Towards Making Network Function Virtualization a Cloud Computing Service

Windhya Rankothge, Universitat Pompeu Fabra, windhya.rankothge@upf.edu
Jiefei Ma, Imperial College, j.ma@imperial.ac.uk
Franck Le, IBM Research, fle@us.ibm.com
Alessandra Russo, Imperial College, a.russo@imperial.ac.uk
Jorge Lobo, ICREA-Universitat Pompeu Fabra, jorge.lobo@upf.edu

***Abstract*—By allowing network functions to be virtualized and run on commodity hardware, NFV enables new properties (e.g., elastic scaling), and new service models for Service Providers, Enterprises, and Telecommunication Service Providers. However, for NFV to be offered as a service, several research problems still need to be addressed. In this paper, we focus and propose a new service chaining algorithm. Existing solutions suffer two main limitations: First, existing proposals often rely on mixed Integer Linear Programming to optimize VM allocation and network management, but our experiments show that such approach is too slow taking hours to find a solution. Second, although existing proposals have considered the VM placement and network configuration jointly, they frequently assume the network configuration cannot be changed. Instead, we believe that both computing and network resources should be able to be updated concurrently for increased flexibility and to satisfy SLA and Qos requirements. As such, we formulate and propose a Genetic Algorithm based approach to solve the VM allocation and network management problem. We built an experimental NFV platform, and run a set of experiments. The results show that our proposed GA approach can compute configurations to to three orders of magnitude faster than traditional solutions.**

I. Introduction

Network Functions Virtualization (NFV) [1] proposes to virtualize functions such as load balancers, firewalls, intrusion detection devices and WAN accelerators, traditionally running on proprietary hardware appliances, and instead have them operate as virtual software components on commoditized hardware. In addition to reducing costs and time to market, NFV enables new properties such as elastic scaling: NFV allows the flexible increase and decrease of resources (e.g., CPU, memory) allocated to each VM instance, and the number of VM instances, to satisfy the dynamic and fluctuating service demands. However, for Virtual Network Function to be offered as a service (e.g., to virtualize the 3G/4G Mobile Core Network and IMS), several research problems still need to be solved [2].

In this paper, we focus and propose a new service chaining algorithm. Researchers have typically relied on mixed Integer Linear Programming (ILP) to optimize VM allocation and network management [3]. However, we argue that this approach is too slow to satisfy dynamic traffic demands and requirements (e.g., flash events). Our own experiments show that to implement 10 network functions optimizing both computing and network resources, ILP calculations take more than 2 hours. In addition, although several works have considered both VM allocation and network management jointly [4] [5], most of them have assumed that only one of the two components could be modified: For example, [3] [4] [5] [6] assume that the computing resources allocation can be updated but not the network configuration.

Instead, we believe that both computing resources and network configuration should be able to be updated concurrently: such approach offers more flexibility, and allows NFV's SLA and QoS requirements to be met [7]. We take advantage of software-defined/OpenFlow infrastructure [8] to facilitate network management, and maximize network resource utilization. As such, in order to quickly optimize both server and bandwidth resources simultaneously, we developed a Genetic Programming based approach: Genetic Algorithms (GA) were developed in the field of artificial intelligence and are search heuristics inspired by the process of natural selection, and used to generate useful solutions to optimization and search problems [9]. GA relies on different techniques including mutations and crossover of non-optimal solutions to generate new solutions, and ranks the solutions according to a fitness function. The population (i.e., the number of solutions considered) is maintained around the same size since solutions with the lowest fitness function values are dropped from the population. Note that ILP objective functions can be the fitness function in a GA. We modeled the optimization of the VM and network resource allocation problem with a GA that returns the best fitted solution after a fixed amount of generations have been explored.

We built an experimental NFV platform to better understand and study research issues related to NFV. The architecture and the complexity of our experimental NFV 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. In particular, we validated our proposal, and our results show that although GA may not provide the optimal solution, GA can decide the computing and network allocations for tens to hundreds of policies in a 64 server environment on the order of seconds. Our proposed GA approach can therefore compute configurations two to three orders of magnitude faster than traditional solutions. We believe that such time reduction would be essential for NFV to be offered as a service.

The rest of the paper is organized as follows. Section II presents our experimental NFV platform. Section III introduces its management system. Section IV describes the implementation of GA based network function placement and dynamic scaling out/in. Section V shows the evaluation of the implementation.

II. Overview of Experimental Platform: Network Function Center (NFC)

NFV is bringing closer the possibility to truly migrate enterprise data centers into the cloud. However, for a Cloud Service Provider to offer such services, many research questions still need to be addressed: e.g., when and where should new virtual network functions be instantiated?, How to scale up/down resources to satisfy traffic demands and guarantee QoS? How can network configuration be updated on-demand to guarantee service chaining, especially in the events of virtual network function creation and deletion?

We are building an experimental platform that we call Network Function Center (NFC) to study research issues related to NFV and network function. We want to develop the NFC using a simple but comprehensive architecture where different types of studies and experiments can be conducted. We will start making several simplifying assumptions that we might later relax if needed. In a NFC, we assume a network function is implemented on a VM that can be deployed in any server in the network, contrary to traditional network functions that are hardware based middle-boxes in fixed places in the network. In this section, we briefly describe the functionality we expect from a NFC and the proposed architecture.

A. Functionality

Table 1

Traffic Flow	Policy	Expected Traffic
10.1.0.0/24 - *, HTTP	Firewall – IDS - Proxy	300 GB
20.1.0.0/24 – 172.0.0.0/24, HTTP	IDS - Proxy	100 GB

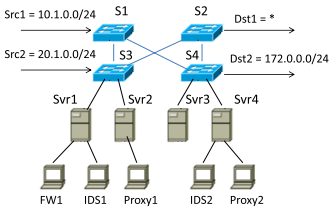

Table 2

Policy	Physical Sequence
Firewall – IDS - Proxy	S1-S3-Svr1-FW1-Svr1-IDS1-Svr1-S3-Svr2-Proxy1-Svr2-S3-S2
IDS - Proxy	S3-S2-S4-Svr4-IDS2-Svr4-Proxy2-Svr4-S4

Fig. 1. Network Function Center Snapshot

We assume NFC delivers virtualized network functions to clients on a subscription basis. To receive services from NFC, a client needs to provide the following three specifications to a NFC: (1) types of functions required, (2) interconnectivity between these functions, and (3) expected traffic load to be generated by these functions. The abstractions used to solicit this information from a client will depend on the functions. The specification could be complex as a high-level description of a virtualized network where each network's endpoint is connected to predefined Virtual Networks [10]; or a simple specification as a set of network services chains through which different classes of traffic must go through [11] (e.g., all traffic from 10.0.0.0/24 must traverse IDS-Firewall-Proxy). The specifications can be (automatically or semi-automatically) translated into a collection of Directed Acyclic Graphs (DAG) connecting sources to destinations of data flows in which the intermediate nodes in a graph path represent network functions that must be applied to the traffic flow going through the path. The first and the last node in a path are the source and the destination of the flow and may or may not be hosted inside the NFC. Each DAG must be accompanied with capacity information for each node (i.e., function capacity requirements) and information about traffic characteristics that will go through the different paths (e.g., traffic class, expected throughput). As an illustration, we assume a NFC that provides network functions represented as chains of services. As illustrated in Table 1 of Figure 1, the client request comes to the NFC in the form of

- Policy (chain of required network functions)
- Ingress and egress locations of client's traffic flow
- Expected volume of the traffic flow

Once the client request is accepted by the NFC, the client's traffic is redirected to the NFC to traverse the network functions. The NFC must guarantee that the client's traffic traverse all the network functions in the correct order. In addition, the NFC is expected to increase/decrease the number of network functions instances and paths for the traffic flow according to the application's dynamic needs and agreements with the client.

B. Architecture

The overall architecture of a 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 network services. Servers in the NFC are used to deploy the virtual machines (VMs) where network services run. A network service is implemented as a software on a VM.

Figure 1 represents a snapshot of a NFC. It depicts the placement of network services to implement the two policy chains in Table 1. Table 2 shows the physical sequences of switches and network services the client's traffic will go through. Client1 wants his traffic flow coming from 10.1.0.0/24 to any destination to go through the policy chain of Firewall-IDS-Proxy network services. To grant his request a firewall service and a IDS service are implemented on two VMs at Server1 and a Proxy service is implemented on a VM at Server2. The network architecture depicted in the figure represents a tree-like architecture typical of datacenters, but our NFC architecture does not assume any particular network structure. As network element reconfiguration becomes a more active part of the management, these standard architectures may change for the benefit of the management. The architecture Management System is described in more detail in the next section.

III. NFC Management System

The goal of the NFC Management system is to automate arrangement, coordination and management of NFC components to maximize on-demand client requests whilst guaranteeing QoS. Figure 2 gives an overview of the initial proposed NFC

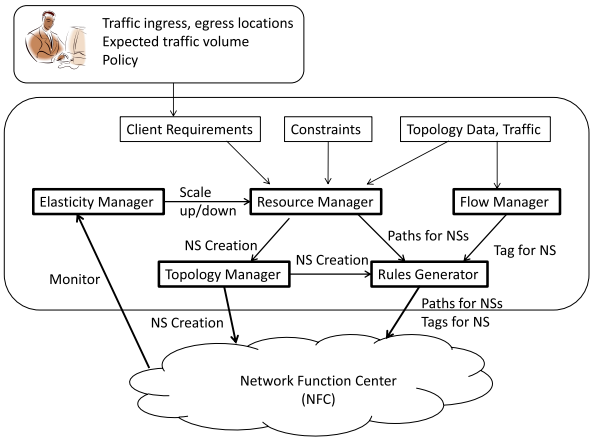

Fig. 2. NFC Management System

Management System architecture. The term "NS" in the figure refers to any requested Network Function.

The NFC Management System requires three high-level inputs: (1) Client Requirements given as a set of annotated DAGs, (2) Topology data and Traffic and (3) Resource constraints. Each client's requirement is annotated with the traffic flow's ingress and egress locations, flow properties (source destination IP addresses, source destination ports), expected volume of traffic traversing each path in the network functions chain. Topology data and traffic describes the placement of current network functions, paths between servers, links between switches, available capacities of links, switches and servers. Resource constraints specify (1) server and switch resources (CPU, memory, TCAM sizes, etc.) and (2) bandwidth capacity for each link in the topology.

A. Process of NFC Management System

The process 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.

Once a new client request is submitted, Resource Manager takes decisions on the placement of network functions and 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 resources utilization. According to parameters such as network traffic, applications' requirements and agreements with clients, the Elasticity Manager takes decisions on when to increase/decrease the instances of network functions and paths for the traffic flows. Once the Elasticity Manager makes its decision, the information is passed to the Resource Manager which then determines the possible changes to the placement of the network functions instances and paths. The Topology Manager, Flow Manager, and Rules Generator configure the network according to decisions taken by the Resource Manager and Elasticity Manager.

The five key modules are described with more detail in the following sub-sections.

1) Resource Manager: The Resource Manager module takes the network's traffic, topology data, constraints and client requirements as inputs. For each client requirement, the Resource Manager decides: (1) the acceptance or rejection of the client request and (2) if the request is accepted, then for each network function, it identifies: (a) either a non-used VM that can be re-used to run the required network function or a server where a new VM can be created to run the required network service, and (b) a path for traffic flow between every two network functions directly connected in the DAG based on expected volume of traffic specified in client requirement and the available bandwidth of links in the NFC.

In addition, responding to requests from the Elasticity Manager to scale resources, the Resource Manager decides a new set of network functions assignments and paths for existing client's traffic flow. In the NFV context, scaling of network functions include: (1) scaling out/in by changing number of instances and (2) scaling up/down by changing memory/CPU capacity/storage of the existing instance [1]. In our research, we have considered the scaling out/in situation. So for scaling out, the decision of whether to create a new service instance in a more suitable location or re-use a dormant service to save service instance creation time, may depend on several factors such as the location of the dormant service and its capabilities. Reassignment of functions and paths allows cloud service providers to maximize their resources. However, the transition from one configuration of network functions and flow assignment to another could create transient congestion which could degrade applications performances.

The combination of all these factors has to be considered by the Resource Manager to decide resource allocation. A popular technique that has been used for VM allocation [3] and network management [11] is to model resource allocation as a mixed Integer Linear Programming (ILP) optimization problem. There is the intrinsic constraint that ILP optimization is a NP-complete problem, and even when solutions are obtained for special classes, it might be too slow for continued adjustments of the system configuration causing inappropriate hysteresis in the reaction. We advocate the more realistic approach of looking for a good feasible configuration and do not expect to find optimal solutions like the ones returned by ILP. We need to explore approximations techniques. In our prototype, we have modeled the problem as finding the *best fitted* solution according to a Genetic Algorithmic (GA) modeling [9] of the problem. Details of the implementation can be found in Section IV.

2) Topology Manager: The Topology Manager module is responsible for maintaining up to date state of the physical infrastructure of the NFC. It keeps an inventory of all functions running or dormant in the system. It knows about all physical paths between servers and paths used by all the traffic flows. It also maintains information about current traffic and service demands. It will be the source of data for any analytic needed to be done about the NFC. It is also in charge of the instantiation or re-use of the necessary network functions as well as the provisioning needed according to the instructions given by the Resource Manager. Creation of network functions may include deploying a VM in a server, installing the necessary software and starting the necessary processes required by the network functions.

3) Flow Manager: The aim of the NFC is to provide flexibility in regard to network functions placement which can be placed anywhere in the network. Hence, the sequence of switches in the physical network that the traffic from a client has to follow, may have loops (the same traffic flow may visit the same server for two different functions more than once). If so, the combination of original source and destination information of a packet, is not sufficient to identify the network functions this packet has gone through so far. Making the situation more complicated, many network functions modify packet headers. Therefore the original source and destination information of a packet can be changed during the flow [11]. Hence, it is essential for the Flow Manager module to find mechanisms to come up with unique identification for the state of a packet in a flow path. This identification can be done through tunnelling mechanism [10] or can be added into headers of packets going through the network service.

4) Elasticity Manager: One of the main objectives of NFC is to support scaling out/in network functions according to the network traffic and dynamic needs of applications. Therefore, the Elasticity Manager monitors the network and servers to determine when to scale out/in the resource allocation to meet the traffic demands according to the SLAs and QoS agreements. Finding the exact network function(s) or path(s) which are causing the bottleneck is essential because of the costs involved in running new network functions as well as the impact reallocating functions and flow paths may cause in service quality and traffic lost due to switching delays. Also, it is important to decide the right type an amount of resources to increase/decrease to achieve the demand and avoid the potential for some kind of thrashing phenomenon.

5) Rules Generator: The Rules Generator module generates data plane configuration for the switches to route traffic through the appropriate sequence of network functions from their source to destination according to client requirements. It is this module that directly takes advantage of SDN and OpenFlow. Because the NFC controls the switches, routing inside the NFC network is done entirely using OpenFlow VLAN and MPLS tags adapting the ideas from [11]. This is done to avoid all the problems that modifications of packet headers by middle boxes can cause [12]. Rules Generator uses the identification tag issued by the Flow Manager to infers mappings between the incoming and outgoing traffic flows of a network service and to identify the traffic flow that a packet belongs to. Tags are added at ingress point before the traffic has been gone through any network function, thus letting the traffic be identified by the Source and Destination headers and they are removed at the egress point.

Furthermore, when updating the network configuration as a result of scaling needs, the Rules Generator follows the per-flow consistency introduced in [13] to avoid any inconsistencies in transient traffic and reduce traffic lost. The update mechanism works by stamping every incoming packet with a version number and modifying every configuration so that it only processes packets with a set version number. To change from one configuration to next, it first populates the switches in the middle of the network with new configurations guarded by the next version number. Once that is completed, it enables the new configurations by installing rules at the perimeter of the network that stamp packets with the next version number. This method makes network updates faster and cheaper, by limiting the number of rules or switches affected.

IV. Initial Implementation of Resource Manager

As the initial stage of developing NFC Management System, we have focused on the implementation of the Resource Manager. Also, we have implemented the Flow Manager and Rules Generator modules as the supporting modules to evaluate the Resource Manager module. In this section, we describe with some detail the implementation of the Resource Manager module.

A. Genetic Programming (GP) Approach

As explained in section III-A1, the Resource Manager module has two main responsibilities: (1) New function provisioning: upon receipt of a new set of policies, resources are identified within the given physical network constraints and already allocated resources for the existing policies and (2) Scaling out/in: upon the request from the Elasticity Manager to support scaling out/in the resources, decides a new set of network functions assignments and paths for existing client's traffic flow. These two activities are implemented independently but both use Genetic Algorithm (GA) as the mechanism to allocate resources.

GAs are a part of evolutionary computing and were introduced as a computational analogy of adaptive systems [9]. 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.

The GAs [9]:

1) Randomly generate an initial population F(0) with n full solutions f
2) Compute and save the fitness u(f) for each individual full solution f in the current population F(t)
3) Generate F(t+1) by selecting i full solutions from F(t)
4) Produce offspring by applying genetic operators to population F(t+1)
5) Repeat step 2 until satisfying solution is obtained.

Following the terms used with GA concepts, a possible configuration state (represented by servers and paths assignments) of the NFC is considered as a full solution f, if it is an allocation of resources for all the policies in the system. The population F(t) consists of n full solutions which represents different possible configuration states for the NFC. If there are m number of policies in the NFC, then each full solution contains m number of partial solutions, each partial solution representing the allocation of resources (i.e., servers and paths) for each policy.

$$F1 = w_1 \tfrac{1}{M}.T_s + w_2 \tfrac{1}{L}.U_l + w_3(1 - \tfrac{1}{L}.T_l) \quad (1)$$

$$F2 = w_1 \tfrac{1}{M}.T_s + w_2 \tfrac{1}{L}.U_l + w_3(1 - \tfrac{1}{L}.T_l) + w_4 \tfrac{1}{M}.C_s + w_5 \tfrac{1}{L}.C_l \quad (2)$$

TABLE I. Fitness Functions

M	Total no. of servers
T_s	Total no. of servers used in the configuration solution
L	Total no. of links
T_l	Total no. of links used in the configuration solution
U_l	% of total links capacity used in a configuration solution
C_s	Total servers changed from previous state to current state
C_l	Total links changed from previous state to current state
w_1, w_2, w_3, w_4, w_5	Weighting factors

TABLE II. DESCRIPTION OF PARAMETERS USED IN FITNESS FUNCTIONS

1) Handling new services provisioning: For new services provisioning, the Resource Manager uses network's traffic, topology data, server constraints and client requirements as inputs. In step 1, the Resource Manager generates the initial population F(t). It performs a selection for the initial assignment of network functions and paths for each new policy request, within the given physical network constraints and previously allocated resources for the existing policies. We have used Depth First Search (DFS), so that the servers and paths are selected by searching through the whole search space and selecting the first solution we come across. The configuration state (network functions and paths) that the Resource Manager comes up with DFS for a new policy request is a partial solution that combined with the partial solutions of each of the existing policies form a full solution.

After the initial population is generated, the fitness function 1 (F1) given in Table I is used to measure how good a full solution is (step 2). F1 takes into account: servers capacity, links capacity, number of links used and number of servers used with respect to the total physical usage of the network and network resources available. Weights of parameters (w_1, w_2 and w_3) can be decided according to the preferences, whether to give more priority to server utilization or links utilization.

As we are trying to maximize the server and network utilization, fittest solutions are those for which the function returns the smallest value. So the full solutions that returns small values are preferred and in step 3 they are selected as the best solutions for next generation population.

In step 4, the Resource Manager performs mutations and crossover for randomly selected partial solutions of a full solution and generate a new full solution. We have considered two types of genetic operators to produce offspring: (1) mutation and (2) crossover. Crossover and mutation perform two different roles. Crossover is a convergence operation which is intended to pull the population towards a local min/max. On the other hand, Mutation is a divergence operation. It 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, crossover happen more frequently and mutation, being a divergence operation, happen less frequently.

In our implementation, mutation is achieved via (1) Replacement where we try to place the network function in a different server and (2) Re-wiring where we try to find a different path between given two network functions. For the Re-placement mutation, first we select a random full solution from the population and a random partial solution of the selected full solution. Then, we select a random network function in the partial solution and try to find a new server where it can be placed on. If a new server is available to place the selected function, then we find paths accordingly between the network function and its successor by considering the new placement. For the Re-wiring mutation, similar to Replacement mutation, first we select a random full solution from the population and a random partial solution of the selected full solution. Then we select a random network function in the partial solution and try to find a new path to its successor.

For the crossover process in the implementation, first we select two random full solutions from the population and a random partial solution from each selected full solution. Then, we try to check whether the configuration given in 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.

The newly generated full solution is added to the existing set of full solutions, which is known as the current population. This process is continued until x number of generations are explored. In the final generation, the full solution that gives the best fitness value is selected as the best configuration for the new policy implementation.

2) Handling scaling out/in: When the Elasticity Manager decides a network function or a path has to be scaled out/in, the Resource Manager starts with the current state and performs an initial selection for the re-assignment of resources (new servers and paths) of a set of network functions that are scaling. As previously explained, the initial selection is done using DFS. The partial solutions relevant to the scaling up are modified according to the results of the search.

The fitness function 2 (F2) given in Table I is used to measure how good a full solution is. It uses additional parameters representing the changes to the current system. While trying to maximize the server and network utilization, we want to minimize the changes to the current system because drastic re-arrangements of the system configuration will cause unacceptable deterioration of functions during the transition time. Therefore, according to the requirements, weights of the parameters (w_1, w_2, w_3, w_4 and w_5) can prioritize server and link utilization or minimize server or link changes.

The mutations and crossovers are carried out only to the partial solutions which were changed because of the scaling out. 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.

V. EVALUATION

In this section we describe results of experiments which were carried out to test the two main functions of the Resource Manager: (1) New functions provisioning and (2) Scaling out of existing functions. Also, we will show a comparison of two configuration update mechanisms used by the Flow Manager and the Rules Generator: (1) Rules are updated in all switches simultaneously and (2) Uses versioning tags to maintain per-flow consistency.

A. Simulation Setup

Our prototype NFC management system have been developed 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.

Our prototype network makes use of Software Defined Networks (SDN) to allow programmatic control over the traffic flow and easy reconfiguration of the physical network. We have implemented the physical structure in Mininet [14], used "Ryu" [15] as SDN controller, and dumb OpenFlow switches as network services [8].

In our experiments, we have considered a new functions provisioning to be a set of policies (chains of network functions). The policies used in experiments are generated randomly with at least 2 but no more than 7 functions. We have considered 5 types of network functions where each network function needs different number of capacity units. The types of network functions in a policy are also selected randomly. All experiments were carried out in a machine with an Intel core i7-4500u processor and 8GB of RAM.

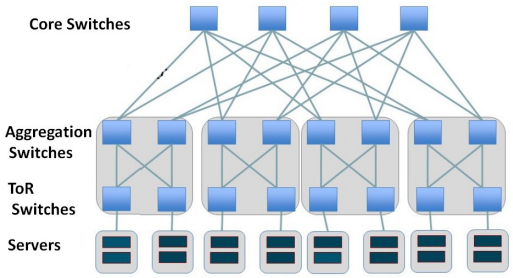

Fig. 3. K-fat tree architecture

We work with four types of environments : (1) 64 servers with 72 switches, (2) 32 servers with 30 switches, (3) 16 servers with 20 switches, and (4) 8 servers with 8 switches, under a k fat-tree architecture. Figure 3 shows a 4-ary fat-tree. A typical k-ary fat-tree network 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 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 and $(k/2)$ servers. Each aggregation switch connects to $(k/2)^2$ switches on the core layer.

B. Handling new policy requests

A popular technique that has been used for VM allocation [3] is to model the resource allocation as ILP optimization problem. There is the intrinsic constraint that ILP optimization is a NP-complete problem, and even when solutions are obtained for special classes, it might be too slow. We carried out 3 experiments to calculate the time taken to find a solution to implement 10, 30 and 50 network functions with ILP. ILP took 2.3, 4.6 and 7.2 hours respectively. Even though an ILP formalization of the problem gives an exact solution, ILP calculation is unfeasible for NFC because to find an optimal configuration, ILP takes a lot of time even for a small number of network services. Therefore, for NFC prototype, we have modelled the Resource Manager with the GP approach which finds the the *best fitted* solution according to a Genetic Algorithmic after a fixed amount of generations have been explored.

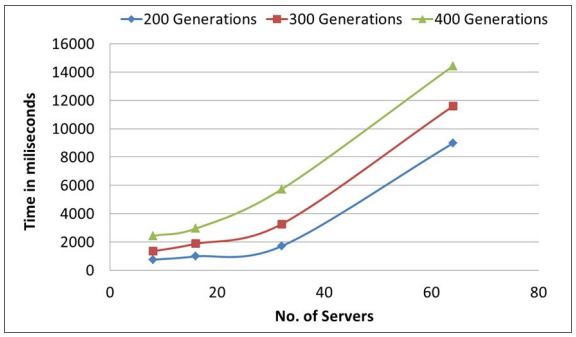

Fig. 4. Total time to implement 100 policies with 200,300,400 generations

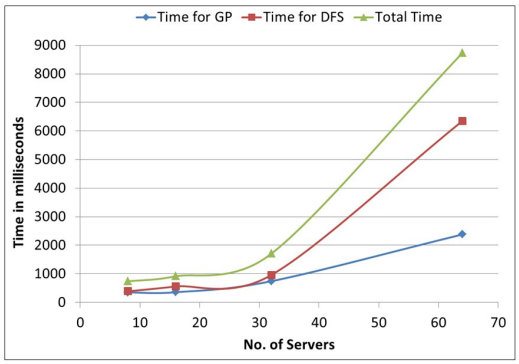

Fig. 5. GP time vs DFS time to implement 100 policies with 200 generations

When handling new provisioning request, the Resource Manager processes the policy chains in the request sequentially. For each policy, it starts by performing a DFS for the assignment of servers and paths for the policy (i.e. a partial solution) under network and server constraints of already allocated resources for the existing policies. After the initial solution for the policy is found, with the use of GP approach, the Resource Manager performs mutations and crossovers for randomly selected partial solutions of a full solution and generate new full solutions.

Figure 4 gives the average total time taken to implement new functions provisioning requested comprising of 100 policies chains (this would be about 500 network functions) in 4 types of NFC environments: (1) 8 servers, (2) 16 servers, (3) 32 servers and (4) 64 servers with 200, 300 and 400 generations. These total times include: (1) time taken for Resource Manager to perform DFS to come up with an initial partial solution for each policy and (2) time taken to run the GP over generations to come up with a better full solution. The growth of the graph is exponential and it seems to becoming smooth when the number of servers are increasing. The increase is due to the large number of choices, specially at the beginning when all resources are available, in larger NFCs.

In order to understand the effect of the size better we get the average times for the DFS part and the GP part for 200 generations separately. In Figure 5 shows that it is indeed the DFS that takes large portion of the time. This can be addressed by using simple heuristics to find the allocation of a few of the

first policies and only start running DFS when the heuristics fail to find quick solutions.

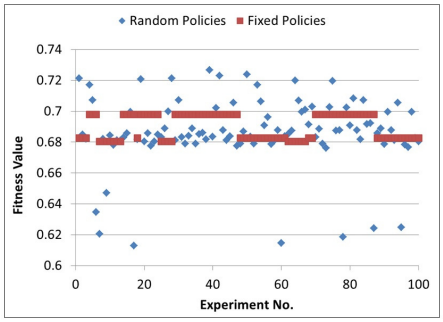

Fig. 6. Fitness values when implementing 100 policies

Since we are processing policies in a new provisioning request sequentially, we needed to check the impact of the order in the results. Figure 6 shows that order has not much effect. We have fixed 100 policies and processed them in 100 random orders for environment (2). Only three different fitness function values were obtained with a mean value of 0.689 a standard deviation (SD) of 0.008. When 100 random policies were selected 100 times and processed the mean was 0.686 and the SD 0.022.

C. Handling scaling out requests of existing policies

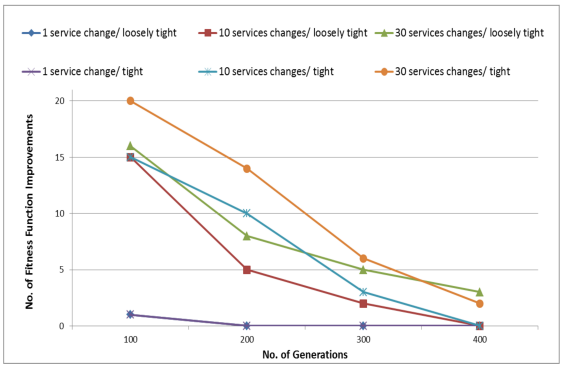

Fig. 7. Impact of No. of generations for improvement of solution

As explained in section IV, handling scaling out requests of existing policies, starts with the Resource Manager performing a DFS for the reassignment of a set of network functions and the corresponding paths. In a first set of experiments we wanted to figure out what was the impact of the number of functions scaling out simultaneously and the initial state of the system before the scaling out starts. We used two types of environments for the experimental setup: (1) an environment where 80% of the server and links capacity are full (tight environment) and (2) an environment where only 50% of the server and links capacity are full (loosely tight environment). We carried out three sets of experiments in each environment. One in which 30 functions were scaling out simultaneously, one in which 10 functions were scaling simultaneously and a third one in which only one function is scaling out. The results are summarized in Figure 7.

The first observation is that there is little benefit in running the GP when the number of functions scaling simultaneously is small. One can avoid running the GP and use directly the DFS solution. Correspondingly, GP optimization offers more gains for larger numbers of simultaneous requests. When implementing 10 policy changes simultaneously, in 37.5% of experiment runs, the full solution was improved over generations. When implementing 30 policy changes simultaneously, the full solution was improved in 65% of the runs.

The second observation is that the tight environments get more improvements than the loosely tight environments. Over all the experiments, in fully tight environments, the fitness function of the full solutions was improved in 52.5% of the runs by the GP, while in the loosely tight environments, there were changes only in 37.5%.

The last observation is that in all the cases, most of the improvement in the fitness function happens early on, and after 200 generations the improvements decrease significantly. Hence, we will present the performance numbers based on runs of 200 generations.

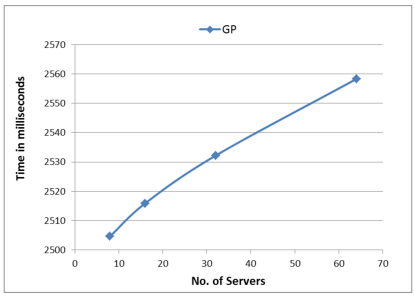

Fig. 8. GP with 200 generations to implement 30 policies changes

In the second set of experiments we collected data from 30 runs of 200 generations when 30 functions were scaling simultaneously. We measured the time at two points. First, the time the Resource Manager took to perform DFS to find to initial solution, and then the time it took to performs the mutations and crossovers in GP. We run the experiments in the 4 environments: (1) 8 servers, (2) 16 servers, (3) 32 servers and (4) 64 servers. The results for the GP part is shown in figure 8. We observed that the time taken by the DFS is insignificant with respect to the time taken by the GP part. For the DFS part, the speed of growth is decreasing with the increasing of the number of servers. For the GP part, the growth is linear with respect to the number of servers.

D. Updating the physical network configurations

As explained in sections III-A5, as part of the scaling out/in, the Rule Generator has to reconfigure the data plane dynamically. As the initial step, we have experimented with two different implementations found in existing works: (1) Rules are updated in all switches simultaneously, and (2) versioning tags are used to maintain per-flow consistency [13]. The first implementation introduces an average of 7% packet loss while second implementation reduce it to average of 1.5%. As a part of future work on the Rules Generator module, we are planing to explore more methods of updating the physical network configurations to reduce the packet loss.

VI. Next Challenges

In order to truly integrate network functionality into commercial offerings of Cloud services, we advocate for a more aggressive integration of the network and server infrastructure. Advances in SDN and standards such as OpenFlow make this integration feasible. However, this integration does not come without new challenges.

The most immediate challenge comes with the network functions placement, that is the increase in the dimensionality of the optimization space. In its most basic form, we want to simultaneously optimize the placement of functions (in the servers) and traffic load (by finding the appropriate paths in the network). The larger number and the non-linear dependency of the variables involved indicate that typical mixed ILP approaches might not be the appropriate tool to approach the problem – we don't want exact solutions, we want fast solutions. The time typically required for ILP to compute a solution is unsatisfactory, especially considering that client's traffic characteristics may change quickly. Even though we have used a linear equation as the fitness function for our implementation, it is not necessary for the fitness function of a GA to be linear. One can derive a non-linear fitness function, that can produce better feasible solutions in a given time constraint and that are robust to progressive changes in the system, since drastic re-arrangements of the system configuration will cause unacceptable deterioration of functions during the transition time. We have also assumed a management system that is mostly agnostic to the semantics of functions, but it is likely that with knowledge about the functions during optimization better and faster results can be obtained. We need to look for approximation algorithms that can cope with non-linear objective functions and we are planing to explore meta-heuristics in operation research and hybrid machine learning methods which are practical improvements based on semantic knowledge injected to the theoretical models of optimization and learning.

The second challenge is to determine when to scale out/in the resource allocation to meet the traffic demands according to the SLAs and QoS agreements. Finding the exact network function(s) or path(s) which are causing the bottleneck and deciding the right number of resources to increase/decrease to achieve the demand is important.The most basic method to scale out/in is to monitor system-level metrics (server and links utilization) and determining whether to scale out/in based on a threshold. However, threshold-based algorithms do not capture the complex interaction among multiple resource parameters (server and links) and the potential diversity of traffic types. Determining the right set of thresholds for them to simultaneously achieve the right SLA and QoS for each type of traffic would be difficult. Often the thresholds are set based on ad-hoc measurements and past experiences. We are planing to explore machine learning techniques, in particular reinforcement learning, to learn the behaviour of the applications and automatically adapt to changes. The learning algorithm can be augmented with heuristics to improve the responsiveness and guide the algorithm itself.

Furthermore, when updating the network configuration as a result of scaling needs, the Rules Generator should avoid any inconsistencies in transient traffic and reduce traffic lost. We are planing to explore more methods of updating the physical network configurations, so that the changes to the current system will be minimum and unacceptable deterioration of functions during the transition time will avoided.

VII. Related Work

Traditionally network functions have been implemented as hardware based middle-boxes. CoMB [16] proposes an architecture which implements these middle-boxes as software-based, virtualized entities. [17], [18] present platforms to manage software-based middle-boxes. With the popularity of NFV, [19] discusses the possibility of outsourcing enterprise middle-box processing to the cloud. FeatureAPI [20] introduces a policy language to map policies onto the underlying cloud network, so that external Feature Providers can easily provide network functions to enterprises. [21] highlights the customer expectations when they outsource the network functions. Stratos [6] is the first work that presents a framework for external Network Functions Providers. Stratos assumes a cloud service exists and to handle traffic flows and redirections, it associates each middle-box with a virtual switch that provides network functionalities. These switches decide where to send traffic, and if the traffics crosses from one physical server to another, it uses the traditional routing provided by the cloud network.

Virtualized data centers are envisioned to provide better management flexibility, lower cost, scalability, better resources utilization and energy efficiency [22], [23], [24]. The placement of the NFVs in the physical machines and use of network bandwidth are crucial for the performance of virtualized data centers. [3] has considered the VM placement problem in a cloud system only with respect to network resources utilization. Their approach uses integer linear programming and takes in the order of minutes to decide the placement of 1024 VMs in the data center of 16 servers. [5] proposes a Markov approximation technique to jointly address the VM placement and routing problem. Their approach takes in the order of seconds to decide the placement of a VM. However, it decides the location of one VM at a time. Instead, we argue that to handle dynamic and fluctuating service demands (e.g., flash events), a large number of policies and VMs may need to be handled concurrently. [4] focuses on network interface of machines as the network resource to optimize with the server resources. They assume that the network interconnecting the machines has full bisection bandwidth, so that considerations of bandwidth of the links are put aside.

VIII. Final Remarks

In this paper we introduced a new service chaining algorithm, based on Genetic Algorithms (GA) which jointly optimize VM allocation and network management. Our approach offers more flexibility, allowing both computing resources and network configuration to be updated concurrently. We discussed architecture and functionalities of the experimental platform: NFC and the simulation setup of the study. Our results showed that although GA may not provide the optimal solution, GA can solve the computing and network allocations for tens to hundreds of policies in a 64 server environment on the order of seconds.

ACKNOWLEDGMENT

This research was sponsored by the U.S. Army Research Laboratory and the 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 author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the 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.

REFERENCES

[1] ETSI, "Network functions virtualisation white paper 1," *SDN and OpenFlow World Congress*, 2013.

[2] E. NFV, "Network functions virtualisation white paper 3," *SDN and OpenFlow World Congress*, 2014.

[3] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *GIIS'12*, 2012.

[4] F. Wuhib, R. Yanggratoke, and R. Stadler, "Allocating compute and network resources under management objectives in large-scale clouds," in *Journal of Network and Systems Management*, 2013.

[5] J. Jiang, T. Lan, S. Ha, M. Chen, and at el, "Joint vm placement and routing for data center traffic engineering," in *INFOCOM'12*, 2012.

[6] A. Gember, R. Grandl, A. Anand, T. Benson, and A. Akella, "Stratos: Virtual middleboxes as first-class entities," *Technical Report TR1771 : University of Wisconsin-Madison*, 2013.

[7] S. Jain, A. Kumar, S. Mandal, J. Ong, and L. P. et al, "B4: Experience with a globally-deployed software defined wan," in *ACM SIGCOMM'13*, 2013.

[8] "Openflow 1.4 specifications," https://www.opennetworking.org/sdn-resources/onf-specifications/openflow.

[9] M. Melanie, *An Introduction to Genetic Algorithms*, 1999.

[10] R. Cohen, K. Barabash, B. Rochwerger, L. Schour, D. Crisan, R. Birke, C. Minkenberg, M. Gusat, R. Recio, and V. Jain, "An intent-based approach for network virtualization," in *IM 2013*.

[11] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *ACM SIGCOMM'13*.

[12] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul, "Flowtags: enforcing network-wide policies in the presence of dynamic middlebox actions," in *HotSDN'13*, 2013.

[13] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," in *ACM SIGCOMM'12*, 2012.

[14] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *ACM HotNets'10*, 2010.

[15] "Ryu sdn controller," http://osrg.github.io/ryu/.

[16] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *NSDI'12*.

[17] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *HotNets-XI*, 2012.

[18] J. Martins, M. Ahmed, C. Raiciu, M. H. Vladimir Olteanu, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *NSDI'14*, 2014.

[19] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: network processing as a cloud service," in *ACM SIGCOMM'12*, 2012.

[20] G. Gibb, H. Zeng, and N. McKeown, "Outsourcing network functionality," in *HotSDN'12*, 2012.

[21] S. K. Fayazbakhsh, M. K. Reiter, and V. Sekar, "Verifiable network function outsourcing: requirements, challenges, and roadmap," in *HotMiddlebox'13*, 2013.

[22] M. Bari, R. Boutaba, R. Esteves, and L. Granville, "Data center network virtualization : A survey," in *IEEE Communication Surveys*, 2014.

[23] N. M. K. Chowdhury and R. Boutaba., "Network virtualization : State of the art," in *IEEE Communication Magazine*, 2009.

[24] A. Fischer, J. Botero, M. Till Beck, and H. de Meer, "Virtual network embedding : A survey," in *IEEE Communication Surveys*, 2013.