Leveraging Heterogeneous NFV Platforms to Upgrade Service Function Chains in DCNs

Yuhan Xue and Zuqing Zhu†
School of Information Science and Technology, University of Science and Technology of China, †Email: {zqzhu}@ieee.org

***Abstract*—The advances on SmartNICs and programmable data plane (PDP) switches motivate people to deploy virtual network functions (vNFs) on these devices directly for better packet processing performance. Hence, this work considers how to leverage heterogeneous network function virtualization (NFV) platforms to upgrade the provisioning of vNF service chains (vNF-SCs) in a datacenter network (DCN). We design a novel time-efficient heuristic, to plan the service upgrade under a limited budget such that the QoS improvement on latency reduction can be maximized. Specifically, our algorithm first chooses nodes in the DCN to upgrade (*i.e.*, by either equipping SmartNICs on certain servers or replacing switches with PDP-SWs), and then adjusts vNF-SC provisioning accordingly.**

***Index Terms*—Network function virtualization (NFV), Heterogeneous NFV platforms, Service function chain (SFC).**

I. Introduction

Recently, the booming of cloud computing has stimulated the exploration of new network architectures [1], and novel transmission and switching technologies have made the physical infrastructure of networks more ready for adaptive service provisioning [2–4]. Meanwhile, the advances on programmable packet processing hardware (*e.g.*, SmartNICs and programmable data plane (PDP) switches) have promoted the idea of in-network computing [5], which enables the execution of virtual network functions (vNFs) in forwarding devices without affecting their performance on normal packet forwarding [6, 7]. Therefore, the success of network function virtualization (NFV) [8] can be further expanded, if we consider these hardware platforms (*i.e.*, SmartNICs and PDP switches (PDP-SWs)) together with their software counterparts (*e.g.*, virtual machines (VMs)) to realize heterogeneous NFV environments [9–12]. This is because the hardware platforms have superior packet processing performance, and thus they can provide service providers (SPs) unprecedented capabilities to address stringent quality-of-service (QoS) demands [6, 7].

Previously, to explore the benefits of heterogeneous NFV platforms, people have studied how to provision vNF service chains (vNF-SCs) with them [9–12]. However, the problem of upgrading existing vNF-SCs has not been considered, while it is as important as that of the initial provisioning. This is because in a practical network, service upgrades are inevitable, and can even be frequent to adapt to the ever-increasing service demands [13]. Note that, service upgrades involve incremental deployment and adjustment, which makes the optimizations of them significantly different from provisioning vNF-SCs from scratch. There are a few studies on the reconfiguration of vNF-SCs in the literature. For instance, the study in [8] tried to reconfigure vNF-SCs to adapt to the movement of clients, and Hu *et al.* [14] proposed to leverage vNF-SC re-organization and task scheduling to address the overheads related to VM-based vNF-SC provisioning. Nevertheless, to the best of our knowledge, the service upgrade of vNF-SCs, which utilizes heterogeneous NFV platforms containing SmartNICs, PDP-SWs and VMs, has not been addressed yet.

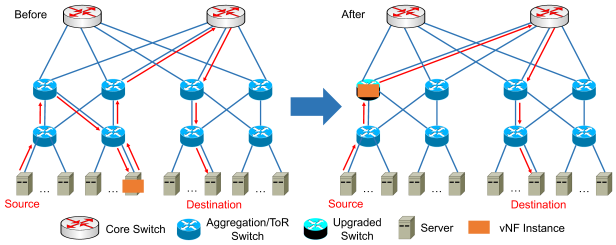


Fig. 1. An example on service upgrade of a vNF-SC.

This work leverages heterogeneous NFV platforms to optimize the service upgrade of vNF-SCs. Specifically, we consider a datacenter network (DCN) as the substrate network (SNT), and study how to first choose nodes in it to upgrade (*i.e.*, by either equipping SmartNICs on certain servers or replacing switches with PDP-SWs) and then adjust the provisioning schemes of existing vNF-SCs accordingly. We propose a novel heuristic to tackle the optimization time-efficiently, which first aggregates vNF-SCs to form vNF forwarding trees (vNF-FTs), and then adjusts the provisioning schemes of vNF-SCs by remapping the vNF-FTs to the SNT. Our simulation results verify the performance of the proposed algorithms.

The rest of the paper is organized as follows. In Section II, we explain the problem considered in this work. The time-efficient heuristic is presented in Section III. Section IV discusses the simulations for performance evaluation. Finally, we summarize the paper in Section V.

II. Problem Description

NFV realizes network services by instantiating vNFs on general-purpose software/hardware platforms, and thus the usages of proprietary hardware systems can be minimized. This involves deploying vNFs on substrate nodes (SNs) and interconnecting them to form various types of virtual topologies [15–17]. The procedure is similar to the famous virtual network embedding (VNE) [18, 19]. Fig. 1 provides an example to explain the advantage of upgrading vNF-SCs with heterogeneous NFV platforms. Before the service upgrade, the source, vNF and destination of a vNF-SC are on three different racks in the DCN. The service upgrade uses a PDP-SW to

replace one aggregation switch that is on the vNF-SC's routing path, and makes it possible to deploy vNFs there. Then, the vNF-SC's routing path gets shortened by 4 hops. Moreover, as the packet processing in the PDP-SW is much faster than that in the original VM on a server, the end-to-end latency of the vNF-SC can be further reduced. Note that, end-to-end latency is an important QoS parameter to many network services.

We model the SNT as $G(V, E)$, where V and E are the sets of SNs and substrate links (SLs), respectively. As the SNT is a DCN, there are two types of SNs in it, *i.e.*, switches and servers. We assume that before the service upgrade, all the switches are traditional ones and thus vNFs can only be deployed on servers with VMs. The upgrade can either equip SmartNICs on certain servers, or replace certain switches with PDP-SWs. After the upgrade, we migrate vNFs to the programmable hardware. Here, the differences between SmartNICs and PDP-SWs are 1) PDP-SWs have better packet processing performance (*i.e.*, shorter processing latencies) than SmartNICs, and 2) SmartNICs have more IT resources for instantiating vNFs (*i.e.*, the memory for storing related flow tables) because they are equipped on servers and thus do not need to use many IT resources on normal packet forwarding as PDP-SWs do. Since SmartNICs and PDP-SWs support vNFs with programmable hardware, they provide much shorter processing latencies than VMs. The IT resources on servers for VMs are the most abundant among the three NFV platforms.

All the vNF types that are supported in the SNT are included in a set M. Considering the heterogeneity among PDP-SWs, SmartNICs and servers, we assume that if a type-m vNF is deployed on a PDP-SW/SmartNIC/server, it consumes different IT resources, and the corresponding processing capabilities are also different. Meanwhile, the latency reductions are various, if the vNF is migrated from a server to a PDP-SW/SmartNIC. Hence, we need to optimize the service upgrade of vNF-SCs. We take the vNF-SC provisioning schemes before the upgrade as the input, and aim to maximize the QoS improvement on latencies under a fixed budget for PDP-SWs and SmartNICs. The resource constraints are also considered to ensure that the vNF-SCs are deployed correctly after the service upgrade.

III. Algorithm Design

To solve the service upgrade problem time-efficiently, we design a heuristic, namely, FTA, which involves three steps: 1) merging vNF-SCs to build vNF-FTs, 2) expanding each vNF-FT to a mapping tree (MT) according to $G(V, E)$, and 3) determining node upgrade schemes and updating vNF-SC deployments accordingly based on the MTs. *Algorithms* 1-3 in the following subsections will describe the steps in detail. The notations used in the algorithm design are as follows.

- $G(V, E)$: the topology of the SNT.
- M: the set of all the vNF types supported in the SNT.
- $SC_i = \{s_i, d_i, \{f_{i,1}, \cdots, f_{i,l}, \cdots, f_{i,N_i}\}, b_i, t_i\}$: the i-th vNF-SC, where s_i and d_i are the source and destination nodes, respectively, b_i is its bandwidth demand, t_i is its QoS requirement on end-to-end latency, and N_i is the number of vNFs in its vNF-SC. We have $R = \{SC_i, \forall i\}$.
- $f_{i,l}^m$: the boolean that equals 1 if the l-th vNF ($f_{i,l}$) in SC_i is a type-m vNF ($m \in M$), and 0 otherwise.
- Ω: the total upgrade budget for PDP-SWs and SmartNICs.
- ϕ^S/ϕ^N: the cost of a PDP-SW/SmartNIC.

A. Constructing vNF-FTs

Algorithm 1 explains how we merge existing vNF-SCs to obtain vNF-FTs. *Lines* 1-2 are for the initialization. Here, we put vNF-SCs, whose sources and destinations are in the same server racks, into the same group. For instance, if vNF-SCs have their sources and destinations in the racks whose ToR switches are u and v ($u, v \in V$), respectively, they are put into group $g_{u,v}$. Then, the for-loop that covers *Lines* 3-25 builds a vNF-FT for each group. *Line* 3 initializes the vNF-FT for each group $g_{u,v}$ and sets its root as $\mathcal{T}.rt = u$. Next, we use the for-loop covering *Lines* 5-16 to check all the vNFs in each vNF-SC in sequence and build the vNF-FT accordingly. For each vNF-SC, we start from its source (*i.e.*, u) and introduce $\hat{v}$ to the current node that is being considered in the vNF-FT $\mathcal{T}$ (*Line* 6). If the vNF $f_{i,l}$ in SC_i has already been included in the $(l+1)$-th level of $\mathcal{T}$, we update the counter of the corresponding node $\mathcal{T}.nd$ (*Lines* 9-10). Otherwise, we create a new node to represent $f_{i,l}$ and insert it in $\mathcal{T}$ (*Lines* 12-13). Fig. 2 gives an illustrative example on the operation principle of *Algorithms* 1 and 2. It can be seen that each vNF-FT records the orders and numbers of vNFs in the vNF-SCs in a same group, where the number on each vNF denotes the value of the counter for the vNFs in the same type.

We use the for-loop that covers *Lines* 17-23 to trim the obtained vNF-FT $\mathcal{T}$ based on two empirical thresholds ξ_{lvl} and ξ_{cnt}. This is because to facilitate effective service upgrade, we should give higher priorities to upgrade the vNFs, which are shared by more vNF-SCs. Specifically, if in vNF-FT $\mathcal{T}$, a node for one vNF is in a level that is lower than ξ_{lvl} and its counter is smaller than ξ_{cnt}, we merge the subtree rooted in it with the subtree whose root is in the next level, represents the same vNF, and has the smallest counter (*Lines* 19-21). After merging the subtrees, we also update the counters of the related nodes accordingly. *Line* 24 inserts the trimmed vNF-FT in set FT. The time complexity of *Algorithm* 1 is $O(|R|^2 \cdot \max_i(N_i+1))$, where R is the set of existing vNF-SCs, and $\max_i(N_i+1)$ is the longest chain length of the vNF-SCs in R.

B. Expanding vNF-FTs to Obtain MTs

For each vNF-FT obtained by *Algorithm* 1, *Algorithm* 2 obtains the mapping schemes of the vNF-SCs aggregated in it based on the status of the SNT, such that the subsequent *Algorithm* 3 can leverage these mapping schemes to finally achieve effective service upgrade. Here, we refer to the mapping schemes of the vNF-SCs in a vNF-FT as a mapping tree (MT), and need to point out that the mapping schemes in an MT are just intermediate but not the final ones after the upgrade. *Line* 1 is for the initialization, and all the MTs initially use the same structures of their vNF-FTs. Then, the for-loop that covers *Lines* 2-20 expands each vNF-FT based on the status of the SNT to get an MT.

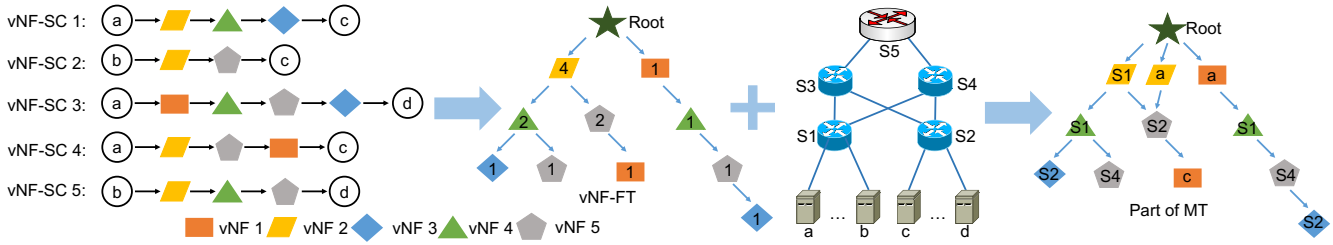


Fig. 2. Example on constructing vNF-FTs and obtaining MTs in *Algorithms* 1 and 2.

Algorithm 1: Building vNF-FTs based on vNF-SCs

Input: vNF-SCs $\{SC_i\}$, empirical thresholds ξ_{lvl} and ξ_{cnt}.
Output: Set of vNF-FTs FT.

1 $FT = \emptyset$;
2 group vNF-SCs based on their sources and destinations;
3 **for** *each group* $g_{u,v}$ **do**
4 $\mathcal{T} = \emptyset$, $\mathcal{T}.rt = u$;
5 **for** *each vNF-SC* SC_i *in* $g_{u,v}$ **do**
6 mark the current node in $\mathcal{T}$ as $\hat{v} = \mathcal{T}.rt = u$;
7 **for** *each vNF* $f_{i,l}$ *in* SC_i *in sequence* **do**
8 **if** $f_{i,l}$ *is in the* $(l+1)$*-th level of* $\mathcal{T}$ **then**
9 find node $\mathcal{T}.nd$ in the $(l+1)$-th level of $\mathcal{T}$ that represents $f_{i,l}$;
10 $\mathcal{T}.nd.cnt = \mathcal{T}.nd.cnt + 1$, $\hat{v} = \mathcal{T}.nd$;
11 **else**
12 create a new node $\mathcal{T}.nd$ to represent $f_{i,l}$ and insert it in $\mathcal{T}$ as a child node of $\hat{v}$;
13 $\mathcal{T}.nd.cnt = 1$, $\hat{v} = \mathcal{T}.nd$;
14 **end**
15 **end**
16 **end**
17 **for** *each level* l *in vNF-FT* $\mathcal{T}$ **do**
18 **for** *each node* $\mathcal{T}.nd$ *in level* l *of* $\mathcal{T}$ **do**
19 **if** *(*$l \le \xi_{lvl}$*) AND (*$\mathcal{T}.nd.cnt \le \xi_{cnt}$*)* **then**
20 merge the subtree rooted in $\mathcal{T}.nd$ with a proper one whose root is in level $(l+1)$;
21 **end**
22 **end**
23 **end**
24 insert $\mathcal{T}$ in FT;
25 **end**
26 **return**(FT);

Algorithm 2: Expanding vNF-FTs to get MTs

Input: Set of vNF-FTs FT, SNT $G(V, E)$.
Output: Set of MTs MT.

1 $MT = FT$;
2 **for** *each vNF-FT* $\mathcal{T} \in MT$ **do**
3 **for** *each level* l *in vNF-FT* $\mathcal{T}$ **do**
4 **for** *each node* $\mathcal{T}.nd$ *in level* l *of* $\mathcal{T}$ **do**
5 $flag = 0$;
6 **for** *each SN* v *that is a feasible server* **do**
7 **if** v *can carry the vNFs in* $\mathcal{T}.nd$ **then**
8 mark v as the SN for $\mathcal{T}.nd$;
9 update substrate resources in SNT;
10 $flag = 1$;
11 **break**;
12 **end**
13 **end**
14 **if** $flag = 0$ **then**
15 split $\mathcal{T}.nd$ into two small nodes evenly, insert them in level l of $\mathcal{T}$ (next to $\mathcal{T}.nd$), connect subtree of $\mathcal{T}.nd$ to them correctly, and remove $\mathcal{T}.nd$ from $\mathcal{T}$;
16 **end**
17 **end**
18 **end**
19 obtain the shortest path between each pair of SNs that carry adjacent vNFs in vNF-SCs and insert the switches on the paths as nodes in the corresponding MT in MT;
20 **end**
21 **return**(MT);

For each node $\mathcal{T}.nd$ in level l of $\mathcal{T}$, *Lines* 6-13 tries to find a feasible server to embed it. Here, to avoid remapping vNF-SCs to use over-length substrate paths, we only select the feasible server from the racks that are either the source and destination racks of the vNF-SCs using the vNF in $\mathcal{T}.nd$ or their adjacent racks (*i.e.*, the racks that share the same aggregation switch with the source and destination racks). If such a server can be found, we store the server in MT $\mathcal{T}$ as the SN for $\mathcal{T}.nd$ (*Lines* 7-12). Otherwise, we split $\mathcal{T}.nd$ into two new nodes, distribute the vNFs in $\mathcal{T}.nd$ to them evenly, and replace $\mathcal{T}.nd$ with them in MT $\mathcal{T}$ (*Lines* 14-16). The new nodes will be checked in the subsequent iterations of the for-loop that covers *Lines* 4-17. The procedure of obtaining an MT is also explained in Fig. 2. Specifically, according to the vNF-FT and the status of the SNT, we split the node for type-2 vNFs in the second layer of the vNF-FT into two nodes, and map them to *Switch* $S1$ and *Server* a, respectively, while the mapping of other vNFs is not affected by the node split.

When the algorithm proceeds to *Line* 19, every node in a vNF-FT has been mapped to a server in the SNT, and thus its MT has been obtained. Hence, *Line* 19 takes care of the link mapping of all the related VLs, and all the switches, which are on the substrate paths for the VLs, are also recorded in the MT. In the subsequent service upgrade, we can replace these switches with PDP-SWs and deploy vNFs on them. The time complexity of *Algorithm* 2 is $O(|R|^2 \cdot \max_i(N_i + 1) \cdot |V|)$.

C. Upgrading SNT for vNF-SC Provisioning

Finally, *Algorithm* 3 determines how to upgrade the SNT with programmable hardware for vNF-SC provisioning based on the MTs in MT. Note that, *Algorithm* 2 already ensures that each node in an MT stores a vNF and the SN to carry it (*i.e.*, a feasible mapping scheme), and the vNF in a node that is closer to the root of an MT is generally shared by more vNF-SCs. Meanwhile, *Line* 19 in *Algorithm* 2 makes sure that the SN to carry the vNF in a node of an MT can be either a server or a switch. Hence, the idea of *Algorithm* 3 is to check each MT from its root and tries to upgrade the SNs used by it greedily, until the budget of the upgrade has been used up.

Algorithm 3: Upgrading SNT according to MTs

```
Input: Set of MTs MT, set of vNF-SCs R, budget Ω
1  M = ∅, Cost = 0, flag = 0;
2  for each MT T ∈ MT do
3      for each level l in MT T do
4          for each node T.nd in level l of T do
5              if flag = 0 then
6                  if the SN v for T.nd is not in M then
7                      determine the programmable hardware
                       (with a cost φ) used to upgrade vNFs
                       in T.nd based on the type of SN v;
8                      if Cost + φ ≤ Ω then
9                          insert SN v in M and upgrade it
                           with programmable hardware;
10                         Cost = Cost + φ;
11                     else
12                         flag = 1;
13                     end
14                 end
15             end
16             if flag = 0 then
17                 for each vNF-SC uses the vNF in T.nd do
18                     for the vNF-SC, map vNF in T.nd
                       and subsequent vNFs to use the
                       upgraded SN v greedily;
19                     update T and resources in SNT;
20                 end
21             else
22                 break;
23             end
24         end
25     end
26 end
```

Line 1 is for the initialization, where $\mathcal{M}$ stores the SNs selected to be upgraded with programmable hardware, $Cost$ records the current upgrade cost, and $flag$ is the boolean to indicate whether the budget has been used up. The multi-level for-loops from *Line* 2 to *Line* 26 check the MTs to determine the scheme of the upgrade. For a node $\mathcal{T}.nd$ in an MT $\mathcal{T}$, *Lines* 5-15 select the SN for it to upgrade if the budget permits (the budget is allocated to each groups equally). Note that, the type of the SN for $\mathcal{T}.nd$ has already been determined by *Algorithm* 2, and thus the SN will be upgraded with a PDP-SW or a SmartNIC if it is a switch or server, respectively (*Line* 7). Next, *Lines* 17-20 upgrade the provisioning schemes of the vNF-SCs that use the vNF in $\mathcal{T}.nd$ to fully use the newly-upgraded SN v. Note that, for such a vNF-SC, we remap not only the vNF in $\mathcal{T}.nd$ but also the subsequent vNFs to SN v if the resources on SN v are enough, to maximize its utilization and minimize bandwidth usage in the SNT (*Line* 18). The time complexity of *Algorithm* 3 is $O(|R|^3 \cdot \max_i(N_i + 1))$.

IV. Performance Evaluation

A. Small-Scale Simulations

We first use a small-scale DCN that consists of 5 switches and 2 server racks, where each rack includes 2 servers (*i.e.*, each ToR switch connects to 2 servers). The switches and servers are arranged in the way similar to that in Fig. 1. We consider $|M| = 4$ types of vNFs and the processing latency of each type of vNF is within $[150, 300]$ μs. The latency reduction achieved after migrating a vNF from a server to a PDP-SW or a SmartNIC is set as $\delta_m^S \in [75, 150]$ μs or $\delta_m^N \in [40, 75]$ μs, respectively, based on the analysis in [20]. We assume that the propagation delay of each SL is 1 μs. We set the processing capacity of each vNF as $b_m^S = 100$ Gbps, $b_m^N = 10$ Gbps, and $b_m^V = 1$ Gbps, respectively, when it is deployed on a PDP-SW/SmartNIC/server, and the IT resource usage of each type of vNF is within $[1, 4]$ units. The IT resource capacity of an SN is set as $C_v^S = 200$, $C_v^N = 500$, and $C_v^V = 1000$ units.

Each SL between a server-ToR switch pair has a capacity of 1 Gbps, and the capacities of other SLs are selected based on $1:1$ convergence ratio and the SNT topology. Each vNF-SC includes $[2, 4]$ vNFs, its bandwidth demand uniformly distributes within $[1, 10]$ Mbps, and its end-to-end latency requirement is randomly selected from $\{200, 600, 1000\}$ μs with probabilities of $\{0.3, 0.4, 0.3\}$, respectively. The cost of a PDP-SW/SmartNIC is $\phi^S = 100$ or $\phi^N = 10$, respectively.

The simulations consider two scenarios, *i.e.*, adequate budget and insufficient budget, whose budgets are set as $\Omega = 50$ and $\Omega = 200$, respectively. We design a greedy-based heuristic based on first-fit vNF placement and shortest path routing for the initial deployment of vNF-SCs. Based on this heuristic, we develop a benchmark, namely, NFTA, which does not consider the aggregation of SCs, but upgrades SNs in descending order of their resource usages. We also formulate a simple integer linear programming (ILP) model to solve the problem. We use the QoS improvement ($\varpi^{in} - \varpi^{out}$), and the total latency reduction ($\tau^{in} - \tau^{out}$) to evaluate the service upgrade.

Table I shows the simulation results, which indicates that the ILP always provides the largest QoS improvement and longest latency reduction. However, its running time is much longer than that of the two heuristics. Our proposed algorithm (FTA) always outperforms the benchmark (NFTA), and it approximates the exact solutions from the ILP better than NFTA. This is because FTA analyzes the existing vNF-SC provisioning schemes more comprehensively, and realizes service upgrade such that the upgraded SNs can be shared by more vNF-SCs.

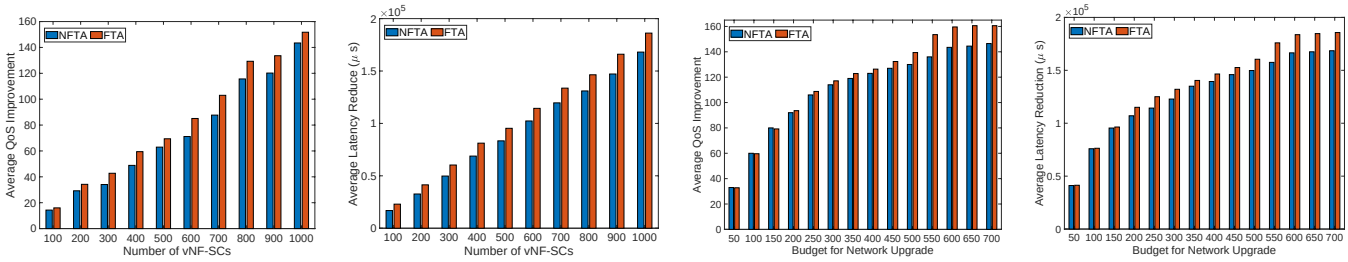
B. Large-Scale Simulations

We then evaluate the performance of FTA and NFTA in a relatively large DCN that consists of 10 switches and 4 server racks. Each rack includes 10 servers, the topology is as that in Fig. 1, while the remaining parameters are unchanged. Note that, compared with real-world DCNs, the size of the DCN considered in the simulations here is still relatively small. However, to reduce the complexity of network management, the operator usually divides its DCN into many point-of-deliveries (PoDs) and manage them in a modular way. Therefore, our simulations can be understood as about the service upgrade for a PoD in a large-scale DCN.

We average the results from 20 independent simulations to get each data point. Fig. 3 illustrates the simulation results. We first compare the QoS improvement and latency reduction from FTA and NFTA in Figs. 3(a) and 3(b), when fixing the budget for service upgrade as $\Omega = 800$. We observe that

TABLE I
PERFORMANCE COMPARISONS OF ILP AND HEURISTICS

Number of vNF-SCs	Budget for Network Upgrade Ω	QoS Improvement ($\varpi^{in} - \varpi^{out}$)			Latency Reduction ($\tau^{in} - \tau^{out}$)			Set of Upgraded SNs			Running Time (Seconds)		
		ILP	FTA	NFTA	ILP	FTA	NFTA	ILP	FTA	NFTA	ILP	FTA	NFTA
60	50	22	13	12	8115	7193	7045	$[7, 8, 9]$	$[7, 9]$	$[7, 8]$	43.31	0.25	0.01
60	200	22	15	12	9234	7465	7134	$[6, 7, 9]$	$[6, 7, 9]$	$[5, 7, 8]$	157.56	0.26	0.01
100	50	39	30	28	14328	11824	11325	$[7, 8, 9]$	$[7, 8, 9]$	$[7, 8, 9]$	105.10	0.33	0.02
100	200	39	34	30	14600	12328	11941	$[6, 8, 9]$	$[6, 7, 8, 9]$	$[5, 7, 8, 9]$	264.33	0.35	0.02



(a) Average QoS improvement (b) Average latency reduction (c) Average QoS improvement (d) Average latency reduction

Fig. 3. Large-scale simulation results.

FTA provides higher QoS improvement and longer latency reduction, and this confirms its effectiveness. Next, we would like to see the algorithms' performance when the number of in-service vNF-SCs is fixed as $1,000$ but the budget increases. The results in Figs. 3(c) and 3(d) indicate that when the budget is small, NFTA and FTA perform similarly, but as the budget increases, the performance gap between FTA and NFTA enlarges gradually. This is because when the budget is small, the flexibility of service upgrade becomes very limited.

V. CONCLUSION

This paper studied how to leverage heterogeneous NFV platforms to optimize the service upgrade of vNF-SCs. Specifically, we considered a DCN as the SNT, and tried to first choose SNs in it to upgrade with SmartNICs/PDP-SWs and then adjust vNF-SC provisioning accordingly. We proposed a novel time-efficient algorithm, to plan the service upgrade under a limited budget such that the QoS improvement on latency reduction is maximized. Simulation results confirmed the effectiveness of our proposals.

ACKNOWLEDGMENTS

This work was supported in part by the NSFC projects 61871357 and 61771445, CAS Key Project (QYZDY-SSW-JSC003), and SPR Program of CAS (XDC02070300).

REFERENCES

[1] P. Lu *et al.*, "Highly-efficient data migration and backup for Big Data applications in elastic optical inter-datacenter networks," *IEEE Netw.*, vol. 29, pp. 36–42, Sept./Oct. 2015.

[2] Z. Zhu, W. Lu, L. Zhang, and N. Ansari, "Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing," *J. Lightw. Technol.*, vol. 31, pp. 15–22, Jan. 2013.

[3] L. Gong *et al.*, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. 836–847, Aug. 2013.

[4] Y. Yin *et al.*, "Spectral and spatial 2D fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks," *J. Opt. Commun. Netw.*, vol. 5, pp. A100–A106, Oct. 2013.

[5] N. Zilberman, "In-network computing," Apr. 2019. [Online]. Available: https://www.sigarch.org/in-network-computing-draft/.

[6] S. Grant, A. Yelam, M. Bland, and A. Snoeren, "SmartNIC performance isolation with FairNIC: Programmable networking for the cloud," in *Proc. of ACM SIGCOMM 2020*, pp. 681–693, Jul. 2020.

[7] K. Zhang, D. Zhuo, and A. Krishnamurthy, "Gallium: Automated software middlebox offloading to programmable switches," in *Proc. of ACM SIGCOMM 2020*, pp. 283–295, Jul. 2020.

[8] J. Liu *et al.*, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Serv. Manag.*, vol. 14, pp. 543–553, Sept. 2017.

[9] C. Sun, J. Bi, Z. Zheng, and H. Hu, "HYPER: A hybrid high-performance framework for network function virtualization," *IEEE J. Sel. Areas Commun.*, vol. 35, pp. 2490–2500, Nov. 2017.

[10] L. Cui *et al.*, "Enabling heterogeneous network function chaining," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, pp. 842–854, Sept. 2019.

[11] L. Dong *et al.*, "On application-aware and on-demand service composition in heterogenous NFV environments," in *Proc. of GLOBECOM 2019*, pp. 1–6, Dec. 2019.

[12] L. Dong, N. L. S. da Fonseca, and Z. Zhu, "Application-driven provisioning of service function chains over heterogeneous NFV platforms," *IEEE Trans. Netw. Serv. Manag., in Press*, 2020.

[13] J. Musovic *et al.*, "The analysis of geometry, coverage and capacity of heterogeneous networks," in *Proc. of MIPRO 2018*, pp. 453–457, 2018.

[14] Y. Hu and T. Li, "Enabling efficient network service function chain deployment on heterogeneous server platform," in *Proc. of HPCA 2018*, pp. 27–39, Feb. 2018.

[15] W. Fang *et al.*, "Joint spectrum and IT resource allocation for efficient vNF service chaining in inter-datacenter elastic optical networks," *IEEE Commun. Lett.*, vol. 20, pp. 1539–1542, Aug. 2016.

[16] M. Zeng, W. Fang, and Z. Zhu, "Orchestrating tree-type VNF forwarding graphs in inter-DC elastic optical networks," *J. Lightw. Technol.*, vol. 34, pp. 3330–3341, Jul. 2016.

[17] Y. Wang, P. Lu, W. Lu, and Z. Zhu, "Cost-efficient virtual network function graph (vNFG) provisioning in multidomain elastic optical networks," *J. Lightw. Technol.*, vol. 35, pp. 2712–2723, Jul. 2017.

[18] L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," *J. Lightw. Technol.*, vol. 32, pp. 450–460, Feb. 2014.

[19] L. Gong, H. Jiang, Y. Wang, and Z. Zhu, "Novel location-constrained virtual network embedding (LC-VNE) algorithms towards integrated node and link mapping," *IEEE/ACM Trans. Netw.*, vol. 24, pp. 3648–3661, Dec. 2016.

[20] J. Sonchack, J. Aviv, E. Keller, and M. Smith, "Turboflow: Information rich flow record generation on commodity switches," in *Proc. of EuroSys 2018*, pp. 1–16, Apr. 2018.