A Fast Robust Optimization-based Heuristic for the Deployment of Green Virtual Network Functions

Antonio Marotta[a,*], Enrica Zola[b], Fabio D'Andreagiovanni[c], Andreas Kassler[a]

[a] *Karlstad University, Universitetsgatan 2, 65188, Karlstad, Sweden*
[b] *Universitat Politècnica de Catalunya, C. Jordi Girona, 1-3, 08034 Barcelona, Spain*
[c] *Sorbonne Universités, Université de Technologie de Compiègne, CNRS, Heudiasyc UMR 7253, CS 60319, 60203 Compiègne, France*

Abstract

Network Function Virtualization (NFV) has attracted a lot of attention in the telecommunication field because it allows to virtualize core-business network functions on top of a NFV Infrastructure. Typically, Virtual Network Functions (VNFs) can be represented as chains of Virtual Machines (VMs) or containers that exchange network traffic which are deployed inside datacenters on commodity hardware. In order to achieve cost efficiency, network operators aim at minimizing the power consumption of their NFV infrastructure. This can be achieved by using the minimum set of physical servers and networking equipment that are able to provide the quality of service required by the virtual functions in terms of computing, memory, disk and network related parameters. However, it is very difficult to predict precisely the resource demands required by the VNFs to execute their tasks. In this work, we apply the theory of robust optimization to deal with such parameter uncertainty. We model the problem of robust VNF placement and network embedding under resource demand uncertainty and network latency constraints using robust mixed integer optimization techniques. For online optimization, we develop fast solution heuristics. By using the virtualized Evolved Packet Core as use case, we perform a compre-

*Corresponding author
Email addresses: antonio.marotta@kau.se (Antonio Marotta), enrica@entel.upc.edu (Enrica Zola), d.andreagiovann@hds.utc.fr (Fabio D'Andreagiovanni), andreas.kassler@kau.se (Andreas Kassler)

hensive evaluation in terms of performance, solution time and complexity and show that our heuristic can calculate robust solutions for large instances under one second.

Keywords: Network Function Virtualization (NFV), Robust Optimization (RO), VNF, 5G, VNF Placement Heuristic, Datacenter

1. Introduction

Recently, Service Providers are migrating vendor specific hardware and software that implement their network functions towards the Cloud. In such Network Function Virtualization (NFV) [1], Virtualized Network Functions (VNFs) run inside Virtual Machines (VMs) or containers on commodity servers. NFV is expected to lead to significantly reduced CAPEX and OPEX due to the elasticity and scalability of the cloud paradigm, which significantly simplifies the VNF operation and management. Virtualization inside modern datacenters enables resources consolidation, leading towards green strategies to manage both compute and network infrastructure where VNFs are hosted.

Important tools are server consolidation strategies that migrate VMs/containers towards the fewest number of servers and power down unused ones to save energy. As VNFs are composed of a set of VNF Components (VNFC) that need to exchange data over the network under capacity and latency constraints, the networking plays also an important part. By using Software Defined Networking (SDN), one can dynamically adjust the network topology and available capacity by powering down unused switch ports or routers that are not needed to carry a certain traffic volume [2], thus consuming the least amount of energy at a potential expense of higher latency. Green strategies try to place the VNF components onto the fewest amount of servers and to adjust the network topology and capacity by powering down unused switches and ports to match the demands of the VNFCs. Such design of the VNF placement and virtual network embedding can be formulated as a mathematical optimization problem, and efficient heuristics can be designed to quickly solve the problem.

In classical mathematical optimization, it is assumed that all data involved in an optimization problem are known exactly when the problem is solved. However, this assumption does not hold in most real-world problems, in which data are often *uncertain*, i.e. not known with precision when the problem is solved. As an example, one can think about the unpredictable fluctuations in the traffic generated by users in telecommunication networks (see, e.g., [3, 4]). The decision maker could solve the problem simply using an estimate of these uncertain data. However, this could have potentially bad effects, as minimum variations in the input data may impact the optimality and the feasibility of the solution (see [5, 6, 7] for a thorough discussion). Solutions that neglect data uncertainty may turn out to be infeasible and thus useless in practice. Therefore, it is crucial to include data uncertainty in the optimization model.

Recently, Robust Optimization (RO) has been proposed in the optimization community as a methodology for dealing with data uncertainty. RO has had a great success in the last decade, thanks to its accessibility and computational efficiency. It essentially consists in taking into account data uncertainty under the form of additional constraints included in the model to cut off solutions that may turn infeasible or suboptimal, if variations in the input data occur [5]. The application of robustness allows to achieve a trade-off between protection from parameter deviations, which may lead to Service Level Agreement (SLA) violations (e.g. in terms of CPU utilization of the virtual components or network latency), and the well-discussed *price of robustness* [3, 6, 8, 9] due to higher cost (e.g. energy consumption) required to protect from parameter deviations [10]. In [11], we proposed a model for Robust Green VNF placement based on RO, which balances the power consumption for the Virtual Network Infrastructure (VNI) deployment and the protection from resource demand deviations of the individual virtual network functions. However, the model is too complex to solve for online optimization and does not account for traffic load induced latency at intermediate switches but rather assumes a fixed link latency.

In this paper, to the best of our knowledge we are the first to present a fast heuristic to solve the problem of Robust Green VNF placement and net-

work embedding with the aim of reducing the overall power consumption of the NFV Infrastructure while considering latency constraints for the service chains. The algorithm powers down compute servers, network switches and links, while taking into account the presence of data uncertainty in terms of VNF resource demands. Our heuristic iteratively solves three subsequent problems to deploy all the VNFs in a robust way. The first problem (step 1) deals with the allocation of each VNF component by minimizing the servers' power consumption and the total network traffic matrix that VNFCs inject. We propose both exact and heuristic approaches. In step 2, the allocation is made robust by using a fast greedy heuristic, which calculates both the set of migrations required to protect the placement from resource demand deviations and the updated traffic matrix. In step 3, we solve the splittable flow routing problem with latency constraints. We model the queueing delay that VNFCs may experience as a function of the link capacity and the processing load which can be modelled through an M/M/K/1 queueing system [12]. We perform a comprehensive evaluation in terms of performance, solution time and complexity using the virtualized Evolved Packet Core and we show that we can calculate robust solutions for large instance sizes in less than a second.

The reminder of the paper is structured as follows. The related work is discussed in Section 2. In Section 3, the problem is formulated, while the heuristic is illustrated in Section 4. Section 5 presents and discusses the numerical results. Finally, Section 6 concludes the paper and draws attention to future work.

2. Related Work

Conventional resource allocation aims at efficiently allocating computing and storage resources, with little effort on ensuring the network performance of the ongoing services. Recently, new approaches have been proposed that abstract the services in the form of virtual infrastructures for resource allocation. The Network Function Virtualization concept brings flexibility, manageability and reliability to the network; characteristics that are crucial for the definition and

further deployment of future network architectures (e.g., 5G). Consequently, the VNF placement has received a lot of attention in recent years. The resource allocation problem for virtual infrastructures is tackled in [13]. The proposed approach extends the rounding technique used for the traditional VNE problem, while minimizing mapping conflicts introduced by the virtual infrastructure embedding problem. In [14], an optimization model is presented to solve the resource allocation for network service chains, by taking into account network latency as a combination of processing, packet queuing and propagation delay. The resource allocation problem for wireless virtual networks is formulated in [15], and a heuristic algorithm based on the Bottom-Left algorithm is developed. As shown, the resource utilization is increased with spectrum aggregation. An Integer Linear Programming (ILP) model is formulated in [16] for allocating VNFs in order to minimize the total network related cost and the resources fragmentation. [17] tackles the VNFs placement problem with the aim of minimizing the total network load overhead, by considering the data plane delay and the control plane overhead. In [18], two constraint-based heuristics are applied for the deployment of a virtualized Evolved Packet Core and the results are shown in terms of average number of used CPU cores and aggregate throughput. The joint problem of VNF scheduling and traffic steering is studied in [19] as a mixed ILP (MILP), with the goal of minimizing the makespan/latency of the overall VNFs' schedule. A genetic algorithm-based heuristic is proposed to reduce the complexity of the formulated problem.

Besides resource-efficient virtual network (VN) mapping or cost-efficient VN mapping, another important issue in cloud-based data centers is the amount of power or energy that is consumed. Consequently, a lot of research effort has focused on energy-aware (green) strategies. Authors in [20] propose an energy-aware virtual network embedding (VNE). An energy cost model is proposed and an ILP is formulated for the energy-aware VNE problem. The authors also proposed two efficient algorithms: a heuristic-based algorithm and a particle-swarm based optimization algorithm. Energy-aware VNE is also considered in [21]. The authors propose an efficient heuristic to assign virtual nodes to

appropriate substrate nodes based on priority, where existing activated nodes have higher priority for hosting newly arrived virtual nodes. [22] uses a game theoretic based approach to consolidate resources and find a balance between energy efficiency and network resiliency in the telecommunication domain. By employing mixed-integer programming, [23] proposes a power-efficient resource provisioning technique in cloud-based data centers, while complying with SLAs. As their optimization problem is NP-hard, the authors also propose a heuristic to efficiently solve it. Authors in [24] propose an optimization method to minimize energy consumption for a backbone network while respecting capacity constraints on links and rule space constraints on routers. An exact ILP formulation is presented first and an efficient greedy heuristic algorithm is introduced. The time limitation is also taken into account in [25], where the VM-placement and routing problem is investigated to provide resource guarantees. Further, VM-migration is exploited to improve power saving and resource utilization. Also in this case, a fast online heuristic is developed to allocate resources on the basis of the request's duration and bandwidth demand.

To the best of our knowledge, none of the above works deal with the uncertainty on the input data to their optimization models or heuristics (e.g., users requests, power consumption, CPU demand, etc.). However, it is well known that the solution of an optimization problem often exhibits high sensitivity to the input data perturbations. Consequently, ignoring uncertainty in input data can lead to solutions which are suboptimal or even infeasible [6, 9] when used in reality. On the other hand, the theory of robust optimization has been applied already successfully in other areas to cope with parameter uncertainty. The OpenFlow VN design problem is addressed in [26], where traffic uncertainty and statistical multiplexing are taken into consideration. The problem is modelled as a robust optimization program to jointly determine admission control for VN and routing for virtual links. A robust cloud resource provisioning algorithm is proposed in [27], where the over-provisioning and under-provisioning costs are minimized and various types of uncertainty are considered. The numerical study shows that the solution obtained from their algorithm achieves

robustness. The problem of sharing the infrastructure of a backhaul network for routing is tackled in [28]. In particular, the authors consider the revenue maximization problem for the physical network operator when subject to stochastic traffic requirements of multiple virtual network operators and prescribed SLAs. A robust MILP is formulated to study the trade-off between revenue maximization and the allowed level of uncertainty in the traffic demands. An original robust cutting-plane algorithm is proposed in [29] to address the uncertain nature of the jamming problem in wireless networks. A robust optimization approach for the VNE problem is investigated in [30], which is based on a robust MILP formulation using the $\Gamma-$robustness model. They also propose a MILP-based two-phase heuristic but do not consider latency constraints or compute demands of the VNFs. In our previous work [11], the problem of the robust green VNF placement was addressed. This new work goes a step further and proposes a heuristic that solves the problem fast also for big instances of the VNI, thus making our proposal suitable for online optimization. In addition to [11], we now consider latency constraints on service chains in the problem formulation. The heuristic then calculates network paths between the servers hosting the communicating VNF components that have the required capacity and fulfil those latency constraints while considering both the propagation and queueing induced latency.

3. Problem Formulation

We consider the VNI as the set of hardware resources (compute and network infrastructure) hosting a certain number of VNFs inside a virtualized data center. Each VNF is composed of service chains, which are a group of VNFCs with a set of traffic demands and a maximum tolerable latency. In particular, the traffic demands specify how much traffic the first component in a service chain sends to the second one, which forwards it after some processing to the third one and so on. The latency of a service chain is the sum of the experienced delays on the used paths, on which all the demands of the service chain are for-

warded and includes the propagation latency and the latency due to queueing. We take into account a set J of servers and a network graph $G(N, E)$, where N represents the set of network nodes and E denotes the links among them. Given the family of service chains, which are defined as a specific number of traffic demands between couples of a subset $\bar{V} \subset V$ of VNFCs, the objective of the problem is to allocate all the VNFCs on the servers and to find the network routes that satisfy the traffic demands while minimizing the VNI overall power consumption, given the latency, resource and bandwidth capacity budgets. A compact expression of the objective function is:

$$min \quad f = P_{VNI} = P_{servers} + P_{switches} \tag{1}$$

Regarding the power model for the compute infrastructure, we assume that the CPU of a server is the most power consuming part [31] and use a simple model as in [32, 33]. Each server j has an idle power consumption $P_{idle,j}$ (the power needed by the server when just powered on) and a maximum power consumption $P_{max,j}$ (all the CPUs run at maximum utilization). In between, the power consumption follows a linear model dependent on the CPU utilization (due to the virtual components that consume the CPU cycles of the server where they are allocated to).

For the switch power consumption, we consider two components as in [34]: a static component due to the chassis and the line cards, and a dynamic one dependent on the powered-on ports operating at a specific rate and characterized by a total utilization. For example, [35] provides an overview on the power consumption of three different 48-port switch models. For a specific switch, they show that the power consumption is 151W when the switch is idle and all the ports are powered down, while it increases to 184W when all the ports are enabled and to 195W when all the ports serve traffic at 1 Gbps. As the traffic-dependent power component is very small compared to the power consumption due to the static components and powered on ports, in this work we neglect the former as in [35].

4. A Fast Heuristic For Green and Robust VNF Placement (GRVP)

In order to cope with data uncertainty, in this work we follow the concept of Γ-Robustness [9], which allows taking into account an uncertainty budget, namely a maximum number of variables that may be affected by parameter deviation. This allows us to trade-off between the protection level against parameter deviations and the cost of the robust solution, which is usually higher than the one obtained in the deterministic case. Nevertheless, solving directly a robust optimization model [11] by using an exact solver, such as IBM ILOG CPLEX [36], may require a very long time (see also the discussion for two distinct robust network design problems in [4, 37]), especially if the problem needs to be solved for a large set of Γ values. Consequently, we develop a fast heuristic, which we call **Green Robust VNFs Placement (GRVP)**, to solve the formulated problem by dividing it into three sub-steps, namely the *VNFCs Placement*, the *Robust Heuristic* and the *Latency Constrained Flow Routing*, see Fig. 1.

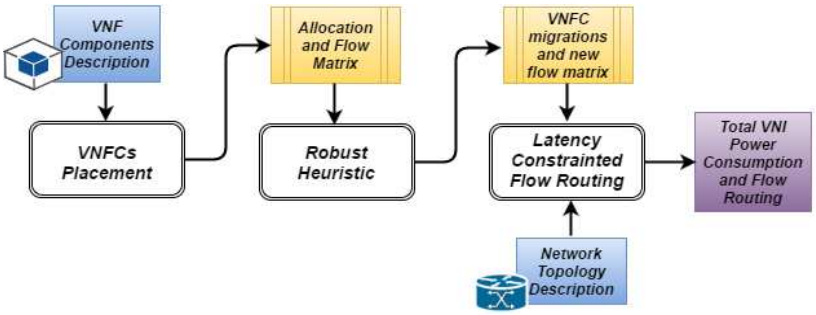


Figure 1: GRVP Heuristic

We briefly explain each step of the GRVP heuristic in the following. Fig. 2 illustrates the overall problem, which is to embed a set of service chains with uncertain resource demands into a given compute and network infrastructure.

1. The first step allocates the VNFCs belonging to the different service chains to a group of servers, each one having a different energy profile. When allocating each component, the objective is to obtain a balance between

the minimum servers' power consumption and the total traffic injected into the network. This is the traffic exchanged by a VNFC m_1 and VNFC m_2, considering that they are allocated to different servers associated to two different network nodes. As can be seen from Figure 2, such total traffic demand would be all the traffic sent from all the servers sending towards their access switches (sum of green, red and violet demands). Would all VNFCs be allocated to the same server, then such demand would be zero. The allocation obtained in this step guarantees no protection from deviations in terms of resource demands of each VNFC, since the average resource demand values are taken into account in this step (i.e., assuming the resource demand is fixed and known precisely). Two outputs are obtained from this first step: the VNFC allocation scheme (e.g. VNFC1 is allocated to server 3, VNFC3 is allocated to server 4 and VNFC2 is allocated to server 6) and the total network flow demands between each node (e.g. VNFC1 is injecting green and violet traffic, etc..). To solve this step, two methods are proposed: a classical optimization model based on Mixed Integer Linear Programming (MILP) and a fast First Fit Clustering Allocation (FFCA) online heuristic.

2. The second step is a greedy heuristic to make the placement immune from a certain number (Γ) of deviating parameters, namely the resource demands of the VNFCs allocated on each server. For example, if server 3 could host VNFC1 at average demand -but not with the maximum demand specified- while server 5 could host the maximum demand of VNFC1, then VNFC1 would migrate from server 3 to server 5. The heuristic tries to migrate away as few VNFCs as possible from those servers in which the remaining free-resource level is not sufficient to accommodate up to Γ components with a maximum deviation on their nominal resource demand. For more than one VNFCs allocated to a server, one needs to consider all possible combinations or the worst case. For example, let us assume

   - a server j with a total CPU of 1.0 (each 0.1 stands for 1 virtual core);

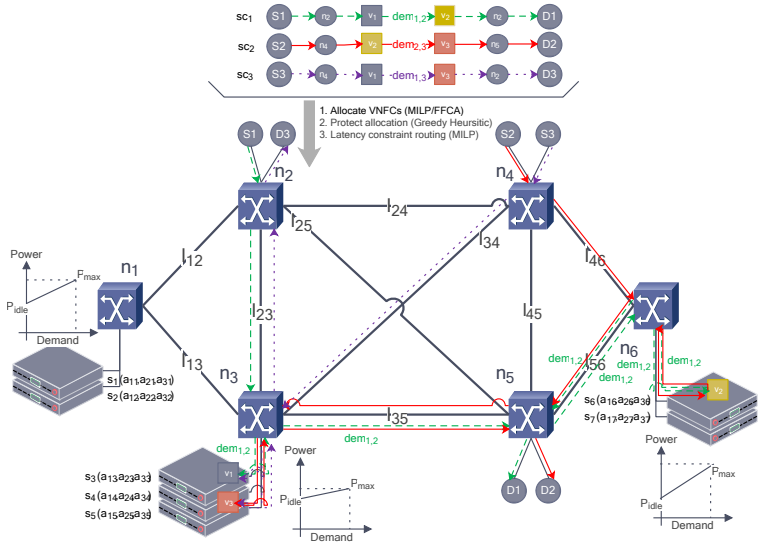


Figure 2: VNF embedding problem into Network and Compute Infrastructure.

- two VNFCs m_1 and m_2 allocated on j, respectively demanding for 0.4 and 0.5 units of CPU;
- a protection factor, Γ, equal to 2;
- a maximum deviation in the CPU demands by the components which account for 30% of the actual demand ($\Delta_r = 30\%$);

then the CPU demands of m_1 and m_2 may deviate up to 0.52 and 0.65, respectively. Consequently, if both VNFCs at the same time deviate, server j can not accommodate both VNFCs any longer (i.e., the total CPU demand is higher than the available CPU at server j). Thus, we need to migrate away one VNFC (and possibly power on another server to host this VNFC) in order to make our placement robust against the demand variations. On the other hand, if we assume $\Gamma = 1$ (i.e., we want to protect against the deviation of only one VNFC demand), we still need to migrate away one VNFC, because in the worst case VNFC m_2 may deviate up to 0.65 CPU units while we assume no deviation for m_1 consuming 0.4, thus the total demand is again higher than the available CPU at server j. The output of the second step will be a set of migrations

that are necessary in order to ensure an allocation which is immune to a maximum number Γ of parameter deviations. Moreover, this step also updates the traffic matrix[1].

3. Once the VNFCs are placed in a robust way in step two, we use the updated traffic flow matrix for routing the traffic between the servers where VNFCs have been placed in the previous step. Here, we only need to consider flows that inject traffic into the network (we do not need to route flows between VNFCs allocated on the same host). In order to compute the routing for the traffic flows, we develop a splittable and latency constrained flow routing model. The aim is to find the minimum power consumption in the network when determining the routing decision for each demand belonging to a specific service chain. To this end, we need to guarantee that the sum of the delays suffered by each demand on the possible paths is not greater than the one tolerated by the service chain. To clarify, suppose we have one service chain (sc_1) with three VNFCs and the following list of demands (dem) and latency (lat, expressed in ms):

   - $sc_1 = \{m_1, m_2, m_3\}$
   - $dem = \{(m_1, m_2, 10), (m_2, m_3, 20)\}$
   - $lat = \{(sc_1, 50)\}$

   If the demand between m_1 and m_2 (i.e., 10) is sent on the path 1-2, with a total latency of 9 ms, and the second demand between m_2 and m_3 (i.e., 20) is forwarded on the path {2-3, 3-4} with a total latency of 21 ms, the total suffered latency for the service chain will be 30 ms (which is less than 50 ms, namely the maximum tolerable latency for the service chain sc_1).

[1]Note that we do not perform real live migration of VNFCs at this stage. Rather, the migrations are virtual ones in order to create a robust placement out of a placement that may not be robust after the first step. Only after all the steps are finalized, we have calculated our placement that will be enforced by the NFV orchestrator.

4.1. Step 1: Initial VNFCs Placement

We consider a set of VNFs V to allocate on the set of servers J, which offer a set of resources I: the VNFs are associated with a set C of service chains involving a set M of VNFCs. Each VNFC $m \in M$ can be run on a single VM or container and we denote by $\bar{r}_{i,m}$ its request for resource $i \in I$. Each server $j \in J$ is connected to one single node of the network, denoted by $n(j) \in N$ and can provide a maximum amount $a_{i,j}$ of each resource $i \in I$. Given this basic notation, we present all the parameters and the decision variables involved in the optimization model to solve the initial VNF placement in Table 1.

Input parameters:	
$a_{i,j}$	amount of resource $i \in I$ available at server $j \in J$
$\bar{r}_{i,m}$	amount of resource $i \in I$ requested by VNFC $m \in M$
$P_{\text{idle},j}$	idle power consumption of server j
$P_{\text{max},j}$	maximum power consumption of server j
dem_{m_1,m_2}	amount of traffic to be sent from VNFC m_1 to m_2, for each $m_1, m_2 \in M : m_1 \neq m_2$
$n(j)$	network node in N to which the server $j \in J$ is connected
Decision variables:	
$x_{j,m}$	is equal to 1 if $m \in M$ is allocated to $j \in J$ and 0 otherwise
y_j	is equal to 1 if server $j \in J$ is active, 0 otherwise
$al_{n,m}$	is equal to 1 if m is associated to network node n, 0 otherwise
p_j	is the power consumption of server $j \in J$
$z_{n_1,n_2}^{m_1,m_2}$	is 1 if the traffic from m_1 to m_2 is sent from node n_1 to n_2 $\forall m_1, m_2 \in M : m_1 \neq m_2, \forall n_1, n_2$: $n_1 \neq n_2$, 0 otherwise
traf_{n_1,n_2}	is the total traffic between the node n_1 and n_2 $\forall n_1, n_2$: $n_1 = n_2$

Table 1: Initial VNF Placement Model Parameters

The Mixed Integer Linear Programming model in Table 2 allocates all the VNFCs to the physical servers to minimize (2): 1) the sum of the final servers' power consumption, normalized to the total power consumption; and 2) the traffic to be injected into the network, normalized to the total traffic demand (tot_dem, which is the sum of all the traffic demands dem_{m_1,m_2}). As the problem is a multi-objective optimization problem, we normalize each single objective component to obtain values between 0 and 1 which we then multiply with a weight, set to 0.5. This is because we want to find a good balance between minimizing both the total server's power consumption and the total network flow.

A non-active server ($y_j = 0$) has a 0 power consumption; on the contrary, the power consumption is linearly increasing with the CPU utilization. The latter is computed as the sum of the CPU units requested by each VNFC allocated to the server, normalized to the total available resource amount (4).

Table 2: Initial (non-robust) VNFCs Placement Model

$$\min f = \tfrac{1}{2} \cdot \textstyle\sum_{j \in J} \frac{P_j}{P_{\max,j}} + \tfrac{1}{2} \cdot \frac{\sum_{n_1,n_2 \in N: n_1 \neq n_2} \mathrm{traf}_{n_1,n_2}}{tot\_dem} \tag{2}$$

$$s.t. \tag{3}$$

$$P_j = P_{\mathrm{idle},j} \cdot y_j + (P_{\max,j} - P_{\mathrm{idle},j}) \cdot \frac{\sum_{m \in M} \bar{r}_{i,m} \cdot x_{j,m}}{a_{i,j}} \quad \forall j \in J,\ i \in I : i = CPU \tag{4}$$

$$\textstyle\sum_{j \in J} x_{j,m} = 1 \quad \forall m \in M \tag{5}$$

$$y_j \leq \textstyle\sum_{m \in M} x_{j,m} \quad \forall j \in J \tag{6}$$

$$y_j \geq x_{j,m} \quad \forall j \in J, m \in M \tag{7}$$

$$\textstyle\sum_{m \in M} \bar{r}_{i,m} \cdot x_{j,m} \leq a_{i,j} \cdot y_j \quad \forall j \in J, i \in I \tag{8}$$

$$al_{n,m} = \textstyle\sum_{j \in J: n(j) = n} x_{j,m} \quad \forall n \in N, m \in M \tag{9}$$

$$z_{n_1,n_2}^{m_1,m_2} \leq al_{n_1,m_1} \quad \forall m_1, m_2 \in M : m_1 \neq m_2, \forall n_1, n_2 \colon n_1 \neq n_2 \tag{10}$$

$$z_{n_1,n_2}^{m_1,m_2} \leq al_{n_2,m_2} \quad \forall m_1, m_2 \in M : m_1 \neq m_2, \forall n_1, n_2 \colon n_1 \neq n_2 \tag{11}$$

$$z_{n_1,n_2}^{m_1,m_2} \geq al_{n_1,m_1} + al_{n_2,m_2} - 1 \quad \forall m_1, m_2 \in M : m_1 \neq m_2, \forall n_1, n_2 : n_1 \neq n_2 \tag{12}$$

$$traf_{n_1,n_2} = \textstyle\sum_{m_1,m_2 \in M: m_1 \neq m_2} \mathrm{dem}_{m_1,m_2} \cdot z_{n_1,n_2}^{m_1,m_2} \quad \forall n_1, n_2 : n_1 \neq n_2 \tag{13}$$

Constraint (5) expresses that each VNFC must be allocated to exactly one server. Constraints (6) and (7) express that a server is active when at least one VNFC is allocated to it, otherwise it is inactive. Moreover, (8) ensures that the total amount of resources available at each server is not exceeded. (9) defines the (binary) variable $al_{n,m}$, which is equal to 1 if VNFC m is allocated to some server connected to node n, otherwise it is 0. Constraints (10-12) link the binary variables al_{n_1,m_1}, al_{n_2,m_2} to the binary variable $z_{n_1,n_2}^{m_1,m_2}$, determining if the traffic has to be routed from node n_1 to node n_2, depending on how the VNFCs m_1, m_2 are allocated with respect to n_1, n_2. If either al_{n_1,m_1} or al_{n_2,m_2} is equal to 0, then no traffic related to VNFCs m_1, m_2 is sent from node n_1 to node n_2 and thus the variable $z_{n_1,n_2}^{m_1,m_2}$ is forced to 0. This also holds if both the

decision variables are equal to 0. In contrast, when both al_{n_1,m_1} and al_{n_2,m_2} are equal to 1, then the right hand side of (12) is equal to 1 and this forces $z_{n_1,n_2}^{m_1,m_2}$ to be equal to 1, expressing the fact that the amount of traffic from m_1 to m_2 has to be sent from node n_1 to n_2. Finally, in (13), given the demands and their directions, the traffic between any two nodes n_1 and n_2 is computed.

Algorithm 1 First Fit Clustering Allocation (FFCA)

```
1: Input: sc, servers
2: Output: allocation
3: order clustered servers in decreasing order of available resources
4: for each service chain c ∈ C do
5:     for each VNFC m belonging to c do
6:         first_fit_allocate(servers, m)
7:     end for
8: end for
```

The first phase, which places the VNFCs in an energy/network efficient way, is modelled as an MILP and calculates the optimal placement. However, the runtime for large instances may be prohibitive due to its complexity. In order to speed up the first phase, we alternatively propose a **First Fit Clustering Allocation (FFCA)** heuristic (Algorithm 1), based on a simple policy. The heuristic iterates through all servers and checks, if they are connected to a specific network node. The nodes are taken into account according to their position in the network, meaning that the node with id 0 is considered first. The outcome is a set of clustered servers per each network node, which are then ordered in a decreasing fashion of the available amount of considered resource. The heuristic tries to allocate all the components of a specific service chain to the servers connected to the same network node, with the aim of reducing the amount of traffic to inject into the network. Algorithm 1 includes two nested loops over the service chains and the VNCFs and has a complexity of $O(|C||M|)$, where by M we denote the overall set of virtual network function components and by C we denote the overall set of service chains.

4.2. A Greedy Heuristic for Robust VNFCs Placement

The placement obtained in the first phase does not take into account resource demand variations for the VNFCs, as it only considers the average resource demands when performing the initial placement decisions. Consequently, the resulting placement may lead to possible SLA violations that may occur if a certain number of VNFCs in some service chains have actual resource requirements that deviate from their expected demand to the maximum. In [11], we proposed an optimization model in which the resource requirement of each VNFC is not known precisely, but may vary within a well-defined interval. We specified a maximum allowed deviation from the mean resource demand which may lie within a symmetrically distributed range with an upper and lower bound. In more details, we assume that a specific VNFC m requires an expected nominal amount $\bar{r}_{i,m}$ of resource i (e.g. memory or CPU) associated with a symmetric maximum deviation, $\hat{r}_{i,m} \geq 0$. Hence the actual resource demand may vary within the interval $[\bar{r}_{i,m} - \hat{r}_{i,m}, \bar{r}_{i,m} + \hat{r}_{i,m}]$. Based on the theory of robust optimization [5], we protect now the allocation by allowing a maximum number of components, given by Γ, to deviate from the expected demand at the same time. Consequently, after phase 1 some servers may have a resource utilization that is not protected against demand deviations. Therefore, this step tries to move away VNFCs from such servers in order to make room for potential demand deviations for a given protection level. Γ denotes the so-called *uncertainty budget* of the problem, which is based on the assumption that uncertain coefficients in different constraints are not correlated and on the observation that it is unlikely that all the coefficients may deviate to their worst possible value at the same time. The pseudo-code of our heuristic which tries to migrate away as few VNFCs as possible from a server that may run into potential contention, given the uncertainty budget Γ, is shown in Algorithm 2. In particular, we define the maximum deviation of each parameter $\bar{r}_{i,m}$ as a percentage ($\omega\%$) of the nominal value ($\hat{r}_{i,m} = \frac{\omega \cdot \bar{r}_{i,m}}{100}$).

The algorithm accepts as input the lists of VNFCs, the idle and active servers after the initial placement, the traffic demands matrix, the uncertainty budget

Algorithm 2 Greedy Heuristic for Robust VNFCs Placement

```
1: Input: list_vms, active_servers, idle_servers, demands, Γ, ω
2: Output: mig_list, traffic_matrix
3: order idle servers in decreasing order of energy efficiency
4: for each active server j do
5:     vms = {}
6:     get the allocated vms on server j and the resource protection for j according to Γ and ω
7:     while get_available_resource(j) ≤ res_protection do
8:         if iteration ≥ max_iter OR vms.size() == 0 then
9:             break while
10:        end if
11:        success = False
12:        m = get_vm_to_migrate (list_vms, vms, demands, num_vms)
13:        for each active server k != j do
14:            if k ∉ protected_list AND allocate_and_protect(m, k, Γ, ω) then
15:                success = True
16:                dest = k
17:                break for
18:            end if
19:        end for
20:        if success = False then
21:            for Each idle server h do
22:                if allocate_and_protect(m, h, Γ, ω) then
23:                    success = True
24:                    dest = h
25:                    break for
26:                end if
27:            end for
28:        end if
29:        if success = True then
30:            migrate(m, dest), add migration in mig_list and update the traffic matrix
31:            if get_protection_space(j, Γ, ω) ≥ get_free_resource(j) then
32:                add j in the protection_list
33:            end if
34:        end if
35:        remove m from vms
36:    end while
37: end for
```

and the maximum relative deviation ω. It calculates the (virtual) migrations list and the updated traffic matrix after the (virtual) migrations are applied. The heuristic checks each active server: for each server and its allocated VNFCs obtained from step 1, we compute the amount of resources that are needed to deal with a certain number Γ of components deviating at the maximum from their nominal demand. If the number of allocated VNFCs is less than the uncertainty budget Γ, all the VNFCs must be considered in the computation,

otherwise they are ordered in a decreasing fashion and the first Γ are taken into account (lines 4-6). If for the examined server there is not enough spare resources in order to account for the deviating VNFCs to their maximum value, then the algorithm tries to free the required amount of resources by migrating some VNFCs to other possible physical machines. The VNFCs to migrate are selected according to the following policy. First, we order the allocated VNFCs in terms of decreasing amount of traffic to exchange with other components on different servers. If there are more VNFCs with the same amount of traffic to choose from, the one with the resource demand closer to the gap to free on the server is selected (lines 7-11). The `for` cycle (line 13) tries to find an already active server to migrate the chosen component to, by verifying if the allocation is possible and by assuring at the same time the protection from the total deviation (Algorithm 3). If the (virtual) migration is successful, then the current server is stored, otherwise the search is carried out among the idle servers and a new server needs to be powered on (lines 13-27). If a server is eventually found, the (virtual) migration is performed and the allocation, together with the traffic matrix, is updated, accordingly. The source host is checked again: if the migration has freed enough space to cope with the uncertainty budget, then it is added to the protected servers list; if this condition does not hold, the next iteration of the `while` cycle tries to migrate a different VNFC (lines 28-36).

The complexity of Algorithm 2 is determined by the three nested loops (defined in lines 4, 7 and 13) involving the set of servers, the set of virtual machines and the set of servers. The algorithm thus has a complexity of $O(|J|^2|VM|)$, where by VM we denote the overall set of virtual machines (each virtual machine hosts one VNFC in our assumption) and by J the set of servers. The complexity of the Algorithm 3 is instead determined by the presence of a single loop over the VNCFs and the complexity is thus $O(|VM|)$, where by VM we denote the overall set of virtual machines.

Note that our strategy to make the placement robust by migrating away VNFCs from servers where there is a potential resource contention is quite conservative. This is because in our heuristic, the Γ-protection is ensured per each

Algorithm 3 Allocate and protect

Input: m, s, Γ, ω
Output: *possible*
get the allocated vms on s
$total\_res = 0$, $possible = False$
if vms.size() $\leq \Gamma$ **then**
    stop = vms.size()
else
    stop = Γ
end if
for m = 1:stop **do**
    $total\_res = total\_res + (\bar{r}_{i,m} + \hat{r}_{i,m})$
end for
if s.max_avail_res $\geq total\_res$ **then**
    $possible = True$
end if
return *possible*

active server. This is in contrast to an exact Robust MILP formulation, which is more opportunistic in considering all possible combinations at the expense of a significant longer runtime, which is not suitable for optimization. Hence, our algorithm is expected to calculate solutions very fast that provide a very good protection at the expense of higher energy cost than the theoretical optimal solution.

4.3. Latency Constrained Flow Routing

Once the final allocation for each VNFC has been found and safeguarded from possible resource demand deviations, the last step consists in finding the routing paths for the traffic demands among the VNFCs allocated on different hosts. As the goal is minimizing the energy consumption, we try to power down as many switches and switch ports as possible and route the traffic along those paths that have enough capacity, while fulfilling the latency requirements of the service chains. Differently from the optimization model in [11], we assume that each flow demand can be routed on splittable paths[2] and the link delay is not a static input parameter of the problem. Instead, each network link

[2]We argue that by using multipath transport layer and SDN, such splittable paths can be enforced.

is characterized by a latency that is computed as the sum of the propagation delay (fixed input parameter depending on the length of the link itself) and a queuing delay (depending on the processing load due to the traffic sent over the link). The processing delay can be considered as an average queuing delay, dependent on the incoming traffic rate, the configuration of the buffer size and the link capacity. Each queue is characterized by a delay that can be modelled according to the M/M/1/K queuing system. We apply the same procedure as in [38] and approximate the queueing induced latency using piecewise linearization.

The optimization model defining the Latency Constrained Flow Routing is shown in Table 4. The objective is to minimize the total power consumption due to all the active network devices and ports as in (14). The power consumption of a switch is zero if it is not used (all of its ports are idle). This is the case in which all the VMs belonging to the service chains, which are exchanging traffic among them, are allocated to servers packed in the same rack. Since we are considering each rack connected to a network switch, this allocation scheme causes all the traffic to be internal to the racks and, as a consequence, no packet will be sent to the upper layer switch. Thus, for the model the switches will be inactive; otherwise, if there is traffic flowing from one rack to another, the interested switches will be active and their consumption will be computed as the sum of a static idle power and the consumption of the active ports (16). In (17), the flow conservation constraint is expressed: given a node n and a traffic demand dem, if the source component is allocated to n and the destination is associated to another node, the sum of the incoming flows and the exiting ones is equal to the demand itself. The difference is equal to the opposite of the demand if the source component is not allocated to n, whereas the destination of the traffic is hosted on n. If n is just a transit node, the difference is zero.

The flow on a given link should be less or equal to the demand itself, if the link is used to carry that traffic (18). If the demand on a specific link is zero, that link should not be active for the demand and the port can be powered off (19). The total amount of traffic on a link is just the sum of all the demands forwarded on it and it should not be greater than the link capacity (20-21).

Input Parameters:	
$P_{s,n}$	is the static power consumption of node n
dem_{m_1,m_2}	is the traffic demand with value dem from VNFC s to d
$e.(src, dst, pw, lat, cap)$	are the source, destination, power, latency and capacity of link e
$path_{p,e}$	is 1 if link e belongs to the possible path p
$sc_{c,m}$	is 1 if the component m belongs to the service chain c
lat_c	is the maximum tolerable latency for service chain c
Decision Variables:	
y_n	is 1 if the node n is active, 0 otherwise
P_n	is the power consumption of the node n
f_e^{dem}	is the flow demand dem on the link e
h_e^{dem}	is 1 if the link e is carrying the demand dem
H_e	is 1 if the link e is used for any traffic
F_e	is the total traffic on link e
$load_{n,e}$	is the load of the node n considering its outgoing link e
$del\_qu_{n,e}$	is the queueing delay of n considering the buffer on the outgoing link e
del_link_e	is the total delay of link e
$latsub_{e,d}$	is the suffered delay by the demand dem on the link e
$path\_lat_{p,d}$	is the total delay suffered by demand dem on the possible path p

Table 3: Latency Constrained Flow Routing Model Parameters

The load of each network node buffer is computed per outgoing link: if n is the source node of the link e, then its load is computed as the sum of the demands forwarded through it, normalized to the total capacity of the link; otherwise it is set to zero (22). The constraint (23) expresses the piecewise linearisation of the node queuing delay according to the coefficients α_i and β_i, which are obtained through the linear interpolation of the curve from [38]. The node queuing delay is set to zero when the node is not active (24). The total delay of link e can be computed as the sum of the latency of link e and the node queuing delay (25).

The delay a traffic demand can suffer on a link e can be expressed as $delay_{e,d} = h_{e,d} \cdot del\_link_e$. The problem of this equation is that it requires the product of two decision variables: the first one indicating that the link is actually used for routing the demand and the total delay of the link, given by the sum of the propagation latency and the queuing delay of its source node. That equation can be linearised by introducing another decision variable $latsub_{e,d}$ and the constraints (26-27-28), where M_2 is an upper bound to the latency suffered by a demand on a link. The latency suffered by the traffic demand on a possible

Table 4: Latency Constrained Flow Routing Model

Flow Routing Model

$$\min \sum_{n \in N} P_n \tag{14}$$

$$s.t. \tag{15}$$

$$P_n = P_{s,n} \cdot y_n + \sum_{e:e.s=n} H_e \cdot e.pw \quad \forall n \tag{16}$$

$$\sum_{e:e.d=n} f_e^d - \sum_{e:e.s=n} f_e^d = \begin{cases} d.dem & \text{if } al_{n,d.s} = 1 \ \& \ al_{n,d.d} = 0 \\ -d.dem & \text{if } al_{n,d.s} = 0 \ \& \ al_{n,d.d} = 1 \\ 0 & \text{otherwise} \end{cases} \quad \forall n, \forall d \tag{17}$$

$$f_e^d \leq h_e^d \cdot d.dem \quad \forall e, d \tag{18}$$

$$h_e^d \leq f_e^d \quad \forall e, d \tag{19}$$

$$F_e = \sum_d f_e^d \quad \forall e \tag{20}$$

$$F_e \leq e.cap \cdot H_e \quad \forall e \tag{21}$$

$$load_{n,e} = \begin{cases} \frac{\sum_d f_e^d}{e.cap} & \forall e : e.s = n \\ 0 & \text{otherwise} \end{cases} \quad \forall n, d \tag{22}$$

$$\alpha_i + \beta_i \cdot load_{n,e} \leq del\_qu_{n,e} + (1 - y_n) \cdot M_1 \quad \forall e, n \tag{23}$$

$$del\_qu_{n,e} \leq M_1 \cdot y_n \quad \forall e, n \tag{24}$$

$$del\_link_e = e.lat + del\_qu_{e.s,e} \quad \forall e \tag{25}$$

$$latsub_{e,d} \leq del\_link_e \quad \forall e, d \tag{26}$$

$$latsub_{e,d} \leq M_2 \cdot h_{e,d} \quad \forall e, d \tag{27}$$

$$latsub_{e,d} - del\_link_e \geq -M_2 \cdot (1 - h_{e,d}) \quad \forall e, d \tag{28}$$

$$path\_lat_{p,d} = \sum_e latsub_{e,d} \cdot path_{p,e} \quad \forall p, d \tag{29}$$

$$\sum_{p,d \ : \ (sc_{c,d.s} \cdot sc_{c,d.d})=1} path\_lat_{p,d} \leq lat_c \quad \forall c \tag{30}$$

$$y_n \leq \sum_{e:e.s=n \ : \ e.d=n} H_e \quad \forall n \tag{31}$$

$$y_n \geq H_e \quad \forall n, \forall e : \text{e.s=n || e.d=n} \tag{32}$$

path is given by the sum of the delays on each link composing the considered path (29). For each service chain c we need to ensure that the sum of the delays suffered by each demand, belonging to c, on each used possible path is not greater than the maximum tolerable one, lat_c (30). Finally, the constraints (31) and (32) assure that a switch is active if and only if at least one of its ports is active, otherwise it is considered as idle. We note that if a solution is not found

in the last step, the heuristic is not able to modify the allocation in order to guarantee a feasible solution; this is instead left as future work.

5. Experimental Results

5.1. Evaluation Setup

In the numerical evaluation, we consider the deployment of a virtual Evolved Packet Core (vEPC) network inside a VNF Infrastructure. By considering the control plane (CP) load in terms of events per hour, we use [18] to compute the number of instances required per each type of VNFC that are needed to sustain the estimated hourly traffic bundle. In particular, we take into account different configurations for the number of VNFCs, a maximum allowed deviation from resource demands of 40% from their nominal value, and a protection factor (Γ) ranging from 0 (no protection) up to a maximum value. Our physical network topology, where the service chains need to be embedded, is organised in three layers: each rack has a single bidirectional link to the upper switch, while the switches in the second layer are connected in a full mesh with the ones in the top layer. For the sake of simplicity, all the links have a capacity of 1 Gbps and a latency randomly selected between 1, 2 and 3 ms.

As our model is to the best of our knowledge the first one to consider robustness for latency aware Green VNF placement and network embedding, we cannot compare it directly against related work. Instead, we provide a comprehensive evaluation in terms of performance, solution time and degree and price of robustness.

5.2. Evaluation of Step 1 - FFCA heuristic

In Figure 3, we plot the energy consumption (left) and the network flows (right) using the placement model from Table 2 when solved by CPLEX [36] using the well known branch-and-cut algorithm. We compare those values with the ones obtained by the FFCA heuristic (Algorithm 1). On the x-axis we vary the instance size configurations in terms of CP load (leading consequently

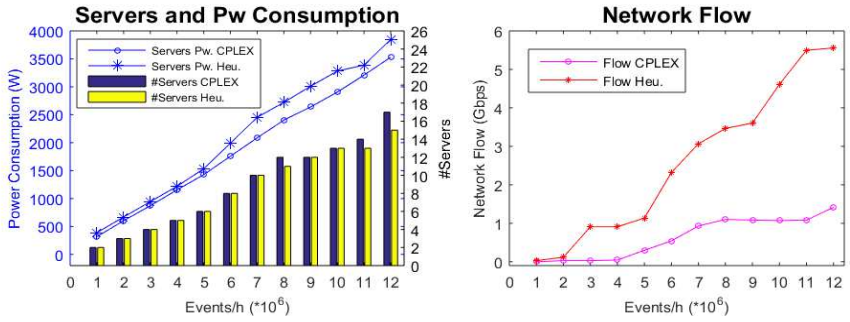


Figure 3: Power Consumption and Remaining Network Flow for the Optimal Placement Model and the FFCA Heuristic

to more VNFCs and higher problem complexity), starting from 10^6 ev/h (28 VNFCs) up to $12 \cdot 10^6$ ev/h (310 VNFCs). In the graph on the left, we show the number of active servers (right y-axis), while the server's power consumption is shown on the left y-axis. These results show the outcome of step one of our algorithm considering the deterministic allocation without any robustness or latency constraints.

The FFCA heuristic shows very encouraging results in terms of used servers and their total power consumption. The heuristic shows even better results compared to the optimal model when considering only the number of used servers for some configurations (the improvement of the results is between 7.15% and 11.76%). But the optimal solver has better results in terms of finding a balance between total power consumption and remaining network traffic. This is also due to the fact that the FFCA heuristic does not take into account the power model of different servers, as it simply uses a first-fit allocation policy to reduce the flows to inject into the network. The FFCA heuristic shows worse results in terms of network flows when the control plane load is increasing because of the vEPC service chains' characteristics. This is also because one component can be part of different service chains, which makes it harder for the heuristic to attenuate the traffic, especially when the CP load is increasing.

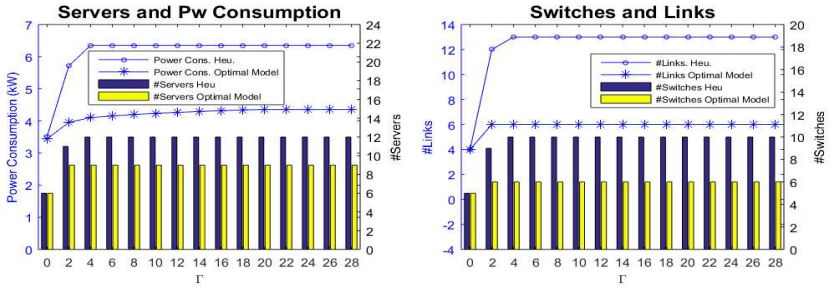


Figure 4: Comparison between the GRVP (with Optimal Placement Model) and the original model [11] (10^6 ev/h, $\omega = 40\%$)

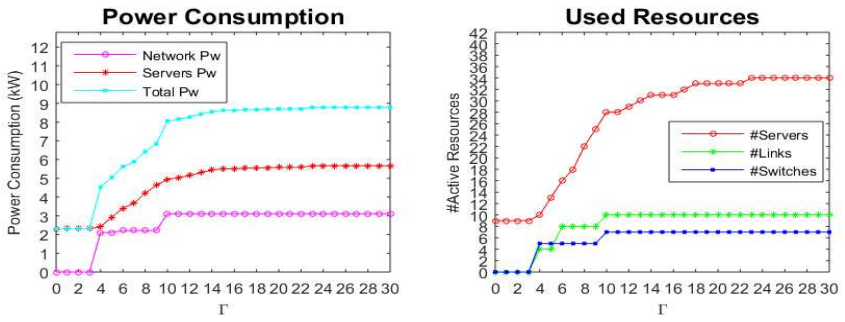
5.3. GRVP and FFCA heuristic combined

In Figure 4, we compare the GRVP heuristic (using the optimal placement model in Table 2 as step 1, followed by the robustifying part and the latency aware flow routing in step 3) with the results of the model in [11]. For fair comparison, we extended [11] to take into account latency due to increased traffic demand with the corresponding constraints. The difference to our heuristic is that [11] with latency extensions uses Γ robustness principle and consequently protects the whole placement (all servers) from demand deviations. Therefore, our heuristic is more conservative as we protect each individual server from demand deviations. We used Matlab [39] and the ROME toolkit [40] for solving the extended model in [11]. Because of the high complexity involved, we can only solve a small instance to optimality ($1 \cdot 10^6$ ev/h), and we vary from $\Gamma = 0$ (no protection) to the maximum protection $\Gamma = 28$.

The figure shows the number of used physical servers and their total power consumption in the left-graph and the active switches and links in the right one, both for the GRVP heuristic and for the model in [11] with latency aware flow routing. It is interesting to observe that the heuristic approach and the optimal solver achieve very similar results when no protection is applied ($\Gamma = 0$): in particular, the number of used servers and networking elements are the same, while the total power is only 2% higher than the optimal one. Consequently, our heuristic provides excellent results when no protection is desired. When $\Gamma \geq 4$

the heuristic activates around 25% more servers to cope with the uncertainty in demands compared to the exact model. The used links and switches also stabilize when $\Gamma \geq 4$ to values that are 53.85% and 40% worse than the optimal results, respectively. Besides, the heuristic has a total flow that is 75.31% higher with respect to the optimal solver in the worst case (1328 traffic units against only 328 computed by CPLEX and ROME). The difference in terms of power consumption is not too excessive, as the heuristic calculates placements that have between 2% and 35.37% higher power consumption than the one given by the exact model. The reason why our heuristic has higher energy consumption compared to the extended model from [11] is mainly because the heuristic is much more conservative as it protects each server individually from demand deviations, while [11] considers all potential combinations of parameter uncertainty. Consequently, for a targeted constraint violation probability, the heuristic requires a lower Γ compared to the optimal model. As we will see later, the benefits of our approach is that it is suitable for online optimization while [11] is too complex to solve reasonably sized instances in short time.

Figure 5: Results for GRVP with FFCA for 8M events and maximum deviation $\omega = 40\%$



In Figure 5, the results of the GRVP heuristic using FFCA allocation techniques for a control plane load equal to $8 \cdot 10^6$ events per hour are presented. The graph on the left shows the power consumption of the servers, network nodes and the total power consumption of the VNF Infrastructure when Γ is

increased from 0 (no protection) up to 30 (beyond this value of Γ, no valuable changes in the results were observed). The greatest increase in terms of total power consumption (15.22%) is observed when Γ increases from 9 to 10: the number of links, switches and servers change from 8 to 10, 5 to 7 and 25 to 28, respectively, to sustain the possible demand deviations. For $\Gamma = 23$ the active servers are stable to 34, while the used switches and links settle to 7 and 10 when Γ is equal to 10. By having a close look at these graphs, the VNI operator can decide if it should protect its VNF deployment from more components that may deviate in terms of resource demands, or be more power conservative consequently leading to less protection in terms of potential SLA violations. The full protection comes at a greater cost in terms of power consumption of the VNF Infrastructure which is 73.74% higher in comparison to the value obtained without any protection ($\Gamma = 0$).

In Figure 6, we show the results obtained by the GRVP with FFCA for the number of used servers, the total network flow and the number of activated links and total power consumption of the VNF Infrastructure for different values of Γ and increasing CP load (4M up to 20M). As can be seen, the total power consumption increases both for increasing protection applied and higher number of signalling events being served.

5.4. Price of Robustness and Runtime Evaluation

Finally, we investigate the additional price to pay for robust solutions in terms of higher energy consumption for protecting against uncertainty for a given Γ. We solve the problem for a given Γ using our heuristic phase 1 and 2 using GRVP with FFCA without the flow routing. For the robust solution calculated after step 2, we create 10.000 different instances of our input variables as follows. For each instance, if a VNFC requires a_{vr} units of CPU, we modify its demand to fall randomly within its upper and lower interval bound. After updating the CPU utilization on each server according to the random values calculated within the given bounds, we check the resource budget constraint and compute the number of constraint violations due to the input parameter

Figure 6: Results of GRVP with FFCA for different configurations ($\omega = 40\%$)

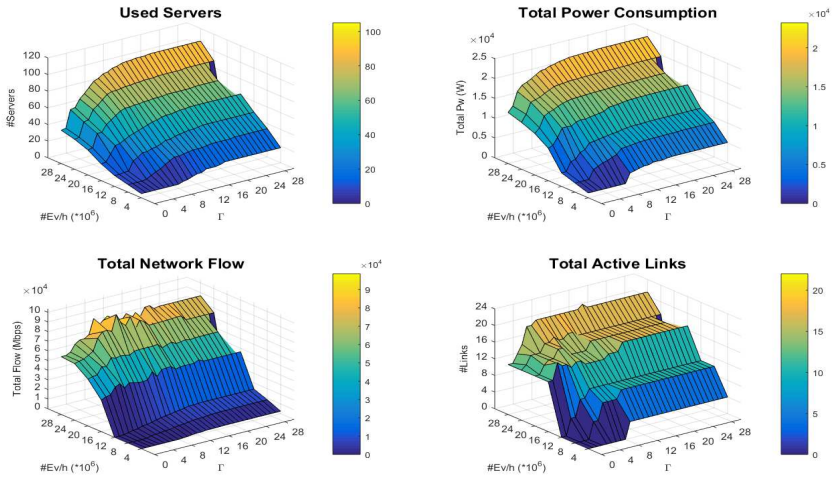
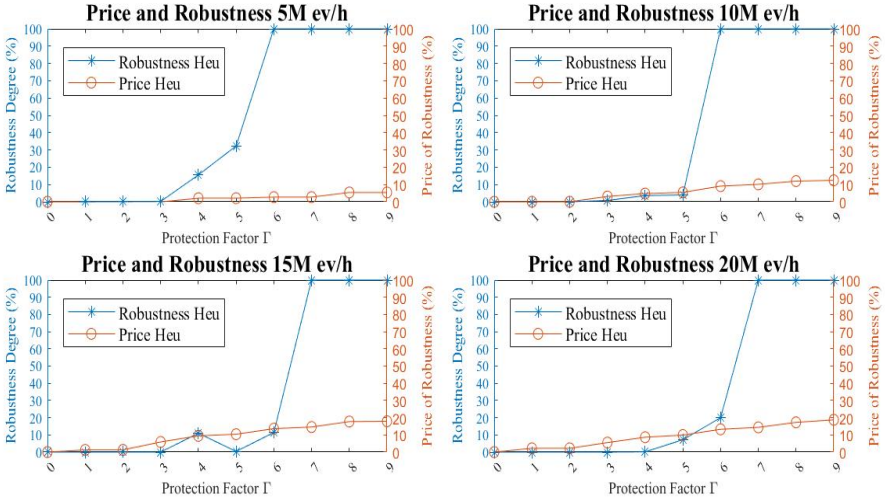


Figure 7: Degree and Price of Robustness for GRVP with FFCA for different protection levels



variation within the given bounds. We calculate the *robustness degree* as:

$$robustness = 1 - \frac{\#violations}{\#runs} \tag{33}$$

In addition, we calculate the price of robustness as the increase in the objective function (i.e., the total server power consumption) compared to the best value achieved when no protection is applied ($\Gamma = 0$):

$$price_{(\Gamma=x)} = \frac{total\_power_{(\Gamma=x)} - total\_power_{(\Gamma=0)}}{total\_power_{(\Gamma=0)}} \tag{34}$$

Fig. 7 shows the robustness degree (in blue) and the price of robustness (in red) as the protection factor increases for four different configurations of the vEPC: 5, 10, 15 and 20 M ev/h for different protection level Γ from 0 to 10. When $\Gamma = 0$, we do not protect against uncertainty and thus no additional resources are needed. This results in the lowest cost also having the lowest protection factor. When Γ increases, more servers and links are activated to protect the allocation from the demand deviations leading to higher energy consumption. For example, when $\Gamma = 6$ and ev/h=20M, we need around 15% more energy to protect at a robustness degree of around 20%. Interestingly, when $\Gamma = 7$, the degree of robustness is 100%, meaning that no SLA violations occur as the servers are all properly overprovisioned for the given workload to cope with demand uncertainty. This is due to the conservative nature of our heuristic.

Selecting a proper Γ is up to the Cloud Operator as the protection factor achieves a trade-off between additional costs in terms of energy consumption and the desired degree of robustness. A more conservative NFVI operator would like to protect its VNFI more from demand deviations, and consequently would select a larger Γ. However, more servers and network elements would be needed leading to higher costs to run the infrastructure. A more opportunistic operator would select a lower Γ leading to a potential higher constraint violation probability, which may lead to increased resource contention and ultimately also to SLA violations at the benefit of significant cost savings.

Finally, in figure 8 we show the execution times of the heuristic in (Algorithm 2). In particular, we fix the protection level Γ to 5 and plot the execution times

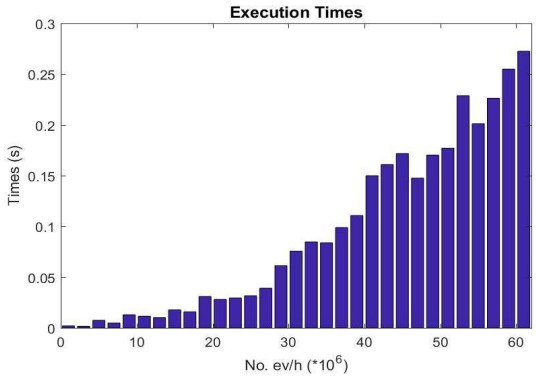


Figure 8: Execution Times for GRVP with FFCA for different problem sizes

starting from 1 million events per hour up to 60 millions events (1800 VMs in total). As shown in the figure, the heuristic performs very well and we are able to calculate a robust solution within 0.268s for very large instance sizes.

6. Conclusions and Future Work

In this paper, we propose a fast three-phase heuristic to tackle the problem of designing a power efficient Virtual Network Infrastructure under uncertainty of resource demands. In the first phase, we solve the problem of placing the VNF components on the servers in an energy efficient way while at the same time minimizing the resulting traffic matrix, without considering robustness. We propose both an exact method and a fast heuristic based on clustering and greedy strategies. The resulting initial placement is made robust in phase two by exchanging VNFCs among servers in a specific way that protects the servers from resource demand deviations of individual VNFCs, while at the same time trying to power on the minimum amount of servers and minimizing the total traffic matrix injected into the network. Finally, in step three, we solve the latency and capacity constrained routing problem to embed the service chain traffic into the substrate network. We consider queueing induced latency which depends on the amount of flows routed on a link.

Our approach can help a Telecom Operator in the planning decision making

by finding a balance between protection from demand uncertainty of the VNFs and a higher cost in terms of additional energy consumption required due to more servers and network elements needed to protect from uncertainty. We show that our heuristic can solve large instances and achieves reasonable results with respect to the optimal solution. Our future work will consist in improving the heuristic to reduce the gap from the optimal solution, also by considering the integration of local search strategies such as greedy randomized adaptive search (GRASP) into the algorithm. Another important future step is to implement the fast heuristic into the orchestrator of an ETSI MANO framework for NFV Orchestration.

Acknowledgement

Part of this work has been funded by the Knowledge Foundation of Sweden through the Profile HITS, by the Spanish Government and ERDF through CICYT project TEC2013-48099-C2-1-P, and by the German Federal Ministry of Education and Research (BMBF grant 05M2013 - VINO: Virtual Network Optimization).

References

[1] Chiosi M., et. al., Network Functions Virtualisation - Introductory white paper (2015).
URL `https://portal.etsi.org/nfv/nfv_white_paper.pdf`

[2] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, N. McKeown, Elastictree: Saving energy in data center networks, in: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, NSDI'10, USENIX Association, Berkeley, USA, 2010, pp. 17–17.

[3] T. Bauschert, C. Büsing, F. D'Andreagiovanni, A. M. Koster, M. Kutschka, U. Steglich, Network planning under demand uncertainty with robust op-

timization, IEEE Communications Magazine 52 (2) (2014) 178 – 185. doi:10.1109/MCOM.2014.6736760.

[4] F. D'Andreagiovanni, J. Krolikowski, J. Pulaj, A fast hybrid primal heuristic for multiband robust capacitated network design with multiple time periods, App. Soft Comp. 26 (2015) 497–507. doi:10.1016/j.asoc.2014.10.016.

[5] D. Bertsimas, D. B. Brown, C. Caramanis, Theory and applications of robust optimization, SIAM Rev. 53 (3) (2011) 464–501. doi:10.1137/080734510.
URL http://dx.doi.org/10.1137/080734510

[6] Ben-Tal A. and El Ghaoui L. and Nemirovski, A.S., Robust Optimization. Princeton University Press, Princeton Series in Applied Mathematics (2009).

[7] C. Büsing, F. D'Andreagiovanni, New Results about Multi-band Uncertainty in Robust Optimization, in: R. Klasing (Ed.), Experimental Algorithms, Vol. 7276 of LNCS, Springer, Heidelberg, 2012, pp. 63–74. doi:10.1007/978-3-642-30850-5_7.

[8] D. Bertsimas, S. M., The Price of Robustness, Oper. Res. 52 (2004) 35–53. doi:10.1287/opre.1030.0065.

[9] D. Bertsimas, A. Thiele, Robust and Data-Driven Optimization: Modern Decision-Making Under Uncertainty. INFORMS Tutorials in Operations Research: Models, Methods, and Applications for Innovative Decision Making (2006).

[10] E. Zola, A. J. Kassler, Optimising for Energy or Robustness? Trade-offs for VM Consolidation in Virtualized Datacenters under Uncertainty, Optimization Letters (2016) 1–22.doi:10.1007/s11590-016-1065-x.

[11] A. Marotta, A. J. Kassler, A power efficient and robust virtual network functions placement problem, in: 28th International Teletraffic Congress

(ITC 28), Würzburg, Germany, 2016.
URL http://i-teletraffic.org/_Resources/Persistent/b4860cf049ff9b40b2d216e277a2e2a5d965467b/Marotta2016.pdf

[12] G. Joshi, E. Soljanin, G. Wornell, Queues with Redundancy: Latency-Cost Analysis, SIGMETRICS Perform. Eval. Rev. 43 (2) (2015) 54–56. doi:10.1145/2825236.2825258.

[13] R. Yu, G. Xue, X. Zhang, Towards Min-Cost Virtual Infrastructure Embedding, in: 2015 IEEE Global Communications Conference (GLOBECOM), 2015, pp. 1–6. doi:10.1109/GLOCOM.2015.7416953.

[14] A. Baumgartner, V. S. Reddy, T. Bauschert, Combined Virtual Mobile Core Network Function Placement and Topology Optimization with Latency Bounds, in: Software Defined Networks (EWSDN), 2015 Fourth European Workshop on, 2015, pp. 97–102. doi:10.1109/EWSDN.2015.68.

[15] F. T. Hsu, C. H. Gan, Resource Allocation with Spectrum Aggregation for Wireless Virtual Network Embedding, in: Vehicular Technology Conference (VTC Fall), 2015 IEEE 82nd, 2015, pp. 1–5. doi:10.1109/VTCFall.2015.7391117.

[16] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, On Orchestrating Virtual Network Functions in NFV, CoRR abs/1503.06377.
URL http://arxiv.org/abs/1503.06377

[17] A. Basta, W. Kellerer, M. Hoffmann, H. J. Morper, K. Hoffmann, Applying NFV and SDN to LTE Mobile Core Gateways, the Functions Placement Problem, in: Proceedings of the 4th Workshop on All Things Cellular: Operations, Applications, & Challenges, AllThingsCellular '14, ACM, New York, NY, USA, 2014, pp. 33–38. doi:10.1145/2627585.2627592.

[18] F. Z. Yousaf, P. Loureiro, F. Zdarsky, T. Taleb, M. Liebsch, Cost Analysis of Initial Deployment Strategies for Virtualized Mobile Core Net-

work Functions, IEEE Communications Magazine 53 (12) (2015) 60–66. doi:10.1109/MCOM.2015.7355586.

[19] L. Qu, C. Assi, K. Shaban, Delay-Aware Scheduling and Resource Optimization With Network Function Virtualization, IEEE Transactions on Communications 64 (9) (2016) 3746–3758. doi:10.1109/TCOMM.2016.2580150.

[20] S. Su, Z. Zhang, A. X. Liu, X. Cheng, Y. Wang, X. Zhao, Energy-Aware Virtual Network Embedding, IEEE/ACM Transactions on Networking 22 (5) (2014) 1607–1620. doi:10.1109/TNET.2013.2286156.

[21] S. Jia, G. Jiang, P. He, J. Wu, Efficient Algorithm for Energy-aware Virtual Network Embedding, Tsinghua Science and Technology 21 (4) (2016) 407–414. doi:10.1109/TST.2016.7536718.

[22] A. P. Bianzino, C. Chaudet, D. Rossi, J. L. Rougier, S. Moretti, The Green-Game: Striking a Balance between QoS and Energy Saving, in: Teletraffic Congress (ITC), 2011 23rd International, 2011, pp. 262–269.

[23] G. Sun, V. Anand, D. Liao, C. Lu, X. Zhang, N. H. Bao, Power-Efficient Provisioning for Online Virtual Network Requests in Cloud-Based Data Centers, IEEE Systems Journal 9 (2) (2015) 427–441. doi:10.1109/JSYST.2013.2289584.

[24] F. Giroire, J. Moulierac, T. K. Phan, Optimizing Rule Placement in Software-Defined Networks for Energy-aware Routing, in: 2014 IEEE Global Communications Conference, 2014, pp. 2523–2529. doi:10.1109/GLOCOM.2014.7037187.

[25] A. Dalvandi, M. Gurusamy, K. C. Chua, Time-Aware VMFlow Placement, Routing, and Migration for Power Efficiency in Data Centers, IEEE Transactions on Network and Service Management 12 (3) (2015) 349–362. doi:10.1109/TNSM.2015.2443838.

[26] S. S. W. Lee, K. Y. Li, K. Y. Chan, Y. C. Chung, G. H. Lai, Design of Bandwidth Guaranteed OpenFlow Virtual Networks Using Robust Optimization, in: 2014 IEEE Global Communications Conference, 2014, pp. 1916–1922. doi:10.1109/GLOCOM.2014.7037088.

[27] S. Chaisiri, B.-S. Lee, D. Niyato, Robust Cloud Resource Provisioning for Cloud Computing Environments, in: Service-Oriented Computing and Applications (SOCA), 2010 IEEE International Conference on, 2010, pp. 1–8. doi:10.1109/SOCA.2010.5707147.

[28] C. Caillouet, D. Coudert, A. Kodjo, Robust Optimization in Multi-Operators Microwave Backhaul Networks, in: Global Information Infrastructure Symposium - GIIS 2013, 2013, pp. 1–6. doi:10.1109/GIIS.2013.6684375.

[29] F. D'Andreagiovanni, Revisiting Wireless Network Jamming by SIR-based Considerations and Multiband Robust Optimization, Optimization Letters 9 (8) (2014) 1495–1510. doi:10.1007/s11590-014-0839-2.

[30] S. Coniglio, A. M. C. A. Koster, M. Tieves, Virtual Network Embedding under Uncertainty: Exact and Heuristic Approaches, in: Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the, 2015, pp. 1–8. doi:10.1109/DRCN.2015.7148978.

[31] P. R. Panda, B. V. N. Silpa, A. Shrivastava, K. Gummidipudi, Power-efficient System Design, 1st Edition, Springer Publishing Company, Incorporated, 2010.

[32] S. H. Lim, B. Sharma, G. Nam, E. K. Kim, C. R. Das, Mdcsim: A multi-tier data center simulation platform, in: 2009 IEEE International Conference on Cluster Computing and Workshops, 2009, pp. 1–9. doi:10.1109/CLUSTR.2009.5289159.

[33] M. Pedram, I. Hwang, Power and performance modeling in a virtualized server system, in: 2010 39th International Conference on Parallel Pro-

cessing Workshops (ICPPW), IEEE Computer Society, 2010, pp. 520–526. doi:doi.ieeecomputersociety.org/10.1109/ICPPW.2010.76.

[34] D. Boru, D. Kliazovich, F. Granelli, P. Bouvry, A. Zomaya, Energy-efficient Data Replication in Cloud Computing Datacenters, Cluster Computing 18 (1) (2015) 385–402. doi:10.1007/s10586-014-0404-x.

[35] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, N. McKeown, Elastictree: Saving energy in data center networks, in: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, NSDI'10, USENIX Association, Berkeley, CA, USA, 2010, pp. 17–17.
URL http://dl.acm.org/citation.cfm?id=1855711.1855728

[36] IBM Cplex (Last accessed: April 2017).
URL http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[37] F. D'Andreagiovanni, A. Nardin, Towards the fast and robust optimal design of wireless body area networks, App. Soft Comp. 37 (2015) 971–982. doi:10.1016/j.asoc.2015.04.037.

[38] O. Dobrijevic, A. J. Kassler, L. Skorin-Kapov, M. Matijasevic, Q-POINT: QoE-Driven Path Optimization Model for Multimedia Services, Springer International Publishing, Cham, 2014, pp. 134–147. doi:10.1007/978-3-319-13174-0_11.

[39] Matlab (Last accessed: April 2017).
URL http://se.mathworks.com/products/matlab

[40] Rome Robust Optimization Made Easy User Guide (Last accessed: April 2017).
URL http://www.robustopt.com/references/ROME_Guide_1.0.pdf