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 up-
coming 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 net-
work 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

II. RELATED WORK

Allocating resources in NFV networks is similar to appli-
cation 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. 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.

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

\( n \in N \) are represented by \( E_{in} \) and \( E_{out} \) 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 \( C \) on this physical network. A service chain \( C \in C \) 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 \( \Gamma^c \). All VM requests of a service chain \( C \) are contained in the set \( R^V(M)(C) \).
- **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 \( \Gamma^s \) 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 \( R^s(C) \).

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 \( R(C) = R^V(M)(C) \cup R^s(C) \). The collection \( R \) contains all requests in the model, aggregated over all of the service chains in \( C \).

All computational nodes in \( n \in N^c \) have a resource capacity \( C_n^\gamma \), for all resource types \( \gamma \in \Gamma^c \). Similarly, all service nodes \( n \in N^s \) offer a limited amount of resources \( C_n^\gamma \), 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 \( r \in R \). 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, \( A_{rn}^s \), that defines whether a resource request \( r \in R \) is allocated on a node \( n \in N \). An integer decision variable \( IC_n^\gamma \) specifies the number of times that a service \( s \in S \) 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 \( IC_n^\gamma \) 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 \( C_n^\gamma \). The total amount of used resources is composed out of the resources used for handling VM requests, represented as \( U^V(M) \) and shown in Equation (2), and out of the resources used to provide service instances, represented as \( U^s \) (shown in...
Objective function. This is however outside of the scope of this paper. To minimize the cost of using specific servers using additional weights in the objective function, ensuring it is minimized during optimization. To this resource use may also be added directly to the objective function. The number of cores used can be used for this.

\[ U^V_M(n, \gamma) = \sum_{r \in R^V_M} A^n_r \times D^\gamma(r) \]  

Equation (2).

\[ U^S(n, \gamma) = \sum_{s \in S} IC^n_s \times D^\gamma(s) \]  

Equation (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).

\[ \forall n \in N : \forall s \in S : \forall \gamma \in \Gamma^s : T^\gamma(s, n) \geq N^\gamma(s, n) \]  

Equation (4).

\[ T^\gamma(s, n) = IC^n_s \times C^\gamma_s \]  

Equation (5).

\[ N^\gamma(s, n) = \sum_{r \in R^s} A^n_r \times D^\gamma(r) \]  

Equation (6).

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

\[ \forall r \in R : \sum_{n \in N} A^n_r = 1 \]  

Equation (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 \( n \).

\[ \forall n \in N^c : \forall r \in R : U_n \geq A^n_r \]  

Equation (8).

The model uses the service request \( r \) 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.

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 (\( \in \Gamma^s \)) 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.

\[ U^+ = \sum_{(\gamma, n) \in N^+} U^V_M(n, \gamma) + U^S(n, \gamma) \]  

Equation (9).

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

\[ \min \left( U^+ + \sum_{n \in N^c} U_n \right) \]  

Equation (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.

---

**C. Adding network-awareness**

To add network-awareness to the model, the flow between two requests \((r_1, r_2) \in R^2\) 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.

\[ \forall (r_1, r_2) \in D : \forall n \in N : A^n_{r_1} + \sum_{e \in E_{out}} F(e, r_1, r_2) = A^n_{r_2} + \sum_{e \in E_{in}} F(e, r_1, r_2) \]  

Equation (11).

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

\[ \forall (r_1, r_2) \in D : \forall n \in N : A^n_{r_1} + \sum_{e \in E_{out}} F(e, r_1, r_2) \leq 1 \]  

Equation (12).

\[ \forall (r_1, r_2) \in D : \forall n \in N : A^n_{r_1} + \sum_{e \in E_{in}} F(e, r_1, r_2) \leq 1 \]  

Equation (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.

\[ \forall e \in E : \sum_{(r_1, r_2) \in D} F(e, r_1, r_2) \times D(r_1, r_2) \leq C(e) \]  

Equation (14).

\[ \forall (r_1, r_2) \in L : \sum_{e \in E} F(e, r_1, r_2) \times L(e) \leq L(r_1, r_2) \]  

Equation (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...
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, \( \alpha \) and \( \beta \) 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.

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 \( \alpha \) and \( \beta \) 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 \( \beta \) 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.
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.

ACKNOWLEDGMENT

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

REFERENCES