Coordinated Allocation of Service Function Chains

Michael Till Beck
Ludwig-Maximilians-Universität München, Germany
michael.beck@ifi.lmu.de

Juan Felipe Botero
Universidad de Antioquia, Medellin, Colombia
juanf.botero@udea.edu.co

***Abstract*—Network Functions Virtualization (NFV) is an emerging initiative to overcome increasing operational and capital costs faced by network operators due to the need to physically locate network functions in specific hardware appliances. In NFV, standard IT virtualization evolves to consolidate network functions onto high volume servers, switches and storage that can be located anywhere in the network. Services are built by chaining a set of Virtual Network Functions (VNFs) deployed on commodity hardware. The implementation of NFV leads to the challenge: How several network services (VNF chains) are optimally orchestrated and allocated on the substrate network infrastructure? In this paper, we address this problem and propose CoordVNF, a heuristic method to coordinate the composition of VNF chains and their embedding into the substrate network. CoordVNF aims to minimize bandwidth utilization while computing results within reasonable runtime.**

***Index Terms*—Network Function Virtualization, Chaining of Network Functions, Orchestration and Allocation**

I. Introduction

Nowadays, with the proliferation of middleboxes in the Internet, the provided network services can be viewed as chains of Network Functions (NFs). Fig. 1 shows an example of a specific network service whose packets shall pass through: 1) a firewall, 2) a deep packet inspector, 3) an encryption server, 4) a network monitor, and 5) a decryption server. Currently, each one of these NFs is implemented in a middlebox (hardware appliance) that is physically located inside the network.

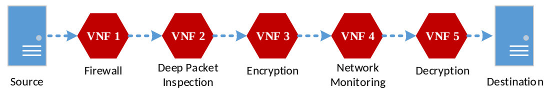

Fig. 1: Service as a Chain of Network Functions

Network Functions Virtualization is an emerging initiative driven by industry; it is being standardized by the European Telecommunications Standards Institute (ETSI) in a joint effort that includes 37 of the world's major service providers [1], [2]. In NFV, standard IT virtualization evolves to consolidate all the current network functions onto high volume servers, switches and storage that can be located anywhere in the network. Services are deployed by chaining a set of Virtual Network Functions (VNFs) in the commodity hardware. This decouples functionality from location, allowing software to be located at the most appropriate places. As a consequence, services can be deployed in the same hardware that can concurrently execute more than one functionality.

One of the main changes NFV brings is the dynamics and flexibility introduced by virtualization to deploy chains of VNFs in network locations that are more convenient to a predefined operator's objective; this differs from the current static network function chain placement that depends on the physical location of the middleboxes in the network [3]. In NFV, the chain of VNFs composing an end-to-end network service is called Virtual Network Function Forwarding Graph (VNF-FG) [4]. This graph is composed by the ordered set of VNFs that the network service runs in order to fulfill its attributes (e.g. reliability, availability, security and performance). Fig. 2 shows the deployment of an end-to-end network service between two substrate nodes in a NFV-enabled substrate network (SN) infrastructure.

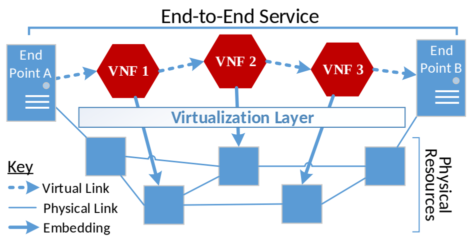

Fig. 2: NFV-enabled Architecture (based on [3])

In NFV, a network service is composed by a number of VNFs. That is, a packet must pass through a set of VNFs to be part of the offered network service. As VNFs are software, one of the main challenges that arises is: How to allocate the set of VNFs that compose a network service in the most adequate way, with respect to the service provider goals, inside the SN? We call this challenge the *NFV resource allocation problem* (NFV-RA). In turn, NFV-RA can be split into two correlated sub-problems:

1) **Chain composition**: A network service is a chain of VNFs (VNF-FG), so dependencies between VNFs have to be considered. The order of VNFs is often flexible, due to the fact that several chains can fulfill the same service: Some VNFs have to be placed in a specific order (e.g. a decryption VNF must be after an encryption one), while others are flexible in that regard.
2) **VNF-FG embedding**: When the most adequate VNF-FG is chosen for the requested service, the main subsequent challenge is the optimization of the resource allocation with regard to a specific objective (e.g. maximization of remaining network resources, minimization SN's power consumption, optimization of a specific QoS metric, etc.). In an NFV scenario, resource allocation

refers to the deployment of the VNF-FG on the SN. Therefore, Virtual Machines (VMs) hosting those VNF instances have to be deployed inside the SN. Here, we name this challenge the *Virtual Network Function Forwarding Graph Embedding* problem (VNF-FGE).

This paper proposes a heuristic approach called *CoordVNF*. The contributions of this novel approach are twofold:

1) **Coordinated allocation**: CoordVNF solves both sub-problems in a coordinated way, that is, while the VNF-FG is composed, it is allocated in the SN. In this way, CoordVNF improves the likelihood of success in the allocation as the building of the VNF-FG depends, in each step, on its successful allocation.
2) **Reasonable runtime**: The heuristic nature of Coord-VNF allows it to compute results within reasonable runtime with negligible performance effects.

II. Related Work

This section is divided into two parts. First, it introduces the Virtual Network Embedding (VNE) problem and explains how it differs from the NFV-RA problem. The second part describes existing approaches to solve the resource allocation problem in NFV.

A. Virtual Network Embedding

Embedding virtual networks in a SN is the main resource allocation challenge in network virtualization and is usually referred to as the VNE problem [5], [6]. VNE deals with the allocation of virtual resources both in nodes (mapped to substrate nodes) and links (mapped to substrate paths). Computing optimal VNEs is a NP-hard optimization problem [5], and likewise, the VNF-RA problem is.

Although sharing several characteristics, VNE and NFV-RA are not the same: VNE deals with static virtual network topologies where virtual nodes are arranged in a fixed, predetermined order. Likewise, embedding constraints are also immutably assigned to virtual entities. Contrary, NFV-RA deals with VNF chains (VNF-FGs) that are generated based on Virtual Network Function Requests (VNFRs) where the placement of the VNFs is flexible (i.e. one service can be described by several VNF-FGs). Performance constraints of VNF instances change depending on how much traffic is routed to them: E.g., computing resource demands of a transcoding VNF vary depending on how many multimedia data have to be transcoded. Also, bandwidth demands change depending on the ordering of VNF instances.

Even when the VNF-FG is built, VNF-FGE differs from VNE. In VNE, each virtual node represents an isolated VM in the SN. In contrast, several VNFs of the same type may share the same VM; a VNF mapping does not imply the instantiation of a new VM. In fact, to avoid the deployment of many VMs, a VNF can be embedded in an already instantiated VM that is hosting VNFs of the same type.

However, VNE approaches can be a good starting point to build NFV-RA solution strategies. In fact, the approach presented in this paper is inspired by a backtracking-based VNE algorithm and implements a similar strategy for finding valid allocation options [7].

B. Resource allocation in NFV

Few approaches have been proposed to solve the NFV-RA problem so far. One of them is proposed in [8]. Here, NFV-RA sub-problems are solved in two separated stages: 1) The requested functions using a greedy heuristic that generates a VNF-FG trying to reduce the data rate of flows and 2) once the service request has been chained, the VNF-FGE problem is solved exactly by using a Mixed Integer Quadratic Constrained Program (MIQCP) with regard of three different optimization criteria: a) maximize remaining data rate on network links, b) increasing energy efficiency and c) minimizing the latency of the created paths. Solving VNF-FGE with an exact algorithm entails a huge cost in running time for medium to large-size scenarios. Also, solving the problem in separated stages does not guarantee that the generated VNF-FG is able to be embedded in the second stage, even if there is a viable solution for the NFV-RA problem.

Another recent approach is introduced in [9]. Here, the authors propose three greedy algorithms and one tabu search meta-heuristic to solve the allocation of VNFs to virtual machines. However, this set of algorithms considers that VNF-FGs are embedded in a set of VMs that are previously allocated in the SN by means of a virtual datacenter embedding algorithm [10]. That is, the embedding of the VNF-FG does not influence VMs deployment, and therefore, the challenge discussed differs from the VNF-RA problem.

In contrast to the approach presented in [8], CoordVNF proposes a coordinated heuristic that generates and embeds VNF-FGs simultaneously. As shown in the evaluation section, CoordVNF is able to solve the VNF-RA problem in reasonable runtime, even for larger network scenarios. In contrast to [9], CoordVNF does consider the deployment of VMs in the SN, which depends on the allocation of VNFs.

III. Problem definition

This section describes the NFV-RA problem, please see Table I for its formal notation.

The objective of a NFV-RA algorithm is to embed a set of VNF embedding requests (VNFRs) on top of a shared SN infrastructure in an efficient way. The algorithm has to consider placement constraints and dependencies between VNFs. Fig. 3 depicts a VNFR and two possible chainings of its VNF instances (notation is described in Table I). For each VNFR, the initial data rate of the network flow is defined and the substrate nodes where the flow initiates and terminates are specified. VNFs can split (or merge) the traffic flow: If a VNF has more than one outgoing link, the traffic flow is split into several subsequent sub-flows (cf. Fig. 3a), if it has more than one incoming link, it merges several sub-flows. For each outgoing link, the relative traffic rate is defined. For instance, for a deep packet inspection VNF separating incoming data into two streams (e.g., TCP and non-TCP traffic), it can be

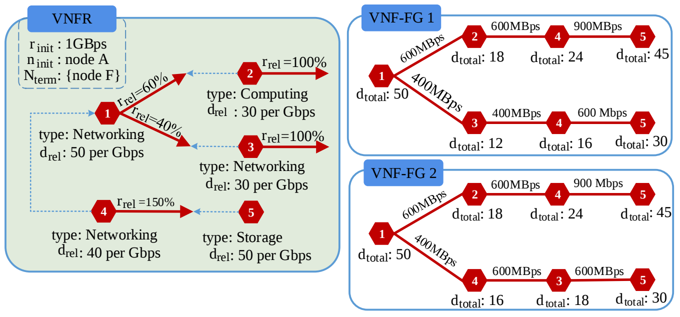

Fig. 3: Chaining of VNFs

specified that 60% of the incoming traffic is forwarded to a VNF 2 and 40% to VNF 3 (cf. VNF 1, 2, and 3 in Fig. 3a).

Depending on the ordering of the VNFs, bandwidth demands of the network flow changes. The ordering of VNFs is flexible, but has to consider dependencies between VNFs; e.g., the network flow first has to traverse a set of VNFs before it arrives at a specific VNF (cf. dotted line in Fig. 3a from VNF 4 to VNF 1, meaning that VNF 4 depends on VNF 1 and must therefore be executed after VNF 1). Dependencies are depicted as dotted lines between VNFs and VNF links in Fig. 3a (cf. VNFs 2 and 3, both pointing to the VNF links of VNF 1). As described in Table I, dependencies are specified for the incoming links of a VNF and refer to outgoing links of other VNFs; this enables the definition of VNFs which should only selectively be placed on some of the sub-flows: E.g., a VNF separating the flow into TCP and non-TCP traffic forwards only the TCP traffic to a TCP optimizer VNF.

Based on the dependencies, valid chaining options of VNFRs are derived. Fig. 3b depicts two possible chaining options for the VNFR shown in Fig. 3a). For each VNF, one or more VNF instances are created: This is due to the fact that in some scenarios, if the network flow is split, traffic has still to be processed by the same types of VNFs, even if traffic is not routed through the same VNF instances (in Fig. 3b, both chains require two instances of VNF 4).

Furthermore, placement constraints of the VNFs have to be considered. Each VNF specifies whether it needs to be placed on a storage/networking/computing node. Hosting capabilities of substrate nodes are limited; for performance considerations, only a limited number of VMs can be deployed on top of a substrate node, assuming that otherwise hosting overhead is becoming too large. Similar to related work, it is assumed here that once a VNF instance has been deployed on top of a substrate node, the VM which is hosting the operating system of that VNF instance can be shared among other VNF instances of the same type [8]. E.g., if two database VNF instances and a file server VNF instance are allocated to a node, only two VMs need to be installed since the two database instances are assigned to one shared VM. Moreover, the amount of required processing capacities is specified for handling the network flow. E.g., a VNF performing video encoding should always be embedded on top of a computing node and demands 500MHz of CPU processing capacities to encode 100MBit/s. The amount of required capacities depends on the amount of data handled by that VNF instance.

TABLE I: NFV-RA Model

$\mathbf{G} = (\mathbf{N}, \mathbf{L})$: The substrate network G is modeled as a graph with nodes N and links L	Substrate Network
host : $\mathbf{N} \rightarrow \mathbb{Z}$: Residual hosting capabilities of substrate nodes for hosting a limited number of VMs	
$\mathbf{res_{node}}$: $\mathbf{N} \rightarrow \mathbb{Z}$: Residual processing capacities; processing capacities are provided by a substrate node for processing computing/storage/networking tasks of the VNF instances	
$\mathbf{res_{link}}$: $\mathbf{L} \rightarrow \mathbb{Z}$: Residual bandwidth resources provided by substrate links for hosting virtual links	
$\mathbf{type}_s$: $\mathbf{N} \rightarrow \{\mathbf{comp}, \mathbf{stor}, \mathbf{net}\}$: Determines the type of a substrate node; a substrate node can only host VNFs of the same type $(\text{type}_v(v) = \text{type}_s(n))$, e.g., a computing node can only host computing instances	
$\mathbf{VNFR}^i(\mathbf{V}^i)$: The i-th VNFR with VNFs V^i	VNFR
$\mathbf{type}_v^i$: $\mathbf{V}^i \rightarrow \{\mathbf{comp}, \mathbf{stor}, \mathbf{net}\}$: Determines the type of a VNF	
$\mathbf{l}_{\mathbf{out}}^i$: $\mathbf{V}^i \rightarrow L^i$: Determines the outgoing VNF links; a VNF with multiple outgoing links splits the flow into several sub-flows	
$\mathbf{l}_{\mathbf{in}}^i$: $\mathbf{V}^i \rightarrow L^i$: Determines the incoming VNF links; a VNF with multiple incoming links merges several sub-flows	
$\mathbf{r}_{\mathbf{init}}^i$: $\mathbb{Z}$: Initial data rate of a VNFR	
$\mathbf{n}_{\mathbf{init}}^i \in \mathbf{N}$: Initial substrate node of the flow	
$\mathbf{N}_{\mathbf{term}}^i \subset \mathbf{N}$: Substrate nodes where the flow terminates	
$\mathbf{d}_{\mathbf{rel}}^i$: $\mathbf{V}^i \rightarrow \mathbb{R}$: Relative processing capacity demands of a VNF	
$\mathbf{r}_{\mathbf{rel}}^i$: $\mathbf{L}^i \rightarrow \mathbb{Z}$: Relative link traffic rate	
$\mathbf{req}^i$: $\mathbf{L}^i \rightarrow \{L\}$: Dependencies of a VNF; those are defined for the incoming edges of a VNF and refer to outgoing links of other VNFs. This also allows the specification of VNFs that get selectively deployed on specific sub-flows. An assignment of a VNF instance is only valid if traffic has first been routed through instances of all the required VNFs.	
$\mathbf{VNF\text{-}FG}^{i,j}(\mathbf{V}_{\text{inst}}^{i,j}, \mathbf{L}_{\text{inst}}^{i,j})$: The j-th VNF-FG of VNFR^i; A VNF-FG is modeled as a graph with VNF instances $V_{\text{inst}}^{i,j}$ and links $L_{\text{inst}}^{i,j}$. For each VNF/VNF link that is defined in a VNFR, there can be multiple instances in a VNF-FG (cf. Fig. 3)	VNF-FG
$\mathbf{d}_{\mathbf{total}}^{i,j}$: $\mathbf{V}_{\text{inst}}^{i,j} \rightarrow \mathbb{R}$: Determines total demands of a VNF instance in a VNF-FG; total demands are computed based on the relative processing capacity demands as defined in the VNFR and the amount of traffic that is routed to that instance. A substrate node $n \in N$ can only host a VNF $v \in V_{\text{inst}}$ if $\text{d}_{\text{total}}(v) \leq \text{res}_{\text{node}}(n)$	
$\mathbf{r}_{\mathbf{total}}^{i,j}$: $\mathbf{L}_{\text{inst}}^{i,j} \rightarrow \mathbb{Z}$: Total link traffic rate	

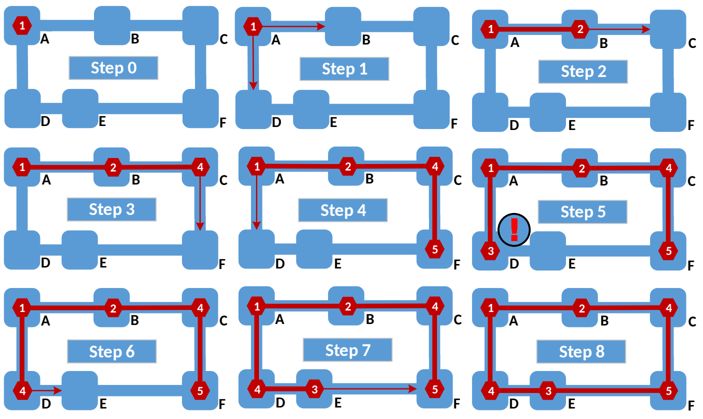

Fig. 4: CoordVNF Example

IV. A Coordinated Embedding Approach

This section introduces the CoordVNF algorithm. CoordVNF solves the NFV-RA problem in one coordinated step, which leads to runtime improvements in scenarios where no valid embedding of a VNF can be found: in this case, a two-step approach discards all previously computed embedding steps and re-starts from scratch with another chaining result [8]. Contrary, CoordVNF makes use of the backtracking concept: It recursively tries to find valid assignment options for the VNF instances; if it fails at some point, it discards the last embedding steps and iteratively tries to embed alternative chaining options.

The CoordVNF algorithm is introduced as follows: First, the main idea is described based on a simple example. Then, the algorithm is described in detail.

A. CoordVNF Example

CoordVNF handles the NFV-RA problem in one coordinated step: the chain composition problem and the VNF-FGE are coherently solved. Fig. 4 depicts a simplified example, based on the scenario presented in Fig. 3. Here, an end-to-end service has to be deployed between substrate nodes A and F. For this example, it is assumed that a substrate node can host no more than one VM ($\text{host}(n) = 1, \forall n \in N$). Furthermore, search depth is limited to a maximum path length of 1.

CoordVNF starts at the initial substrate node of the VNFR, node A. First, the algorithm tries to attach the first VNF, VNF 1. Therefore, from node A, a breadth-first-search is performed in order to find all substrate nodes within a specified range that provide sufficient resources for hosting VNF 1. Since node A itself provides sufficient hosting and processing resources, VNF 1 can be directly assigned to A (step 0 in Fig. 4).

VNF 1 splits the current flow into 2 subsequent sub-flows; First, CoordVNF aims to embed the first sub-flow. Therefore, a VNF is selected where all requirements have been met so far (VNF 2). For hosting VNF 2, there are 2 possible substrate node candidates, node B and D (step 1). First, CoordVNF tries to assign VNF 2 to B (step 2). Then, VNF 4 can only be assigned to node C (step 3), and finally, VNF 5 is hosted on terminating node F (step 4). Now, the first sub-flow was successfully embedded and the algorithm returns to node A, trying to attach the second sub-flow (step 4): To this end, first, CoordVNF assigns VNF 3 to node D (step 5). Now, let's assume here that node E is not able to host instances of VNF 4. As no alternative VNF can be selected, the algorithm is not able to continue embedding. Therefore, a backtracking step has to be performed, and the assignment of VNF 3 to node D is undone. Now, instead of VNF 3, VNF 4 is hosted on node D (step 6). As a result, VNF 3 can then be successfully embedded on nodes E (step 7). Finally, VNF 3 is connected to the VNF instance of VNF 5 which is hosted on terminating node F (step 8); to this end, node demands of VNF 5 are updated due to the fact that the total amount of traffic that needs to be processed by VNF 5 is increased by the current sub-flow. Note that CoordVNF realized that VNF-FG 1 could not be embedded in the SN and, recursively, it reversed a composition decision and successfully embedded VNF-FG 2.

B. Detailed Description

Algorithm 1 presents VNFCoord's pseudo-code. The algorithm starts at the initial substrate node of the VNFR (function VNFCOORD). First, the REC function is invoked:

In each recursive step, first, the set of suitable VNF instances the current sub-flow can be passed through is selected based on the specified dependencies. If no additional VNFs need to be attached, the recursive function returns the set of embeddings that have been computed so far (line 6).

Otherwise, the algorithm iteratively tries to find valid allocation options for the demanded ressources. Therefore, CoordVNF has to find suitable substrate nodes providing sufficient resources for hosting demanded node capacities of the sub-flow. At the same time, for each VNF candidate, valid embeddings for the link connecting the VNF instance that was mapped in the previous step to the current candidate has to be found. To this end, absolute capacity demand of the VNF instance is calculated based on the amount of traffic routed to the particular VNF instance. Then, breadth-first search is performed starting from the substrate node hosting the predecessor (line 8). This way, CoordVNF determines a list of candidate substrate nodes fulfilling capacity demands of the VNF candidate. To reduce maximum path length between two embedded VNF instances, and for runtime considerations, depth of the breadth-first search is limited by the *maxPathLength* parameter.

The algorithm tries to map the VNF candidate to a substrate candidate node and to allocate bandwidth resources between the substrate node and the node hosting the previously mapped VNF (line 10). Therefore, CoordVNF iteratively selects adequate substrate candidates, beginning with those that are immediately connected to the predecessor. Once a valid embedding has been found, CoordVNF tries to find valid embeddings for all subsequent VNFs by invoking the recursive function (line 18). If one of the sub-flows could not be embedded successfully, backtracking is performed: in this case, the last assignments are revoked and the algorithm tries to embed one of the alternative candidate options (line 22-27).

V. Evaluation

In this section, first, CoordVNF's performance is evaluated for different *maxPathLength* parameter settings. Then, CoordVNF is compared to MIVNF, the approach presented in [8]. CoordVNF is implemented and evaluated using the ALEVIN simulation framework [11].

A. CoordVNF's maxPathLength Parameter

After a VNF instance has been mapped to a substrate node, CoordVNF scans for other substrate nodes in close proximity to the first node. Those nodes are then considered as candidates for hosting the subsequent VNF instance. This section evaluates the *maxPathLength* parameter which limits the maximum distance between the mapped node and the substrate candidates.

```
1:  function VNFCoord($\mathbf{VNFR}^i$,G)
2:      return Rec($\mathbf{VNFR}^i$,G,$\mathbf{n}^i_{\mathbf{init}}$,$\mathbf{r}_{\mathbf{init}}(\mathbf{VNFR}^i)$,{})
3:
4:  function Rec($\mathbf{VNFR}^i$,G,prevSNode,dr,$M$)
5:      VNFs ← VNFOptions($\mathbf{VNFFG}^i$,$M$)
6:      if VNFs=∅ then
7:          return $M$
8:      candidates ← BFS($\mathbf{VNFR}^i$,G,prevSNode,VNFs,dr,$M$)
9:      sortCandidates(candidates)
10:     for all c ∈ candidates do
11:         $M'$ ← MAP_NODE_DEMANDS(c,dr)
12:         $M'$ ← $M'$ ∪ MAP_LINK_DEMANDS(c,dr,prevSNode)
13:         success ← True
14:         if all incoming flows of the VNF instance assigned then
15:             $M''$ ← $M'$
16:             for all l ∈ $l^i_{\text{out}}$(VNF) do            ▷ embed subflows
17:                 dr′ ← dr · d(l)
18:                 $M_c$ ← Rec($\mathbf{VNFR}^i$,G,c.node,dr′,$M''$)
19:                 if $M_c \neq \emptyset$ then
20:                     $M''$ ← $M''$ ∪ $M_c$
21:                 else
22:                     success ← False
23:                     UNDOALL($M''$)
24:                     break
25:         if success=True then
26:             return $M''$
27:     return ∅                                           ▷ backtracking
```

Algorithm 1: VNFCoord Algorithm

1) Methodology: The evaluation based on randomly created SN topologies with 50 nodes were generated using the Waxman model [12]. Each substrate node offers capabilities for hosting 4 different VNF types, providing randomly assigned processing capacities between 50-100 units. Bandwidth resources for substrate links were randomly chosen between 50-100 bandwidth units.

In each scenario, 10 VNFRs were generated based on a set of randomly generated VNFs: 5 VNFs were assigned to each VNFR. Dependencies between VNFs were randomly chosen. Initial data rate of the VNFRs was set to 50 bandwidth units, and the amount of required processing capacities to 1 capacity unit/bandwidth unit. Relative traffic rate of the VNFs was randomly chosen between 50% and 150%.

For each parameter combination, 50 scenarios were generated in order to ensure stability of the results that are shown with a confidence interval of 95%.

2) Results: The *maxPathLength* parameter limits the maximum distance between two connected VNF instances that are embedded into the substrate networks. This effect is shown in Fig. 5a, which depicts the average and maximum path length between VNF instances. For bigger *maxPathLength* values, path lengths increase. As shown in Figs. 5c and 5d, this also leads to increasing bandwidth usage per link and higher embedding cost. In line with related work, embedding cost is defined as the sum of reserved processing capacities and bandwidth resources [5].

So, in general, a small value should be chosen for this parameter. However, choosing the *maxPathLength* parameter too restrictively leads to scenarios where CoordVNF is not able to find a valid embedding of the VNF-FGs. This is depicted in Fig. 5b: Here, the rate of successfully embedded VNFRs is shown. When *maxPathLength* is chosen too small, the number of successfully embedded VNFRs decreases.

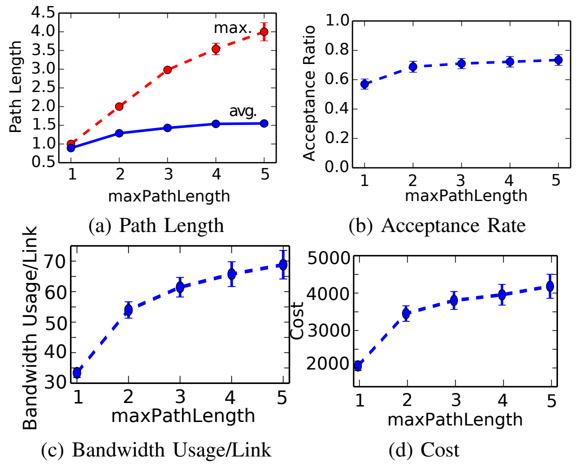

Fig. 5: CoordVNF's *maxPathLength* Parameter (1)

Due to the fact that *maxPathLength* limits the maximum path length between substrate nodes, it implicitly limits the number of substrate candidates considered in each recursive step. This is depicted in Fig. 6a: For increasing *maxPathLength* values, the number of backtracking steps performed by CoordVNF increases due to the fact that the algorithm considers more alternative substrate nodes to embed the VNFs. The influence of *maxPathLength* is most notable in scenarios where a valid embedding cannot be found at once. Therefore, besides the average number of backtracking steps, the maximum number of steps is also depicted here. An increasing number of backtracking steps directly affects runtime of CoordVNF, as shown in Fig. 6b.

Therefore, there is a tradeoff between maximum runtime of the algorithm, acceptance rate, and embedding cost. Concluding from the figures, *maxPathLength*$=2$ is a good tradeoff in these scenarios.

B. Comparing CoordVNF to MIVNF

This subsection compares CoordVNF to the MIVNF approach. MIVNF is the only approach presented so far aiming to solve the embedding problem for VNF chains. Evaluation of MIVNF could be performed based on the original implementation, since the authors of MIVNF gratefully provided the source code used to produce the results presented in their paper

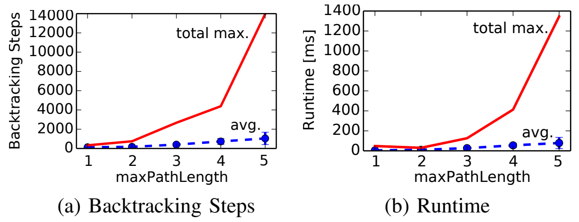

Fig. 6: CoordVNF's maxCandidates Parameter (2)

TABLE II: Runtime Comparison

|N|	CoordVNF	MIVNF
10	16ms	1.8s
15	6ms	41.5s
20	9ms	11m55s
25	4ms	33m45s
30	7ms	12h56m
50	6.6s	–
100	26.6s	–
200	1m50s	–
300	4m48s	–
400	9m42s	–
500	17m43s	–

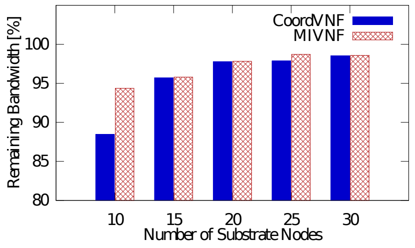

Fig. 7: Remaining Bandwidth Resources

[8]. MIVNF supports three optimization strategies (minimizing bandwidth usage, increasing energy efficiency, and minimizing path latency). Since CoordVNF aims to reduce embedding cost, MIVNF's bandwidth minimizing strategy was chosen for comparing both approaches. To evaluate both algorithms, various substrate networks of different sizes were randomly generated. Furthermore, a VNFR with 6 VNFs was selected that was also used by the authors of MIVNF (it is based on the NFV use cases presented in [13]).

Table II and Fig. 7 depict performance measurements for both CoordVNF and MIVNF. MIVNF aims to solve the problem in two steps: first, the chaining of the VNFs (e.g., in terms of bandwidth demands) is computed; then, in the second step, the best chaining result is embedded. In MIVNF, the embedding step is formulated as a MIQCP. MIVNF aims to find the optimal result for the embedding step. In contrast, the heuristic CoordVNF algorithm computes both the chaining of the VNFs and the embedding of the VNFs in one single step. Table II compares runtime of MIVNF and CoordVNF. While both approaches were able to embed all VNFRs successfully, runtime of CoordVNF is significantly faster in these scenarios than MIVNF. While CoordVNF terminates within the range of milliseconds, MIVNF takes several seconds even for such small-sized scenarios. For CoordVNF, Table II also depicts runtime measurements for even larger SN topologies. As shown here, CoordVNF performs very well with respect to runtime even for networks with 500 nodes.

Fig. 7 depicts the remaining bandwidth for both algorithms. As shown here, the cost that CoordVNF pays off to have better runtime than MIVNF is a slightly worse performance (5% in the worst case) with respect to embedding quality.

VI. Conclusion and Future Work

This paper introduces CoordVNF, an heuristic approach to solve the NFV-RA problem in a coordinated way. First evaluation results indicate that CoordVNF is significantly faster in scenarios with a relatively small number of VNFs than existing approaches. Consequently, CoordVNF is able to compute embedding results in large substrate network topologies. Quality of the embedding results remain comparable between CoordVNF and existing approaches, showing only negligible degradations for CoordVNF.

Future work will be devoted to the evaluation of CoordVNF in online scenarios where VNFRs dynamically arrive and depart. CoordVNF is currently evaluated by the authors in large-scale scenarios with a large number of VNFs. Furthermore, reconfiguration of already allocated VNF-FGs to improve the acceptance of arriving VNFRs is an exciting branch of research that is left for future work.

Acknowledgements

This research has partly been supported by the CODI project 2014-856 of "Universidad de Antioquia UdeA" and "Sistema Nacional de Regalías (Proyecto convenio 512C-2013 código SUI 20139080)".

References

[1] ETSI, "Network Functions Virtualisation," http://www.etsi.org/technologies-clusters/technologies/nfv.

[2] M. Ciosi and et al, "Network Functions Virtualisation (NFV): Network Operator Perspectives on Industry Progress," http://portal.etsi.org/NFV/NFV_White_Paper2.pdf, ETSI, Tech. Rep., October 2014.

[3] G. ETSI, "Network Functions Virtualisation (NFV): Architectural Framework," *ETSI GS NFV*, vol. 2, p. V1, 2013.

[4] ——, "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV," *ETSI GS NFV*, 2013.

[5] A. Fischer, J.-F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: A survey," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 4, pp. 1888–1906, Fourth 2013.

[6] M. T. Beck, A. Fischer, J. F. Botero, C. Linnhoff-Popien, and H. de Meer, "Distributed and scalable embedding of virtual networks," *Elsevier Journal of Network and Computer Applications*, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804515001393

[7] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*. New York, USA, Aug. 2009, pp. 81–88.

[8] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, Oct 2014, pp. 7–13.

[9] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turk, and S. Davy, "Design and evaluation of algorithms for mapping and scheduling of virtual network functions," in *Network Softwarization, IEEE 1st International Conference on*, April 2015.

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

[11] M. T. Beck, C. Linnhoff-Popien, A. Fischer, F. Kokot, and H. de Meer, "A simulation framework for virtual network embedding algorithms," in *Telecommunications Network Strategy and Planning Symposium (Networks), 2014 16th International*, Sept 2014, pp. 1–6.

[12] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal of Selected Areas in Communication*, vol. 6, no. 9, pp. 1617–1622, Dec. 1988.

[13] W. Liu and et al., "Service Function Chaining (SFC) Use Cases, Active Internet-Draft, IETF Secretariat, Internet-Draft," 2014.