A Partition-Aware VNF Placement Methodology for FPGA-Equipped NFVIs

Victor S. Guerra, Gabriel L. Nazar

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{vsguerra,glnazar}@inf.ufrgs.br

Abstract. *In the context of Network Function Virtualization (NFV), Field Programmable Gate Arrays (FPGAs) can be used to reduce bottlenecks introduced by the substitution of dedicated hardware middleboxes by virtualized implementations. The problem of placing Virtualized Network Functions (VNFs) on FPGA-equipped NFV infrastructures, however, imposes additional challenges that require an accurate modeling of the FPGA fabric. More specifically, simultaneous sharing of the FPGA requires careful partitioning of its resources into fixed regions that can be dynamically reconfigure and to which functions can be mapped. In this work, we will demonstrate that accurate modeling of the FPGA partitions into the placement solution is crucial to achieve solutions that are guaranteed to be viable. Experimental results obtained through two Integer Liner Programming models will be used to demonstrate that partition awareness can avoid invalid solutions caused by overestimation of the device capabilities, with a small impact on the number of allocated user requisitions.*

1. Introduction

The Network Function Virtualization (NFV) paradigm has seen a significant increase in popularity over recent years, primarily focusing on substituting specialized hardware, known as middleboxes, with general-purpose hardware. These middleboxes, traditionally implemented with Application-Specific Integrated Circuits (ASICs) and designed for high-performance execution of specific network functions (NFs), are now replaced by more versatile hardware. A shift from specialized to generic hardware typically implies a performance trade-off. To mitigate this, Field-Programmable Gate Arrays (FPGAs) can be employed to offer both high performance and reconfigurability, essential features for NFV. FPGAs demonstrate improved performance compared to general-purpose processors (GPPs) for many relevant NFV applications, primarily due to their ability to implement dedicated acceleration architectures at the hardware level. This feature enables their efficient application in many NFs, such as firewalls, Deep Packet Inspection (DPI), Advanced Encryption Standard (AES), and load balancing.

However, the shared use of FPGAs in an NFV Infrastructure (NFVI) still poses significant challenges [Niemiec et al. 2020]. The placement of Virtualized Network Functions (VNFs), organized as Service Function Chains (SFCs), consists of deciding in which NFVI Point of Presence (NFVI-PoP) each VNF will be executed [Li and Qian 2016, Laghrissi and Taleb 2019, Sun et al. 2022]. Function placement must, therefore consider available resources in each NFVI-PoP and available bandwidth in each link to place multiple SFCs, aiming at maximizing the efficient use of the infrastructure while meeting user requirements. As will be shown in this work, for FPGA-equipped NFVIs, careful modeling of device resources and their organization is fundamental to enable their efficient use.

Traditional FPGA designs assume the whole device will be used by a single application during its entire lifetime, which is reasonable for many FPGA use cases. This assumption hides the long latency of FPGA reconfiguration since the device is seldom reprogrammed. On the other hand, in a typical NFVI in which the number of applications and users potentially exceeds the number of devices, device sharing becomes crucial for cost reduction, in line with the expected benefits of NFV. Due to the long programming latencies, relying solely on time sharing is not practical for continuously operating functions, which are commonly found in NFV, as it introduces long and frequent service interruptions. Fortunately, current FPGAs support Dynamic Partial Reconfiguration (DPR), which allows reprogramming parts of the device while the remaining resources stay operational [Xilinx 2021]. This functionality is very attractive for NFV, as it allows NF instances to be quickly combined on the fly in a single FPGA, with no interruption to other instantiated functions. DPR, however, demands a priori partitioning of the devices' resources into dynamically reconfigurable regions. These regions dictate how many functions can be simultaneously allocated in each device and how many resources will be available for each of them, a limitation that is often not accurately modeled in previous work.

Thus, in this work we propose a partition-aware VNF placement approach, able to place VNF instances not only on NFVI-PoP but also in specific FPGA partitions within each NFVI-PoP. Additionally, we demonstrate that a simplified modeling of FPGA resources, which is often employed, can lead to function allocations that are not feasible when the device-specific partitioning restrictions are taken into account. The main contributions of this work can be summarized as follows:

- An Integer Linear Programming (ILP) model for SFC placement that takes into account individual FPGA partitions, aiming at maximizing the instances' total value, which is related to their use of resources, used by the placed SFCs in a network with FPGA-equipped NFVI-PoPs.
- A novel optimistic algorithm to determine whether sets of allocated VNFs can possibly fit within a certain FPGA model. With this algorithm, one can determine if allocations made by partition-unaware approaches are possibly viable or certainly not feasible.
- An experimental evaluation of the proposed ILP-based approach with diverse network sizes, FPGA models, and NFs, demonstrating its applicability and scalability. Experimental results are also compared to those obtained by optimistic partition-unaware models, indicating that partition awareness is not only necessary, but has a small impact on the total value of allocated SFCs.

The remainder of this work is structured as follows. Section 2 presents related work on FPGA use in NFV, with emphasis on function placement. Section 4 presents the proposed ILP model for partition-aware placement. Section 5 describes the algorithm used to detect invalid function allocations in FPGAs. Section 6 presents and discusses the experimental results. Finally, section 7 concludes this work.

2. Related Work

Works that focus on network function placement in PoPs are well described in the literature. The approaches are different and emphasize diverse ways to achieve the objectives, which usually aim to minimize the amount of resources used or place a higher number of functions in a network. The usage of FPGAs in these works is not so well explored because of the variety of distinct hardware components in heterogeneous infrastructures in which the VNFs can be deployed.

There are works that aim to reduce latency, like [Xu et al. 2018], which introduces CoNFV, a framework that combines the cloud and end-hosts to achieve low latency, low cost, and high flexibility simultaneously, or [Zhang et al. 2017] that focuses on efficiently placing VNF chains across the network and effectively scheduling requests to service instances to minimize average response latency.

Numerous techniques have been developed to address complex problems and yield satisfactory results. However, for the problem at hand, the most popular and effective approach is the use of Integer Linear Programming (ILP) and its variations. Researchers such as [Paganelli et al. 2021], [Dong et al. 2021], and [Sharma et al. 2020] have successfully employed ILP and its variations to solve similar problems. Simple heuristics are often used, as [Qi et al. 2019] applying greedy and multi-stage graph algorithms. Some works use more complex approaches like [Solozabal et al. 2020] using Neural Combinatorial Optimization theory for VNF placement, demonstrating competitive results.

Observing the majority of works in the area focuses on improving the deployment of functions to mobile network infrastructure. Mobile context is mainly focused as its demand is expanding uninterruptedly and is strictly constrained, keeping low latency and high throughput. Some works that target this objective are [Leivadeas et al. 2019] and [Doan et al. 2023].

Given all the possible hardware combinations that can be used in a heterogeneous infrastructure, there are some authors who have chosen FPGA, using its flexibility and high efficiency as motivation. [Niemiec et al. 2020], [Sharma et al. 2020] and [Lopes et al. 2023] demonstrate good examples of how it can be employed, extracting the best of this type of hardware and justifying the reason it is a feasible solution.

Surveys in this area cover a wide range of approaches to NFV placement, covering more topics about it, such as recent network function orchestration frameworks, highlighting the advantages and disadvantages of different placement strategies, as shown by [Li and Qian 2016], [Laghrissi and Taleb 2019] and [Santos et al. 2021].

As can be noted, previous works on VNF and SFC placement have addressed important variations of the problem. However, the use of FPGA devices demands placement algorithms that are aware of their specificities, most notably of the DPR partition structure, as DPR is a key feature to enable device sharing. Previous work that addressed function placement in FPGA-equipped NFVIs, such as [Sharma et al. 2020] and [Draxler and Karl 2019], have typically used simplified models that do not fully capture the limitations of DPR partitioning. Thus, in this work, we provide a model that considers DPR regions, and we also demonstrate that approaches that are unaware of this concept may lead to unfeasible allocations.

3. FPGA

Field Programmable Gate Array is a semiconductor device that stands out for its programmability, aiming to merge ASICs' low latency and high processing capability with the flexibility of GPPs. Current its hardware-based design enables fast and efficient data processing. Current FPGAs comprise heterogeneous resources such as: Configurable Logic Blocks (CLBs), that include Look Up Tables (LUTs) and Flip-Flops (FFs); Block RAM (BRAM) to implement internal memories; Digital Signal Processing (DSP) blocks for efficient arithmetic operations; and Input/Output Blocks (IOBs), for communication with external devices. These resources are distributed along the hardware as columns that form a matrix. The definition of partitions, which is a requirement for DPR, must follow specific rules and consider minimum resource requirements, allowing efficient component allocation.

Reprogramming an FPGA can either be complete or partial, depending on the requirements and availability of resources. A total reconfiguration is necessary when the current functions are no longer needed or when a new function requiring substantial resources becomes a priority. During this process, the FPGA's operation is temporarily halted. On the other hand, partial reconfiguration involves updating just one partition's configuration, allowing the rest of the device to continue functioning normally. This approach enables better management of ongoing activities within the FPGA. This study uses Xilinx FPGAs, which feature Dynamic Function eXchange (DFX) for partial reconfiguration. The details for using DFX can be found in [Xilinx 2021].

4. Model and approach

The main goal of this work is to provide an approach for adequate placement of functions in FPGA devices in NFVIs and to demonstrate the crucial role of accurate modeling resources and partitions. For this purpose, ILP models will be presented in the following subsections.

4.1. Definition

This work first establishes the problem's context, which revolves around virtualizing network functions. To do this, we define a connected and weighted graph, denoted by $G_{net} = (N, L)$, where N represents the network's nodes, and L represents its links. Each node has limited computing resources, denoted by $r(n) = (CLB, BRAM, DSP)$, comprising the main hardware components employed by VNFs. We use the Python library networkX, [NetworkX 2014], to simulate the graph, ensuring it is connected and modeling various network functions that can be virtualized and run by this system. Each network function has a set of different implementations, occupying different points in the design space that include resource use and performance. Each implementation is defined by a resource requirement tuple, including CLBs, BRAMs, DSP blocks, maximum latency, and minimum data throughput.

Constructing partitions is a crucial aspect of this work, as it involves allocating resources to implement all supported functions with minimal wasted resources. The partition should be rectangular and follow specific rules considering the physical characteristics of the FPGA. The smallest possible resource unit for a partition is a column of resources delimited by configuration rows along the device, which can vary for each FPGA model. For the FPGA family used in this work, the partition height must be a multiple of the column size, and the width can be delimited as needed. The allocation of resources should consider the non-uniform distribution of resource columns and routing problems, which hamper the complete utilization of any given partition. To avoid such issues, we allocate 25% more CLBs than indicated by the synthesis tool and reserve at least 10% of the device area for the static area, which is responsible for communicating partitions with external components.

4.2. Model

When it comes to optimizing the placement of virtualized network functions in an infrastructure, there are different approaches that can be taken. The focus of this work is not to create a new heuristic, but instead to model the hardware in a realistic way, and evaluate the impact of such a model on function allocation. We present two approaches to modeling FPGAs for VNF allocation: one is partition aware, i.e., it includes the explicit modeling of the reconfigurable partitions of the FPGA fabric, which are defined beforehand by designers and impose restrictions on how resources can be used for multiple functions; the other, deemed partition unaware, uses the simplified view of FPGA resources as a collection of blocks that can be freely assigned to functions as necessary as [Sharma et al. 2020] and [Draxler and Karl 2019]. As will be seen in this work, the partition unaware approach is simpler and uses fewer computational resources, but may lead to impossible allocations. Thus, it is recommended mainly for the early stages of the project and evaluation of NFVIs.

Each approach aims to maximize the value that can be achieved in an NFVI configuration, which is the sum of the value defined for each requisition allocated to the NFVI-PoPs. The approaches differ since partition awareness enforces that each partition can fit only one function, even if it has remaining resources. The partition-unaware model does not have partitions. Thus, the functions can take resources as needed.

Sets and Indices

- N: Set of nodes in the graph.
- R: Set of requisitions.
- P: Set of paths between nodes.
- F: Set of functions for each requisition.
- S_n: Set of resource sets for node n.

Parameters

- $\text{Latency}_{n_1,n_2}$: Latency between nodes n_1 and n_2.
- $\text{Throughput}_{n_1,n_2}$: Throughput between nodes n_1 and n_2.
- $\text{Resources}_{n,s}$: Resources available in resource set s at node n.
- Req_r: Tuple containing source node, destination node, maximum latency, minimum throughput, function chain, and value for requisition r.

Decision Variables

- $x_{n,s,r,f,p}$: Binary variable indicating if function f of requisition r is allocated in resource set s at node n along path p.
- y_r: Binary variable indicating if requisition r is selected.

Objective Function

Maximize the total value of allocated requisitions:

$$\text{Maximize} \quad Z = \sum_{r \in R} y_r \times \text{Value}_r \tag{1}$$

Constraints

- **Resource Constraints**: Ensure that the resource set has enough resources to allocate the requisition.

  $$\forall n \in N, s \in S_n, r \in R, f \in F, p \in P : x_{n,s,r,f,p} \times \text{ReqResource}_{r,f} \leq \text{Resources}_{n,s}$$

- **Latency and Throughput Constraints**: For each path of each requisition, ensure that the total latency does not exceed the maximum allowed and the minimum throughput is achieved.

  $$\forall r \in R, p \in P : \text{TotalLatency}_p \leq \text{MaxLatency}_r$$

  $$\forall r \in R, p \in P : \text{TotalThroughput}_p \geq \text{MinThroughput}_r$$

- **Link Throughput Constraint**: For each link between nodes, the total throughput used by all requisitions traversing this link must not exceed the link's throughput capacity.

  $$\forall \text{s\_node}, \text{d\_node} \in N : \sum_{\substack{r \in R, \\ p \in P, \\ f \in F, \\ i \in \{1, \ldots, |p|-1\}}} x_{\text{path}[i], \text{set\_idx}, r, f, p} \times \text{MinThro}_r \leq \text{LinkThro}_{\text{s\_node}, \text{d\_node}}$$

- **Function Allocation Constraint**: Ensure all functions of an allocated requisition are allocated along the same chosen path.

  $$\forall r \in R : y_r \times |F_r| \leq \sum_{n \in N, s \in S_n, f \in F, p \in P} x_{n,s,r,f,p}$$

- **Unique Allocation of Node-Resource Set Pair**: Ensure that each node-resource set pair is allocated to at most one function of one requisition.

  $$\forall n \in N, s \in S_n : \sum_{r \in R, f \in F, p \in P} x_{n,s,r,f,p} \leq 1$$

- **Allocation Limit**: Limit the allocation for each function of each requisition to at most one resource set.

  $$\forall r \in R, f \in F : \sum_{n \in N, s \in S_n, p \in P} x_{n,s,r,f,p} \leq 1$$

  For the partition-unaware model, instead of using the constraint *Allocation Limit* above, we used the constraint *Resource Capacity Constraint* below.

- **Resource Capacity Constraint**: Ensure that the total allocation of resources at each node does not exceed its available capacity.

  $$\forall \text{node} \in N, \text{set\_idx} \in S_{\text{node}} : \sum_{\substack{r \in R, \\ p \in P, \\ f \in F}} x_{\text{node}, \text{set\_idx}, r, f, p} \times \text{ReqResource}_{r,f} \leq \text{Resources}_{\text{node}, \text{set\_idx}}$$

5. Invalid allocation detection

The algorithm proposed in this work aims to find invalid allocation in an FPGA. It takes into consideration some hardware particularities, such as how the resources are distributed physically. The distribution along the chip can be seen as an example in Figure 1.

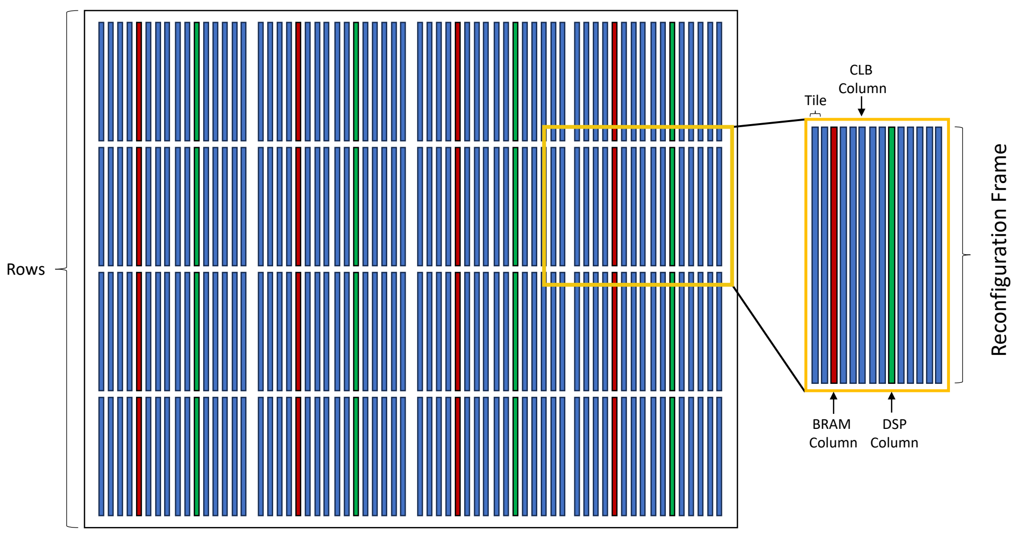

Figure 1. Reconfiguration Block Illustration.

The FPGA configuration memory, which stores the configuration bitstream, is divided into frames, which are organized in columns and rows. The configuration frame is minimum addressable unit of the configuration memory. Each FPGA family is designed with a unique frame size, which accommodates a predetermined number of resources, such as CLBs or BRAMs, with variations existing between models. The minimum partition size, with regard to computational resources, is defined by the frame height. This characteristic determines the lowest number of resources that can form a functional partition. For example, in the KU040, a model used in this work, the height of the configuration frame is 60 CLBs, or 12 BRAMs, or 24 DSPs. Therefore, a partition that requires only 5 CLBs will have to physically occupy 60 CLBs, as it needs to reserve a value that is a multiple of 60, even if not all are used.

The system partitions are composed of multiple configuration blocks. As each column in the resource matrix represents a unique resource type, it is advisable to construct taller rather than wider partitions when dealing with large partitions. This method ensures that resources are allocated wisely. For a detailed understanding of the algorithm employed, please refer to Algorithm 1.

Algorithm 1 Invalid allocation detection algorithm

```
Input: List of requisitions allocated using unaware approach $R$
Output: A list of requisitions that were not allocated (invalid) $I$
$I \leftarrow \emptyset$
for each requisition $r$ in $R$ do
    $Valid \leftarrow False$
    $min\_Tile\_CLB \leftarrow \lceil r.CLB/60 \rceil$
    $min\_Tile\_BRAM \leftarrow \lceil r.BRAM/12 \rceil$
    Check for the FPGA model to get total_rows, total_columns, and resource distribution
    for $column \leftarrow 1$ to $total\_columns$ do
        for $row \leftarrow 1$ to $total\_rows$ do
            if $min\_Tile\_CLB <= row \times column$ then
                $min\_BRAM \leftarrow row \times \frac{column}{min\_Tile}$
                if $min\_BRAM >= min\_Tile\_BRAM$ then
                    $Valid \leftarrow True$
                    Remove the used resources from the FPGA
                    break
    for $column \leftarrow 1$ to $total\_columns$ do
        for $row \leftarrow 1$ to $total\_rows$ do
            if $min\_Tile\_BRAM <= row \times column$ then
                $min\_CLB \leftarrow row \times \frac{column}{min\_Tile}$
                if $min\_CLB >= min\_Tile\_CLB$ then
                    $Valid \leftarrow True$
                    Remove the used resources from the FPGA
                    break
    if $Valid \neq True$ then
        $I \leftarrow I \cup \{r\}$
```

The algorithm idea is to check the requisitions that were previously allocated by the and verify their validity, giving an optimistic view of the hardware. For each requisition in the list the minimum number of tiles of each computational resource is calculated. Tile here is a single column from a reconfiguration block, as seen in Figure 1, being the minimal unit of resources that can be allocated. From the requisition list, we get the node and FPGA to which the function was allocated. For each FPGA, there is a different resource distribution along the hardware. This information is important because it defines the min_Tile value that represents the gap between two different resources, like for every 4 CLBs columns, there is a column of BRAM. The for loops are responsible for iterating over the number of columns and rows till the number of tiles of each resource inside this area fits the required amount. We focus on CLBs and BRAMs, which are the resources most heavily used by the evaluated VNFs, although the algorithm could be extended to cover other resource types. This logic is used to find the best aspect ratio that minimizes the amount of resources that the requisition needs to use. Thus, we assume each function is allocated to an actual partition of optimal dimensions. If the requisition does not fit these constraints, it is not valid, then it will be appended to the invalid list, and this instance of solution is considered invalid.

The purpose of this approach is to verify the validity of allocations performed by the solver by detecting any invalid requisitions. Furthermore, this method enforces the minimum area of usage and subsequently expands the rectangular shape as the counter increases. The algorithm works by commencing with a thin and tall partition, which is then horizontally expanded. The partition's height-to-width ratio is used to determine the quantity of a given resource it contains. The method is particularly beneficial when dealing with functions that require an uneven distribution of resources since it aids in the efficient partitioning of the device, minimizing resource and space wastage.

6. Experimental Results

The purpose of this section is to evaluate the scalability and efficiency of the ILP model by conducting simulation experiments. We will start by explaining the simulation environment used in our assessment and then proceed to present the results we obtained from running experiments on the ILP model.

6.1. Experimental Setup

The ILP model was constructed using the Gurobi solver API for Python version 11. The solver ran on an AMD Ryzen 5 5600 6-Core Processor @ 3.5 GHz with 32GB of RAM.

Table 1 shows the network functions considered in the simulations with their required resources. Different types of functions have varying resource numbers, allowing for various possibilities. Some latency values were not provided by authors, and for those cases, we consider random numbers drawn from the range of latencies of functions of the same type. Table 2 displays the FPGA models that were considered in the simulations, along with their corresponding specifications and the number of partitions that were used to run the tests. They were chosen by modeling three different sizes of partitions, to average fit the size of the network function chosen. Large has around 20.000 CLBs and 500 BRAMs, medium has around 10.000 CLBs and 300 BRAMs, and last, the small one has around 3.000 CLBs and 80 BRAMs, which may vary slightly from each model. Table 2 also lists how many of each partition size were placed in each FPGA model. For the KU040, the smallest FPGA, two different randomly assigned partition schemes were used, one with only one large partition, and one with one medium and two small partitions.

Function Type	CLB	BRAM	Throughput (Gbps)	Latency (μs)	Reference
Firewall	1150	5	2.9 - 6.6	1.84 - 4.2	[Ajami and Dinh 2011]
Firewall	8537	1	2	23 - 212	[Lopes et al. 2021]
Firewall	8123	241	92.16	73	[Fiessler et al. 2017]
DPI	8377	37	0.8	278	[Lopes et al. 2021]
DPI	8612	438	0.8	2778	[Lopes et al. 2021]
DPI	15206	36	14.4	-	[Yang et al. 2008]
DPI	713	96	90	-	[Jiang and Prasanna 2009b]
DPI	5154	407	80	-	[Jiang and Prasanna 2009a]
DPI	6048	399	102.6	-	[Fong et al. 2012]
AES	2532	0	49.38	-	[Smekal et al. 2018]
AES	4095	0	59.3	2	[Zodpe and Sapkal 2020]
AES	2304	0	45	-	[Soliman and Abozaid 2011]
AES	9561	450	119.3	-	[Henzen and Fichtner 2010]
AES	486	0	3.44	-	[Liu et al. 2015]

Table 1. Network Function Used

The system generates a unique network topology for every instance, with the size of the topology based on the number of network nodes. The number of nodes gradually increases by increments of 5 until it reaches a maximum of 30. Each size is run through multiple instances with entirely fresh topologies that accurately reflect their corresponding number of nodes. The number of links is a multiple of nodes, using 30% more, and creates a list of requisitions 5 to 7 times the number of nodes. To generate values for the creation of different network topologies, a range of values is employed to achieve specific variations. For instance, values ranging from $20\mu s$ to $200\mu s$ are utilized to define latency, while data throughput values are drawn from four fundamental values: 40 Gbps, 100 Gbps, and 200 Gbps. The total value achieved for each instance is the sum of values from each requisition, which is a function value calculated based on the resources used by it, i.e., we assume higher fees are charged for more resource-intensive functions. This approach can create a diverse range of network topologies with varying characteristics that cater to specific requirements.

Table 2. Product specifications

	KU040	**KU095**	**VU190**
CLB Blocks	30.300	67.200	134.280
Block RAM Blocks	600	1.680	3.780
I/O DLLs	40	64	120
DSP Slices	1.920	768	1.800
150G Interlaken	0	2	9
100G Ethernet	0	2	9
Large Partition	0/1	2	4
Medium Partition	1/0	1	2
Small Partition	2/0	3	3

6.2. Experiments

Figure 2 compares the mean value achieved by each size of topology, where each row represents a version of the model. The red one is the approach unaware of partitions, and the green represents the awareness. The bars in each dot are the standard deviation values from each size of topology. As expected, the mean value is always lower in the partition aware as it follows the hardware constraints, allocating some resources that will not be used. The mean value follows a linear rate as the number of nodes in the topology grows. Nonetheless, we note that, on average, the partition-aware approach is able to allocate 78.9% of the requisitions allocated by the partition-unaware model. It should be noted, however, as will be seen next, that using a partition-unaware model may lead to unfeasible allocations.

The validity of the allocation algorithm was tested through simulations to assess the frequency of invalid allocations when implementing the unaware approach. The figure 3 shows the proportion of invalid solutions to the total number of instances, highlighting the occurrence of faulty allocation. Thus, although an unaware model is able to allocate more functions, there is a relevant probability that an invalid solution will be produced, when designers actually try to fit the selected functions in the FPGA fabric. In other words, neglecting to factor in partitions during FPGA modeling can lead to inaccurate values, ultimately resulting in an inadequate computational resource for the network infrastructure.

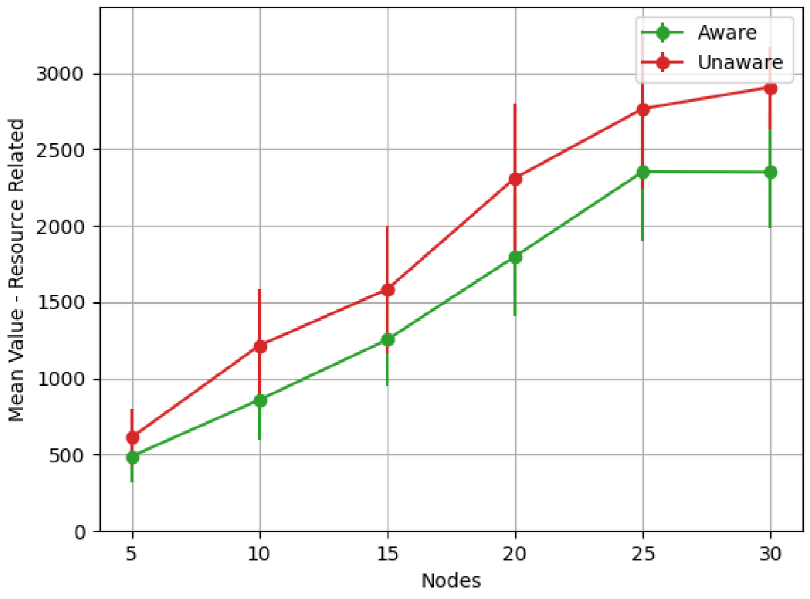

Figure 2. Mean number of allocated requisitions achieved by each approach per number of nodes in the topology

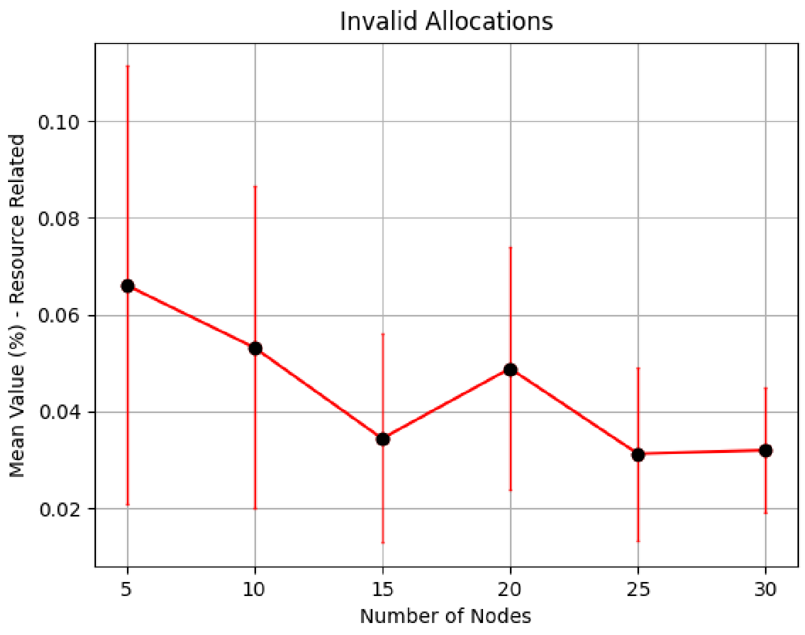

Figure 3. Mean ratio for invalid allocation

7. Conclusion

This paper presented an ILP-based VNF placement scheme for FPGA-equipped NFVIs. Results highlighted the importance of considering hardware limitations and partition-awareness while designing network infrastructures. The study's simulations showed that neglecting partitions can lead to inaccurate allocations and insufficient computational resources. However, adopting a partition-aware approach can help the network achieve more precise values and better resource allocation, resulting in a highly efficient and effective infrastructure.

Future works that are being targeted focus on trying to reduce the optimistic view of the algorithm, not generalizing the number of different resource columns but using a real distribution of the hardware instead.

Acknowledgements

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), grant #21/2551-0002089-0, the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), grant #305355/2023-6, and the São Paulo Research Foundation (FAPESP), grant #2021/06947-6.

References

Ajami, R. and Dinh, A. (2011). Embedded network firewall on fpga. In *2011 Eighth International Conference on Information Technology: New Generations*, pages 1041–1043.

Doan, T. V., Nguyen, G. T., Reisslein, M., and Fitzek, F. (2023). Sap: Subchain-aware nfv service placement in mobile edge cloud. *IEEE Transactions on Network and Service Management*, 20:319–341.

Dong, L., da Fonseca, N. L. S., and Zhu, Z. (2021). Application-driven provisioning of service function chains over heterogeneous nfv platforms. *IEEE Transactions on Network and Service Management*, 18(3):3037–3048.

Draxler, S. and Karl, H. (2019). Spring: Scaling, placement, and routing of heterogeneous services with flexible structures. In *Network Softwarization (NetSoft)*, pages 115–123. 2019 IEEE Conference.

Fiessler, A., Lorenz, C., Hager, S., Scheuermann, B., and Moore, A. W. (2017). Hypafilter+: Enhanced hybrid packet filtering using hardware assisted classification and header space analysis. *IEEE/ACM Transactions on Networking*, 25(6):3655–3669.

Fong, J., Wang, X., Qi, Y., Li, J., and Jiang, W. (2012). Parasplit: A scalable architecture on fpga for terabit packet classification. In *2012 IEEE 20th Annual Symposium on High-Performance Interconnects*, pages 1–8.

Henzen, L. and Fichtner, W. (2010). Fpga parallel-pipelined aes-gcm core for 100g ethernet applications. In *2010 Proceedings of ESSCIRC*, pages 202–205.

Jiang, W. and Prasanna, V. (2009a). Large-scale wire-speed packet classification on fpgas. pages 219–228.

Jiang, W. and Prasanna, V. K. (2009b). A fpga-based parallel architecture for scalable high-speed packet classification. In *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 24–31.

Laghrissi, A. and Taleb, T. (2019). A survey on the placement of virtual resources and virtual network functions. In *Communications Surveys and Tutorials*, pages 1409–1434. IEEE.

Leivadeas, A., Kesidis, G., Ibnkahla, M., and Lambadaris, I. (2019). Vnf placement optimization at the edge and cloud †. *Future Internet*, 11:69.

Li, X. and Qian, C. (2016). A survey of network function placement. In *2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 948–953.

Liu, Q., Xu, Z., and Yuan, Y. (2015). High throughput and secure advanced encryption standard on field programmable gate array with fine pipelining and enhanced key expansion. *IET Computers & Digital Techniques*, 9(3):175–184.

Lopes, F. B., Nazar, G. L., and Schaeffer-Filho, A. E. (2021). Vnfaccel: An fpga-based platform for modular vnf components acceleration. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 250–258.

Lopes, F. B., Schaeffer-Filho, A., and Nazar, G. (2023). Modular vnf components acceleration with fpga overlays. *IEEE Transactions on Network and Service Management*, 20:846–857.

NetworkX (2014). Networkx — networkx documentation. Accessed: 2024-01-28.

Niemiec, G. S., Batista, L. M. S., Filho, A. E. S., and Nazar, G. L. (2020). A survey on fpga support for feasible execution of vnfs. In *Communications Surveys & Tutorials*, pages 504–525. IEEE, VOL. 22, NO. 1, FIRST QUARTER 2020.

Paganelli, F., Cappanera, P., Brogi, A., and Falco, R. (2021). Profit-aware placement of multi-flavoured vnf chains. In *Cloud Networking (CloudNet)*, pages 48–55. 2021 IEEE 10th International Conference.

Qi, D., Shen, S., and Wang, G. (2019). Towards an efficient vnf placement in network function virtualization. *Computer Communications*, 138:81–89.

Santos, G. L., Bezerra, D., Rocha, , Ferreira, L., Moreira, A., Gonçalves, G., Marquezini, M., Recse, A., Mehta, A., Kelner, J., Sadok, D., and Endo, P. (2021). Service function chain placement in distributed scenarios: A systematic review. *Journal of Network and Systems Management*, 30.

Sharma, G. P., Tavernier, W., Colle, D., and Pickavet, M. (2020). Vnf-aapc: Accelerator-aware vnf placement and chaining. In *Computer Networks*. Elsevier, Computer Networks 177 (2020), 107329.

Smekal, D., Hajny, J., and Martinasek, Z. (2018). Comparative analysis of different implementations of encryption algorithms on fpga network cards. *IFAC-PapersOnLine*, 51(6):312–317. 15th IFAC Conference on Programmable Devices and Embedded Systems PDeS 2018.

Soliman, M. I. and Abozaid, G. Y. (2011). Fpga implementation and performance evaluation of a high throughput crypto coprocessor. *Journal of Parallel and Distributed Computing*, 71(8):1075–1084.

Solozabal, R., Ceberio, J., Sanchoyerto, A., Zabala, L., Blanco, B., and Liberal, F. (2020). Virtual network function placement optimization with deep reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 38:292–303.

Sun, J., Zhang, Y., Liu, F., Wang, H., Xu, X., and Li, Y. (2022). A survey on the placement of virtual network functions. *Journal of Network and Computer Applications*, 202:103361.

Xilinx (2021). Vivado design suite user guide: Dynamic function exchange (ug909). `https://docs.xilinx.com/r/en-US/ug909-vivado-partial-reconfiguration`. [Online; accessed March 27, 2023].

Xu, Z., Cui, Y., and Jiang, Y. (2018). Confv: An endhost-cloud collaborated network function virtualization framework. *2018 2nd IEEE Advanced Information Management,Communicates,Electronic and Automation Control Conference (IMCEC)*, pages 1280–1284.

Yang, Y.-H. E., Jiang, W., and Prasanna, V. K. (2008). Compact architecture for high-throughput regular expression matching on fpga. ANCS '08, page 30–39, New York, NY, USA. Association for Computing Machinery.

Zhang, Q., Xiao, Y., Liu, F., Lui, J. C., Guo, J., and Wang, T. (2017). Joint optimization of chain placement and request scheduling for network function virtualization. *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 731–741.

Zodpe, H. and Sapkal, A. (2020). An efficient aes implementation using fpga with enhanced security features. *Journal of King Saud University - Engineering Sciences*, 32(2):115–122.