Interference Aware Network Function Selection Algorithm for Next Generation Networks

Venkatarami Reddy
Department of CSE
IIT Hyderabad, India
cs17resch01007@iith.ac.in

Gaurav Garg
Department of CSE
IIT Hyderabad, India
cs16mtech11022@iith.ac.in

Bheemarjuna Reddy Tamma
Department of CSE
IIT Hyderabad, India
tbr@iith.ac.in

Antony Franklin A
Department of CSE
IIT Hyderabad, India
antony.franklin@iith.ac.in

***Abstract*—Service Function Chaining (SFC) is used to steer the traffic to a specific set of Network Functions (NFs) (such as load balancer, proxy, firewall, etc.) based on the type of traffic and operator policy. Handling the massive amount of user traffic envisioned in the next generation networks using traditional techniques is costly and tedious. By leveraging advanced technologies such as Network Functions Virtualization (NFV) and Software Defined Networking (SDN), NFs can be deployed as software instances on Virtual Machines (VMs) (also called as Virtual Network Function (VNF)). Network operators widely place different types of VNFs at different locations to meet the user traffic demands. Multiple VNF instances on the same physical server compete for common resources such as network I/O bandwidth, CPU cycles, cache memory, and main memory which can lead to severe performance interference, which is ignored in existing NF selection mechanisms. However, increasing the SFC acceptance rate of SFC requests with an effective selection of required VNFs under the constraint of end-to-end latency is still an open problem. Since this problem is NP-Hard, we propose a heuristic algorithm based on dynamic programming which efficiently selects the required VNFs and steers the traffic by considering the interference effect. Results show that the proposed algorithm improves the average SFC acceptance rate by 29% as compared with existing methods.**

I. Introduction

In traditional networks, all the Network Functions (NFs) (*e.g.,* Deep Packet Inspection (DPI), Firewall, etc.) are implemented in hardware-based middleboxes [1]. The number of devices connected to the Internet is expected to be more than 28 Billion and Internet traffic could reach 4.8ZB per year by 2022 [2]. Because of this rapid increase in data traffic demand, the diverse requirements of users are quite difficult to be satisfied using dedicated hardware. In this respect, telecom operators are migrating to cloud native networks, which can change the way networks are deployed, operated, and also the way customers can enable the services on the fly [3].

Software Defined Networking (SDN) and Network Functions Virtualization (NFV) are two fundamental building blocks of cloud native networks. NFV intends to decouple the NFs like Firewall, NAT, etc., from the proprietary hardware appliances and enables them to run on Virtual Machines (VM) (also called as Virtual Network Functions (VNFs)) on commodity servers. It helps to improve the resource utilization by creating new instances on the fly based on traffic demands. With the help of SDN, which separates the control plane from the data plane of the network, traffic will be managed dynamically and steered to the required VNFs. By using both SDN and NFV, cloud native networks can handle more traffic effectively in a flexible way to meet the dynamic user requirements with less CAPital EXpenditure (CAPEX) and OPerational EXpenditure (OPEX).

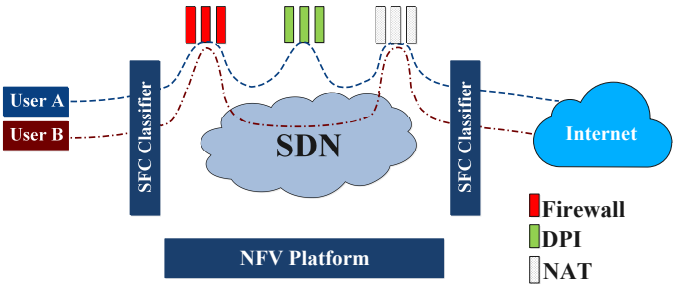


Figure 1: Service Function Chaining: Use Case.

Telecom operators provide a diverse set of services, where each service request needs a set of NFs in a particular order. When VNFs are stitched together in the specified order to fulfill the service requirement, they form a Service Function Chaining (SFC) [4] [5]. Each SFC request has some specific requirements such as throughput and end-to-end latency. An SFC classifier distinguishes the traffic flow based on its type and steers it through the required NFs. As shown in Figure 1, traffic from User A needs to be serviced by NFs Firewall, DPI, and NAT in order. Therefore, whenever SFC Classifier gets the traffic from User A, it steers the traffic to the selected instances of required VNFs in order based on the rules deployed on it.

To save energy and improve resource utilization, an operator may deploy multiple VNF instances on the same server [6]. These co-located VNFs at a server may compete for the shared resources like network I/O bandwidth, CPU cycles, cache memory, and main memory leading to performance interference [7], and eventually increasing the response time of the requests. The performance degradation due to this interference depends on how frequently and what resources are the co-located VNFs competing for. In this paper, the interference delay of any VNF instance is considered as the time difference between the processing delay of the VNF instance when it runs with other co-located VNFs on a server from the delay when it runs exclusively. In [6] [8], the authors show that interference can cause up to 50% of throughput degradation which is quite severe for critical services and it is observed that there exists a linear dependency between throughput degradation and interference delay. So, when mul-

tiple instances of a VNF are available at different servers, selecting an instance considering interference delay becomes important and can have a significant effect on the performance. Existing VNF selection approaches [9] [10] [11] do not take interference delay into account, which could lead to violation of SLA requirements. Therefore, an efficient selection of VNFs to form SFCs by considering link delay, processing delay, and also, interference delay is an important aspect that needs to be addressed. In this paper, we address the problem of VNF selection to provision the SFC requests and traffic steering in telecom networks.

The main contributions are as follows :

- We show how VNF response time varies when it is co-located with other VNFs by conducting real-time experiments.
- With the aim of maximizing the acceptance ratio of SFC requests, we propose an Interference Aware Network Function Selection (IANFS) algorithm which uses the Dynamic Programming approach and considers the expected VNF interference delay based on the co-located VNFs, along with other delays.
- To show the efficiency of our algorithm, through simulations we compare IANFS with an existing algorithm named Latency-aware SFC Steering (LSFCS) [9] which selects the VNFs based on available capacity of link and VNF instances without considering interference effect.
- The results show that IANFS outperforms the existing scheme LSFCS by 29% for the metric average acceptance ratio of SFC requests.

II. Related Work & Motivation

In this section, we first review existing works on VNF selection and VNF interference. Then we present motivation behind the proposed work.

A. Related Work

VNF Selection: Several works study the problem of placement and selection of VNFs to provision the SFC request by considering different metrics and scenarios. In [12], the authors introduced a flexible resource allocation model to meet the strict QoS requirements of different services by allocating extra resources dynamically to the already running VNF instances instead of creating new ones. The proposed model considered the linear dependency between resources allocated to a VNF instance and its processing delay. To minimize the bandwidth consumption, joint topology design and SFC mapping for Telco clouds is explored in [13]. Shundan et al. [9] formulated the VNF selection and traffic steering problem as an ILP and proposed a heuristic algorithm which maximizes the traffic throughput based on the available capacity of link and VNF instances under the constraint of end-to-end latency.

VNF interference: NFV provides more flexibility and agility of VNF creation, but its performance depends not only on configuration of the server but also on the type of its co-located VNFs. In [6] and [14], the authors have studied the problem of performance interference among various VNF instances and investigated the root cause for this interference concerning different resources like CPU, cache, network I/O. Proctor [7] presented a real-time performance monitoring infrastructure based on online statistical learning approach which can detect performance intrusion and identify the root cause of service violation. When migrating a VM from one server to another, iAware [15] chooses a server node with the least interference effect among the co-located and migrating VMs.

Differing from previous works, while selecting VNF instances for SFC requests, we also consider the interference effect due to co-located VNF instances in the same server node. We propose a heuristic algorithm to achieve the high SFC acceptance rate.

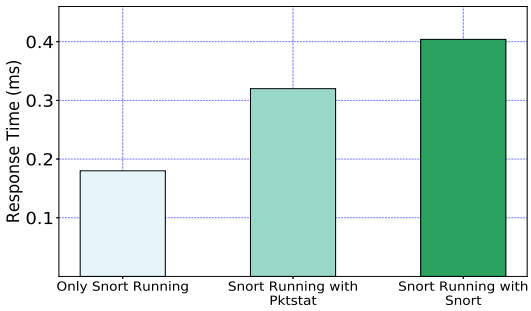


Figure 2: Motivational plot demonstrating the effect of VNF with co-located VNF on response time.

B. Motivation

To discuss how performance interference plays a vital role, we experiment with two VNFs. The two VNFs chosen for experimentation are Snort [16] and Pktstat [17]. Snort is an open-source intrusion detection system which analyzes real-time traffic and performs protocol analysis and content searching. Snort is configured with ten thousand detection rules, and it sends an alert message to the system when an spurious packet is detected. In this way, Snort can read packet up to application layer. Similarly, Pktstat displays a real-time summary of packet activity and can read the packet header up to the transport layer. We use two servers in our experimental setup. Snort and Pktstat are run in one server, called VNF server. Another server is utilized as a traffic server, which generates iperf traffic and sends it to the VNF server. We use Wireshark to calculate the time taken by a packet to get serviced by the VNF. VNF server and traffic server are configured on a machine with Intel Xeon E5-1650 v4 3.60 GHz CPU with 12 cores, equipped with 32 GB of RAM, and running on Ubuntu 16.04 LTS 64-bit OS.

As shown in Figure 2, we observe that the response time of Snort increases when it runs with other VNFs. It is worth noting that the response time of Snort is less when it runs alone than when it runs along with Pktstat.

Therefore, we conclude that the co-located VNFs have an impact on its performance and this makes it very necessary for an operator to take this performance interference into account while placing and/or selecting VNF instances.

III. System Model and Problem Statement

A. System Model

The service provider network is modelled as an undirected graph G = (V, E), where V is the set of servers (nodes) which are connected to switches and E is the set of edges which interconnect the switches. Each server node is capable of hosting multiple instances of different VNFs based on its capacity. These instances are represented as a set K (*e.g.,* K ∈ {NAT, Firewall, DPI}). Every VNF is characterized by a value based on the resources allocated to it called as throughput or capacity of the VNF which represents how many bits the VNF can process per second. To provide a diverse set of services to customers, an operator of network G offers d different types of VNFs. Let F = $\{ f_1, f_2, f_3, \ldots, f_d \}$ be the set of different VNFs. Multiple instances of the same VNF are deployed at different server nodes to achieve reliability and to load balance the traffic from different locations.

Each service request s_m is represented as a quintuple $(src_m, dst_m, tol\_lat_m, sc_m, b_m)$, where src_m and dst_m are the source and destination nodes, respectively. tol_lat_m is the tolerable end-to-end latency of the SFC request. sc_m is the service chain of the request, and b_m is the bandwidth requirement of the SFC. sc_m consists of L number of VNFs which are interconnected in sequence, represented as sc_m=$\{f_{m1}, f_{m2}, \ldots, f_{mL}\}$, where $f_{mj} \in$ F is j^{th} VNF in sc_m. We assume that the VNFs in each SFC are unique, and various service requests can share the same VNF instance.

To verify if end-to-end latency requirements of SFC requests are satisfied, we model latency contributions as follows: (i) processing delay(d_{sf}) is the time taken to process the request at VNF f;(ii) interference delay(d_{if}) arises because of co-located VNFs present at the same node competing for shared resources;(iii) link delay(d_{pl}) is time taken to transfer a packet from one node to another. All together, the end-to-end latency (d_{es}) of an SFC request can be calculated as follows:

$$d_{es} = \sum_{f \in sc_m} (d_{sf} + d_{if}) + \sum_{l \in path_{sc_m}} d_{pl}$$

B. Problem Statement

The considered network comprises of a set of switches and a centralized controller where the orchestrator is located. The controller manages these switches and deploys new flow rules whenever needed. The compute nodes are connected to the switches on which different VNF instances can run. The value pair written at each VNF denotes [processing delay (ms), interference delay (ms)]. As shown in Figure 3, [4, 2] at VNF A1 means that A1 takes processing delay of 4 ms and suffers interference delay of 2 ms because of the presence of co-located VNF D1 on the same node. Moreover, each edge e ∈ E which connects two nodes is associated with a link delay. After the selection of required VNFs for any SFC request by the orchestrator, the controller deploys new rules to the switches, which helps to steer the traffic flows.

We illustrate our problem statement with an example given in Figure 3. The SFC requests are provided as input to the

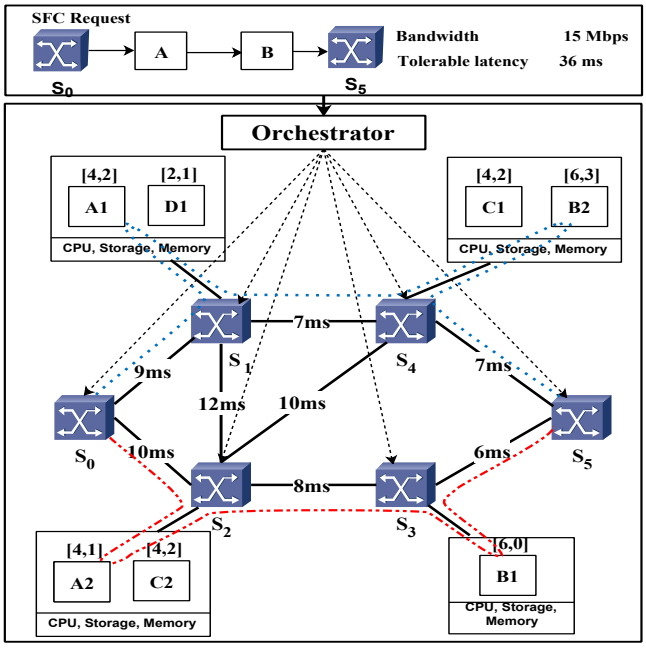


Figure 3: An example of NF selection for SFC provisioning.

orchestrator. Our proposed algorithm, IANFS, is placed at the orchestrator which takes SFC requests as input and effectively selects the required VNFs to provision it with the constraint of end-to-end latency. In the considered example, the input SFC request s_m is (S_0, S_5, 36ms, {A, B}, 15Mbps) and we assume that each VNF instance in the network has sufficient capacity to satisfy the request. As shown in Figure 3, the shortest path scheme prefers to choose a path with the minimum end-to-end latency without considering interference effect. Thus, it selects the path $S_0 \rightarrow S_1 \rightarrow S_4 \rightarrow S_5$ with the end-to-end latency of 33 ms. Although the end-to-end latency is less than the tolerable latency, the path chosen is unreliable because the selected VNFs suffer from performance interference of co-located VNFs. When interference delay is also considered, the end-to-end latency reaches 38 ms which is more than the tolerable latency. Hence, the request should not be accepted. However, the proposed IANFS scheme (explained in Section IV) chooses the path $S_0 \rightarrow S_2 \rightarrow S_3 \rightarrow S_5$ with the end-to-end latency of 34 ms which considers interference delay also at the time of VNF selection.

IV. Heuristic Algorithm for VNF Selection

The primary objective here is to design an algorithm which efficiently selects the required VNFs to provision the SFC requests. The algorithm considers different delays which come across in the chosen path like link delay, processing delay, and interference delay under the constraint of end-to-end latency. Since this problem is NP-Hard [9], we propose a heuristic algorithm named IANFS based on dynamic programming to achieve the objective.

Algorithm 1 provides an overview of the proposed approach IANFS. It takes network topology G = (V, E) and a set of SFC requests as inputs and returns acceptance rate Υ as output. We first initialize Υ (line 1) and the number of accepted requests μ (line 2) to zero. The total number of SFC requests

Algorithm 1 IANFS Algorithm

Input: $G(V, E)$, S

Output: Path P_s for each $s_m \in$ S and Acceptance Ratio.

1: $\Upsilon \leftarrow$ 0; /* Acceptance ratio */
2: $\mu \leftarrow$ 0; /* Number of accepted SFC requests */
3: $\lambda \leftarrow \Sigma_s$ s ; /* Total number of SFC requests */
4: **for** each $s_m \in$ S **do**
5: $P_s \leftarrow$ SFPusingDP(G, s_m);
6: **if** $P_s \neq$ *null* **then**
7: $\mu \leftarrow \mu$ + 1 ;
8: Provision the SFC request s;
9: Update the link and VNF capacities in path P_s
10: **end if**
11: **else**
12: Reject the SFC request s;
13: **end for**
14: $\Upsilon \leftarrow \mu/\lambda$;
15: **return** Υ;

is stored into a variable λ (line 3). For each SFC request s_m, it finds the efficient path which includes the required VNFs with minimum delay by calling the function named *SFPusingDP*. This function returns a path as output if it finds the path for the input request which satisfies the end-to-end latency constraint. Otherwise, it returns *null* as output which means that none of the paths satisfies the input constraints. After processing all the SFC requests, the IANFS algorithm updates the acceptance ratio Υ.

A SFC request contains L number of VNFs $\{f_1, f_2, \ldots, f_L\}$ (in order), starting from the source src_m and ending at the destination dst_m. To find the path, we build an L-stage graph from G based on the ordered set of VNFs in input SFC request, where L is the length of the input request. There are L+2 stages in total, where the source and destination are fixed at 0^{th} stage and $(L+1)^{th}$ stage, respectively. The required VNF instances may reside in multiple nodes. There are L stages between source VNF and destination VNF where each stage contains the VNF instances of that particular VNF running. For example, the number of nodes in stage L is equal to the number of instances of VNF f_L running in the network. The shortest link delay between each stage is computed using the $Dijkstra$ algorithm.

Algorithm 2 provides the details of *SFPusingDP*(G, s_m). In this function, we use different arrays to store different considered delays. We denote the processing delay of VNF f as $pd[f]$, link delay between node x and y as $ld[x][y]$, and interference delay of VNF f on node x as $id[x][f]$. We first initialize the parameters (line 1-line 4) and then remove the nodes and edges where capacity of the required VNF is not sufficient along with its associated edges. After building the L-stage graph, the algorithm finds the path with minimum end-to-end latency from source src_m (stage 0) to destination dst_m (stage L+1) and stores the delay into the variable *calc_delay*(from line 6 to line 17). p_node stores the considered nodes in each stage to get the path with minimum delay. If the *calc_delay* satisfies the requirement of s_m, then the algorithm finds the path by using the array p_node and returns. Otherwise, the algorithm will return *null* as output which indicates that the input SFC request is rejected.

Algorithm 2 SFPusingDP(G, s_m)

Input: $G(V, E)$, s_m

Output: Path for request s_m.

1: calc_delay $\leftarrow$ 0; /* Minimum calculated delay*/
2: d $\leftarrow$ INFINITE; /* A 2-D array to store minimum delay at each stage */
3: p_node $\leftarrow$ 0; /* A 1-D array used to store selected node information on previous stage */
4: path $\leftarrow$ 0; /* A 1-D array to store the selected VNFs locations */
5: G' is the resultant graph after removing nodes and edges which do not have sufficient capacity from G;
6: Build an L-stage graph from G' where L is the length of the SFC;
7: **for** i $\leftarrow$ 1 to L+1 **do**
8: **for** each node m in stage i **do**
9: **for** each node n in stage i-1 **do**
10: **if** (d[n][i-1] + ld[m][n]) $<$ d[m][i] **then**
11: d[m][i] $\leftarrow$ (d[n][i-1] + ld[m][n] + pd[m] + id[m][i]);
12: p_node[m][i]$\leftarrow$ n;
13: **end if**
14: **end for**
15: **end for**
16: **end for**
17: calc_delay $\leftarrow$ d[dst][L+1];
18: **if** calc_delay $\leq tol\_lat_m$ **then**
19: **for** i=L to 1 **do**
20: path[i] $\leftarrow$ p_node[path[i+1]][i];
21: **end for**
22: **return** path;
23: **else**
24: **return** $null$;
25: **end if**

The computational complexity of our algorithm depends upon the number of stages and the maximum number of nodes in any stage. If S_{max} is the maximum number of nodes in any stage, then its computational complexity is *O(L * S^2_{max})*.

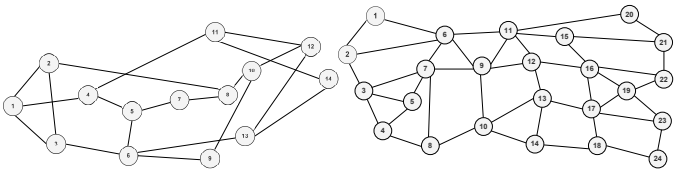
V. Performance Evaluation

In this section, we evaluate the performance of the proposed algorithm on two different well known network topologies which are 14-node, 21-link NSFNET network and 24-node, 43-link USA backbone IP network(USNET), as shown in Figures 4a and 4b. To evaluate the performance, we develop a C++ based simulator. Each experiment is repeated 100 times and results are plotted with 95% confidence interval.

A. Simulation setup

In our simulations, we choose four different types of VNFs. To show the interference effect, we deploy two different VNF instances at each compute node which are chosen randomly. The available capacity of each node is represented by a single value instead of explicitly differentiating the type of resources like CPU, memory, and storage. The source and destination of each SFC request is uniformly distributed among all node pairs. The number of VNFs in each SFC request is normally

distributed between 1 and 4. The definitions of end-to-end latency and requested bandwidth are considered from NGMN white paper [18]. The range of processing delay of each VNF is considered from [12]. The range of interference delay based on the combination of co-located VNF is taken from [6]. All the parameters assumed for simulations are shown in Table I.



(a) NSFNET Topology (b) USNET Topology

Figure 4: Network topologies used in the study.

Performance metrics: To evaluate the proposed approach, we consider acceptance ratio and effective throughput as the performance metrics. We compare these metrics by running the existing approach LSFCS [9] and the proposed approach, IANFS. The LSFCS approach provisions SFC requests based on available capacity of links and VNF instances without considering the interference effect. Acceptance ratio is calculated as the ratio of the number of successfully provisioned SFC requests which satisfied the end-to-end latency requirement to the total number of SFC requests. Effective throughput is the sum of throughputs of all the accepted requests.

Table I: Simulation Parameters.

Parameter	Range
Length of SFC requests	1 - 4
Requested bandwidth	10 - 50 Mbps
End-to-End latency	90 - 110 ms
Link delay	15 - 25 ms
Processing delay	5 - 10 ms
Interference delay	1 - 5 ms
Link Capacity	600 - 1800 Mbps
Node Capacity	600 - 1800 Mbps

B. Simulation Results

Acceptance Ratio vs Service Arrivals: Figures 5a and 6a present the average rate of accepted SFC requests achieved by both algorithms on NSFNET and USNET topologies, respectively, where the length of SFC is fixed as 3. From these figures, we can observe that the acceptance ratio decreases as the number of SFC requests increases due to insufficient resources. However, IANFS always achieves better acceptance ratio than the LSFCS algorithm. The main reason behind it is that the LSFCS algorithm provisions the SFC requests based on the available capacity of link and VNF instances without considering the interference effect. After adding interference delay for each selected VNFs in the chosen path using LSFCS, the total delay may exceed the tolerable end-to-end latency. Thus, LSFCS cannot accommodate those SFC requests. Thus, IANFS algorithm achieves a higher average acceptance ratio, and it accepts up to 29% more SFC requests over LSFCS.

We also show the comparison of effective throughputs for both these algorithms. From Figures 5b and 6b, we see that effective throughput increases with an increase in the number of SFC requests. Similar to the acceptance ratio, effective throughput obtained by IANFS is always higher than LSFCS approach. Since effective throughput is based on the number of accepted requests and the number of accepted requests increases with the total number of SFC requests, effective throughput also increases.

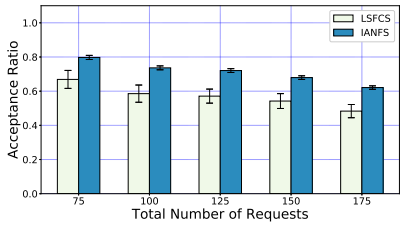
Acceptance Ratio vs Length of SFC: Figures 5c and 6c depict the average rate of accepted SFC requests with respect to length of SFC which varies from 1 to 4 where the number of SFC requests is fixed at 100. The VNFs for each SFC request are selected randomly. The results show that as the length of the SFC increases, the acceptance ratio decreases. When the length of SFC is either 1 or 2, we can see that almost all the requests are accepted. Compared to LSFCS, IANFS increases the acceptance ratio up to 45% on average. It is because of the same reason as explained previously. Similarly, Figure 5d shows how effective throughput changes with different SFC lengths for NSFNET respectively.

Results with varying link capacity: Figure 5e shows the impact of link capacity on SFC acceptance ratio. In this experiment, we assumed the VNF instance capacity as 1200 Mbps and the number of SFC requests as 100. As shown in Figure 5e, the acceptance ratio increases with the varying link capacity. Compared with LSFCS, IANFS increases acceptance ratio up to 26% on average. It shows that IANFS performs best with the variation of link capacity for both topologies. The trend observed for USNET topology is very similar to NSFNET topology.

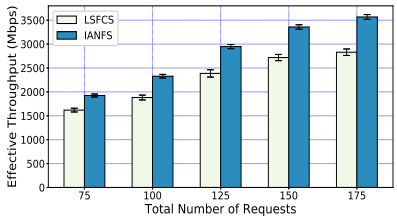
Results with varying VNF instance capacity: Figure 5f shows the impact of VNF instance capacity on SFC acceptance ratio for NSFNET topology. In this simulation, we assumed the link capacity as 1500 Mbps and the number of SFC requests as 100. As shown in Figure 5f, the acceptance ratio increases with the increase of VNF instance capacity. Compared with LSFCS, IANFS increases acceptance ratio up to 34% on average. It shows that IANFS performs better with the variation of VNF instance capacity for both the topologies.

VI. Conclusions

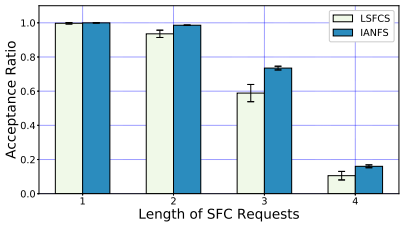
In this work, we have proposed an efficient Interference-Aware Network Function Selection Algorithm (IANFS) to provision the SFC request and steer the traffic. Given the SFC request, the proposed IANFS selects the required VNFs using dynamic programming approach with the constraint of meeting end-to-end latency. In this process, we have also considered interference delay into consideration along with the other delays, i.e., VNF processing delay and link delay. The interference delay arises because of co-located VNFs at the same compute node which was not considered in existing works and it could play a significant role in many delay sensitive 5G use cases. The simulation results showed that the proposed IANFS improves the average acceptance rate by 29% as compared with existing methods. As part of future work, we plan to consider the variation of interference effect when load of the co-located VNF changes and evaluate



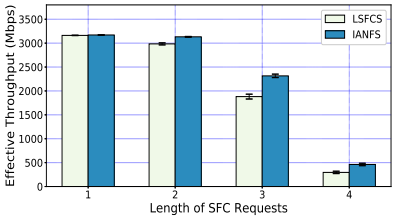
(a) Acceptance Ratio vs Service Arrivals.



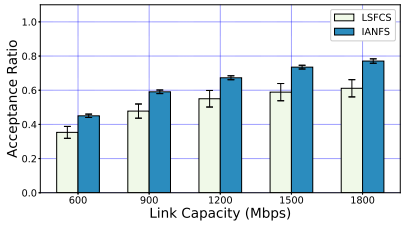
(b) Effective Throughput vs Service Arrivals.



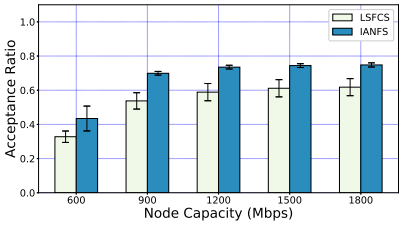
(c) Acceptance Ratio vs SFC Length.



(d) Effective Throughput vs SFC Length.

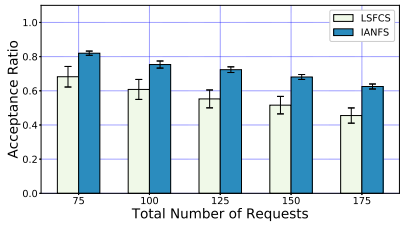


(e) Acceptance Ratio vs Link Capacity.

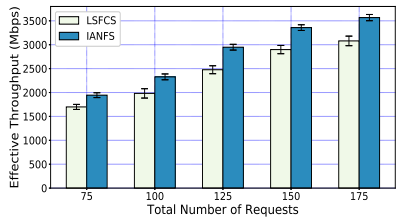


(f) Acceptance Ratio vs Node Capacity.

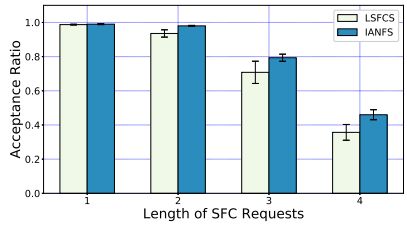
Figure 5: Simulation results of NSFNET topology.



(a) Acceptance Ratio vs Service Arrivals.



(b) Effective Throughput vs Service Arrivals.



(c) Acceptance Ratio vs SFC Length.

Figure 6: Simulation results of USNET topology.

our approach on an experimental setup using LTE/5G-Core modules (OAI) and open source NFV MANO (OSM).

ACKNOWLEDGEMENT

This work was partially supported by the projects "Visvesvaraya PhD Scheme" and "Converged Cloud Communication Technologies", MeitY, Govt. of India.

REFERENCES

[1] Rashid Mijumbi et al. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1):236–262, 2016.

[2] Cisco Visual Networking Index: Forecast and Trends, 2017–2022.

[3] Sameerkumar Sharma et al. A cloud-native approach to 5g network slicing. *IEEE Communications Magazine*, 55(8):120–127, 2017.

[4] Joel Halpern and Carlos Pignataro. Service Function Chaining (SFC) architecture. Technical report, 2015.

[5] C Zhang et al. L4-l7 service function chaining solution architecture. *Open Networking Foundation, ONF TS-027*, 2015.

[6] Chaobing Zeng et al. Demystifying the performance interference of co-located virtual network functions. In *Proc. of IEEE Conference on INFOCOM*, pages 765–773, 2018.

[7] Ram Srivatsa Kannan et al. Proctor:detecting and investigating interference in shared datacenters. In *Proc. of IEEE Conference on Performance Analysis of Systems and Software(ISPASS)*, pages 76–86, 2018.

[8] Tootoonchian Amin et al. Resq: Enabling slos in network function virtualization. In *Proc. of 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[9] Shundan Jiao et al. Joint virtual network function selection and traffic steering in telecom networks. In *Proc. of IEEE Conference on Global Communications (GlOBECOM)*, pages 1–7, 2017.

[10] Ahmed M Medhat et al. Orchestrating scalable service function chains in a nfv environment. In *Proc. of IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5, 2017.

[11] Ahmed M Medhat et al. Near optimal service function path instantiation in a multi-datacenter environment. In *Proc. of IEEE Conference on Network and Service Management (CNSM)*, pages 336–341, 2015.

[12] Abdelhamid Alleg et al. Delay-aware vnf placement and chaining based on a flexible resource allocation approach. In *Proc. of IEEE Conference on Network and Service Management (CNSM)*, pages 1–7, 2017.

[13] Cao Ye, Zilong et al. Joint topology design and mapping of service function chains for efficient, scalable, and reliable network functions virtualization. *IEEE Network*, 30(3):81–87, 2016.

[14] Junhee Park et al. Performance interference of memory thrashing in virtualized cloud environments: A study of consolidated n-tier applications. In *Proc. of IEEE Conference on Cloud Computing (CLOUD)*, pages 276–283. IEEE, 2016.

[15] Fei Xu et al. iaware: Making live migration of virtual machines interference-aware in the cloud. *IEEE Transactions on Computers*, 63(12):3012–3025, 2014.

[16] https://www.snort.org/.

[17] https://linux.die.net/man/1/pktstat/.

[18] NGMN Alliance. Ngmn 5g white paper. 2015.