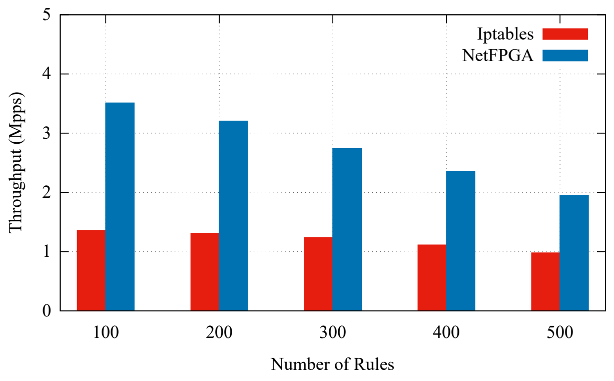


Fig. 7. Architecture of a NetFPGA project.



(a) DPI throughput analysis in bits per second (bps).



(b) Firewall throughput analysis in packets per second (pps).

Fig. 8. Performance Analysis and Comparison

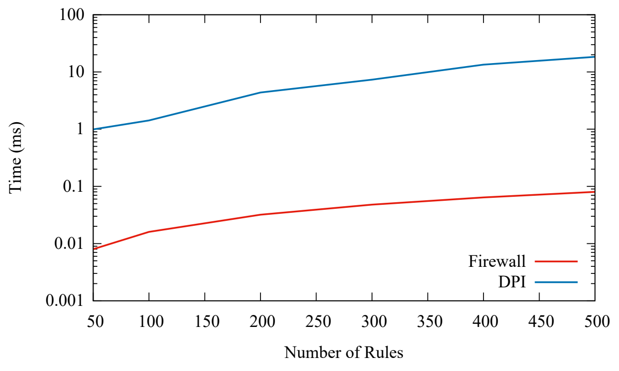


Fig. 9. Memory Reconfiguration Time

This analysis exhibits a better throughput performance presented by the FPGA approach over pure software solutions. In the study of the firewall and DPI use cases, both functions analyze multiple fields of the packet header in parallel as a single word. This characteristic makes them suitable for allocation at the FPGA, as mentioned in Section III. The translation to a finite automaton [21] ensures the DPI implementation maintains its throughput even for large rulesets, although at the cost of increased memory use, as presented in Table I. The firewall implementation, on the other hand, performs sequential processing of rules, affecting its performance as the ruleset grows. As a result, however, its resource usage is low, meaning that parallelism can be exploited over multiple VNFC instances if higher throughput is necessary. The memory and FPGA resources used will be evaluated in Section IV-B4.

2) Reconfiguration and Downtime: Both firewall and DPI functions operate rule checking on packets, employing a ruleset determined by the network operator through the VNFC Allocation Controller. The modifications after deployment expected for these functions comprise, particularly, ruleset changes. As the FPGA is idle while rules are allocated, it is important to measure the required time to apply new rulesets. Both implementations use rulesets stored in FPGA memory, which can be modified, as required, through PCIe communication. The time consumption of the DPI and firewall implementations for ruleset storage on FPGA memory is presented in Fig. 9. The results are presented on a log scale due to the large difference in memory utilization of both functions.

This evaluation exhibits the downtime of FPGA when modifications are required. The firewall time consumption for performing memory reconfiguration is notably low compared to the number of rules stored, increasing linearly as the ruleset increases. The DPI time consumption begins very low, increasing proportionally with the size of DFA states. The difference in the complexity of the rules causes this notable disparity of reconfiguration time. Firewall rules are simple, addressing packet fields to actions. DPI rules are complicated regexes, generating multiple DFA states to perform rule checking.

3) Instantiation Latency: To measure the time to deploy VNFs as described in Section III-D, we analyzed the instantiation time of the firewall and DPI implementations. We assume that the bandwidth between controller, repositories, and NFV PoP are equal and provide 100 Mbps. The bandwidth for a SelectMAP FPGA bitstream reconfiguration is 3.2 Gbps for the employed FPGA family [26]. Previous FPGA-enabled NFV platforms do not emphasize decoupling VNFC instantiation and configuration. As such, this analysis can be viewed as a comparison between VNFAccel and these platforms.

Assuming the worst-case scenario, in which all operations are sequentially performed as shown in Fig. 6, the time for VNFs instantiation is presented in Fig. 10. We analyzed the four major steps of the VNF deployment for FPGA: the download of the bitstream and of the required ruleset, and the configuration of the FPGA and of the VNFCs, i.e., the overlay configuration with the desired ruleset.

For software solutions, the time to transmit rules to a PoP ranges from 23μs to 212μs for the iptables firewall and from 278μs to 2778μs for the Snort DPI, for the same evaluated ruleset sizes. In a scenario that the PoP has a cached FPGA configuration file for the functions, these times would be comparable to those observed for FPGAs. The software instantiation time presents a significant difference between Snort and iptables ruleset size, reinforcing the disparity of function complexity exhibit in Section IV-B2. The snort ruleset size varies between 30 to 280 kB while the iptables ruleset varies from 2 to 28 kB.

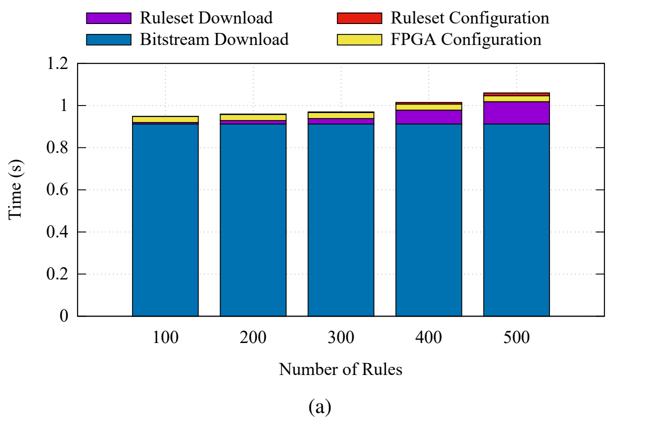
The FPGA instantiation time shows the most time-consuming operations are the bitstream download and FPGA configuration. NetFPGA utilizes several FPGA elements to create the abstraction layer, generating bitstreams with 8 to 12 MBytes, regardless of the complexity of the function. Nevertheless, the time for bitstream download and configuration of FPGA combined does not reach 1 second. If necessary, these times can be reduced by caching frequent VNFCs in the PoPs. This is relevant in scenarios where frequent context changes among multiple VNFs take place. The ruleset size does not affect noticeably the firewall instantiation time due to the relative small size of rulesets. In the DPI implementation, however, the ruleset size influences the instantiation time. Still, the combination of ruleset download and configuration time is low compared to the bitstream download.

The total instantiation time shows the contrast between the two reconfiguration methods discussed in Section III-C. A fixed-configuration VNFC would only require the bitstream download and the FPGA configuration, while an overlay implementation requires all four steps. The times shown in Fig. 10, however, indicate that the fixed configuration does not provide a significant reduction in instantiation time compared to overlays. On the other hand, if a new ruleset is required, the overlay VNFC can accommodate it in a few milliseconds, while a fixed approach requires from minutes to hours to perform a full synthesis and bistream generation.

4) Resources Usage: We evaluated the use of resources with the increase of the maximum ruleset size in DPI and firewall implementations. Table I presents the use of CLBs, and Block RAM (BRAM), the main internal memory component for FPGAs.

This analysis exposes the costs in memory use of the overlay approach. The variation on CLBs is very low with the increase of the ruleset, due to the static logic of the implementations. However, BRAM consumption exhibits the modules' dependency on memory. The DFA-based DPI implementation experiences a high expansion in the number of states, increasing memory consumption. Conversely, the firewall implementation stores its rules directly, thus consuming less memory, although larger rulesets will still impact BRAM usage.

For comparison, Table II presents the memory usage of the software (RAM) and FPGA (BRAM) solutions, as we increase the ruleset size. As FPGA embeds functionality directly on the hardware, it can provide very efficient memory use, allowing the storage of numerous rules within its internal memory. For the Firewall case, the minimal memory is limited to the size of one single BRAM.



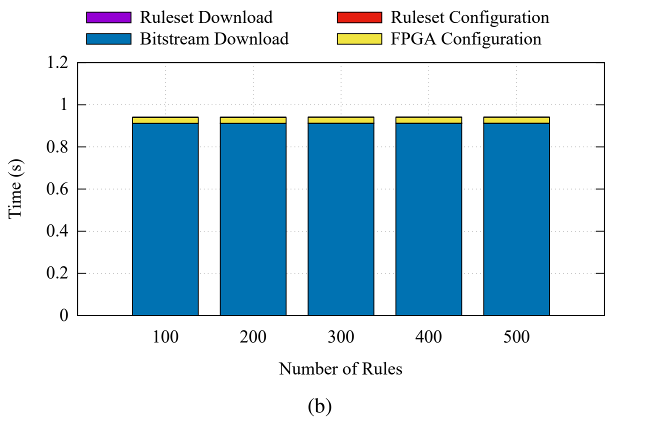


Fig. 10. Instantiation Time of DPI (a) and Firewall (b) Implementations

TABLE I
FPGA RESOURCES USAGE

	Maximum Ruleset size	CLBs	BRAM
Firewall DPI	50	6545 (12.8%) 8377 (16.4%)	1 (0.2%) 37 (8.3%)
Firewall DPI	100	7641 (14.9%) 8407 (16.5%)	1 (0.2%) 86 (19.3%)
Firewall DPI	200	7876 (15.1%) 8449 (16.5%)	1 (0.2%) 128 (28.7%)
Firewall DPI	300	8182 (15,4%) 8502 (16,6%)	1 (0.2%) 239 (53.7%)
Firewall DPI	400	8253 (16.1%) 8554 (16.7%)	1 (0.2%) 353 (79.3%)
Firewall DPI	500	8537 (16.7%) 8612 (16.9%)	1 (0.2%) 438 (98.4%)

TABLE II
CPU AND FPGA MEMORY USAGE

	Max. ruleset size	SW RAM (kB)	FPGA BRAM (kB)
DPI	50	154844	166.5
Firewall		1257	4.5
DPI	100	156128	387
Firewall		1347	4.5
DPI	200	157176	576
Firewall		1419	4.5
DPI	300	158416	1075.5
Firewall		1484	4.5
DPI	400	162740	1588.5
Firewall		1554	4.5
DPI	500	165756	1971
Firewall		1596	4.5

V. RELATED WORK

Several VNFs modularization and management platforms have been proposed in the past years. In this section, we summarize recent work on VNFC platforms and implementations.

The OpenBox project [27] is a software-defined platform for development and management of VNFs. OpenBox exploits modularization, offering the abstraction of complete functions while utilizing VNFCs in the real processing traffic. The platform combines and deploys the components upon virtual instances. Similarly to our proposed infrastructure, it employs VNFCs to compose different VNFs, although not exploring hardware acceleration. The platform and our work are complementary in the management and deployment of VNFCs.

P4NFV [28] is an NFV architecture with a flexible data plane, comprising software network functions as well as programmable hardware for VNF deployment, using the P4 language. It proposes an abstraction layer above a hybrid physical layer, containing commodity servers and programmable hardware switches. However, the prototype analyzes only GPP implementations. This work observes the need for heterogeneous fabrics to support NFV, complementing our research in hybrid platforms for VNFs deployment, in the P4-based context.

CoNFV [29], a configurable NFV system, proposes a controller to a heterogeneous and scalable NFV platform. FPGA and processor are dynamically and separately configured by an autonomous controller. VNFs are allocated either on VMs upon processors, with low throughput or on FPGA, with high throughput. The controller migrates, creates, and destructs functions between those options in real-time to meet functional and network needs. There are other proposals to enable NFV upon FPGA, such as VirtManager [30], a Virtual Open Systems' technology that enables FPGA virtualization. The system operates by dividing the FPGA in accelerators, controlled by a context manager. This increases the management and control over the FPGA, and also decreases reconfiguration time.

The hybrid platforms discussed above [29], [30], similarly to VNFAccel, employ the combination of FPGA and GPP to support the execution of network functions. However, those implementations do not exploit the advantages of each device for each VNFC operation. Compared to these related work, VNFAccel can improve the development and performance of VNFs for FPGA or GPP through the proposed guidelines and the focus on efficient reconfiguration of FPGA-based VNFCs.

TABLE III
RELATED WORK

	Implementation	Research Papers
DPI	NFA	[31]
	DFA	[32], [33]
Firewall	Comparator-Based	[34], [35]
VNF Infrastructure	GPP	[27], [28]
	GPP-FPGA	[29], [30]

Regarding specific VNF implementations, previous work focused on efficiency and performance. In FPGA-based DPI implementations, Non-deterministic Finite Automata (NFAs) are widely used due to the multiple active states, suitable to parallel architectures. In [31], NFAs are mapped directly to VHDL, generating highly optimized multi-character matching in FPGA. Another approach to the DPI function is DFA implementations, permitting a more straightforward approach to update the FPGA with different expressions without reconfiguration. The feasible execution of ruleset reconfiguration on FPGA using DFA is observed in [32], [33].

The straightforward method to implement an FPGA-based firewall is to use a comparator-based engine, as used in our implementation, evaluating all rules on all packets. Some works use simple architectures with a sequential evaluation of rules [34], [35], attaining low resource occupation, and low power. These efforts exploit FPGA functionalities seeking the best performance of each VNF. Our current implementation does not aim at maximum performance, but instead at the demonstration of the VNFCs integration in our heterogeneous platform. Moreover, our functions exhibit the positive effects of flexible VNFCs (i.e. using FPGA overlays) in a heterogeneous system.

VI. CONCLUDING REMARKS

In this work, we proposed a platform for accelerating VNF execution in heterogeneous infrastructures. We presented use cases of the VNF modularization, identifying the components used for two network functions. This fragmentation led to the analysis of development and allocation properties of VNFCs. We discussed the design of the heterogeneous FPGA-GPP platform to manage and deploy VNFs through the allocation of VNFCs. Finally, we developed use cases for FPGA, evaluating the performance of two accelerated function compared to pure software solutions. We observed the performance improvement when VNFCs are allocated to the FPGA.

Future work includes the implementation of more VNFCs in both FPGA and GPP context, evaluating and expanding our proposed guidelines. Additionally, the analysis of power consumption, latency of the VNF execution, and resource utilization. Furthermore, we intend to investigate the placement of VNFCs inside the heterogeneous infrastructure, considering different hardware constraints. Other improvements include the implementation of a cache inside the NFV PoPs, which could maintain frequent configurations to decrease the instantiation time.

References

[1] NFV White Paper, "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. Issue 1," Oct. 2012.

[2] ETSI, "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV," Aug. 2018.

[3] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[4] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[5] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, pp. 459–473. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/martins

[6] I. Cerrato, M. Annarumma, and F. Risso, "Supporting fine-grained network functions through intel dpdk," in *2014 Third European Workshop on Software Defined Networks*, 2014, pp. 1–6.

[7] G. S. Niemiec, L. M. S. Batista, A. E. Schaeffer-Filho, and G. L. Nazar, "A survey on fpga support for the feasible execution of virtualized network functions," *IEEE Communications Surveys Tutorials*, vol. 22, no. 1, pp. 504–525, 2020.

[8] Xilinx. (2020) Xilinx Platform Studio - Embedded Development Kit (EDK) Description. [Online]. Available: https://www.xilinx.com/products/design-tools/platform.html

[9] L. Bondan, M. F. Franco, L. Marcuzzo, G. Venancio, R. L. Santos, R. J. Pfitscher, E. J. Scheid, B. Stiller, F. De Turck, E. P. Duarte, A. E. Schaeffer-Filho, C. R. P. d. Santos, and L. Z. Granville, "Fende: Marketplace-based distribution, execution, and life cycle management of vnfs," *IEEE Communications Magazine*, vol. 57, no. 1, pp. 13–19, 2019.

[10] I. Kuon, R. Tessier, and J. Rose, *FPGA Architecture: Survey and Challenges*. Now Foundations and Trends, 2008.

[11] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc)," *IEEE Network*, vol. 28, no. 6, pp. 18–26, 2014.

[12] C. Kachris, G. Sirakoulis, and D. Soudris, "Network function virtualization based on fpgas:a framework for all-programmable network devices," 2014.

[13] A. Kamaleldin, A. Mohamed, A. Nagy, Y. Gamal, A. Shalash, Y. Ismail, and H. Mostafa, "Design guidelines for the high-speed dynamic partial reconfiguration based software defined radio implementations on xilinx zynq fpga," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1–4.

[14] J. Cong, Z. Fang, Y. Hao, P. Wei, C. H. Yu, C. Zhang, and P. Zhou, "Best-effort fpga programming: A few steps can go a long way," 2018.

[15] J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, "Features, design tools, and application domains of fpgas," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, pp. 1810–1823, 2007.

[16] Intel. (2020) High-Level Synthesis Compiler - Intel HLS Compiler. [Online]. Available: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html

[17] Xilinx. (2020) Vivado High-Level Synthesis. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html

[18] H. Kim and K.-I. Choi, "A pipelined non-deterministic finite automaton-based string matching scheme using merged state transitions in an fpga," *PLOS ONE*, vol. 11, no. 10, pp. 1–24, 10 2016. [Online]. Available: https://doi.org/10.1371/journal.pone.0163535

[19] M. Becchi and P. Crowley, "Efficient regular expression evaluation: Theory to practice," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 50–59. [Online]. Available: https://doi.org/10.1145/1477942.1477950

[20] Y. Yang and V. Prasanna, "High-performance and compact architecture for regular expression matching on fpga," *IEEE Transactions on Computers*, vol. 61, no. 7, pp. 1013–1025, 2012.

[21] M. Becchi and P. Crowley, "A hybrid finite automaton for practical deep packet inspection," in *Proceedings of the 2007 ACM CoNEXT Conference*, ser. CoNEXT '07. New York, NY, USA: Association for Computing Machinery, 2007. [Online]. Available: https://doi.org/10.1145/1364654.1364656

[22] NetFPGA. (2020) Netfpga platform page. [Online]. Available: https://netfpga.org/

[23] Digilent. (2020) NetFPGA-1G-CML [Reference.Digilentinc]. [Online]. Available: \url{https://reference.digilentinc.com/reference/programmable-logic/netfpga-1g-cml/start}

[24] NetFilter. (2020) Iptables packet filtering. [Online]. Available: https://www.netfilter.org/projects/iptables/

[25] Snort. (2020) Snort - open source intrusion prevention system. [Online]. Available: https://www.snort.org/

[26] Xilinx. (2013) Partial Reconfiguration User Guide. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/ug702.pdf

[27] V. Sateesh, C. Mckeon, J. Winograd, and A. DeHon, "Pipelined parallel finite automata evaluation," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, 2019, pp. 108–116.

[28] M. He, A. Basta, A. Blenk, N. Deric, and W. Kellerer, "P4nfv: An nfv architecture with flexible data plane reconfiguration," in *2018 14th International Conference on Network and Service Management (CNSM)*, 2018, pp. 90–98.

[29] X. Zhang, X. Shao, G. Provelengios, N. K. Dumpala, L. Gao, and R. Tessier, "Scalable network function virtualization for heterogeneous middleboxes," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 219–226.

[30] M. Paolino, S. Pinneterre, and D. Raho, "Fpga virtualization with accelerators overcommitment for network function virtualization," in *2017 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2017, pp. 1–6.

[31] Y.-H. E. Yang, W. Jiang, and V. K. Prasanna, "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*, ser. ANCS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 30–39. [Online]. Available: https://doi.org/10.1145/1477942.1477948

[32] J. Kastil and J. Korenek, "Hardware accelerated pattern matching based on deterministic finite automata with perfect hashing," in *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, 2010, pp. 149–152.

[33] W. Lin, Y. Tang, B. Liu, D. Pao, and X. Wang, "Compact dfa structure for multiple regular expressions matching," in *2009 IEEE International Conference on Communications*, 2009, pp. 1–5.

[34] F. D. Pereira and E. D. M. Ordonez, "Ssdr - reconfigurable firewall: Reconfiguration model performance," in *2008 4th Southern Conference on Programmable Logic*, 2008, pp. 253–256.

[35] A. P. Antonov, A. S. Filippov, and O. V. Mamoutova, "Next generation fpga-based platform for network security," in *2016 18th Conference of Open Innovations Association and Seminar on Information Security and Protection of Information Technology (FRUCT-ISPIT)*, 2016, pp. 9–14.