An SLA-Aware Network Function Selection Algorithm for SFCs

Gaurav Garg*, Venkatarami Reddy*, Vanlin Sathya†, Antony Franklin A*, Bheemarjuna Reddy Tamma*
* Indian Institute of Technology, Hyderabad - INDIA, † University of Chicago - USA
Email: {cs16mtech11022, cs17resch01007, antony.franklin, tbr}@iith.ac.in, vanlin@uchicago.edu

***Abstract*—5G is expected to support diverse services with different service requirements. Network Slicing (NS) is a new paradigm which aims to assign different services to a logically dedicated network. A network slice can host multiple services of similar requirements. The traffic flows belonging to a service are processed by a sequence of Virtual Network Functions (VNFs) which form a Service Function Chain (SFC). To utilize the resources efficiently, an operator can share the same VNF instance for multiple SFCs. The CPU utilization of a VNF instance can increase when new flows are accepted. This may increase the VNF delay which can result in the violation of SLA requirements of the existing flows. Therefore, an efficient VNF selection to form SFCs while meeting the SLA requirements is an important aspect that needs to be addressed. In this paper, we address the VNF selection problem for an SFC with the goal of supporting more number of SFCs by considering the dynamic variation of VNF delay. We develop a heuristic algorithm namely PENDANT which considers the effect of CPU utilization on VNF delay and migrates flows to other VNF instances to avoid SLA violations.**

I. Introduction

Among the many visions that 5G has promised, the need for the 5G network to support a diverse set of services is very essential. The legacy one-size-fits-all model fails to address this issue because these networks are not designed to differentiate between the demands of services. Network Slicing (NS) is a new fundamental capability of 5G infrastructure which brings a flexible network architecture to support diverse services and allows multiple virtual networks to be created on the top of a common shared physical infrastructure [1]. A NS can host one or more services of similar requirements. Each traffic flow belonging to a service is processed by a sequence of Virtual Network Functions (VNFs) which forms a Service Function Chain (SFC). To meet the user demands from different locations, an operator may choose to deploy VNF instances of different types across multiple Network Function Virtualization (NFV) nodes.

For any flow with SFC request, the selected VNF's location determines its end-to-end latency which primarily comprises of link delay and VNF delay. VNF delay of any VNF instance is computed as the difference between the timestamps of the packet on the output and input interfaces. After the selection of VNF instances, it is also important to guarantee the SLA requirements for all the flows during their lifetimes. Network traffic passing through a VNF instance changes dynamically depending on the number of flows and their varying traffic rates leading to an increase in VNF delay. This increase in VNF delay is majorly a result of queuing delay which varies with the utilization of physical resources and may cause increase in the end-to-end latency of the flows. To avoid the end-to-end delay violation, traffic flows can be steered through alternative VNF instances to meet the SLA requirements by a process known as flow migration. All the existing approaches [2] [3] on VNF selection overlook the effect of CPU utilization on VNF delay which could incur SLA violations for the existing flows. Therefore, an efficient selection of VNFs to form SFCs while meeting the SLA requirements is an important aspect that still needs to be addressed.

The main contributions of the paper are as follows:

- We show how VNF delay varies with different CPU utilizations for different types of VNFs and formulate the problem of VNF selection to provision SFC requests as a Nonlinear Integer Programming (NLP) model.
- Since the problem is NP-hard [4], we propose a heuristic algorithm, PENDANT based on dynamic programming (DP) approach which considers the variation of VNF delay with its CPU utilization. While choosing any NF instance, PENDANT considers the effect of the already running flows on the instance.
- Through extensive simulations, we show that PENDANT outperforms the existing schemes Shortest Path Service Scheme (SPSS) [2] and Delay-Aware VNF Selection Algorithm for Service Function Chaining (DAVIS) [5] by 30% and 8% of average acceptance ratio, respectively.

II. Related Work & Motivation

A. Related Work

Recently, there have been several works which studied the problem of NF selection in NFV-based telecom networks. The authors of [2] formulated the problem of NF selection and traffic steering as an ILP model and proposed a heuristic algorithm with the objective of maximizing the throughput of SFC requests accommodated in a network. The authors of [6] proposed a VNF chaining and placement model that considers both traffic engineering and network functions virtualization infrastructure cost as the optimization goals. In our previous work [5], we considered the relationship between CPU utilization and VNF delay and proposed an algorithm named DAVIS to maximize the acceptance rate. However, to accommodate a request, DAVIS does not perform flow migrations. Chen et al. [3] build a controller to migrate the flows in NFV/SDN-based environments and design optimal formulation. Differing from previous works, while selecting VNF instances for SFC requests, we also consider the the variation of VNF delay with its CPU utilization.

B. Motivation

To discuss how VNF delay plays a vital role in NF selection, we experiment with three types of VNFs. The three VNFs chosen for the experiment are Snort [7], Pktstat [8] and Nginx [8]. In our experiments, we configured Snort with 30K rules and Nginx is configured to update the port number of packets received at port 80 to port 5001. The VNFs have been configured on a machine with Intel Xeon E5-1650 v4 3.60 GHz CPU, equipped with 32 GB of RAM, 12 cores, and running on Ubuntu 16.04 LTS 6. The VNFs run on VMs, configured with one vCPU and 1 GB of RAM. The background traffic to the VNF is generated using iPerf3 from another machine with similar configuration. A network testing application netperf [9] is used to compute the VNF delay on the VNF instance.

As shown in Figure 1, we observe that the VNF delay of each VNF increases with its CPU utilization. Moreover, the VNF delay also depends on the type of VNF. For example, since NAT reads as well as writes into a packet, the VNF delay of NAT is more than VNF delay of Snort and Pktstat. Therefore, we conclude that the VNF delay of different VNFs vary with CPU utilization, which makes it necessary for an operator to take this variation into account while selecting VNF instances to provision SFC requests.

III. System Model and Problem Formulation

A. System Model

The physical 5G core network is modelled as an undirected graph $G = (V, E)$ where V is the set of servers (nodes) and E is the set of edges which connect the nodes. We assume that all the flows are with SFC requests and each SFC request $r \in R$ is represented by a 6-tuple $(s_r, d_r, lat_r, F_r, b_r, res_r)$ where s_r and d_r are the source and destination nodes, respectively. lat_r is the tolerable end-to-end latency and b_r is the requested bandwidth for the request r. F_r defines the service chain for the request, consisting of L number of VNFs, represented as $F_r = \{f_{r1}, f_{r2}, \ldots, f_{rL}\}$, where $f_{rj} \in F$ is the j^{th} VNF to be traversed by the request r. Multiple instances of VNF type $f \in F_r$ can be deployed on a node $v \in V$. Each request is attributed with a vector of CPU demands that need to be satisfied, denoted by res_r.

B. Nonlinear Integer Programming Formulation

The objective of the proposed Nonlinear Integer Programming (NLP) model is to maximize the total number of accepted

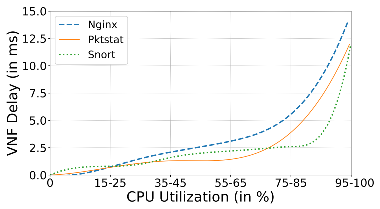


Figure 1: Effect of CPU Utilization (%) on VNF Delay (ms).

Table I: Notation used in the Optimization Model

Notation	Description
$G(V, E)$	Network Topology
R	The set of SFC requests, $r \in R$
lat_r	The end-to-end latency requirement of r
d_e	The latency of link e
d_m	The VNF delay of node m
b_r	The bandwidth of request r
res_r	The CPU resource requirements of request r
res_m	The CPU resources of node m
cap_e	The capacity of link e
Q_r	1, if request r is accepted
x_r^{mfng}	1, if r traverses from VNF g on node n to VNF f on node m
l_r^{mne}	1, if r traverses through link e from node m to node n
z_r^{fm}	1, if r traverses through VNF f on node m

requests, while guaranteeing QoS constraints for each traffic request r.

Eqn. (1) shows the objective function of the optimization model. All the notation used in optimization model is summarized in Table I.

$$\max \sum_r Q_r b_r \; S.T \; Eqn.(1) \; to \; (4)$$

Constraints: For the VNF selection problem, we assume that SFC request r passes through VNF f only once :

$$\sum_{v \in V} z_r^{mf} = 1 \quad \forall f \in F_r, \forall r \in R \tag{1}$$

To guarantee the QoS constraints, we must verify the end-to-end latency requirement :

$$\sum_{e} \sum_{m,n} \sum_{f \in F_r} \sum_{g \in F_r} x_r^{mnug} l_r^{mne} (d_m + d_e) \leq lat_r \qquad \forall r \in R \tag{2}$$

The following Eqns. (3) and (4) assure that the CPU and bandwidth constraints of nodes and physical links are satisfied :

$$\sum_{r \in R} z_r^{mf} res_r Q_r \leq res_m \qquad \forall m \in V \tag{3}$$

$$\sum_{r \in R} x_r^{mfng} z_r^{nf} b_r . Q_r \leq cap_e \qquad \forall e \in E \tag{4}$$

where, Q_r, x_r^{mfng}, l_r^{mne}, y_r^{fg}, z_r^{fm} $\in \{0, 1\}$ are binary decision variables. Since the VNF selection problem is NLP, it cannot be solved in a reasonable time for large networks. Hence, we propose a heuristic algorithm namely PENDANT which is explained in the following section.

IV. Heuristic Algorithm Solution

The abstract of PENDANT algorithm is shown in Alg. 1 which takes $G(V, E)$ and the set of SFC requests R as inputs.

Algorithm 1 : PENDANT Algorithm

Input: $G(V, E)$, $R(r \in R)$
Output: Path P_r for each $r \in$ R and acceptance ratio a_r

```
1:  for each $r \in$ R do
2:      $P_r \leftarrow VNFSelection$(G, $r$)
3:      if $P_r \neq NULL$ then
4:          for each link $e \in P_r$ and VNF node $v \in P_r$ do
5:              Update the available resources
6:          end for
7:      else
8:          Reject the SFC request $r$
9:      end if
10: end for
11: return acceptance ratio;
```

A. VNFSelection function

$PENDANT$ uses $VNFSelection$ function to find the path from source to destination to maximize the number of accepted requests. To find the path between s_r and d_r, we build a multi-stage graph with (L+2) stages. 0^{th} and $(L+1)^{th}$ stages are fixed as s_r and d_r respectively. The elements in each stage represent the nodes which run the VNF type f_L. The detailed description of $VNFSelection$ function is given in Alg. 2.

Algorithm 2 : VNFSelection (G, r)

```
1: Generate multi-stage graph with L stages and ensure that
   the selected node can accommodate $r$ without violating
   the guaranteed latency of the existing flows
2: Find the path $P$ acc. to Eqn. (1) and the corresponding
   constraints to maximize the throughput of the accepted
   requests with the minimum end-to-end delay
3: for each node $i$ in path $P$ do
4:     $P_r \leftarrow$ Delete node $i$ from path $P$ and select other nodes
       of the same VNF type
5:     $var \leftarrow FlowMigration(G, r, P_r)$
6:     if $var = true$ then
7:         return $P_r$
8:     end if
9: end for
```

We use Dijkstra′s algorithm to find the shortest path between the nodes. A node is selected in the path if and only if it can accommodate request r without violating the guaranteed latency of the existing flows. The algorithm retrieves the shortest path according to Eqn. (1) subject to constraints given in Eqns. (2) - (5) by backtracking the optimal states. We invoke $FlowMigration$ function for finding the shortest path P. $FlowMigration$ migrates the existing flows from one instance of the node to another instance on the same node. It returns $true$ if the request r can be accommodated on the path P without violating the guaranteed latency of the existing flows on any of the nodes traversed by P. If it returns false, we delete each node in path P_r and select other VNF nodes of the same VNF type, and if the latency of path P_r is less than the requested latency, we invoke $FlowMigration$ for each path P_r. If $FlowMigration$ returns true for any path P_r, the path is returned to $PENDANT$.

B. $FlowMigration\ function$

When flow r is accommodated on an instance, some existing flows can violate their SLA requirements. Such flows are termed as unsafe flows and rest of the flows are termed as safe flows.

For each node v on path P, the instance with the most number of safe flows is stored in I_{ms}, and the instance with the most number of unsafe flows is stored in I_{mu} (line 2 - line 3). For each flow f provisioned on I_{ms} and I_{mu}, d_f is the difference of current latency of the request from the SLA latency (line 4 - line 6). We select the flows which violate their SLAs from I_{ms} and I_{mu}, sort the flows according to d_f and store them in $unsafe$. Similarly, the flows which do not violate their SLAs are sorted according to d_f (in descending order) and stored in $safe$. After forming sets $unsafe$ and $safe$ from the flows, the algorithm migrates a flow u to $instance_s$ such that $d_s > d_u$ and flows s and u are provisioned on different instances. The intuition here is to migrate the flow which violates its SLA to an instance where no flow gets violated. Therefore, when the new request is provisioned on the instance, none of the existing requests violate their SLAs. Finally, the path is returned to $VNFSelection$ if the achieved latency is within the SLA requirements (line 10 - line 12).

Algorithm 3 : FlowMigration (G, r, P)

```
1: for each node $v$ on path P do
2:     $I_{ms}$ = Instance with most number of safe SFC flows
3:     $I_{mu}$ = Instance with most number of unsafe SFC flows
4:     for each flow $f$ in $I_{ms}$ and $I_{mu}$ do
5:         $d_f$ = $lat_f$ - $lat_{cur}$
6:     end for
7:     $unsafe$ = Sorted set (according to $d_f$) of flows from
       $I_{ms}$ and $I_{mu}$ which violate their SLAs
8:     $safe$ = Sorted set (according to $d_f$, in descending
       order) of flows from $I_{ms}$ and $I_{mu}$ which do not violate
       their SLAs
9:     Migrate u $\in unsafe$ to $instance_s$ and s $\in safe$
       to $instance_u$ s.t. $d_s > d_u$
10:    if any instance on node $v$ does not have enough re-
       sources to provision $r$ then
11:        return false
12:    end if
13: end for
```

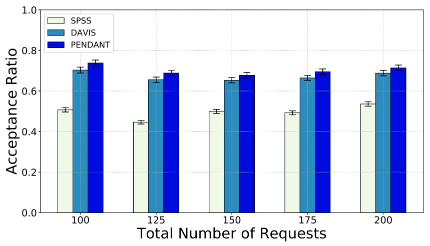
V. Performance Evaluation

In this section, we evaluate the performance of PENDANT on a widely used telecom network topology - RocketFuel with 58 nodes and 108 edges [10]. To evaluate the performance, we have used a simulator developed in C++. Each experiment was repeated 100 times and the obtained results are plotted with 95% confidence interval.

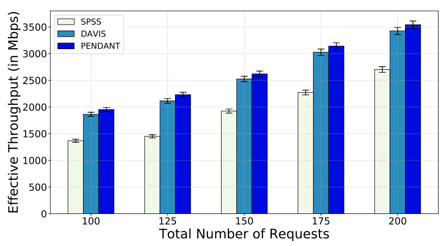
A. Simulation Setup

We test PENDANT on 3 types of VNFs. Each node can host multiple instances of the same VNF type. The type of VNF a node can host is selected randomly. The VNF delays at respective CPU utilizations have been taken from Figure 1. The available capacity at each link is set as 1500 Mbps. According to [11], we configure the nodes and VNF instances with [50 - 200] and [10 - 20] units of CPU resources, respectively. The length of each SFC request is fixed to 3. The end-to-end latency for each request is assumed to be randomly distributed in [20 - 30] ms. Each link is characterized by bandwidth which is randomly selected from [10, 20, 30, 40, 50] Mbps. The CPU resource required by each request is randomly distributed in [1 - 5] units. The link delays are set in the range [5 - 40] ms [10].

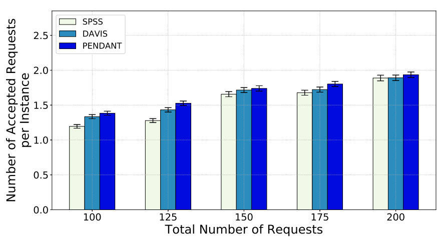
Performance metrics: To evaluate the proposed approach, we consider three performance metrics. *SFC Acceptance Ratio* is calculated as the ratio of accepted number of SFC requests to the total number of SFC requests. *Effective Throughput* is the sum of throughput of all the accepted requests. *Number of Accepted Requests per Instance* is the ratio of total number of successfully provisioned SFC requests to the total number of active instances. This metric signifies the fairness ratio i.e.,



(a) Acceptance Ratio vs No. of SFC Requests.



(b) Effective Throughput vs No. of SFC Requests.



(c) No. of Accepted Requests vs SFC Requests.

Figure 2: Simulation results of RocketFuel topology.

how efficiently the algorithm provisions the requests on the available resources.

Algorithms Compared: We compare the proposed approach PENDANT with two state-of-the-art algorithms. *SPSS* prefers to select a node along the shortest path and does not consider the relationship between CPU utilization and VNF delay. *DAVIS* considers the relationship between CPU utilization and VNF delay. To accommodate requests, the above two algorithms do not migrate the flows from one instance of the node to another.

To show the effectiveness of our solution, we also compare the performance of PENDANT with the optimization model. The optimization model has been solved using GAMS CPLEX solver and since the model takes a long time, we show the results on a small-scale ARPANET [12] topology with 4 nodes and 4 edges. The total number of requests is set as 10 and the results in Table II show that PENDANT gives close results as compared to the optimization model.

Table II: Comparison between Optimization and PENDANT

	Acceptance Ratio	Effective Throughput (in Mbps)
Optimization	0.9	190
PENDANT	0.7	170

B. Simulation Results

Acceptance Ratio: Figure 2a shows the average acceptance ratio achieved by the algorithms on RocketFuel topology. The acceptance ratio of PENDANT is always the highest among the three algorithms. The main reason behind it is that SPSS algorithm provisions the SFC requests based on the available capacities and link delays and does not check whether already provisioned requests violate their SLAs or not. PENDANT provisions the SFC requests considering the current utilization and does not violate other SFCs' SLAs. Thus, PENDANT accepts upto 30% more requests over SPSS and 8% more requests over DAVIS.

Effective Throughput: Figure 2b shows the effective throughput achieved by the three algorithms. We can see that effective throughput increases with the increase in number of SFC requests. The effective throughput is the highest for PENDANT and outperforms SPSS and DAVIS by upto 85% and 11%, respectively.

Number of Accepted Requests per Instance: Figure 2c shows the average rate of SFC requests achieved by the algorithms. The figures show that PENDANT is the most effective in terms of resources used. It means that PENDANT effectively utilizes the given resources to accept more number of requests. This is because PENDANT migrates requests to an already running instance instead of creating a new instance. Hence, it decreases the overall number of instances used to provision the requests by 19% over SPSS and 7% over DAVIS.

VI. Conclusions and Future Work

In this work, we proposed an efficient heuristic algorithm named PENDANT to provision the SFC requests. Given a SFC request, PENDANT selects the nodes while considering their current utilization and steers the traffic through them. We also show that the number of resources used is comparatively less than the existing methods. Extending the algorithm to the scenario where requests can leave the network at any time and release the resources is part of our future work.

Acknowledgements

This work is partially supported by the R&D work undertaken in the project under the Visvesvaraya PhD Scheme of Ministry of Electronics & Information Technology (MeitY), Govt. of India, being implemented by Digital India Corporation and "Converged Cloud Communication Technologies" of MeitY, Govt. of India.

References

[1] NGMN Alliance. Ngmn 5g white paper. 2015.
[2] Shundan Jiao et al. Joint virtual network function selection and traffic steering in telecom networks. In *Proc. of IEEE Conference on GLOBECOM*, pages 1–7, 2017.
[3] C. Sun et al. Enabling nfv elasticity control with optimized flow migration. *IEEE Journal on Selected Areas in Communications*, 36(10):2288–2303, Oct 2018.
[4] A. Fischer et al. Virtual network embedding: A survey. *IEEE Communications Surveys Tutorials*, 15(4):1888–1906, Fourth 2013.
[5] G. Garg, V. Reddy, A. Antony Franklin, and B. R. Tamma. Davis: A delay-aware vnf selection algorithm for service function chaining. In *Proc. of 2019 11th International Conference on Communication Systems Networks (COMSNETS)*, pages 436–439, Jan 2019.
[6] B. Addis et al. Virtual network functions placement and routing optimization. In *Proc. IEEE 4th International Conference on CloudNet*, pages 171–177, Oct 2015.
[7] https://www.snort.org/.
[8] https://linux.die.net/man/1/pktstat/.
[9] https://hewlettpackard.github.io/netperf/.
[10] RocketFuel ISP Topology. https://tinyurl.com/y9f2pp8v. [Online].
[11] Qixia Zhang et al. Adaptive interference-aware vnf placement for service-customized 5g network slices. *Proc. of IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, 2019.
[12] http://www.topology-zoo.org/maps/arpanet196912.jpg. [Online].