Energy-efficient Path Allocation Heuristic for Service Function Chaining

Mohammad M. Tajiki[1], Stefano Salsano[2,3], Mohammad Shojafar[2], Luca Chiaraviglio[2,3], Behzad Akbari[1]

[1]ECE Department, University of Tarbiat Modares, Tehran, Iran - Email: {mahdi.tajiki,b.akbari}@modares.ac.ir
[2]CNIT, Rome, Italy - Email: mohammad.shojafar@cnit.it
[3]University of Rome Tor Vergata, Rome, Italy - Email: {stefano.salsano, luca.chiaraviglio}@uniroma2.it

Abstract—**Service Function Chaining (SFC) is a service deployment concept that promises cost efficiency and increases flexibility for computer networks. On the other hand, Software Defined Networking (SDN) provides a powerful infrastructure to implement SFC. In this paper, we mathematically formulate the SFC problem in SDN-based networks. In this way, the energy consumption of the network is minimized while the traffic congestion is controlled through network reconfiguration. Additionally, a low complex heuristic algorithm is proposed to find a near-optimal solution for the mentioned problem. Simulation results show that the proposed heuristic reconfigures the network in a way that the energy consumption is near-optimal while the SFC requirements are met. Besides, the computational complexity is very low which makes it applicable for real-world networks.**

Index Terms—**Software Defined Network (SDN), Service Function Chaining, Energy Consumption, Resource Management;**

I. Introduction

In real-world networks, traffic may need to pass through different hardware middle-boxes like for example Intrusion Detection Systems (IDS), proxies, firewalls. Network Function Virtualization (NFV) replaces hardware middle-boxes with flexible and innovative software applications known as Virtual Network Functions (VNFs) to reduce the capital and operational expenditures and increase the flexibility of providing the services [1]. On the other hand, Software Defined Networking (SDN) paradigm overcomes the traditional ossification of computer networks and introduces interesting, flexible, and novel service abstractions.

The steering of packets across a set of middle-boxes or more generically across different network functions is called Service Function Chaining (SFC). A Service Chain represents the set of network/service functions that need to be associated to a given flow. Service Chains may be required to process large amount of traffic with QoS constraints. Failure to provide the desired QoS to a Service Chain may lead to violating the service level agreements incurring high penalties for a network provider. Consequently, there are numerous works which focus on providing SFC in SDNs. Through this paper, we refer to Service Functions as VNF. A SFC taxonomy that considers architecture and performance dimensions as the basis for the subsequent state-of-the-art analysis is introduced in [2].

A. Related Work

The authors of [3] study the problem of deploying SFCs over NFV architecture. Specifically, they investigate VNF placement problem for the optimal SFC design across geographically distributed clouds. Moreover, they set up the problem of minimizing inter-cloud traffic and response time in a multi-cloud scenario as an Integer Linear Programming (ILP) optimization problem, along with some other constraints such as total deployment costs and service level agreements (SLAs).

Moreover, in [4] an optimization model based on the concept of Γ-robustness is proposed. They focus on dealing with the uncertainty of the traffic demand. The authors of [5] propose a heuristic algorithm to find out a solution for Service Function Chaining. It employs two-steps flow selection when a SFC with multiple network functions needs to scale out. The authors of [6] propose a scheme which provides flexibility, ease of configuration and adaptability to relocate the VNFs with minimal control plane overhead.

The authors of [7] use ILP to determine the required number and placement of VNFs that optimize network operational costs and utilization without violating service level agreements (SLAs). In [8] an approximation algorithm for path computation and function placement in SDNs is proposed. Similar to [7], they proposed a randomized approximation algorithm for path computation and function placement. In [9] an optimization model to deploy a chain in a distributed manner is developed. Their proposed model abstracts heterogeneity of VNF instances and allows them to deploy a chain with custom throughput without worrying about individual VNF's throughput. The paper [10] considers the offline batch embedding of multiple service chains. They consider the objectives of maximizing the profit by embedding an optimal subset of requests or minimizing the costs when all requests need to be embedded.

Reference [11] solves a joint route selection and VM placement problem. They design an offline algorithm to solve a static VM placement problem and an online solution traffic routing. They expand the technique of Markov approximation to gain their objectives. Although aforementioned solutions are interesting, however, none of the them considers the problem of service chaining with respect to the energy consumption of the servers.

The authors of [12] propose a scheme that uses three different algorithms to do the VNF placement, SFC routing, and VNF migration in response to changing workload. Their objective is to minimize the rejection of SFC bandwidth and reduce the energy consumption. Although their work is pretty interesting, there are some limitations in their approach. First of all, their approach is applicable for networks with predicable traffic, i.e., they suppose that the traffic pattern is repeated in a time interval. Moreover, they assume the amount of network traffic demands for all slots of the time interval and based on this knowledge, they turn on or off servers. Finally, the authors consider all of the possible physical paths as an input to their algorithm which is not an applicable assumption for medium and big networks.

B. Our Contribution

In this paper, in turn, we jointly consider the problem of flow rerouting and server energy consumption in SFC context. Our main objective is to minimize the network energy consumption while the required VNFs are properly delivered to the traffic flows. Specifically, we mathematically formulate the problem in form of ILP and propose an efficient heuristic algorithm to solve the problem in a real-time manner. Our main contributions are listed as follows:

- We mathematically formulate the resource reallocation problem which is a cross-layer optimization problem considering energy and SFC parameters. We use an Integer Linear Programming formulation.
- We propose a suboptimal heuristic to solve the aforementioned optimization problem. The proposed solution is an adaptive approach that is applicable in real-world networks.
- We compare the optimal resolution and the heuristic approach in terms of different metrics and computation time.

The remainder of this paper is organized as follows. Section II overviews the considered architecture and its main components while Section III presents the problem definition and related assumptions. Section IV details the proposed ILP optimal formulation. Section V presents the proposed heuristic algorithm and its computational complexity. The obtained results are detailed in Section VI. Finally, Section VII concludes the paper with some final remarks and outlines open research problems.

II. Reference Architecture

Fig. 1 reports a scheme of the considered architecture. We assume that there is a logically centralized SDN controller computing the forwarding table of the SDN-enabled switches. The controller uses a southbound protocol (e.g., OpenFlow) to dynamically program the switches. The VNFs are executed in servers that are directly connected to the switches (see Fig. 1). We consider the pair (switch, server) as a node. The controller can associate a VNF to a flow by routing the flow through a node in which the switch is connected to a server that supports the needed VNF.

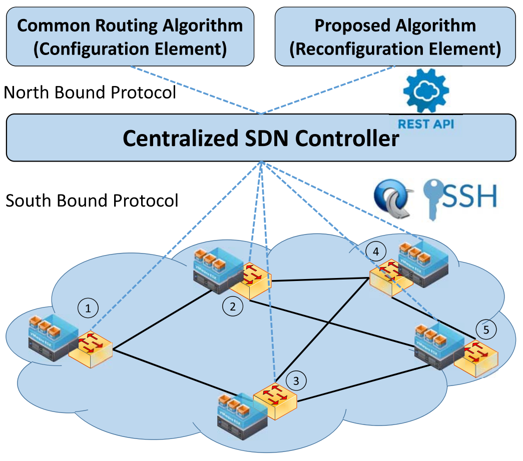


Fig. 1: System Architecture.

The switches along with the forwarding of the packets perform traffic measurements and forward these measurements to the controller. The controller can obtain the current flow matrix, network topology, and servers status via querying the SDN switches and servers.

In order to dynamically adapt the network configuration with respect to the traffic variations, the controller exploits the traffic information to optimally reroute the flows. As can be seen in Fig. 1, the controller includes the following modules:

- *Configuration Module*: when a new flow enters to the network, this element assigns the required resources to the flow. This element does not reroute existing flows and focus on the newly arrived flow.
- *Reconfiguration Module*: This module uses the current status of the network and reconfigures the network in a way that minimizes the servers' power consumption while the flow requirements are guaranteed to be met. This procedure may be applied either periodically or when a link is congested.

In this work we focus on the Reconfiguration module, which represents the more general case. Starting from the algorithms for the Reconfiguration module, it is possible to cover also the the Configuration module with some adaptations (e.g. only one flow is considered in the optimization).

III. System Model and Assumptions

The network is (re)configured in order to optimize the *energy consumption*, while meeting the SFC requirements (i.e. each flow requires to receive a set of VNFs in its path from the source to the destination switch). Moreover, the maximum link utilization and maximum server load should be less than predefined thresholds. Since different VNFs impose different processing loads on the servers, the maximum server load is evaluated considering the rate of the flows and the type of the VNFs. From the point of view of energy consumption, we consider that a server is ON if there is at least one VNF running on it. If no VNF is active on a server, the server is

TABLE I: Main Notation.

	Symbol	Description
Parameters	$\mathcal{N}$	Set of switches, $\vert\mathcal{N}\vert \triangleq N$
	$\mathcal{F}$	Set of flows, $\vert\mathcal{F}\vert \triangleq F$
	$\mathcal{X}$	Set of functions, $\vert\mathcal{X}\vert \triangleq X$
	E	Number of links
	ψ	Maximum number of required functions per flow
	$B_{(i,j)}$	Matrix of link bandwidth between i-th and j-th switches
	μ	Maximum link/node utilization
	C^f	Bandwidth requirement vector for the f-th flow
	s^f	Vector of source switch for the f-th flow
	d^f	Vector of destination switch for the f-th flow
	FP_x	Required processing load for the x-th VNF
	NC_i	Processing capacity for the i-th server
	$FN_{(i,x)}$	VNF x is available in i-th node
	ϵ	Factor of power consumption in IDLE mode
	R_x^f	Requested VNFs for the f-th flow
	$\mathcal{E}_i$	Power consumption for i-th server
Variable	$A^f_{(i,j)}$	Rerouting matrix between i-th and j-th switches with the flow f
	$U^f_{(i,x)}$	Used VNFs for the i-th switch with the flow f that runs the function x
	O_i	Power state of a server: 1 if the i-th server is powered ON, 0 if it is in IDLE mode

IDLE and its energy consumption will be a fraction ϵ of the one in the ON mode.

Table I summarizes the notation used in this paper. Consider a network with N SDN-capable switches, we represent the network topology with a matrix $B_{N\times N}$ where $B_{(i,j)}$ determines the bandwidth of the link from the switch i to the switch j. The number of flows in the network is F.

The input parameters s^f and d^f are used to store the source and destination of each flow, respectively. The routing matrix $A_{N\times N\times F}$ specifies the path selected for each flow, e.g., if $A^f_{(i,j)} \in \{0,1\}$ is equal to 1, then the flow $f \in \mathcal{F}$ crosses the link i to j (i.e., $i \rightarrow j$).

In addition, C_F stores the capacity required by each flow. Considering X different VNFs, each flow can request for at most $Y \leq \psi$ VNFs.

The required processing power of each VNF is denoted by FP_X where $FP_x \in FP_X$ specifies the required processing power of x^{th} function. Besides, the vector NC_N states the processing capacity for each server. In addition, we introduce the binary input parameter R_x^f, which takes value 1 if the x-th function is requested by the f-th flow, 0 otherwise. Note that we are not considering ordering constraints, that is in this model the VNFs can be crossed by the flow in any order. In addition, $FN_{(i,x)}$ is a binary input parameter, taking value 1 if function f is supported by node i, 0 otherwise. Finally, $U^f_{(i,x)}$ is a binary variable, taking the value 1 if flow f receives VNF x on node i, 0 otherwise.

Focusing on energy consumption, The vector $\mathcal{E}_N$ states the energy consumption of nodes where $\mathcal{E}_i$ specifies the energy consumption of nodes i. O_N specifies the modes of nodes (ON/IDLE) in which $O_i = 1$ means that the i-th node is ON mode, otherwise it is IDLE mode.

IV. Optimal Network Reconfiguration (ONR)

In this section, we present the SFC-aware resource reallocation system that aims at minimizing energy consumption of the servers. The objective functions and the considered constraints are described by the following equations (1)-(10):

$$\min_{O} \quad \sum_{i=1}^{N} O_i \cdot \big((1-\epsilon)\cdot \mathcal{E}_i\big), \tag{1}$$

Subject to:

$$\sum_{i=1}^{N} U^f_{(i,x)} \geq R_x^f, \ \forall x \in \mathcal{X}, \ \forall f \in \mathcal{F}, \tag{2}$$

$$\sum_{i=1}^{N} A^f_{(i,j)} \geq U^f_{(j,x)}, \ \forall x \in \mathcal{X}, \ f \in \mathcal{F}, \forall j \in \mathcal{N} - \{s^f\}, \tag{3}$$

$$U^f_{(i,x)} \leq FN_{(i,x)}, \ \forall f \in \mathcal{F}, \ \forall i \in \mathcal{N}, \ \forall x \in \mathcal{X}, \tag{4}$$

$$\sum_{i=1}^{N} U^f_{(i,x)} \leq 1, \ \forall f \in \mathcal{F}, \ \forall x \in \mathcal{X}, \tag{5}$$

$$\sum_{f=1}^{F}\sum_{x=1}^{X} (U^f_{(i,x)} \ \cdot \ FP_x \ \cdot \ C^f) \leq \mu \cdot NC_i, \ \forall i \in \mathcal{N}, \tag{6}$$

$$\sum_{f=1}^{F} \Big(A^f_{(i,j)} \ \cdot \ C^f\Big) \leq \mu \cdot B_{(i,j)}, \forall i,j \in \mathcal{N}. \tag{7}$$

$$\sum_{j=1}^{N} A^f_{(i,j)} - \sum_{j=1}^{N} A^f_{(j,i)} = \begin{cases} 1 & \text{if } i = s^f \\ -1 & \text{if } i = d^f \\ 0 & \text{else} \end{cases},$$

$$\forall f \in \mathcal{F}, \ \forall i \in \mathcal{N}, \tag{8}$$

$$\sum_{j=1}^{N} A^f_{(i,j)} \leq 1, \ \forall i \in \mathcal{N}, \ \forall f \in \mathcal{F}, \tag{9}$$

$$(1 + F \cdot X)\, O_i \geq \sum_{f=1}^{F}\sum_{X=1}^{X} U^f_{(i,x)}, \ \forall i \in \mathcal{N}, \tag{10}$$

$$U^f_{(i,x)}, \ A^f_{(i,j)} \in \{0, \ 1\}, \ \forall i,j \in \mathcal{N}, \ \forall f \in \mathcal{F}, \ \forall x \in \mathcal{X}.$$

where objective function (1) minimizes the number of servers that are required to be turned ON. The constraint (2) indicates that each flow crosses all the required VNFs (i.e. a flow crosses a node in which the required VNF is active). Moreover, constraint (3) imposes that for a given flow, if a VNF is provided in a node, the node is crossed by the flow. Constraint (4) imposes that for a given flow, if a VNF is provided in a node, the node supports this type of VNF. Constraint (5) prevents using a VNF more than once for each flow. Constraint (6) provides a threshold on the maximum load of a node, considering the processing load of all VNFs that are run in the node (which in turn depends on the sum of the rates of the flows that use each VNF). Focusing on the link bandwidth, constraint (7) enforces a threshold on the link utilization between each pair of the switches. Equation (8) provides the typical flow conservation constraints. In order to prevent loops, constraint (9) is applied for each flow. Finally, constraint (10) specifies which servers must be ON (those that deliver at least one VNF to a flow).

V. Heuristic Network Reconfiguration (HNR) Algorithm

The computational complexity of the optimal solution does not allow finding a solution in a reasonable time for realistically sized networks. Therefore, We design a new algorithm, called Heuristic Network Reconfiguration (HNR), to reallocate the resources in an online manner. For each flow, we consider the set of VNFs to be supported which is denoted with K in the Algorithm. Among all instances of the VNFs (that could be located in different nodes) we select the VNF and node which minimizes the incremental energy consumption of the network. If there are multiple options that have equal incremental energy consumption, then the node which is closer to the current node is selected (to this end we use shortest path algorithm considering the link delay as the cost function). Once the first VNF and node is selected, the shortest path from source to the node is evaluated. If there are other required VNFs in the chosen node or on a node along the path which is already in ON state, the VNFs are selected and removed from the set of VNFs to be supported. As an example, consider the flow f requested VNFs 1 and 2 and the selected node and VNF are node a and VNF 1. If node a supports both VNFs 1 and 2 and it has enough processing capacity, then flow f meets both VNFs in node a. If there are still VNFs required for the flow, the procedure is repeated, starting from the chosen node until there are no more VNFs in the set. Algorithm 1 presents pseudo code of the proposed HNR algorithm.

In algorithm 1, *SP* is the selected path for each flow, e.g., $SP^1 = [n_1, n_2, n_3]$ means that flow 1 starts from switch n_1 and then moves to n_2 and n_3, respectively. It should be considered that *HNR* algorithm reroutes the existing flows one-by-one, in other words, it reroutes one flow in each step (line 2). To this end, until all of the requested VNFs are delivered for one flow (line 4) the following steps takes place:

1) links that have available capacity less than the flow rate are removed from consideration (see line 5 the function $Prune\_Links$);
2) the shortest distance of the current node/switch (i.e., current place of the flow) to all other nodes/switches are calculated (see line 6 the algorithm $Dijkstra$). It should be mentioned that just those links that have bandwidth greater than the size of the flow are considered as valid links;
3) nodes that have not enough processing capacity or support non of required functions are removed by setting the distance of reaching them as infinity (see line 7 the function $Prune\_Nodes$). Note that distance is the number of the path hobs;
4) the amount of extra energy that would be imposed by each node is calculated (see line 8 the function $Energy\_Consumption$). Accordingly, if node i is currently idle, then the variable $energy_i \in \overrightarrow{energy}$ is a fraction of the energy consumption in ON mode.
5) *ENS* (standing for Energy-aware Nearest Service) finds the nearest ON node which supports one of the required VNFs (see line 7). If there is not such a node, *ENS* seeks nodes that are in idle mode. In this way, it finds the node with minimum energy consumption. In line 9, sn is the selected node, sp is the shortest path to the selected node, and sf is the set of functions that are selected to be delivered to the flow in the selected node or on an already ON node along the path;
6) the shortest path from sp to the selected node is added to SP^f (see line 10) and the current status is updated to the selected node (see line 12). Besides, the available bandwidth and processing power of the links and nodes that are used in the selected path are reduced by the size of the flow (see lines 13 and 14 the functions $Reduce\_Bandwidth$ and $Reduce\_Node\_Capacity$, respectively).

Algorithm 1 Heuristic Network Re-configuration (HNR)

```
INPUT: $F, K, B, N, C, NC$
OUTPUT: $SP$                     ▷ $SP$: selected path for each flow
 1: $SP = Empty\_Matrix(F)$;
 2: for each flow $f$ in $\mathcal{F}^f$ do
 3:     $CN = s$;                        ▷ $CN$: current node
 4:     while $K$ is not empty do        ▷ $K$: requested VNFs
 5:         $G = Prune\_Links(B)$;
 6:         $distances = Dijkstra(CN, G)$;
 7:         $distances = Prune\_Nodes(distances)$;
 8:         $\overrightarrow{energy} = Energy\_Consumption(\mathcal{N})$;
 9:         $[sn, sp, sf] = ENS(distances, energy)$;
10:         add $sp$ to $SP^f$
11:         remove $sf$ from $K$
12:         $CN = sn$;
13:         $B = Reduce\_Bandwidth(B, C^f, sp)$;
14:         $NC = Reduce\_Node\_Capacity(NC, C^f, sf)$;
15:     end while
16:     $p = Shortest\_Path(CN, Prune(B))$;
17:     add $p$ to $SP^f$
18:     $B = Reduce\_Bandwidth(B, C^f, p)$;
19:     $NC = Reduce\_Node\_Capacity(NC, C^f, sf)$;
20: end for
21: return $SP$
```

When all required VNFs have been included in the path of a flow, the algorithm uses shortest path to directly move to the destination (lines 16-19).

A. Computational Complexity

ONR: The problem can be reduced to capacity-aware multi commodity problem which is categorized as an NP-hard problem.

HNR: The computational complexity of lines 2 and 4 of algorithm 1 are in order of $O(F)$ and $O(\psi)$, respectively.

TABLE II: Hardware Configuration.

Name	**Description**
Processor	Intel-Core(TM) i5-2410M-CPU 2.30GHz
IDE	Standard-SATA AHCI Controller
RAM	4.00 GB
System Type	64-bit Operating System, Windows 10

The order of the computational complexity of lines 5, 13, and 18 are $O(E)$ while it is $O(N)$ for lines 7, 8, 14, and 19. Similarly, the computational complexity of lines 6 and 16 is $O(N \cdot \log N + E)$ while it is $O(\psi + N)$ for line 9. Therefore, the total computational complexity of *HNR* is $O\left(F \cdot \psi \cdot [N \cdot \log N + E + \psi]\right)$.

VI. Numerical Results

In this section, the proposed schemes are evaluated under different network traffic patterns and a real-world network topology. We also discuss the traffic-demand/resource generator and the simulation setup. We use CVX toolbox [13] to solve the *ONR* which is an ILP problem.

A. Simulation Setup

In order to investigate the performance of the proposed algorithm, a traffic-demand/resource generator is proposed. It generates the set of demands and resources with all the needed characterizations. Considering the demands, it generates for example the flow specifications (i.e., rate, source, destination, VNF requirements). Considering the resources, it generates the server processing capacity, the energy consumption parameters, the link capacity and so on. It is important to note that the proposed generator deals with traffic demands (not traffic packets). In the following, the system configuration and the network topology used in our simulations is described. Table II reports the PC configuration of the simulation environment. Table III presents the list of simulation parameters.

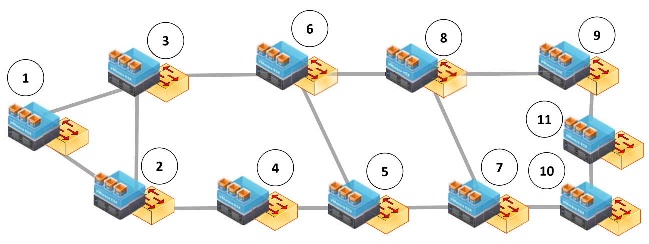


Fig. 2: Abilene Network Topology.

The considered network topology is the Abilene one [14], which is shown in Fig. 2. We set the link capacity $B(i,j) = 1$ [Gbps] for all the links that interconnect the switches. In addition, the links that connect a servers to a switch in a node have a capacity equal to the sum of the other links of the switch, so that there is no bottleneck in the local communication between a switch and a server. Each server has two states: ON (i.e., working with full rate energy consumption) and IDLE (in which case it is using a fraction ϵ of the full rate energy consumption). We assume that the processing power of a server NC is proportional the sum of the capacity of all incoming links. We assume that the energy consumption of a server $\mathcal{E}_i$ is related to its processing power. In particular, we define as parameters the maximum and minimum energy consumption ($\mathcal{E}_{max}$, $\mathcal{E}_{min}$) of a server. We assign $\mathcal{E}_{max}$ to the server(s) that have the maximum processing power, $\mathcal{E}_{min}$ to the server(s) that have the minimum processing power, and intermediate value between $\mathcal{E}_{min}$ and $\mathcal{E}_{max}$ to the other servers (scaled linearly). In our simulation we set $\mathcal{E}_{min}$=200J and $\mathcal{E}_{max}$=400J.

TABLE III: Simulation Parameters.

Parameter	**Value**
N	11
E	14
X	10
F	$\{35, 41\}$
$\mathcal{E}_{min}$	200 J
$\mathcal{E}_{max}$	400 J
μ	0.8
ψ	5
B	1 $Gbps$

The algorithms are examined in a sequence of five different time slots. We generate a set of traffic demands corresponding to the first time slot. The set of flows (i.e. the number of flows and their source and destination node) and the set of required VNFs for each flow are chosen according to the model reported in [15]. The rate of each flow is generated using a uniform distribution between $b_{min} \cdot B_{(i,j)}$ and $b_{max} \cdot B_{(i,j)}$. We considered b_{min}=0 and selected two different values for b_{max} to differentiate the traffic scenarios. After the first generation of traffic demands, we progressively increase the flow rates (and consequently the load on the network and servers) in four time slots. In each time slot, we randomly increase the rates of each flow so that on average the increase factor is α. In particular, we multiply the rate of each flow by a factor which is chosen between 0 and $2 * \alpha$ with a uniform distribution. For each time slot, the ONR optimal solution and the HNR heuristic are used to assign resources to the flows. In our experiment, four different traffic scenarios are considered by varying b_{max} and α, as shown in IV.

TABLE IV: Traffic Generator Notation and Inputs.

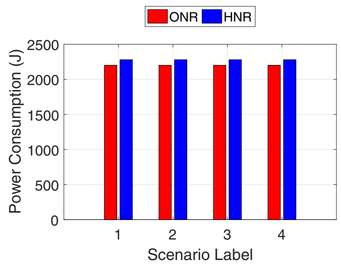
Parameter	**Scenario 1**	**Scenario 2**	**Scenario 3**	**Scenario 4**
b_{max}	0.02	0.02	0.05	0.05
α	0.1	0.3	0.1	0.3

On average, the four scenarios are ordered by increasing load. In scenarios 1 and 2 the initial load is lower ($b_{max} = 0.02$) and rate of increase in the time slots (iterations) is low for scenario 1 ($\alpha = 0.1$) and high for scenario 2 ($\alpha = 0.3$). In scenarios 3 and 4 the initial load is higher ($b_{max} = 0.05$) and the rate of increase (in each iteration) is low for scenario 3 and high for scenario 4. Note also that the average rate of the flows is bigger in scenario 3 and 4 with respect to scenarios 1 and 2.

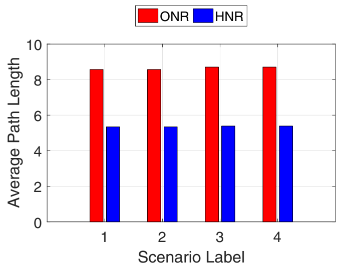
B. Simulation Results

The proposed heuristic algorithm and the optimal solution are compared via five different metrics: power consumption, path length, server/link utilization, and computational complexity.

1) Power Consumption and Path Length: The comparisons of the energy consumption and the average path length of the different traffic scenarios are presented in Fig. 3. As it can be seen in this figure, in all traffic scenarios, the *HNR* algorithm is able to ensure a power consumption close to the one of the *ONR* algorithm. On the other hand, the average path length of *HNR* is less than *ONR*, since the focus of *ONR* is on minimization of the energy consumption.



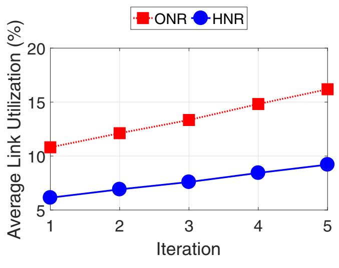
(a) Power Consumption.



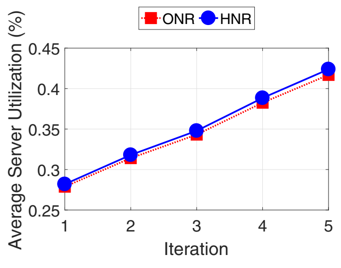
(b) Path Length.

Fig. 3: *ONR* vs. *HNR* Comparisons.

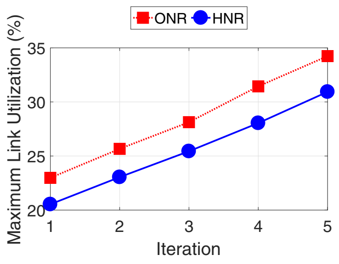
2) Link and Server Utilization: In this section, the proposed schemes are compared based on the link and server utilization measurements. To this end, the average and maximum utilization of both links and servers are measured in different traffic scenarios (depicted in figures 4-7).



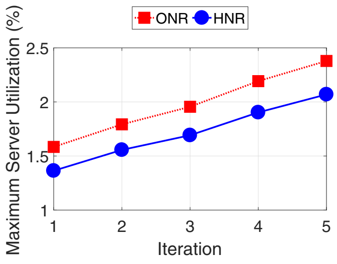
(a) Link Utilization



(b) Server Utilization



(c) Maximum Link Utilization

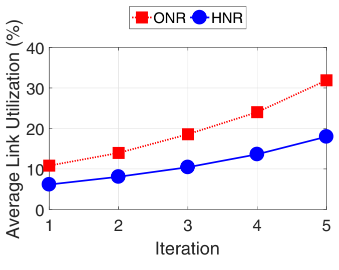


(d) Maximum Server Utilization

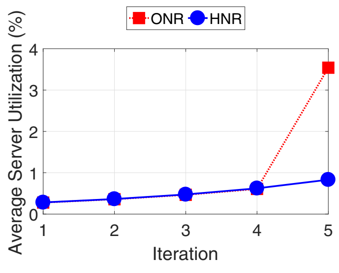
Fig. 4: Scenario 1: Utilization Comparisons.

In our simulation, we set the maximum allowed links/server utilization μ to 0.8. In Figs. 4 (Scenario 1) and 5 (Scenario 2), the maximum link/server utilization always remains well below μ, therefore, increasing the flow rate in different iterations constantly increases both the average and maximum link/server utilization. As it can be seen, the average and maximum link utilization and server utilization of *HNR* is lower than *ONR* algorithm. This is due to the fact that the main goal of *ONR* is to minimize the network energy consumption. As a result, the *ONR* algorithm is more effective in reducing the energy consumption, therefore there will be less servers in ON state. This will clearly increase their utilization (and this is a positive effect). On the other hand, the optimization of energy consumption comes at the price of increasing the network load, but this can be controlled by setting the maximum utilization threshold for the links.

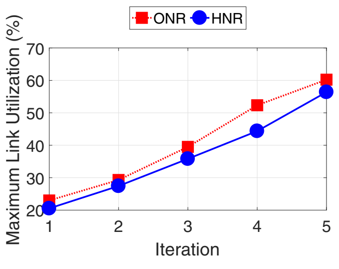
Actually, in the last time slot of Fig. 5, the server utilization of *ONR* has a sharp increase (while for the *HNR* there is a regular increase). This means that the *ONR* has found a solution with a different set of active servers. This may result in a higher average and maximum server utilization because some servers with smaller maximum capacity can be selected. We recall that we assigned to a server a maximum capacity proportional to the sum of the link rates of the node that it is hosting the server.



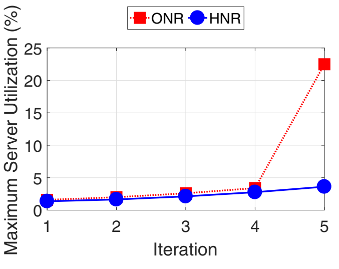
(a) Link Utilization



(b) Server Utilization



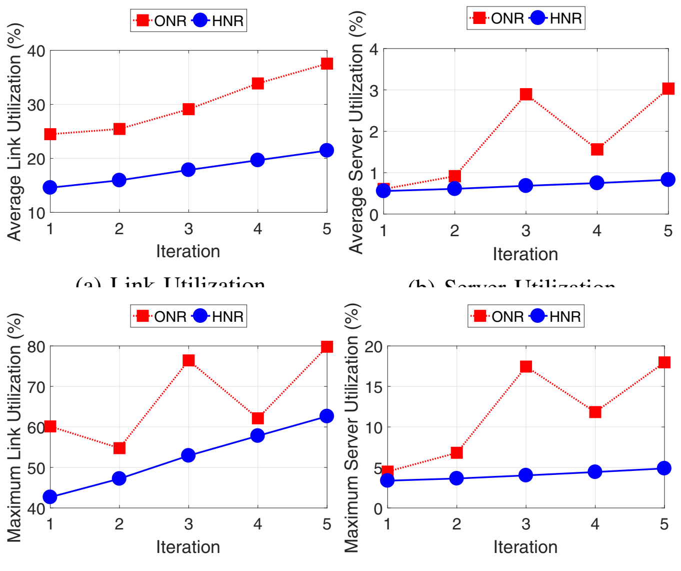
(c) Maximum Link Utilization



(d) Maximum Server Utilization

Fig. 5: Scenario 2: Utilization Comparisons.

In Fig. 6 we are considering the Scenario 3 which has a higher load. In this case, the maximum link utilization for the *ONR* algorithm approaches the maximum allowed link utilization. This results in a rerouting of the flows to avoid the bottleneck link(s) and the different solutions that are found for the different time slots can have a decrease in the maximum link utilization. This happens between the first and the second time slot and between the third and the fourth time slot in Fig. 6 (c). Due to this rerouting, the set of servers which are used to deliver the VNFs to the flows can be changed. With a new set of servers the server utilization (average and maximum) can decrease. This can be seen between the third and fourth time slot in Fig. 6 (b) and (d).



(a) Link Utilization (b) Server Utilization

(c) Maximum Link Utilization (d) Maximum Server Utilization

Fig. 6: Scenario 3: Utilization Comparisons.

In Fig. 7, we report the scenario 4 which has the highest load. The maximum link utilization of *ONR* reaches μ in the second time slot and remains steadily at the threshold (Fig. 7 (c)). This means that the considered optimal solutions will load the network up to the defined constraint in order to minimize the energy consumption. In fact, the server utilization for ONR is higher then HNR, because either less servers are ON, or the selected servers have a lower energy consumption and therefore lower capacity. Having lower capacity, they will achieve a higher utilization to serve the same flows.

3) Computational Complexity: Table V compares the execution time of *HNR* and *ONR* algorithms for different number of flows. As can be seen, the execution time of *HNR* algorithm is very low and it can be used in realistic network size.

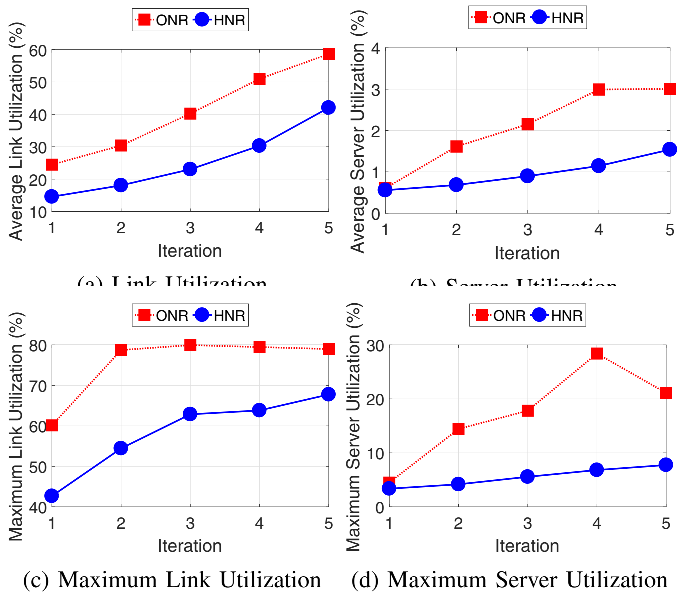
TABLE V: Computational Complexity.

Flow	ONR (s)	HNR (s)
10	16	0.007
250	1382	0.080
500	> 61000	0.163
750	> 61000	0.250
1000	> 61000	0.346
1500	> 61000	0.502
2000	> 61000	0.755

VII. Conclusion and Future Works

In this paper, we formulate the problem of service function chaining in an SDN-based network, with the goal of reducing the overall energy consumption as an Integer Linear Programming (ILP) problem. In our formulation, we control the link and server congestion by putting constraints on their maximum utilization. The resolution of the ILP problem using a solver tool (CVX) is referred to as *ONR*. Since the computational complexity of the proposed optimal solution ONR is high for networks of realistic size, a near-optimal heuristic called *HNR* is proposed. The proposed ONR and HNR solutions were compared in terms of power consumption, average path length, link/server utilization, and computational complexity. It is shown that *HNR* reconfigures the network achieving a near-optimal energy consumption, while it is still applicable for real-world networks.

In our ongoing work (see also [15]) we are considering the ordering constraints in the chain of VNFs and QoS parameters such as end-to-end delay in the formulation. Additionally, we are considering three modes for the servers (ON, OFF, and IDLE), allowing a node to be completely switched off. Another future direction of work is to consider switch failure probability in the routing algorithms, for example to enforce that the end-to-end probability of failure is less than a predefined threshold.



(a) Link Utilization (b) Server Utilization

(c) Maximum Link Utilization (d) Maximum Server Utilization

Fig. 7: Scenario 4: Utilization Comparisons.

VIII. Acknowledgment

This work has received funding from the Horizon 2020 EU project SUPERFLUIDITY (grant agreement No. 671566).

References

[1] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, “Virtual network embedding: A survey,” *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.
[2] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, “Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges,” *IEEE Communications Magazine*, vol. 55, no. 2, pp. 216–223, 2017.
[3] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, “Optimal virtual network function placement in multi-cloud service function chaining architecture,” *Computer Communications*, vol. 102, pp. 1–16, 2017.

[4] V. S. Reddy, A. Baumgartner, and T. Bauschert, “Robust embedding of vnf/service chains with delay bounds,” in *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*. Palo Alto, CA, USA: IEEE, 2016, pp. 93–99.

[5] B. Zhang, P. Zhang, Y. Zhao, Y. Wang, X. Luo, and Y. Jin, “Co-Scaler: Cooperative scaling of software-defined NFV service function chain,” in *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*. Palo Alto, CA, USA: IEEE, 2016, pp. 33–38.

[6] S. Kulkarni, M. Arumaithurai, K. Ramakrishnan, and X. Fu, “Neo-NSH: Towards scalable and efficient dynamic service function chaining of elastic network functions,” in *Innovations in Clouds, Internet and Networks (ICIN), 2017 20th Conference on*. Paris, France: IEEE, 2017, pp. 308–312.

[7] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba, “On orchestrating virtual network functions,” in *Network and Service Management (CNSM), 2015 11th International Conference on*. Barcelona, Spain: IEEE, 2015, pp. 50–56.

[8] G. Even, M. Rost, and S. Schmid, “An Approximation Algorithm for Path Computation and Function Placement in SDNs,” in *International Colloquium on Structural Information and Communication Complexity*. Helsinki, Finland: Springer, 2016, pp. 374–390.

[9] M. Ghaznavi, N. Shahriar, R. Ahmed, and R. Boutaba, “Service function chaining simplified,” *arXiv preprint arXiv:1601.00751*, pp. 1–7, 2016.

[10] M. Rost and S. Schmid, “Service chain and virtual network embeddings: Approximations using randomized rounding,” *arXiv preprint arXiv:1604.02180*, pp. 1–27, 2016.

[11] J. W. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, “Joint vm placement and routing for data center traffic engineering,” in *INFOCOM, 2012 Proceedings IEEE*. Orlando, FL, USA: IEEE, 2012, pp. 2876–2880.

[12] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, “An Approach for Service Function Chain Routing and Virtual Function Network Instance Migration in Network Function Virtualization Architectures,” *IEEE/ACM Transactions on Networking*, 2017.

[13] M. Grant, S. Boyd, and Y. Ye, “Cvx: Matlab software for disciplined convex programming,” 2008.

[14] internet2.edu, “Abilene network,” Jun 2017, [Online; posted 24-March-2012]. [Online]. Available: https://web.archive.org/web/20120324103518/http://www.internet2.edu/pubs/200502-IS-AN.pdf

[15] M. M. Tajiki, S. Salsano, M. Shojafar, L. Chiaraviglio, and B. Akbari, “Joint Energy Efficient and QoS-aware Path Allocation and VNF Placement for Service Function Chaining (extended version).” [Online]. Available: http://arxiv.org/pdf/1710.02611