VNF-P: A Model for Efficient Placement of Virtualized Network Functions

Hendrik Moens and Filip De Turck
Ghent University – iMinds, Department of Information Technology
Gaston Crommenlaan 8/201, B-9050 Gent, Belgium
e-mail: hendrik.moens@intec.ugent.be

Abstract—**Network Functions Virtualization (NFV) is an upcoming paradigm where network functionality is virtualized and split up into multiple building blocks that can be chained together to provide the required functionality. This approach increases network flexibility and scalability as these building blocks can be allocated and reallocated at runtime depending on demand. The success of this approach depends on the existence and performance of algorithms that determine where, and how these building blocks are instantiated. In this paper, we present and evaluate a formal model for resource allocation of virtualized network functions within NFV environments, a problem we refer to as Virtual Network Function Placement (VNF-P). We focus on a hybrid scenario where part of the services may be provided by dedicated physical hardware, and where part of the services are provided using virtualized service instances. We evaluate the VNF-P model using a small service provider scenario and two types of service chains, and evaluate its execution speed. We find that the algorithms finish in 16 seconds or less for a small service provider scenario, making it feasible to react quickly to changing demand.**

I. Introduction

Using Network Functions Virtualization (NFV), various network functions are migrated from costly hardware appliances to virtualized instances deployed on generic servers. This network architecture increases network flexibility and scalability, as these virtualized services can be instantiated and scaled on-demand using cloud scaling technologies.

Two challenges however remain: (1) While there are many similarities to cloud resource allocation, the NFV architecture is designed to be used within entire service provider networks, and not just within datacenters. In datacenters, high-capacity and high-speed networks are used to connect servers, making the specifics of the underlying network less important. In NFV deployments in networks outside of the datacenter, the importance of network constraints such as bandwidth and latency however increases. (2) In practice, the more expensive dedicated hardware often performs faster and more efficiently than virtualized instances, even though the latter are more flexible. As dedicated hardware is currently widely deployed, it is likely that hybrid deployments will be common, where part of the services are provided by physical hardware. This results in a scenario analogous to a cloud burst: a base load is handled by physical hardware (the private cloud in a cloud burst scenario), while variation in load is handled by dynamically instantiating services (the public cloud in a cloud burst scenario). In this "NFV burst" scenario, a base load is handled by physical hardware while spillover is handled by virtual service instances. This approach is illustrated in Figure 1.

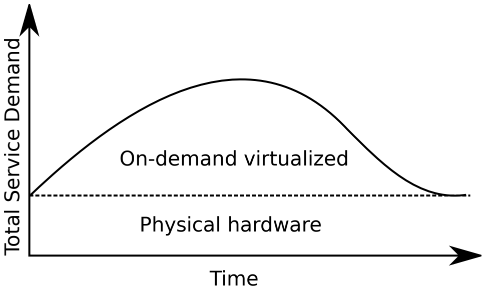


Fig. 1: An NFV burst scenario: physical hardware is fully utilized by a base load, while spillover is handled by utilizing virtualized services.

To resolve these challenges, network and service-aware NFV management algorithms must be developed. In this paper, we present and evaluate a formal Virtual Network Function Placement (VNF-P) model that can be used to allocate resources in hybrid NFV networks. The remainder of this paper is structured as follows. In the next Section, related work is presented. Afterwards, in Section III, the major differences between NFV resource allocation and cloud resource allocation are discussed. The VNF-P model is presented in Section IV. We discuss the evaluation set-up in Section V and present the results in Section VI. Finally, in Section VII we state our conclusions.

II. Related Work

Allocating resources in NFV networks is similar to application placement in datacenters and clouds [1], specifically to network-aware application placement. Many publications [2], [3], [4], [5], [6], [7], [8] focus on either allocating collections of Virtual Machines (VMs), or on adding network-awareness to datacenter resource management algorithms. These works however focus specifically on Infrastructure as a Service (IaaS) clouds where VMs are allocated. Often only datacenter-specific network topologies are considered. In this paper, by contrast, we focus on NFV resource allocation without any restrictions on the underlying network topologies. Additionally we also consider the difference between service requests and VM requests. These two request types must be handled differently, as services may be deployed either on dedicated hardware or on shared service instances that are managed by the service provider while requested VMs are managed by the client who requested the service chain.

The VNF-P is also related to the problem of virtual network embedding in software defined networks [9]. Virtual network embedding focuses on how virtual network requests, in the form of a collection of VMs and their interconnections can be deployed on physical networks. In this paper we extend this embedding approach by defining VNF-P, making it possible to specify both VM requests and service requests, the latter resulting in service provider managed services that may be shared between multiple tenants. We also incorporate the notion of hybrid networks containing both physical devices offering services and virtualized services. Similarly, [10] also focuses on deployment of virtual network functions in pure NFV environments while we also consider a hybrid environment where dedicated hardware for providing services is present.

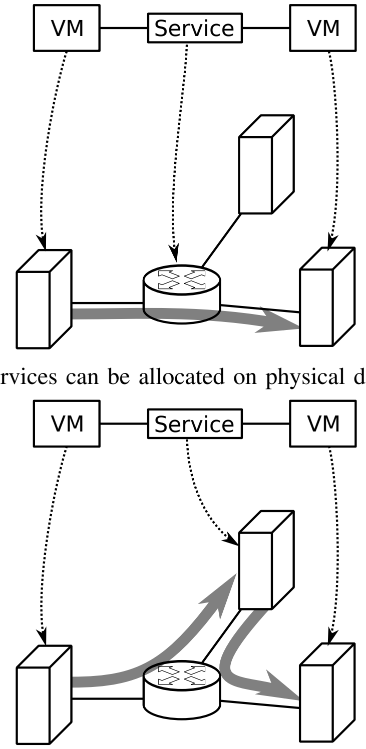
III. NFV Resource Allocation

In NFV networks, a collection of service chains must be allocated on physical network nodes. A service chain is a collection of one or more services or VMs that are chained together to provide specific functionality, and can be represented as a graph containing services and the network demand between these services. In a hybrid network environment, service chains can be allocated either using physical hardware, or by using virtualized instances. These two approaches are illustrated in Figure 2.

A service request can be allocated either on dedicated hardware, or using a service that is deployed by the service provider using a VM. It is also possible for client VMs to be part of a service chain. The key difference between service and VM requests is that a VM that is part of a service chain is managed by the client who requests the service chain, while a VM that provides a service is managed by the service provider. These provider-hosted services can therefore be shared between multiple clients. Because of this, three types of deployment, illustrated in Figure 3, are possible in a hybrid NFV network: (1) VM deployment on physical server nodes, (2) service deployment on physical service nodes, and (3) service deployment on virtualized service instances. Allocating resources for the first two deployment types is similar to network-aware VM allocation in clouds, the only difference being that for VM deployment CPU and Memory are typically used as resource types whereas for service deployment other resource types such as requests per second are more appropriate. Existing resource allocation algorithms can however easily be adapted to work with these different resource types. The addition of the third type of deployment, where services are deployed on shared virtualized service instances, however results in the need for significant changes to existing models.

IV. The VNF-P Model

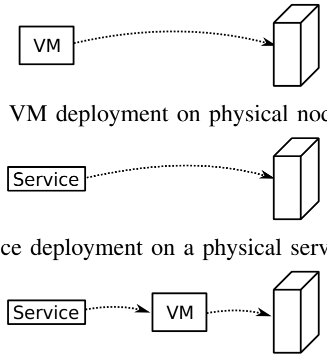
In the following sections we present a formal model for the VNF-P problem. To improve legibility, we first present the model inputs. Next, we present a model that only takes into account the service and VM requests, ignoring the underlying network. Subsequently, we extend the model, adding network-awareness to it.



(a) Services can be allocated on physical devices.

(b) Alternatively, services can be offered using virtualized instances on generic hardware.

Fig. 2: NFV service chain allocation approaches.



(a) VM deployment on physical nodes.

(b) Service deployment on a physical service node.

(c) Service deployment on a virtualized service instance.

Fig. 3: Service and VM allocation approaches.

A. Problem description

The inputs of the complete model are shown in Table I. The network is represented as a graph $G = (N, E)$. The graph consists of a collection of nodes N that represent physical network nodes and edges E between these nodes. Within the model we assume that these edges can be either bidirectional or unidirectional. Incoming and outgoing edges from a node

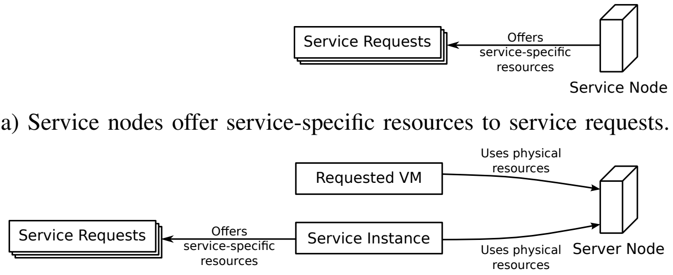
TABLE I: The model input variables.

Symbol	Description
G	The graph $G = (N, E)$ representing the network.
N	The nodes within the network. This collection can be partitioned into a set of computational nodes N^c and a set of service nodes N^s.
E	The edges within the network.
E_n^{in}	The edges that are incoming in node $n \in N$.
E_n^{out}	The edges that are outgoing from node $n \in N$.
$C(e)$	The bandwidth capacity of an edge $e \in E$.
$L(e)$	The network latency of an edge $e \in E$.
$\mathcal{C}$	The service chains that must be allocated on the network.
$R^{VM}(C)$	The VM requests that are part of service chain $C \in \mathcal{C}$.
$R^s(C)$	The service requests that are part of service chain $C \in \mathcal{C}$.
$R(C)$	The resource requests that are part of service chain $C \in \mathcal{C}$. $R(C) = R^{VM}(C) \cup R^s(C)$.
S	The collection of all services that exist within the model.
$D^\gamma(r)$	The resources of type γ needed for a request r. This request can either be a service request or a VM request.
Γ^c	The resource types that are available on a physical computational node. Typically these resources are CPU and Memory.
Γ^s	The resource types that are provided by a service $s \in S$. Typically this is a service-specific resource such as requests per second.
$\mathcal{R}$	A collection containing all of the resource requests of all service chains in $\mathcal{C}$.
C_n^γ	The resource capacity of a node or service n. For computational nodes, $\gamma \in \Gamma^{VM}$. For service nodes offering service s, $\gamma \in \Gamma^s$. A service VM of a service s provides C_s^γ resources of types $\gamma \in \Gamma^s$.
$D(r_1, r_2)$	The network demand between requested VMs or services r_1 and r_2.
$L(r_1, r_2)$	The maximum network latency between requested VMs or services r_1 and r_2.

$n \in N$ are represented by E_n^{in} and E_n^{out} respectively. A node can be any device in the network, such as a computational node, router, SDN controller, access point, network switch or any other type of device. We make a distinction between computational nodes, contained in N^c, and service nodes N^s that offer a specific service.

The objective of the optimization is to allocate a collection of service chains $\mathcal{C}$ on this physical network. A service chain $C \in \mathcal{C}$ consists of a collection of service and VM requests, the network demand between these services and VMs, and possibly network latency restrictions. We make a distinction between service requests and VM requests:

- *VM requests* are requests for the instantiation of a specific VM. This instance must be allocated on physical hardware that is present in the network, where it will use node physical resources such as CPU, memory, and possibly disk space. The physical resource types offered by physical hardware are contained in the set Γ^c. All VM requests of a service chain C are contained in the set $R^{VM}(C)$.
- *Service requests* are requests for capacity of a specific service. These requests can either be allocated directly on dedicated physical hardware, or they can be executed in a shared virtual service instance. A service s makes use of service-specific resources such as *requests per second.* These resource types are contained in the set Γ^s which is defined for every service $s \in S$, and where S is the set of all services. All service requests of a service chain C are contained in the set R^s.



(a) Service nodes offer service-specific resources to service requests.

(b) Server nodes offer physical resources to VM requests. They can also be used to instantiate service instances that offer service-specific resources to service requests.

Fig. 4: The difference between service nodes and server nodes.

The different ways in which VMs and services can be deployed are illustrated in Figure 4. All requests in a service chain C are represented by the collection $R(C) = R^{VM}(C) \cup R^s(C)$. The collection $\mathcal{R}$ contains all requests in the model, aggregated over all of the service chains in $\mathcal{C}$.

All computational nodes in $n \in N^c$ have a resource capacity C_n^γ, for all resource types $\gamma \in \Gamma^c$. Similarly, all service nodes $n \in N^s$ offer a limited amount of resources C_n^γ, for all $\gamma \in \Gamma^s$ and for all $s \in S$.

The edge capacity $C(e)$ and latency $L(e)$ are defined for all edges $e \in E$. The network demand between any requested VMs and services is represented by $D(r_1, r_2)$ for any two requests $r \in \mathcal{R}$. Similarly, a maximum network latency between requested VMs and services $L(r_1, r_2)$ may be specified.

B. Basic model

We first specify a basic model for NFV resource allocation that does not take the underlying network into account and only focuses on the allocation of the VMs and services in the requested service chains.

We define a binary decision variable, A_n^r, that defines whether a resource request $r \in \mathcal{R}$ is allocated on a node $n \in N$. An integer decision variable IC_n^s specifies the number of times that a service $s \in \mathcal{S}$ is instantiated on a computational node $n \in N^c$. We use an integer variable as we assume that it is possible for there to be multiple instances of the same service on a single physical node. If this is not desired, an upper bound of 1 can be specified for the IC_n^s variable preventing this.

Using these decision variables, and the input variables specified previously, a server capacity constraint is specified in Equation (1). This constraint ensures that the total amount of used resources on a server does not exceed the available capacity C_n^γ. The total amount of used resources is composed out of the resources used for handling VM requests, represented as U^{VM} and shown in Equation (2), and out of the resources used to provide service instances, represented as U^s (shown in

Equation (3)).

$$\forall \gamma \in \Gamma^c : \forall n \in N : U^{VM}(n,\gamma) + U^s(n,\gamma) \leq C_n^\gamma \tag{1}$$

$$U^{VM}(n,\gamma) = \sum_{r \in R^{VM}} A_n^r \times D^\gamma(r) \tag{2}$$

$$U^s(n,\gamma) = \sum_{s \in \mathcal{S}} IC_n^s \times D^\gamma(s) \tag{3}$$

It is important to ensure that the IC_n^s decision variables take on the correct values. To achieve this, a constraint must be added ensuring that the number of instances on a node, IC_n^s, offers sufficient resources for the service requests that are allocated on the node. This is expressed in Equation (4). This equation makes use of the total available service resources $T^\gamma(s,n)$ and the total needed service resources $N^\gamma(s,n)$ on the node. The expressions for $T^\gamma(s,n)$ and $N^\gamma(s,n)$ are shown in Equations (5) and (6) respectively.

$$\forall n \in N : \forall s \in \mathcal{S} : \forall \gamma \in \Gamma^s : T^\gamma(s,n) \geq N^\gamma(s,n) \tag{4}$$

$$T^\gamma(s,n) = IC_n^s \times C_s^\gamma \tag{5}$$

$$N^\gamma(s,n) = \sum_{r \in \mathcal{R}^s} A_n^r \times D^\gamma(r) \tag{6}$$

Finally, it is important to ensure that every request is allocated exactly once. This is done using Equation (7).

$$\forall r \in \mathcal{R} : \sum_{n \in N} A_n^r = 1 \tag{7}$$

The objective of the model is to minimize the number of used servers. To determine whether a server node n is used, the binary decision variable U_n is used. Equation (8) ensures that U_n takes on value 1 as soon as a single service or VM is allocated on the node n.

$$\forall n \in N^c : \forall r \in \mathcal{R} : U_n \geq A_n^r \tag{8}$$

It may also be possible for a node to be partially used, e.g. if a datacenter is represented by a single node, the number of servers used within the datacenter is relevant. In this case, this resource use may also be added directly to the objective function, ensuring it is minimized during optimization. To formalize this, we define a collection N^+ containing pairs of resources ($\in \Gamma^c$) and nodes ($\in N^c$) of which the total resource demand must be minimized. We do this using the U^+ variable defined in Equation (9). For the datacenter example, the number of cores used can be used for this.

$$U^+ = \sum_{(\gamma,n) \in N^+} U^{VM}(n,\gamma) + U^s(n,\gamma) \tag{9}$$

Using the U^+ and U_n variables, the optimization objective can be specified as shown in Equation (10).

$$\min \left(U^+ + \sum_{n \in N^c} U_n \right) \tag{10}$$

To further expand the model, it would be trivial to add the cost of using specific servers using additional weights in the objective function. This is however outside of the scope of this paper.

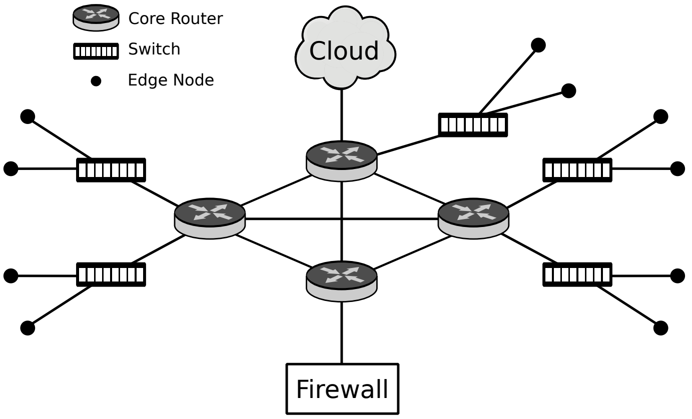


Fig. 5: The evaluation network.

C. Adding network-awareness

To add network-awareness to the model, the flow between two requests $(r_1, r_2) \in \mathcal{R}^2$ over the edges $e \in E$ of the network is modeled using a collection of binary flow decision variables $F(e, r_1, r_2)$. If $F(e, r_1, r_2) = 1$, the edge e is used for the flow (r_1, r_2), otherwise the value of this decision variable must be 0. These flow variables are subject to a flow conservation constraint, shown in Equation (11). For all nodes except the source and sink nodes, the incoming flow must equal the outgoing flow. For the source node, the flow must exceed the out flow, while for the sink node the opposite holds.

$$\begin{aligned} \forall (r_1, r_2) \in D : \forall n \in N : A_n^{r_2} + \sum_{e \in E_n^{out}} F(e, r_1, r_2) \\ = A_n^{r_1} + \sum_{e \in E_n^{in}} F(e, r_1, r_2) \end{aligned} \tag{11}$$

Two additional constraints, shown in Equations (12) and (13) are added to prevent cycles from occurring.

$$\forall (r_1, r_2) \in D : \forall n \in N : A_n^{r_2} + \sum_{e \in E_n^{out}} F(e, r_1, r_2) \leq 1 \tag{12}$$

$$\forall (r_1, r_2) \in D : \forall n \in N : A_n^{r_1} + \sum_{e \in E_n^{in}} F(e, r_1, r_2) \leq 1 \tag{13}$$

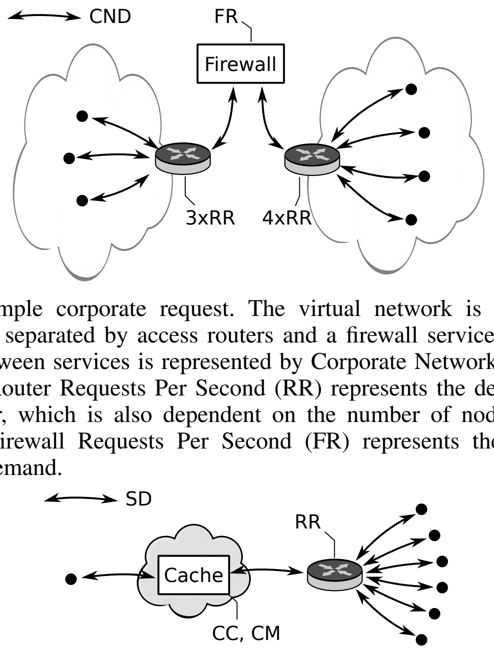
Finally, a capacity constraint, expressed in Equation (14) is needed to ensure that the total flow over an edge does not exceed the available edge capacity. To limit the latency on connections, a latency constraint, shown in Equation (15) is also added to limit the total delay over the entire path.

$$\forall e \in E : \sum_{(r_1, r_2) \in D} F(e, r_1, r_2) \times D(r_1, r_2) \leq C(e) \tag{14}$$

$$\forall (r_1, r_2) \in L : \sum_{e \in E} F(e, r_1, r_2) \times L(e) \leq L(r_1, r_2) \tag{15}$$

V. Evaluation Setup

We evaluate the model by using it to allocate a collection of service chains on a small network, shown in Figure 5. This network represents that of a small service provider, containing



(a) A sample corporate request. The virtual network is split into two parts separated by access routers and a firewall service. The demand between services is represented by Corporate Network Demand (CND). Router Requests Per Second (RR) represents the demand for the router, which is also dependent on the number of nodes in the subnet. Firewall Requests Per Second (FR) represents the firewall service demand.

(b) A sample streaming request. The source node streams to a cache VM instance with Cache CPU Cores (CC) representing CPU demand and Cache Memory (CM) representing memory demand. The cache is connected to a routing service with Router Requests Per Second (RR) demand, which in turn is connected to a collection of edge nodes. The demand between nodes is represented by Stream Demand (SD).

Fig. 6: The two evaluation service chain types. The various abbreviations represent network, service and VM parameters that are used in the evaluation scenario.

10 edge nodes, 5 switches, 4 core routers, a hardware firewall and a small cloud datacenter. We add the number of CPU cores that are used within the datacenter to N^+ to ensure as few datacenter servers are used as possible[1]. We consider two types of service chains: a corporate virtual network, and a streaming scenario. The two service chain types are illustrated in Figure 6.

The corporate virtual network scenario is illustrated in Figure 6a. In this scenario, a number of services is requested to connect a collection of edge nodes using a virtual network. The virtual network is randomly subdivided into a private subnet and a public subnet. The two subnets are connected using two access routers that are in turn connected to a firewall service.

The streaming scenario is illustrated in Figure 6b. Here, a single edge node is the stream source, while multiple edge nodes occur as sinks for the stream. The stream node is connected to a cloud-based cache which is a VM that is hosted by the service chain requester. The cache is connected to a router which is in turn connected to the various sink edge nodes.

Table II shows the network parameters used for the evaluation scenario. Here we assume that physical hardware, such as the hardware firewall and routers, are capable of handling more requests than virtualized instances of these services. The parameters for the various service chains are shown in Table III. Two multipliers, α and β for the corporate and streaming service chain respectively, make it possible to increase or decrease the load of the service chain, making it possible to simulate a scenario where the load on workflows varies throughout the day.

TABLE II: Network parameters used for the evaluation scenario.

Network		
Edge Capacity	100Gbps	
Edge Latency	0.01s	
Number of requests	40 per type	
Edge nodes per request	7	
	Hardware service	Virtual service
Routing service capacity	200000 requests/s	100000 requests/s
Firewall service capacity	5000 requests/s	10000 requests/s

TABLE III: Service chain parameters used for the evaluation scenario.

Corporate	
Corporate Network Demand (CND)	$\alpha \times 0.05$ Gbps
Router Requests Per Second (RR)	$\alpha \times 1000$ requests/s
Firewall Requests Per Second (FR)	$\alpha \times 1000$ requests/s
Streaming	
Stream Demand (SD)	$\beta \times 0.1$ Gbps
Router Requests Per Second (RR)	$\beta \times 10000$ requests/s
Cache CPU Cores (CC)	$\beta \times 8$ cores
Cache Memory (CM)	$\beta \times 16$ GB

The Integer Linear Programming (ILP) model presented in the previous section was implemented using CPLEX 12.4 [11] and Scala 2.11.1 [12]. The evaluations were executed on an Ubuntu 14.04 server with Intel Core i3-530 processor with 4GB of memory. All evaluations were repeated 50 times.

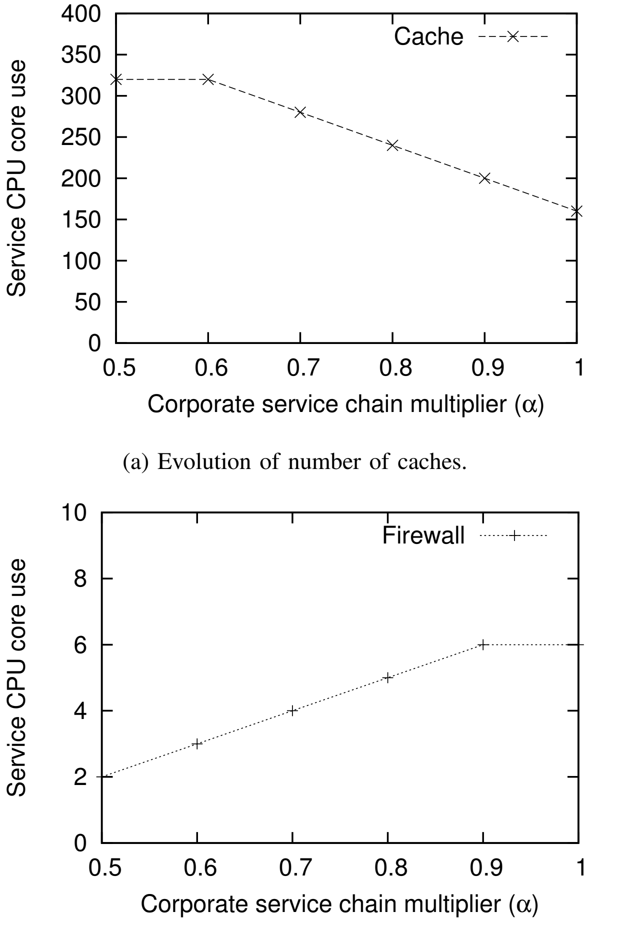
VI. Evaluation Results

We evaluate a scenario where the load of the corporate service chain increases while the streaming service chain load decreases. This is done by modifying the α and β multipliers, letting them vary from 0.5 to 1 and from 1 to 0.5 respectively. This represents a scenario where the load of various flows varies throughout the day (e.g. corporate service chains may have a higher load during working hours, while streaming requests may result in a higher load during the evenings).

Figure 7a shows the number of CPU cores in the datacenter that are used for hosting streaming caches. As the load of streaming service chains decreases (as β decreases), the number of caches decreases as well. As a separate cache is used for each of the service chains using a dedicated VM, a significant number of caches remains active. When the load of corporate service chains is increased, the number of shared firewall service instances gradually increases as shown in Figure 7b. In all of the scenarios there were sufficient hardware routers, ensuring no software routers were instantiated.

The performance of the VNF-P model was evaluated by gradually increasing the number of service chain requests, as shown in Figure 8. We observe that, as the number of service chain requests increases, the time needed to evaluate the model

[1] As discussed in the previous section, N^+ is used to select a specific node resource use decision variable that must be added to the global minimization objective.



(a) Evolution of number of caches.

(b) Evolution of the number of firewall instances.

Fig. 7: Evolution of resources when the load of corporate requests increases (α) while the load of streaming requests decreases ($\beta = 1.5 - \alpha$).

increases more or less linearly. For the evaluated scenarios, running the algorithm never takes more than 16 seconds, implying the algorithm is well-suited for smaller provider networks.

VII. Conclusions and Future Work

In this paper we presented a model for resource allocation in NFV networks, referred to as Virtual Network Function Placement (VNF-P). The model can be used in both pure NFV networks and in hybrid networks containing physical hardware. This makes the model useful in an NFV burst scenario where a base load is handled by physical hardware, while spillover is handled by on-demand virtualized services.

The presented model was implemented as an ILP and was evaluated using a service provider scenario containing two types of service chain requests. As the load of the request types changes, the number of instantiated services changes. The execution time of the algorithm remained less than 16 seconds.

In future work, we will evaluate the model in larger scale scenarios, and develop heuristic management algorithms that will improve the scalability of the management approach.

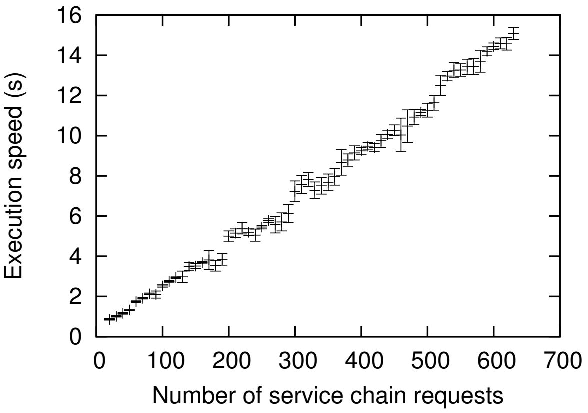


Fig. 8: The performance of the model for increasing numbers of workflows. The bars show the sample standard deviation. $\alpha = \beta = 0.01$, 50 iterations per data point.

Acknowledgment

Hendrik Moens is funded by the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT).

References

[1] B. Jennings and R. Stadler, “Resource Management in Clouds: Survey and Research Challenges,” *Journal of Network and Systems Management*, Mar. 2014.

[2] C. Low, “Decentralised Application Placement,” *Future Generation Computer Systems*, vol. 21, no. 2, pp. 281–290, 2005.

[3] M. Rabbani, R. Pereira Esteves, M. Podlesny, G. Simon, L. Zambenedetti Granville, and R. Boutaba, “On tackling virtual data center embedding problem,” in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE, 2013, pp. 177–184.

[4] F. Wuhib, R. Yanggratoke, and R. Stadler, “Allocating Compute and Network Resources Under Management Objectives in Large-Scale Clouds,” *Journal of Network and Systems Management*, Jul. 2013.

[5] M. Zhani, Q. Zhang, G. Simon, and R. Boutaba, “VDC Planner: Dynamic migration-aware Virtual Data Center embedding for clouds,” in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE, 2013, pp. 18–25.

[6] L. Shi, B. Butler, D. Botvich, and B. Jennings, “Provisioning of requests for virtual machine sets with placement constraints in IaaS clouds,” in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE, 2013, pp. 499–505.

[7] M. Alicherry and T.V. Lakshman, “Network Aware Resource Allocation in Distributed Clouds,” in *IEEE INFOCOM*, 2012, pp. 963–971.

[8] R. Esteves, L. Zambenedetti Granville, H. Bannazadeh, and R. Boutaba, “Paradigm-based adaptive provisioning in virtualized data centers,” in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE, 2013, pp. 169–176.

[9] R. Guerzoni, R. Trivisonno, I. Vaishnavi, Z. Despotovic, A. Hecker, S. Beker, and D. Soldani, “A Novel Approach to Virtual Networks Embedding for SDN Management and Orchestration,” in *Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014.

[10] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, “The Dynamic Placement of Virtual Network Functions,” in *Proceedings of the 2014 IEEE Network Operations and Management Symposium (NOMS)*, 2014.

[11] (2014) IBM ILOG CPLEX 12.4. [Online]. Available: http://www-01.ibm.com/software/integration/optimization/cplex-optimizer

[12] (2014) Scala 2.11.1. [Online]. Available: http://www.scala-lang.org/