International Journal on Applications in Information and Communication Engineering
Volume 4: Issue 1: March 2018, pp 29 - 43. www.aetsjournal.com ISSN (Online) : 2394 - 6237

OPTIMIZING RESOURCE ALLOCATION FOR VIRTUALIZED NETWORK FUNCTIONS IN A CLOUD CENTER USING GENETIC ALGORITHMS

M. PADMAVATHI , M. PRABHU , S. PONNUTHURAI

Abstract— With the introduction of Network Function Virtualization (NFV) technology, migrating entire enterprise data centers into the cloud has become a possibility. However, for a Cloud Service Provider (CSP) to offer such services, several research problems still need to be addressed. In previous work, we have introduced a platform, called Network Function Center (NFC), to study research issues related to Virtualized Network Functions (VNFs). In a NFC, we assume VNFs to be implemented on virtual machines that can be deployed in any server in the CSP network. We have proposed a resource allocation algorithm for VNFs based on Genetic Algorithms (GAs). In this paper, we present a comprehensive analysis of two resource allocation algorithms based on GA for: (1) the initial placement of VNFs, and (2) the scaling of VNFs to support traffic changes. We compare the performance of the proposed algorithms with a traditional Integer Linear Programming resource allocation technique. We then combine data from previous empirical analyses to generate realistic VNF chains and traffic patterns, and evaluate the resource allocation decision making algorithms. We assume different architectures for the data center, implement different fitness functions with GA, and compare their performance when scaling over the time.

Keywords— Network Function Virtualization (NFV), Cloud Resources Optimization, Genetic Algorithms.

I. INTRODUCTION

Network Function Virtualization (NFV) [1] is a promising technology that proposes to move packet processing from dedicated hardware middle-boxes to software running on commodity servers. As such, NFV brings the possibility of outsourcing enterprise Network Function (NFs) processing to the cloud. When an enterprise outsources its NFs to a Cloud Service Provider (CSP), the CSP is responsible for deciding:
(1) where initial Virtual NFs (VNFs) should be instantiated, and (2) what, when and where additional VNFs should be instantiated to satisfy changes in the traffic (scaling) with minimal impact on network performances.

Existing work on cloud resource allocation for VMs are not suitable for cloud resource allocation for VNFs. Optimizing the placement of VMs in a cloud tend to be node-centric as VMs are end-points. Optimizing the placement of VNFs is, in contrast, network-centric: their provisioning normally involves service chains of VNFs rather than individual VNFs. The placement of a service chain requires allocation of server resources (for the VNFs), as well as allocation of network resources (paths) to route traffic flow from one VNF to next VNF in the chain, within the cloud. Furthermore, most of the existing work on placement of VNFs, consider only a part of the problem, by optimizing either host or bandwidth resource, but do not provide an integrated view of computation, storage and networks optimization [2]. Despite some initial efforts [3], [4], [5], the dynamic resource allocation for scaling VNFs presents still many open challenges. One of the challenges is how to achieve scaling, i.e., whether to use horizontal (i.e., installation/removal of VNF instances), or vertical scaling (i.e., allocation/release of host and bandwidth resources to/from a VNF instance) or both. A second challenge is how to resolve potentially conflicting optimization objectives: for instance, re-allocating resources in a way that minimizes changes to current configuration, and therefore minimally disturbes current net-work activities, and at the same time optimize usage of server and network resources [6].

In this paper we argue that mixed Integer Linear Programming (ILP), traditionally used to optimize VM allocation and network management in cloud data centers [7], is not suitable for online scaling of VNFs in response to traffic changes. This is because solving ILP problems can take hours [8]. Instead, one can find suitable approximation algorithms for the optimization. In previous work [8], we started to investigate our hypothesis, by proposing two new resource allocation algorithms, based on Genetic Programming (GP), for the initial placement of VNFs, and the scaling of VNFs to support traffic changes. Similar to recent work in data centers [9], to make more efficient utilisation of the NFC resources, our approach allows both computing resources and network config-urations to be managed concurrently and assumes a Software-Defined/OpenFlow infrastructure [10] to easily reconfigure the physical network. In this paper, we provide a more in-depth analysis. Building upon our previous results [8], [3], we present an improved version of our GP resource allocation algorithms that use more effective genetic operations, conduct an extended evaluation of these algorithms, and discuss a comprehensive analysis of their performance. The specific contributions of the paper are as

M. Padmavathi , Final Year ME-Cse, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India
M. Prabhu , Final Year Me-Cse, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India
S. Ponnuthurai , Assistant Professor, Meenakshi Ramaswamy Engineering College, Thathanur, Tamilnadu, India

follows.

First, we define the Network Function Center Resource Management Problem (NFCRMP) as a set of ILP equations. They address both (1) the resource allocation for new VNFs

provisioning, and (2) the resource allocation for the scaling of existing VNFs to support traffic changes. In the context of resource allocation for new VNFs, the goal is to minimize the required resources (i.e., number of servers, number of links, and average link utilization). In the context of resource allocation for scaling of existing VNFs, the aim is to adjust the resources to satisfy the traffic changes and, at the same time, minimize the number of configuration changes to reduce potential service disruptions, and performance degradation. We developed algorithms to solve the NFCRMP using both a GP and an ILP approach.

Second, we implemented the ILP formulation of the NFCRMP in CPLEX [11] and implemented the approximation algorithms using GP, and compared their performances for small networks. We ran experiments where the objective con-sisted in deploying a set of new VNF chains, and considered VNF chains with 10 VNFs and 20 VNFs in a 4 server environment. In both cases, the GP gave the exact optimal solution as ILP. When we increase the number of servers to 16, GP was able to find solutions fast (average of 47 milliseconds), while the ILP implementation ran for nearly 6 hours, and even crashed without finishing correctly. The results confirm that the time to solve ILP problems renders ILP not suitable for the online deployment and scaling of VNFs, and suggest that the GP generated solutions can provide optimal – or close to optimal – solutions in significantly shorter amount of time, which is more suitable to the NFCRMP problem.

Third, we present a comprehensive evaluation of the pro-posed GP approach for large networks. We ran experiments both for the deployment of new VNF chains, and for the scaling of existing VNF chains, using realistic traffic pat-terns [12]. We considered three network architectures: (1) a k-fat tree architecture [13], (2) a BCube architecture [14], and (3) a VL2 architecture [15]. As the GP process relies on an initial solution, we used simple Depth First Search (DFS) and random approaches to find an initial solution, and considered this solution as the baseline to compare solutions given by GP process after 200 generations. Our results showed that the performance of the algorithms highly depends on the network architecture, as each network architecture has a different num-ber of nodes. The GP process provided an average objective value improvement of up to 7.87% over the initial solution (the baseline) which results in reductions of the average link utilization of up to 28.7%. Furthermore, our results showed that the GP algorithm can decide server and network allocations for hundreds of policies (around 400 VNFs) in a 128 server environment and find reasonable solutions in milliseconds.

Fourth, we study the quality of the GP generated solutions over time. Whenever we need to reallocate resources to scale, we adopted a “local approach” to find solutions, where we limit the resource re-allocation only to policies affected by traffic change. These “local approach” solutions may gradually diverge from the optimal solution of a “global approach” solu-tion given that the local approach solutions strive to minimize not only the required resources, but also the number of changes in the network. We compared the solutions computed by the GP approach with its “global approach” solutions, generated by ignoring the need to minimize the number of changes in the network. Our results showed that, although the “global approach” provides better resource allocations, the solutions require drastic re-arrangements to the current configurations, and therefore is impractical in real scenarios. In contrast, “local approaches” provide reasonable solutions with lesser changes to readjust configurations, without diverging from the “global approach” solutions over time.

The rest of the paper is organized as follows. Related work is presented in Section II. Section III gives a brief description of our experimental NFV platform and its management system, the Network Function Center (NFC). Section IV describes the formulation of Network Function Center Resource Manage-ment Problem (NFCRMP) as a set of ILP equations for new VNFs provisioning and dynamic scaling. Section V describes the GP based resource allocation algorithms implementations and a performance comparison between ILP and GP. Section VI describes our experimental set-up. Sections VII and VIII show the results of a comprehensive performance evaluation of GP based resource allocation algorithms. Our final remarks can be found in Section IX.

II. RELATED WORK

Traditionally Integer Linear Programming (ILP) has been used in cloud resource allocations to optimize VM allocation and network management [7]. However, this approach can be applied only if adjustments to traffic demands are made in the order of hours [8]. Finding an optimal allocation is com-putationally hard [16]. Hence, more practical solutions look for approximations. There have been several recent studies on optimizing orchestration and placement of VNFs using heuristic based approaches. Provisioning requests from cloud’s users involves service chains of VNFs instead of single VNFs [17], [18], [19], [20]. The placement of these VNF chains in physical machines and use of the network bandwidth are therefore crucial for performance of a NFC [21].

Most of the existing solutions focus on the initial placement of VNFs in the cloud, striving to minimize the number of VNFs instances used in the cloud, and possibly an overall network cost, using heuristics based resource allocation algo-rithms [22], [23], [24], [16], [25], [8], [3], [26], [27], [28]. Those existing solutions assume that a VNF instance can be shared across policies. In contrast, our work on initial placement of VNFs explores a different angle of the problem: we assume that a given VNF instance is dedicated to a single policy. This assumption is to provide tenant isolation.

To offer infrastructure resources for client's traffic on an on-demand basis, following the initial placement, VNFs may need to be rescaled over time. The work on scaling of VNFs is limited [3], [4], [5]. Similarly to the initial placement problem, existing solutions for scaling also look for approximations. Li et al. works with VNF chains and solves the initial placement problem by iterating over initial input chains [4]. Scaling is achieved by duplicating full instances of VNF chains and the optimization is very dependent on the datacenter architecture. Milad et al. optimizes one VNF at a time, independent of the order of the VNFs in the VNF chain. In addition, the proposed solution performs a local optimization, as they do not consider optimizing the placement of full VNF chain [5]. In contrast, our model takes a middle approach: we consider chains of VNFs and assume that in a chain the increase of resources demand is likely to happen in isolated VNFs. However, we perform the optimization in the context of the full chain. We optimize server and bandwidth utilization as well as the cost of making modifications. Furthermore, our approach does not depend on the datacenter architecture.

III. OVERVIEW OF EXPERIMENTAL PLATFORM: NETWORK FUNCTION CENTER (NFC)

We are building an experimental platform, called Network Function Center (NFC), to study management issues related to the combined management of VNFs. The architecture and the complexity of our experimental NFC platform are simpler than those defined by existing standardization bodies (e.g., ETSI, IETF) [1]. This simplification allows us to focus on specific research aspects and conduct experiments. Contrary to traditional NFs that are hardware based middle-boxes, deployed at specific locations in the network, the NFC assumes a NF to be implemented by a VM that can be deployed in any server in the Cloud Service Provider network. In this section, we briefly describe the functionality we expect from a NFC and the proposed architecture.

We assume that the NFC delivers VNFs as a service to clients on a subscription basis. To receive services from a NFC, a client needs to provide the following specifications:

(1) the types of required VNFs, and interconnectivity between them (policy), (2) the ingress and egress locations of client's traffic flow, and (3) the initial expected traffic load to be processed by these VNFs. Once the client request is accepted by the NFC, the client's traffic is redirected to the NFC to traverse the VNFs. The NFC must guarantee that the client's traffic traverses all the VNFs in the correct order. However, traffic may change over time, when compared to the initial expected traffic load provided by the client. Therefore, the client can request for a flexible service level agreement where for example, the infrastructure resources are adjusted to satisfy the traffic changes and demands. The NFC is expected to scale resources (horizontally or vertically) to handle the traffic changes according to the agreements with the client.

The NFC consists of two main components: a physical infrastructure, and a management system for the infrastructure. The physical infrastructure comprises a network and a server infrastructure. The network infrastructure provides connectivity for all communications occurring in the NFC and between the NFC and its users. The server infrastructure hosts all VNFs. Servers in the NFC are used to deploy the virtual machines (VMs) where the VNFs run. A NF is implemented as a software on a VM. The NFC should configure its network to route traffic flow from one VNF to the next VNF of the policy (to its successor), according to the order given in the policy.

The goal of the NFC Management system is to automate arrangement, coordination and management of NFC components to satisfy the maximum number of client requests with a specified level of QoS. The NFC Management System is built around five key modules: (1) Resource Manager, (2) Topology Manager, (3) Flow Manager, (4) Elasticity Manager and (5) Rules Generator. The Topology Manager is responsible for maintaining the state of the physical infrastructure of the NFC. This includes information about current network information (i.e, server and link utilization information) and topology data (i.e, network architecture). Once a new client request is submitted, the Resource Manager module considers current network information and topology data (provided by the Topology Manager) in addition to the constraints such as maximum capacity of servers and links, and takes decisions on the placement of VNFs and the paths for the client's traffic to follow inside the NFC. The Resource Manager is also called by the Elasticity Manager. The Elasticity Manager monitors the resource's utilization and takes decisions on when to scale resources for the traffic changes. The Resource Manager then determines the re-allocation of server and network resources to satisfy the new demands based on the current network infor-mation, topology data and constraints. The Flow Manager, and Rules Generator configure the network according to decisions taken by the Resource Manager and Elasticity Manager. More details of the architecture of our NFC can be found in [8].

IV. NETWORK FUNCTION CENTER RESOURCE MANAGEMENT PROBLEM (NFCRMP)

The Resource Manager module has two responsibilities:

1) New policy requests provisioning: upon receiving a new set of policies, the Resource Manager takes into account the physical network, servers constraints, and already allocated resources, to identify the resources where to instantiate the VNFs of the new policy;
2) Scaling of existing policy requests: upon receiving scal-ing requests from the Elasticity Manager, the Resource Manager decides the re-allocation of resources in order to satisfy traffic changes.

In the following section, we formalize our problem, the Network Function Center Resource Management Problem (NFCRMP), as a set of ILP equations for (1) resource allocation for the new policy requests provisioning, and (2) resource allocation for the scaling of existing policy requests to support traffic changes. For the new policy requests provisioning, the NFCRMP aims at minimizing the required server and network resources (e.g., average link utilization.) For the scaling of existing policies, in addition to minimizing the required server and network resources, the NFCRMP also aims at minimizing the number of changes in server and links configurations. Table I provides a description of the key notations used in NFCRMP.

A. New policy requests provisioning

We consider a NFC with M servers and L links. A link l connects a server to a switch, or a switch to another switch. The amount of resource capacity of server m is denoted H_m, and the network capacity of link l is denoted K_l. A path p between two servers (a source and a destination server pair), is composed by two or more links. P denotes the set of the shortest paths between all source and destination server pairs in the NFC. Given a path p (in P) that connects server m1 and m2, Q_p represents the source server (m1), and R_p represents the destination server (m2). The variable E_l^p indicates whether link l is used on path p. As explained in Section VI-D, the definition of the shortest path can vary based on the network architecture type, as each of them have different default maximum hop count for a path between two servers.

TABLE I - SUMMARY OF KEY NOTATIONS

Constants for new policy requests provisioning	
N	No. of VNFs, indexed by n = 1; :::::; N
S_n	Server capacity required for VNF n
B_n	Bandwidth required for VNF n
M	No. of servers, indexed by m = 1; :::::; M
H_m	Capacity of server m
L	No. of links, indexed by l = 1; :::::; L
K_l	Bandwidth of link l
P	No. of paths, indexed by p = 1; :::::; P
Q_p	Source server of the path p
R_p	Destination server of the path p
E_l^p	Indicates whether the link l is used on path p, (0; 1)
$w_1; w_2; w_3$	Weighting factors
Dynamic variables for new policy requests provisioning	
Z_n^m	A binary decision for placing VNF n on server m
A_n^p	A binary decision for routing traffic of VNF n on path p
U	Average of link capacity used percentages
G_m	Server m is used/not
X	Total servers used
F_l	Link l is used/not
Y	Total links used
Additional constants for scaling	
	Previous state's no. of VNFs
	Previous state's no. of servers
N^U	Previous state's no. of links
M^U	Previous state's no. of paths
L^U	Previous state's link l is used on path p
P^U	Previous state's binary decision for placing n on server m
$(E_l^p)^U$	Previous state's binary decision for routing traffic of n on
$(Z_n^m)^U$	path p Weighting factors
$(A_n^p)^U$	
$w_4; w_5$	Additional dynamic variables for scaling
C	Total servers changed from previous state to current state
D	Total links changed from previous state to current state

The number of VNFs running in the NFC is denoted by N. Each VNF n is characterized by its resource requirements:

(1) the required server capacity (S_n), and (2) the required bandwidth: the expected amount of the traffic flow (B_n). For each VNF n in the requested policies, we find a server to place the VNF. The server must support the physical resource requirements of the VNF. Also, if the VNF is not the last VNF of the policy chain, we find a path to route the traffic of the VNF to its successor in the policy chain. The successor is the next VNF(s) according to the sequence in the policy chain. Links in the selected paths should support the bandwidth requirements of the VNF.

We define Z_n^m to be a binary variable for placing VNF n on server m, such that, if VNF n is placed on server m, then Z_n^m = 1, otherwise it is 0.

Let G_m 2 f0; 1g be a binary variable indicating whether server m is used in a configuration solution. Therefore, the total number of servers used in a configuration solution is:

$$X = \sum_{m=1}^{M} G_m$$

The traffic flow of VNF n to its successor is represented by the vector B_n. To configure the routing between VNF n and its successor, we need to find a path in P , joining the servers where VNF n and its successor reside. Therefore, we define A^p_n to be a binary variable that indicates if path p is used to route traffic between VNF n and its successor, such that, if traffic of n to its successor is routed on path p, then A^p_n = 1, otherwise it is 0.

Let F_l 2 f0; 1g be a binary variable indicating whether link l is used in a configuration solution. Therefore, the total number of links used in a configuration solution is: $Y = \sum_{l=1}^{L} F_l$ defined as:

For link l, the percentage of link utilization is

$$\left(\sum_{p=1}^{P}\sum_{n=1}^{N} A_n^p \cdot E_l^p \cdot B_n\right)/K_l$$

Therefore, considering all the links in the configuration solution, the average percentage of link utilization U, is:

$$U = \left(\sum_{l=1}^{L}\left(\left(\sum_{p=1}^{P}\sum_{n=1}^{N} A_n^p \cdot E_l^p \cdot B_n\right)/K_l\right)\right)/L$$

The NFC Management System takes decisions on new policy requests provisioning, with the goals of minimizing the average link utilization and the number of servers used. We assume that if a VNF has to be placed on a server, then the server has to be switched on and the server is considered as a server that is in use. Therefore, we want to minimize the number of servers used, so that the already switched on servers, can be utilized efficiently. Furthermore, as the NFC network has more than one path between most of the (source, destination) server pairs, and these paths mostly use different links, the NFC Management System tries to maximize the number of used links, so that the system is encouraged to use different paths to route traffic between each (source, destination) server pair. This results in distributing traffic over different paths, and reducing the average link utilization. We highlight that since U is an average, we normalize the following optimization function by considering the number of servers M, and links L, and introduce weighting factors w_1;

w_2; w_3 to allow operators to tune the trade-offs between the optimization factors. The NFCMP can be explained as the following constrained optimization:

Minimize

$$w_1 \frac{1}{M}:X + w_2:U + w_3(1 \quad \frac{1}{L}:Y) \qquad (1)$$

Constraints (2a) and (2b) model the server resource constraints of the NFCMP. Constraint (2a) guarantees that each VNF in a policy is placed on one and only one server. Constraint (2b) guarantees that, for each server, if the server is used (captured by G_m), then the total capacity consumed by all VNFs placed on the server does not exceed the total capacity of that server. Constraints (2c) to (2g) model the network resource constraints of the NFCMP. If a VNF is not the last VNF of a policy, constraint (2c) guarantees that the VNF has a path to its successor. Constraint (2d) guarantees that for each link, if the link is used (captured by F_l), then the total bandwidth consumed by the VNFs does not exceed the total bandwidth of that link. Since we are maximizing "the number of links used" in the objective function, constraint (2e) guarantees that a link is counted as "used", only if it is actually used in the configuration solution. Constraints (2f) and (2g) guarantee that the path selected for a VNF to send traffic to its successor, starts from the server where the VNF resides (source server), and ends in the server where the VNF's successor resides (destination server).

subject to

$$\overset{M}{\underset{X}{-}} \quad Z_n^m = 1; 8n \qquad (2a)$$

$$\overset{N}{\underset{X}{-}} \quad Z_n^m:S_n \quad H_m:G_m; 8m \qquad (2b)$$

$$\overset{P}{\underset{X_p}{}} \quad A_n^P = 1; 8n \qquad (2c)$$

$$X_p \quad \bar{X} \quad A_n^P:E_l^P:B_n \quad K_l:F_l; 8l \qquad (2d)$$

$$X_p \quad \bar{X} \quad A_n^P:E_l^P \quad F_l \quad 0; 8l \qquad (2e)$$

$$_{p=1} \quad A_n^P:Q_p \quad _{m=1} \quad Z_n^m:m = 0; 8n \qquad (2f)$$

$$_{p=1} \quad A_n^P:R_p \quad _{m=1} \quad Z_n^{m}2:m = 0; 8(n; n2) \qquad (2g)$$

B. Scaling of existing policy requests

In the scaling situations, the optimization needs to consider the current configurations of the system, so that we can minimize the disturbances to the existing traffic flows when implementing the solutions provided by the optimization to satisfy the new traffic changes. Therefore, while trying to minimize the link utilization, and required servers in the optimization process, we also try to minimize the changes to the current system. We use additional variables to represent the previous state of the NFC, i.e., the state of the NFC before scaling. N^0 , M^0 , L^0 and P^0 represent the number of VNFs, number of servers, number of links and number of paths in the state before the scaling. $(E_l^p)^0$, $(Z_n^m)^0$ and $(A^P{}_n)^0$ represent whether link l is used on path p, the binary decision for placing n on server m, and the binary decision for routing traffic of n on path p in the state before the sclaing.

Hence, the total number of servers changed, C, and total number of links changed, D, from the current state to the new state, is captured by the following equations:

$$C = [\quad N \quad M \quad Z^m \quad (Z^m)^0 \quad] + [N^0 \quad N \quad]$$
$$n{=}1 \quad m{=}1 \qquad n \qquad n \quad j \quad j \qquad j$$

$$D = \quad P_N \quad P_L \quad j_P \quad A^P:E^P \quad (A^P)^0:(E^P)^0$$
$$n{=}1 \quad l{=}1 \quad p{=}1 \; j \quad n \quad l \qquad n \quad l \quad j$$
$$+ \quad N^0 \qquad L \qquad P^0 \, (A^P)^0 \; :(E^P)^0$$
$$=$$
$$n{=}N{+}1 \quad l{=}1 \qquad 1 \quad n$$
$$P \quad P \qquad P_p \qquad l$$
$$P \qquad P \; P$$

The optimization function of the scaling process tries to minimize the server and link changes, in addition to the three parameters introduced in Equation (1) for the optimization of new policy requests provisioning. Therefore, the optimization function of the scaling process includes the parameters C and D. Operators can then freely tune the trade-offs between these five optimization parameters, using the weighting factors w_1; w_2; w_3; w_4; w_5.

The scaling component of the NFCRMP, can therefore be formalised as the following constrained optimization:

Minimize

$$w_1 \frac{1}{M}:X + w_2:U + w_3(1 \quad \frac{1}{L}:Y) + w_4 \frac{1}{M}:C + w_5 \frac{1}{L}:D \; (3)$$

V. NFCRMP: GP APPROACH

As shown in previous studies [24], [16] finding an optimal solution for the VNFs placement is a NP-hard problem. Furthermore, our results in Section VC1 show that finding the optimal solution for the ILP Equation (1) can take hours and is consequently not suitable to meet traffic changes in the NFV context. Instead of finding optimal solutions (e.g., solutions returned by an ILP solver), we, therefore, believe it to be more realistic to look for good feasible configurations. We explore approximation techniques, and model the problem as finding the best fitted solution according to a Genetic Algorithmic (GA) model of the problem, after a fixed amount of generations have been explored. The two main responsibilities of the Resource Manager module new policy requests provisioning, and scaling of existing policy requests are implemented independently but both rely on Genetic Programming (GP) as the mechanism to allocate resources.

GAs are part of evolutionary computing and were introduced as a computational analogy of adaptive systems [29]. They are modelled loosely on the principles of the evolution via natural selection, employing a population of individuals that undergo selection in the presence of variation, inducing operators such as mutation and crossover. A fitness function is used to evaluate individuals, and reproductive success varies with fitness.

GAs can be described by the following five key steps [29]:

1. Generate an initial population F (0) with n full solutions
2. Compute the fitness value u(f) for each individual full solution f in the current population F (t)
3. Generate the next population F (t + 1), by selecting i best full solutions from F (t)
4. Produce offspring by applying the genetic operators to population F (t + 1)
5. Repeat from Step 2 until a satisfying solution

Following the terms used in GA, a possible configuration state of the NFC (represented by the servers and paths assignments for VNFs) is considered as a full solution f, if it is an allocation of server and network resources for all the policies in the system. We call a configuration where only one of the policies has been allocated resources, a partial solution. If there are m number of policies in the NFC, then a full solution contains m number of partial solutions, each representing the allocation of resources (i.e., servers and paths) for each policy. The population F (t) consists of n full solutions which represents different possible configuration states for the NFC. We have considered two types of genetic operators to produce offspring: (1) mutation and (2) crossover. The crossover is a convergence operation which is intended to pull the population towards a local min or max. On the other hand, the mutation is a divergence operation which is intended to occasionally break one or more members of a population out of a local min/max space and potentially discover a better space. Since the end goal is to bring the population to convergence, crossovers happen more frequently (typically every generation). The mutation, being a divergence operation, should happen less frequently, and typically only affects a few members of a population in any given generation.

In our implementation, mutation is achieved via two independent operations: replacement and rewire. In the replacement mutation we try to change the currently allocated server of a VNF we remove the VNF from the current server and try to place it in a different server. In our previous work [8], we tried to change the server of a single VNF of a selected policy. However, in this work, we try to change the server of all VNFs of the selected policy, and try to place all the VNFs of that policy in a different server. Specifically, we select a random full solution from the population and randomly pick a partial solution from the selected full solution. We, then, attempt to find a new server where all VNFs in that partial solution can be placed on. If a new server is available to place the selected VNFs, then we find the necessary paths between selected VNFs and their successors by considering the new placement. We have observed that trying to change the placement of all VNFs of a policy and place them in a different (single) server provides better solutions than trying to change the placement of a single VNF of the policy. The next mutation is the re-wiring, where we try to change the path between two given VNFs and find a different path. Similar to re-placement mutation, we first select a random full solution from the population and randomly pick a partial solution from the selected full solution. Then, we select a random VNF in the partial solution and attempt to find a new path to its successor.

As for crossovers, we first select two random full solutions from the population and randomly pick partial solution from each selected full solution. Then, we check whether the configuration given in the first partial solution can be applied to the second partial solution and vice versa. If both ways are possible, then the configurations of partial solutions will be changed accordingly.

Each generation of the GP approach goes through mutations and crossovers. The newly generated solutions are evaluated according to a fitness function. We use two different fitness functions, one for the new VNFs provisioning, and another for the scaling out/in. These fitness functions are derived according to the optimization functions defined in the ILP formulation, namely equations (1) and (3).

A. New policy requests provisioning: global approach

For the new policy requests, the Resource Manager uses network's traffic, topology data, server constraints and the client requirements as inputs. Within the given physical network constraints and previously allocated resources for the existing policies, first, for each VNF in each new policy request, the Resource Manager selects: (1) a server depending on the server capacity requirement of the requested VNF and (2) a path(s) depending on the expected traffic load for the requested VNF. We have considered two types of initial selections: (1) Depth First Search (DFS), and (2) Random. In the DFS method, servers and paths are selected by searching through the whole search space and selecting the first solution we come across. The random method searches servers and paths randomly anywhere in the network, until a feasible configuration is found. The configuration state (the servers and paths allocation) that the Resource Manager comes up with for a new policy request, i.e, a partial solution. Combination of all partial solutions (each representing a policy) forms a full solution. Second, the Resource Manager applies the fitness function derived from the optimization function in Equation (1), to each full solution. Unless specifically mentioned, for all our experiments, we assumed equal weights for all the parameters in the fitness function. Third, the full solutions that return small values are preferred, and selected as the best solutions for the next generation population. Fourth, the Resource Manager performs "global approach" where it applies genetic operators (mutations and crossover) on randomly selected partial solutions of randomly selected full solutions to generate offsprings, i.e., new full solutions. The last three steps are repeated until x number of generations are explored and the best full solution is selected as the configuration for the new VNFs provisioning.

B. Scaling of existing policy requests: local approach

For the scaling, the Resource Manager starts with the current state and search for the re-assignment of resources (servers and paths) for the set of VNFs that are scaling, using DFS/random methods. Partial solutions relevant to the scaling are modified according to the servers and paths found. The fitness function, derived from the optimization function in Equation (3), is used to measure how good each full solution is. As in the previous new policy requests provisioning, unless specifically mentioned, for all our experiments, we assumed equal weights for all the parameters in the fitness function. In contrast to the "global approach", which is performed during the initial resource allocation process, when scaling we adopt a "local approach". Because we want to minimize the changes to current configurations, mutations and crossovers are carried out only to the VNFs which were changed because of the scaling (not to the all VNFs of the policies that are scaling). As mentioned earlier, the process is continued until x number of generations are explored and the best full solution is selected as the configuration for re-assignment of the policy.

C. NFCRMP: GP approach vs ILP approach

We compare the GP approach with the ILP approach in the case of new policy requests provisioning. We have implemented the ILP formulation of the NFCRMP in CPLEX [11] (version 12.5.1 with default settings) and carried out our experiments in a machine with an Intel core i7-4500u processor and 8GB of RAM.

1) Configuration solution timing:

We have conducted a set of experiments to compare the time taken to find a solution for new policy requests provisioning by ILP and GP approaches. For the GP process, the total time includes both (1) to find an initial solution using DFS, and (2) the GP process over 200 generations. We have considered a small NFC with a k-fat tree architecture. We have assumed an environment with 2 pods and 4 servers, where each pod is connected 2 servers. For a set of policy requests with total of 10 VNFs, ILP took 0.9 seconds while GP took 0.0032 seconds and for a set of policy requests with total of 20 VNFs, ILP took 12.5 seconds while GP took 0.0034 seconds to find the exact optimal solution. However, when we increase the number of pods and servers, for an example, when we have a topology with 4 pods and 16 servers, where each pod is connected 4 servers, ILP implementation ran for nearly 6 hours, but crashed without finishing correctly. Therefore, although the ILP formalisation of the problem gives the optimal solution, the ILP computational time requirement makes it not suitable even for a few VNFs in a large scaled network.

2) Configuration solution quality:

In the next set of experi-ments, we compared the quality of the solution for new policy requests provisioning of ILP and GP approaches. In addition to the objective value, we specifically looked at the number of servers and links have been used and the link congestion in solutions of ILP and GP approaches. We have assumed an environment with 2 pods and 4 servers, where each pod is connected 2 servers. As explained in earlier sections, the GP process can rely on either (1) DFS, or (2) a random approach to find the initial solution. We carried out separate GP process experiments with both types of initial solutions. We explored different classes of problems where we assume that the 10 VNFs are distributed over one, two or three policies. By varying the capacity requirements of VNFs, we observed that there are different classes of these problems, where the differences are based on number of servers required by these VNFs. We made sure to select 3 cases in which the DFS would not give the optimal solution, because we wanted to explore how the GP process improves the solution given by DFS. Specifically: (1) 10 VNFs belong to three policies, but all of them fit onto a single server, (2) 10 VNFs belong to two policies, and they fit onto two servers and (3) 10 VNFs belong to three policies, but they fit onto a single server.

In all 3 sets of experiments, DFS gave better initial solution than random. Therefore, when the DFS solution was given as the input to the GP process, in all 3 cases, within 200 generations, the GP algorithm was able to find solutions with objective values that are exact to the optimal solutions given by ILP. In addition to the objective value, the number of servers and links have been used and the link congestion in solutions of GP approach were similar to the solutions of ILP approach. As described in Section V-A, the DFS method is a good bin-packing strategy, and therefore the solutions given by the DFS uses minimal number of servers required. Also, it introduces less inter-rack traffic, as it tries to place VNFs of a policy in the same server as much as possible. Since the random method selects servers and paths randomly that can be anywhere in the network, it uses much more servers and introduces high inter-rack traffic, as the policy is splitted and VNFs are placed in servers that are in different pods. Because of these reasons, the initial solutions provided by the DFS was better than the random approach. However, as the solution provided by the DFS tended to use fewer links and those links were congested, GP was able to improve the solution by using different paths with different links to distribute the traffic and reduce the average link utilization.

VI. EXPERIMENTS FOR NFCRMP WITH GP APPROACH

Experiments results, described in Section V-C, show that the ILP approach is not suitable for large networks with dynamic requirements, but the GP approach can find reasonable solu-tions fast. We have therefore conducted a more comprehensive evaluation of the performance of the NFC Management System when using our proposed GP approach. The rest of the paper focuses on this evaluation and specifically on the performance of the Resource Manager module when using the GP approach for large networks with dynamic requirements.

We have developed a prototype of the NFC Management System, in C++ and Python. Conceptually, the Resource Manager, Topology Manager, Elasticity Manager and Flow Manager can be seen as controller applications, while the Rule Generator as an extension to the network operating system. The network of our prototype makes use of SDN to allow programmatic control over the traffic flow and easy reconfiguration of the physical network. We have implemented the physical structure in Mininet [30], used Ryu [31] as the SDN controller, and bro firewalls and iptables as VNFs [10]. To conduct a more realistic evaluation, we needed data on:

(1) potential VNFs chains (policies), (2) traffic flows passing through these VNFs chains, (3) how the traffic changes affect the VNFs (scaling) and (4) different data center architectures for the NFC. However, there are no publicly available real data sets on VNF chains and traffic that pass through VNF chains. In our previous work [8], we evaluated the GP approach with randomly generated data (for policies and traffic patterns), but going further, for this paper we have used more realistic data from previous empirical analyses [32], [12] and made some assumptions to derive the required data [33]. We developed four programs to model the gathered data and generated the required data. The data generating process is described in the following sections. All gathered data and data modelling programs are publicly available at [34].

A. Policy requests

When generating policy requests for the NFC, the main factor to be considered is the type (e.g., small, medium, large size network) of the enterprise/user, that is requesting the policies. Depending on the type of the enterprise/user, the total number of VNFs required, the number of VNFs in a policy and types of the VNFs in the policy can vary. The policies used in our experiments are generated based on a study about physical middle-boxes used in enterprise networks [32], which includes figures about types of enterprise networks, number and types of middle-boxes used in them. For our experiments,

following statistics given in [32], we have assumed that we are going to provide services for 4 large enterprise networks, each enterprise network having 100 VNFs. The number of VNFs in a policy follows a truncated power-low distribution with exponent 2, minimum 2 and maximum 7. Therefore, for our experiments, we derive a set of policies for each enterprise, where each set of policies have 100 VNFs and altogether all the policies of four enterprises have 400 VNFs.

B. Traffic flows

When simulating traffic, we need traffic data where owners (enterprises/users) of the flows can be identified, so that we can differentiate the traffic passing through each policy. The traffic load that each enterprise/user is expecting can vary according to their target applications [35]. We consider web-based applications and for the traffic, we rely on empirical data from previous studies [12]. The data set includes an HTTP traffic breakdown of 30,000 users for a day which is measured at three different vantage points of an Italian ISP over a period of 24 hours. The traffic breakdown reports traffic for every 2 hours. We focused on the traffic statistics of 4 enterprises: Megaupload, LeaseWeb, Level3 and Limelight.

In a data center, traffic changes happen throughout the day and according to the amount of these changes, the VNFs should be scaled to satisfy the current traffic demand. A limitation of the HTTP traffic data we are using is that, information was collected at every two hours. Therefore, the first challenge is interpreting the pattern of traffic change over two hours. Other studies (e.g., [36]) show that traffic changes on usual days happen gradually over time. From times when traffic may increase significantly, changes may still increase gradually over 15 minutes time periods [37]. As such, although sudden traffic changes may occur within few minutes, we have assumed a uniform traffic increase/decrease over the 2 hours time intervals. To reflect scaling requirements of all situations, we spread the increase/decrease of number of VNFs (needed for the full 2 hour traffic change) over 2 hours and increase/decrease the capacity of one VNF at a time. The second challenge is identifying the policies affected by each enterprise traffic change. For each enterprise we have x number of policies generated and each policy has a unique traffic passing through its VNFs. When there is a change in total traffic for that enterprise, it is very unlikely that traffic passing through all policies of that enterprise contributed to the change. Therefore, we randomly select a subset of policies from that enterprise, as the policies affected by the traffic change.

C. Scaling

After selecting the policies affected by each enterprise traffic change, the first challenge is deciding which VNF from each policy, needs to be scaled to satisfy the new traffic demands. An earlier study [26] shows that in general no two VNFs will be simultaneously and equally bottlenecked and scaling one VNF in the policy at a time is the best strategy. Hence, assuming the conditions in [26], we randomly select a VNF from each policy as the bottlenecked VNF for which the resource allocation needs to be increase/decrease. The second

challenge is, from the identified VNF instance to scale, how many instances we should add/remove to satisfy the new traffic demand. Here, we are making an assumption: the traffic flowing through the VNF instance is proportional to the capacity of the VNF instance and it is the same for all types of VNFs. Therefore, the initial capacity unit requirement of all types of VNFs is assumed to be the same. Another study [38] shows that if we add more than one instance at a time, we are usually adding more than what is needed and wasting resources. Therefore, we calculated a traffic change threshold to find how many instances we should add/remove to accommodate traffic change, and as explained in Section VI-B, we add/remove one instance at a time. This resulted in 42 significant events over the 24 hours of traffic data. There are two types of events: (1) when the traffic change has reached the threshold, resources have to be reallocated to increase/decrease at least one VNF instance or (2) when the traffic change has not reached the threshold, modify the bandwidth usages of the links of the paths that were effected by the traffic change, to reflect the new traffic amount passing by.

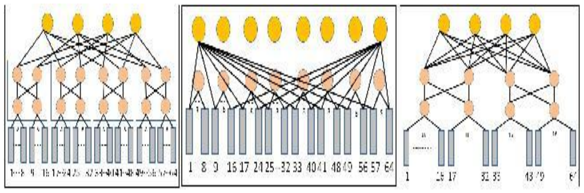

Figure 1: Architectures used for NFC

D. Data center architectures for NFC

We evaluated the performance of the resource allocation algorithm assuming three different data center network architectures for NFC: (1) k fat tree, (2) VL2 and (3) BCube shown in Figure 1. A k-ary fat-tree network [13] has three layers: a core layer, an aggregation layer and a Top-of-Rack (ToR) layer. It consists of $(k=2)^2$ core layer switches and k pods of k switches, half of them aggregation switches and the other half ToR. Each switch in a pod has k ports. The ToR switches are at the bottom of the pod, and the aggregation switches in the middle. In one pod, each ToR switch is connected to every aggregation switch. Each aggregation switch connects to (k=2) switches on the core layer. We have used a 4 fat-tree architecture, which has 20 switches: 4 pods of 4 switches and 4 switches in the core layer. For a NFC with 64 servers, 8 servers are connected to each ToR switch. The network consists of 96 links and 13770 paths connecting all source destination server pairs with maximum number of hops for a path of 6. The VL2 architecture [15] shares many features with an k-ary fat-tree architecture, but the main difference is the core tier and aggregation tier form a Clos topology: the aggregation switches are connected with core one's by forming a complete bipartite graph. We have used a VL2 architecture with 12 switches. For a NFC with 64 servers, the network consists of 88 links and 33760 paths connecting all source destination server pairs with maximum number of hops for a path of 6. In the BCube architecture [14], servers are considered part of the network infrastructure, i.e., they forward packets on behalf of other servers. A BCube is a recursively defined structure. At the level 0, a $BCube_0$ consists of n servers that connect together with a n-port switch. A $BCube_k$ consists of n $BCube_{(k-1)}$ connected with n^k n-port switches. We have used a $BCube_1$ architecture where there are 8 $BCube_0$s, each connected to 8 switches in the next level switches and form the $BCube_1$. Each s server of $BCube_0$s is connected to switch s of $BCube_1$. For a NFC with 64 servers, the network consists of 128 links and 7168 paths connecting all source destination server pairs with maximum number of hops for a path of 4.

VII. EVALUATION FOR NEW POLICY REQUESTS PROVISIONING

In this section, we will describe the results of the experiments that were carried out to evaluate the performances of GP for one of the main functions of the Resource Manager: new policy requests provisioning. Again, all experiments were carried out in a machine with an Intel core i3 processor and 20GB of RAM. As described in Section VI, we assume that we are going to provide services for policy requests of 4 large enterprises, therefore the Resource Manager has to handle new policy requests that consists of a total of 400 VNFs. For our experiments, unless explicitly mentioned, we have assumed situations where 50% of the NFC servers have to be used to allocate resources for policy requests.

A. Comparison of GP, DFS and Random

The GP approach takes an initial solution as the input to the GP process, and tries to improve the given initial solution using genetic operations. As explained in Section V, we use (1) DFS or (2) a random approach to find the initial solution. Therefore, we can use these DFS and random solutions as the baseline to compare the solutions given by the GP process after 200 gener-ation. First, we carried out experiments (50 rounds) to compare the quality of the solutions provided by DFS and random approaches for large networks. For each round of experiment, we derived a fixed set of policies (average of 100 policies) that include a total of 400 VNFs and try to find solutions with DFS and random in a 128 server environment in a 4-fat tree architecture network. We assumed that each server has an initial capacity of 1000 units, each link has an initial capacity of 3000 units and each VNF requires 100 server capacity units. Similar to the results observed in smaller networks (Section V-C2) and for the same reasons, in all the experiment rounds, the initial solutions provided by the DFS was better than the random approach. Furthermore, we extended our experiments to all three architectures (a 4-fat tree, a BCube and a VL2) and explored how the GP process improved the initial solution given by DFS over 200 generations. As the main goal of our optimization is to reduce the average link utilization, so that the network is less congested and future scaling requirements are minimized, the results produced by GP reduced the average link utilization by 28.7%, 3.2% and 14.9% (compared to the DFS solution) respectively for the three architectures.

B. Effect of the number of generations

As explained in Section V, the GP process tries to improve the given initial solution by applying genetic operations over the generations. To explore how GP process improves the initial solution, we conducted 30 round of experiments. For each round, we derived a new set of policies (average of 100 policies) that include a total of 400 VNFs. First, we tried to find the initial solutions with DFS, and then improve the solution using GP process. We count the times that the fitness value was improved during the GP process. We assumed a 128 server environment in a 4-fat tree architecture where each server has an initial capacity of 2000 units, each link has an initial capacity of 6000 units and each VNF requires 75 server capacity units. The important observation was that most of the improvements in the fitness function (52% from total number of improvements) happens early on (during first 100 generations) and after that improvements decrease significantly. In fact there were very few improvements after 400 generations: 6% from the total number of improvements.

C. Effect of the number of servers and nodes in the NFC

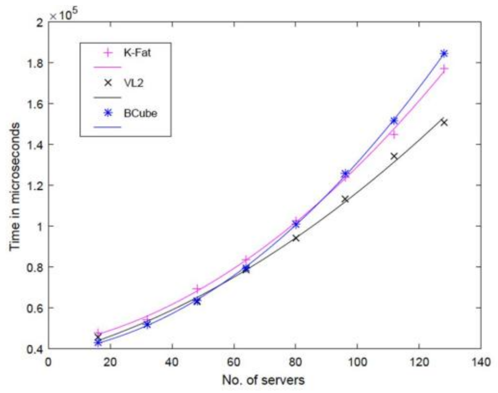

Figure 2: Factors effecting GP timing: No. of Servers

To better understand how the GP approach performs for large networks, we carried out a set of experiments to calculate the total time taken by the GP approach to provide a solution for new policy requests provisioning (we used 87 fixed policy requests that consists of 400 VNFs) in large networks. These total times include the time taken by the Resource Manager:
(1) to perform DFS to come up with an initial solution and (2) to run the GP process over generations to improve the initial solution. With the current implementation of the algorithm, most of the steps of the DFS process, such as finding a server or a path for a VNF in the policy, are performed in logarithmic time (the server capacities are stored in a sorted balanced tree and operations to the tree such as searching and updating can be done in a logarithmic time). However, in the GP process, when we perform a genetic operation and try find an improved solution to grow the population, we keep a copy of the original solution. We observed that the time taken for the GP process is dominated by this copying process. The complexity of copying the original solution, depends on the size of data structures that store servers and links current usage information. In our implementation, a link is represented as a connection between two nodes, where a node can be a server or a switch of the NFC. Even though it is not necessarily that there is a link between each and every node in the network, we used an 2D array to store links usage information, with a row and a column representing each node of the NFC. The total number of nodes in the network depends on two factors: (1) the number of servers in the NFC and (2) physical topology of the NFC (Section VI-D). When we increase the number of servers, the 2D array that stores links usage information grows quadratic. Therefore, when we perform genetic operations, timing for the process of copying the original solution grows quadratic too.

The effect of total number of servers for the timing was examined for three network architectures (4-fat tree, BCube and VL2) separately. We conducted 50 rounds of experiments from each type and calculated the average. We assumed that each server has an initial capacity of 1000 units and each link has an initial capacity of 6000 units. We defined the capacity requirements of VNFs, in a way that 50% of the server capacities is filled. First, in all three types of architectures, the timings for DFS process is significantly smaller compared to timings for GP process. For a network with 128 servers, timings are: (1) 4-fat tree 5489 s, (2) BCube 5204 s and (3) VL2 5236 s and growth of the graph with respect to number of servers is linear. Second, in all three types of architectures, the timings for the GP process is dominated by the process of copying original solution during genetic operations. Figure 2 shows the comparison of time taken for GP process with 200 generations in different architectures when there are 16, 32, 48, 64, 80, 96, 112 and 128 servers in the NFC. We observed, in all three architectures, the growth of the graph is quadratic with respect to the number of servers and when they are plotted in the same figure, three graphs fall on top of each other. Fitting the plots into a quadratic polynomial of the form "p1 x^2 +p2 x+p3", we get within the 95% confidence bounds the coefficient p1, for k-fat tree to be 6.107 (varying from 4.661 to 7.553), BCube to be 5.948 (varying from 5.431 to 6.465) and VL2 to be 3.972 (varying from 2.308 to 5.637).

As we mentioned earlier, the total number nodes in the network depends on two factors: the number of servers and the physical topology. In a situation where there are fixed number of servers in the NFC, the total number of nodes in the NFC will depend on the physical topology. Therefore, the timings for the GP process with a fixed number of servers in different network architectures will vary, depending on the total number of nodes. We explored the time taken for GP process with 200 generations in different architectures with respect to different number of nodes when there are 16, 32, 48, 64, 80, 96, 112 and 128 servers in the NFC. In all three architectures, the growth of the graph is quadratic with respect to the number of nodes in the network and when they are plotted in the same figure, three graphs fall on top of each other.

D. Effect of the state of the NFC

To better understand the effect of the state of the NFC, to the improvements to the solution during the GP process, we carried out 10 set of experiments in four types of 128 server environments of a data center with a 4-fat tree architecture network: (1) an environment where 80% of the server and links capacity is full: Very tight (2) an environment where only 70% of the server and links capacity is full: Tight, (3) an environment where only 50% of the server and links capacity is full: Medium and (4) an environment where only 30% of the server and links capacity is full: Loose. We assumed that each server has an initial capacity of 1000 units and each link has an initial capacity of 6000 units. We used 92 fixed policy requests that consists of 400 VNFs. We count the times that the fitness value was improved during the GP process. The first observation is that, in all types of environments, most of the improvements (80% from total number of improvements) in the fitness function happens early on (during first 100 generations), and after that improvements decrease significantly. The second observation is that, the environments with loosely tight resource availability get more improvements (33% from total number of improvements) than tighter environments (15% from total number of improvements).

E. Effect of the order of policy requests

Since we are processing policies in a new provisioning request sequentially, we needed to check the impact of the order policies in the results. We used fixed 83 policies that includes 400 VNFs and processed them in 100 random orders for an environment of 128 servers in a 4-fat tree architecture. Our results showed that the order of the policies does not impact the quality of the solution. Only five different fitness function values were obtained with an average value of 1.072 and a standard deviation of 0.0058.

VIII. EVALUATION FOR SCALING OF EXISTING POLICY REQUESTS

The second responsibility of the Resource Manager is to find online configuration solutions to implement dynamic scaling requirements of existing policy requests, according to the traffic changes. The following section describes the performance evaluation of the GP approach when handling these dynamic re-allocation of resources in a NFC with 128 servers. As in the previous section, for our experiments, unless explicitly mentioned, we have assumed situations where 50% of the NFC servers have to be used.

A. Effect of the scaling approach used

Scaling of a VNF instance can be done in two ways: (1) vertical or (2) horizontal. Vertical scaling is allocation/release of host and bandwidth resources to/from a VNF instance, whereas horizontal scaling is installation/removal of VNF instances or paths. Vertical scaling is a basic feature of VMs, which adjusts logical partitions of multiple metrics (i.e. CPU, Memory, Bandwidth). So vertical scaling of VNFs can be done adjusting the existing VNF instance with new metrics
of capacities for CPU, Memory and Bandwidth. However, horizontal scaling changes the number of VM instances, which involves running VNF instances on two or more separate VMs hosted on the same or different servers. We assumed a scenario where traffic flow has increased and we have to allocate more server and bandwidth resources (extra resources). For the vertical scaling based approach, first we check whether the server and the path, that is currently used by the existing VNF instance, can handle the total resource requirement. If yes, then we do not need to change the current network configurations (we can use the same path), and we can perform vertical scaling for the existing VNF. If not, we search for a new server and a path that can handle the total resource requirement. In this case, we need to perform a live migration of the VNF [39] and change the current network configurations to redirect traffic to the new path, which causes more changes than performing the vertical scaling. For the horizontal scaling, we check for a server and a path that supports the extra resource requirement, to add a new VNF instance. It doesn't matter whether we can use the original server and path (server and the path that is currently used by the existing VNF instance), with the horizontal scaling, we have to add a new VNF instance and configure a new path to the instance. Therefore, it definitely causes changes in servers and paths.

There are two different aspects to look at when deciding which method to use: (1) selecting a configuration solution based on the fitness function and (2) selecting a configuration solution considering the real implementation. When selecting a configuration solution based on the fitness function, there are two possible scenarios. First, we give more weight to reduce changes to current configurations. For this, vertical scaling is appropriate because if we can allocate/release host resources from/to the VNF in the same server where the existing VNF instance resided, this method does not cause changes to the currently configured path. So the number of links changes are zero. On the other hand, with horizontal scaling, even though we may be able to install/remove VNFs instances in the same server where the existing VNF instance resided, we have to install paths to the new VNF instance or remove paths from the additional VNF instance. Therefore, it causes changes to the links and it is counted in the fitness function. Second, if we give more weight to reduce links congestion, the horizontal scaling is appropriate because, instead of all traffic going to a single instance and links getting congested, traffic can be distributed to both instances and routed through more paths. With our fitness function where we have equal weight for all parameters, we conducted 20 experiments assuming a data center with a 4-fat tree architecture network where each server has an initial capacity of 1000 units, each link has an initial capacity of 3000 units and each VNF initially requires 100 server capacity units. For each round of experiments, we have used a fixed set of policy requests that consists of 400 VNFs as the initial policy requests and carried out scaling out with both methods separately. In 70% of the experiments both methods gave the same fitness value while in 30% of the experiments, the first method gave better fitness values. We have observed that this is because, when the current server and path of the existing VNF instance can handle the total resource requirements, vertical scaling can be performed and therefore no changes introduced.

When selecting a configuration solution considering the real implementation, existing work has shown that each of the aspects has advantages as well as disadvantages [38]. Vertical scaling is better than horizontally scaling because: (1) needs less time for reconfiguration as it needs only metrics adjustment, (2) does not need additional software licenses, (3) does not affect the quantity of VNF instances and (4) does not introduce a coordination or a traffic distribution among multiple VNF instances. However, vertical scaling is limited by capacity of the server or link, because it cannot increase the resources more than the maximum capacity. Also, the consolidation after vertical scaling is more complicated because the fragments caused by it are likely to be irregular. Hence, the cost of migration caused by vertical scaling is higher than horizontal scaling. However, as the vertical scaling based method gave better results in terms of the fitness function, we have used it for rest of the experiments in the paper.

B. Effect of the number of VNFs scaling

To better understand how the NFC behaves when handling different number of VNFs scaling out simultaneously, we carried out 4 sets of experiments (30 rounds from each):
(1) 30 VNFs were scaling out simultaneously, (2) 20 VNFs were scaling out simultaneously, (3) 10 VNFs were scaling out simultaneously, and (4) 2 VNFs were scaling out simultaneously. We have assumed a 4-fat tree architecture with 128 servers, where each server has an initial capacity of 1000 units, each link has an initial capacity of 3000 units and each VNF initially requires 100 server capacity units. For each round of experiments, we have used a fixed set of policy requests that consists of 400 VNFs as the initial policy requests. We count the times that the fitness value was improved during the GP process. The observation is that there is little benefit in running the GP process when the number of VNFs scaling simultaneously is small. The number of improvements when 2 VNFs were scaling was 6% from total number of improvements. One can avoid running the GP process and use directly the DFS solution. As the number of VNFs scaling simultaneously increases, the potential gains provided by the GP optimization also increases. The number of improvements when 30 VNFs were scaling was 41% from total number of improvements. In all cases, most of the improvement in the fitness function (66% from total number of improvements) happens early on (during first 100 generations), and after that the improvements decrease significantly.

C. Effect of the local approach

Although the “global approach” where we are allowed to change configurations of any policy of NFC during the genetic operations, may provide better resource allocations, the solutions may require drastic re-arrangements of the current configurations, hence making them impractical in real scenarios. However, we can use this method to provide us with a baseline to compare against a “local approach” where we limit changes to configurations of policies that are scaling. We conducted experiments for both GP based global and local approaches to compare performances with respect to the resource allocation efficiency over several days. We repeated the data for single day (we derived 42 significant events over the 24 hours of traffic data [see Section VI-C]) for two times to get the data to emulate 2 consecutive days, and therefore we had total of 84 events for 2 days. The global approach is considered as the baseline i.e., the minimization of the parameters relevant to server and links usage. For the local approach, we explored 2 cases. First, we considered a situation where all parameters of the objective function are considered: minimizing resources and changes (all the weights in the Equation 3 are 1) and we call it “local 1”. Second, we considered a situation where only the parameters relevant to changes are considered: “local 2” (w1 = w2 = w3 = 0 and w4 = w5 = 1). The local 2 strategy represented the scaling solutions that in theory minimally disturb the traffic (e.g., packet drops, latency), because it tried to minimize the changes to the servers and links.

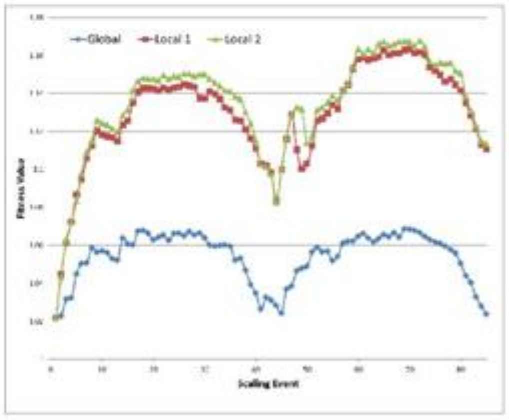

Figure 3: Fitness value comparison: KFat tree architecture

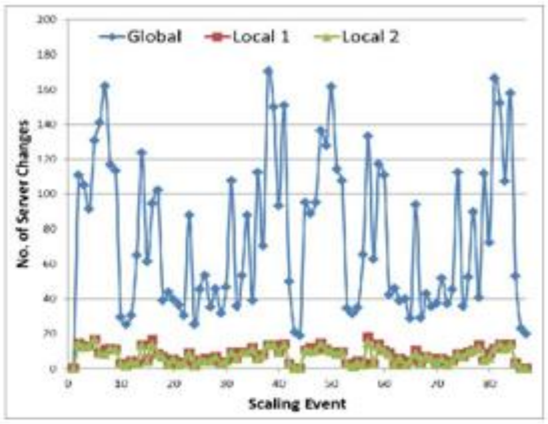

Figure 4: Server changes comparison: KFat tree architecture

Once the solutions have been provided for the 84 events by the global and two local approaches, we manually calculated the value of the each fitness function of the provided solutions, assuming that changes do not count (i.e., $w_4 = w_5 = 0$) and compared the global and local approaches. For each set of experiments, we have used a fixed set of policy requests that consists of 400 VNFs. We have assumed a 128 server network, where each server has an initial capacity of 1000 units, each link has an initial capacity of 3000 units and each VNF requires 100 server capacity units. The fitness value comparison (an average from 5 sets of experiments) for 2 consecutive days for a 4-fat tree architecture network is shown in figure 3. The global approach produced solutions with better resource allocations than the other two. Furthermore, the figure clearly shows that approaches (2) and (3), followed essentially the same behavior (modulo a translation in the y axis) that the behavior of the baseline (1), if we smooth the curves.

A comparison of server changes needed in the configuration solutions (an average from 5 sets of experiments) given by (1) global approach, (2) local approach 1 and (3) local approach 2, after processing each event is shown in the Figure 4. As expected, the global approach caused the largest number of changes. The solutions given by the local approaches had fewer server changes from their previous configuration because the local approaches performed genetic operations only on the partial solutions that are scaling. The interesting observation was that, most of the time both local approaches had the same number of server changes, making the two methods essentially the same. We have noted that these server changes are the unavoidable changes due to the scaling requirements, and not necessarily caused by the genetic operations. In addition, similar to the server changes, we have observed that the solutions given by the global approach has most links changes from their previous configuration. Most of the time both local approaches have the same number of links changes. As shown earlier, local approach 1 (which additionally minimizes usage of servers and links congestion) gave better fitness values than local approach 2. Therefore, local approach 1 provided solutions with better server and network resources utilization without making many changes. Although we have included the local approaches comparison results only for a 4-fat tree architecture network because of the limited pages, we have observed that this behaviour is the same for the other two architectures: BCube and VL2. Therefore, in the rest of the experiments, we have used only local approach 1.

D. Effect of the NFC architecture

Going further, we have compared the behaviour of local approach 1 with global approach for other architectures: BCube and VL2 architectures for 2 consecutive days. The fitness values (an average from 5 sets of experiments) for BCube and VL2 architectures for 2 consecutive days are shown in Figure 5 and 6 respectively. As mentioned earlier, for each set of experiments, we have used a fixed set of policy requests that consists of 400 VNFs. We have assumed a 128 server network, where each server has an initial capacity of 1000 units, each link has an initial capacity of 3000 units and each VNF requires 100 server capacity units. Similar to the 4-fat tree architecture, the global approach produced solutions with better resource allocations than the local approach for BCube and VL2 architectures. Also, they followed essentially the same behaviour (module a translation in space) of the global approach: the baseline. In the results of all three architectures, during most of the scaling events of each day, the fitness values grew. The reason is, in the traffic model we are using, the traffic is increasing until late night of each day.

We have observed that each architecture's fitness values are effected by different parameters of the fitness function. For all three architectures, number of links used and links utilization made the difference in the fitness values and the number of servers used were very similar. In the BCube architecture, the difference between local and global approaches was due to the fact the local approach always used fewer links than the global approach. While the global approach freely used more links over time, the local approach hesitated to use more links because we were trying to minimize number of server and links changes in the local approach. Therefore, the solutions given by the local approach were more congested than the solutions given by the global approach. In the VL2 and 4-fat tree architectures, the number of links used was similar for both the local approach and the global approach, while the difference was on link utilization. When considering the fitness values increase for each day, the VL2 architecture's fitness values for the local approach increased fast with respect to the 4-fat tree and BCube. In the VL2 architecture, servers are located in a more compact manner and it has fewer paths between servers inside the same pod. Therefore, the links got more congested, and when traffic was increasing the link utilization also increased fast. The 4-fat tree architecture has more paths and therefore the servers were not compact. It tried to use more links and made the links less congested. Hence, the 4-fat tree architecture had more smooth effect on the parameters of the fitness function.

IX. FINAL REMARKS

In this paper we present a comprehensive analysis on the proposed GP based resource allocation algorithms for: (1) new VNFs provisioning and (2) scaling of existing VNFs with the traffic changes. We compared the GP approach with traditional resource optimization technique: ILP for a small network and explored the evolution of the GP based algorithm for large networks, over a full day traffic patterns based on more realistic data. The results showed that generating solutions even for 10 VNFs in a relatively small network (16 servers), ILP can take hours while GP takes only few milliseconds. Furthermore, although GP may not provide the optimal solution, GP can decide the computing and network allocations for hundreds of policies (around 400 VNFs) in a 128 server environment and find reasonable solutions on the order of milliseconds. Moreover, our results showed that the GP process provided an average objective value improvement percentage up to 7.87% over the initial solution (the baseline) with a reduction of average link utilization up to 28.7% for the three architectures (a 4-fat tree, a BCube and a VL2). In the evaluation of the algorithms over the time, our results showed that, a "local approach" provides reasonable solutions with lesser changes to the current configurations, and moreover, it does not diverge from the "global approach" solutions over time.

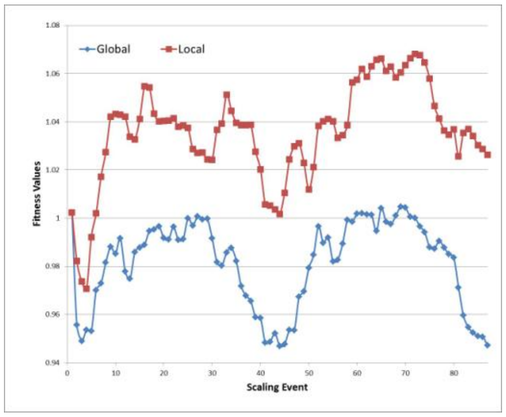

Figure 5: Fitness value comparison: BCube architecture

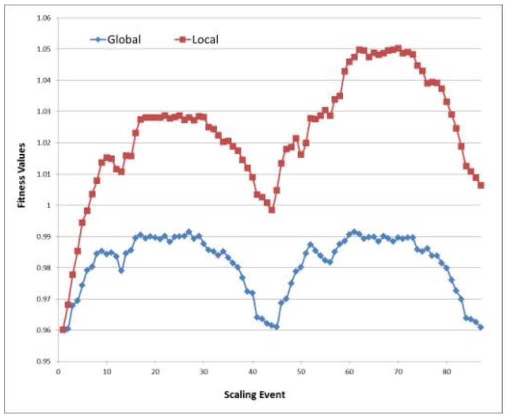

Figure 6: Fitness value comparison: VL2 architecture

In this paper, for both initial policy requests provisioning and scaling, we have considered policies with chains of VNFs, and assumed that one VNF can have only one successor VNF. However, we can easily extend our ILP model to facilitate general policies that can be represented with Directed Acyclic Graphs (DAG), where one VNF can have more than one successor VNF, and the VNF might need more than one path assigned to it so that its traffic can be directed to the successors. We can use a constant to define the required number of paths to route traffic to its successor(s) and modify the constraint 2c in the Section IV to reflect the required number of paths. Furthermore, we have evaluated the behaviour of the scaling out/in resource allocation algorithm, with fitness functions that considered the number of servers used, links used, links congestion, number of server changes and links changes. There are many other linear and non-linear factors that might affect the NFC: traffic lost, delay, cost of VNFs software license [22] and power consumption. Therefore, we need to look for more approximation algorithms that can cope with linear and non linear objective functions. As the future work, we are planning to explore the feasibility of using the concept of meta heuristics in operation research, namely the Iterative Local Search (ILS) approach which is one of the most popular single solution based meta-heuristics. We are planing to compare the performances of the ILS approach with GP and ILP approaches. Also, for a fairer comparison with ILP, we are thinking of an approach to include a warm start for the ILP.

X. ACKNOWLEDGMENT

We would like to thank the reviewers for their comments and suggestions that helped us to improve the paper significantly.

This research was sponsored by U.S. Army Research Laboratory and U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of U.S. Army Research Laboratory, U.S. Government, U.K. Ministry of Defence or U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. Jorge Lobo was partially supported by the Secretaria dUniversitats i Recerca de la Generalitat de Catalunya. Also this work was supported by the Maria de Maeztu Units of Excellence Programme.

References

[1] ETSI, "Network functions virtualisation white paper," SDN and Open-Flow World Congress, 2013.
[2] T. C. V. C. Mathieu Bouet, Jrmie Leguay, "Cost-based placement of vdpi functions in nfv infrastructures," in International Journal of Network Management, 2015.
[3] W. Rankothge, F. Le, A. Russo, and J. Lobo, "Experimental results on the use of genetic algorithms for scaling virtualized network functions," in IEEE SDN/NFV 2015.
[4] Y. Li, L. Phan, and B. T. Loo, "Network functions virtualization with soft real-time guarantees," in IEEE International Conference on Computer Communications (INFOCOM), 2016.
[5] G. Milad, K. Aimal, S. Nashid, A. Khalid, A. Reaz, and B. Raouf, "Elastic virtual network function placement," in Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on, 2015.
[6] N. S. Milad Ghaznavi, Aimal Khan and at el., "Elastic virtual network function placement," in IEEE CloudNet, 2015.
[7] X. Meng, V. Pappas, and at el, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in GIIS '12.
[8] W. Rankothge, J. Ma, F. Le, A. Russo, and J. Lobo, "Towards making network function virtualization a cloud computing service," in IEEE IM 2015.
[9] S. Jain, A. Kumar, and et al, "B4: Experience with a globally-deployed software defined wan," in ACM SIGCOMM '13, 2013.
[10] "Openflow 1.4 specifications," https://www.opennetworking.org/sdn-resources/onf-specifications/openflow.
[11] "Cplex," http://www-01.ibm.com/software/ commerce/ optimization/ cplex-optimizer/.
[12] G. Vinicius, F. Alessandro, M. Marco, and at el., "Uncovering the big players of the web," in ICTMA '12.
[13] C. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," in IEEE Transactions on Computers, 1999.
[14] C. Guo, G. Lu, D. Li, and at el, "Bcube: a high performance, server-centric network architecture for modular data centers," in ACM

SIGCOMM 2009.
[15] A. Greenberg, J. R. Hamilton, N. Jain, and at el, “Vl2: a scalable and flexible data center network,” in ACM SIGCOMM 2009.
[16] R. Cohen, L. Lewin-Eytan, and at el, “Near optimal placement of virtual network functions,” in INFOCOMM 2015.
[17] Z. Qazi, C. Tu, L. Chiang, and at el, “Simple-fying middlebox policy enforcement using sdn,” in ACM SIGCOMM ’13, 2013.
[18] L. Shi, B. Butler, and at el, “Provisioning of requests for virtual machine sets with placement constraints in iaas clouds.” in IEEE IM ’13, 2013.
[19] D. Jayasinghe, C. Pu, and at el, “Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement.” in IEEE SCC ’11.
[20] K. Konstanteli, T. Cucinotta, K. Psychas, and T. Varvarigou, “Admission control for elastic cloud services,” in IEEE Cloud ’12, 2012.
[21] B. Jennings and R. Stadler, “Resource management in clouds: Survey and research challenges,” in JNSM ’14, 2014.
[22] M. Luizelli, L. Bays, L. Buriol, M. Barcellos, and L. Gaspary, “Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions,” in IEEE IM ’15.
[23] M. Bari, S. Chowdhury, and at el, “On orchestrating virtual network functions,” in IEEE/ACM/IFIP CNSM 2015.
[24] T. Lukovszki, M. Rost, and S. Schmid, “Its a match! near-optimal and incremental middlebox deployment,” in ACM SIGCOMM Computer Communication Review: Jan 2016.
[25] M. Ghaznavi, N. Shahriar, and at el, “Service function chaining simplified,” in eprint arXiv:1601.0075.
[26] A. Gember, R. Grandl, A. Anand, and at el, “Stratos: Virtual middleboxes as first-class entities,” Technical Report TR1771, 2013.
[27] S. Clayman, E. Maini, and at el, “The dynamic placement of virtual network functions,” in NOMS 2014.
[28] R. Mijumbi and at el, “Design and evaluation of algorithms for mapping and scheduling of virtual network functions,” in NetSoft 2015.
[29] M. Melanie, An Introduction to Genetic Algorithms, 1999.
[30] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: Rapid prototyping for software-defined networks,” in ACM HotNets’10, 2010.
[31] “Ryu sdn controller,” http://osrg.github.io/ryu/.
[32] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, and at el., “Making middleboxes someone elses problem: network processing as a cloud service,” in ACM SIGCOMM ’12, 2012.
[33] “Data modelling process,” http://arxiv.org/abs/1702.00369v3.
[34] “Test data,” https://github.com/windyswsw/ DataForNFVSDN Experiments.
[35] S. Gebert, R. Pries, D. Schlosser, and K. Heck, “Internet access traffic measurement and analysis,” in ICTMA ’12.
[36] K. Srikanth, S. Sudipta, and at el., “The nature of data center traffic: Measurement and analysis,” in ACM SIGCOMM IM 2009, 2009.
[37] Y. Tarui, “Analyzing the impact of major social events on internet exchange traffic,” in NANOG38, 2009.
[38] X. C. Wenting Wang, Haopeng Chen, “An availability-aware virtual machine placement approach for dynamic scaling of cloud applications,” in Conference on Autonomic and Trusted Computing, 2012.
[39] S. H. J. G. H. C. Clark, K. Fraser and at el., “Live migration of virtual machines,” in NSDI, 2005.