Virtual Network Function Placement

A Report Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Bachelor of Technology

in

Information Technology

by

Group IT-24

to the

Computer Science and Engineering Department

Motilal Nehru National Institute of Technology

Allahabad, Prayagraj, Uttar Pradesh

28 May, 2021

UNDERTAKING

We declare that the work presented in this report titled "*Virtual Network Function Placement*," submitted to the Computer Science and Engineering Department, Motilal Nehru National Institute of Technology Allahabad, Prayagraj, for the award of the *Bachelor of Technology* degree in *Information Technology*, is our original work. We have not plagiarized or submitted the same work for the award of any other degree. In case this undertaking is found incorrect, we accept that my degree may be unconditionally withdrawn.

Sachin Yadav (20178023)
Preyash Gupta (20178072)
Awadhesh Kumar (20178095)

28 May, 2021

MNNIT Allahabad,
Prayagraj

CERTIFICATE

Certified that the work contained in the report titled "*Virtual Network Function Placement*," by *Sachin Yadav (20178023), Preyash Gupta (20178072) and Awadhesh Kumar (20178095),* has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. Shashwati Banerjea

Computer Science and Engineering Dept.

M.N.N.I.T Allahabad, Prayagraj

28 May , 2021

Preface

A good B.Tech. thesis is one that helps you in furthering your interest in a specific field of study. Whether you plan to work in an industry or wish to take up academics as a way of life, your thesis plays a key role.

Your thesis should judiciously combine theory with practice. It should result in a realization of a complex system (software and/or hardware). Given various limitations, it is always better to extend your predecessor’s work. If you plan it properly, you can really build on the experience of your seniors.

Acknowledgements

Foremost, we would like to express our sincere gratitude to our mentor *Dr. Shashwati Banerjea* for the continuous support of our study and research while making this project, for his patience, motivation and enthusiasm and immense knowledge. His guidance helped us in all the time of research and writing of this thesis. We could not have imagined a better advisor and mentor for our mini project research.
We are thankful that we have an amazing team and for all the stimulating discussions and for all the experience and knowledge that we have gained by working together. We would also like to thank our parents and all our friends that have motivated and for never doubting our ability to complete this project.

Contents

Introduction

As we all know, Traditional networks are filled with a massive and ever-growing variety of network functions coupled with dedicated hardware devices, which leads to difficulty in network management. Network Function Virtualization is a paradigm to change such a situation by decoupling network functions from the underlying dedicated hardware and moving them to virtual servers. This kind of decoupling introduces many benefits which include reduction of Capital and Operational Expense. In this project work, we intend to solve a Virtual network function placement problem. Although there are many challenges with network function virtualization (see Appendix A for details).

Currently, most traditional networks are full of various proprietary hardware appliances, which are also called middle-boxes such as security system, Network Address Translator (NAT) and many more. A given service has a strong connection with some specific middle-boxes. For example, launching a new service needs to deploy a variety of middle-boxes and accommodating these middle-boxes is becoming increasingly difficult. In addition, designing dedicated hardware-based protocols and deploying that hardware is extremely hard and time-consuming. Moreover, with the ever-increasing and various service requirements, service providers must scale up their physical infrastructure periodically, which directly leads to high capital expenses and operational expenses.

According to ETSI (European Telecommunication Standard Institute), the idea of NFV (Network Function Virtualization) is recognized as a network architecture which transforms the way of building and operating networks by leveraging standard IT (Information Technology) virtualization technologies and consolidating dedicated hardware-based network functions into standard commercial devices. Instead of installing expensive dedicated Hardwares, network service providers can purchase switches, storage, and servers to run virtual machines that perform network functions. This merges multiple functions into a single physical server and reduces cost and expenses. If a customer wants to add a new network function in network infrastructure, the network service provider can simply spin up a new virtual machine to perform that function. For instance, instead of deploying a new hardware appliance across the network to enable network encryption, encryption software can be deployed on a standardized server or switch already in the network.

This virtualization of network functions decreases dependency on dedicated hardware appliances for network operators, and results in improved scalability and customization across the entire network.

NFV Motivation

In today's networks, there are a lot of middle-boxes (i.e., dedicated hardware appliances). These indicate the forwarding or processing devices that transmits, transforms, filters, inspects, or controls network traffic for the purpose of network control and management. Hence, they are important elements to support traditional network services. Typical examples of middle-boxes include NATs that modify packet source and destination addresses, and firewalls that filter unwanted or malicious traffic, etc. To give an overall view, we summarize the commonly used middle-boxes (i.e., routing/forwarding devices, NAT, Wide Area Network (WAN) optimizer, proxy, firewall, Flow Monitor (FM), Intrusion Detection System (IDS), Intrusion Prevention System (IPS), Deep Packet Inspection (DPI), etc) in Table1.
However, today the categories of middle-boxes are far more than what is mentioned here.

Middle-boxes	**Action**
Routing/Forwarding Device	Routing table update and data forwarding
Load Balancer	Address rewrite or traffic reroute
Network Address Translator	IP address assignment or translation
Security system	Allow, bypass, deny and log only
Proxy	Mapping, rewrite and routing
WAN Optimizer	Shaping, dropping, priority alteration
Flow monitor	Monitoring, capturing, logging
Intrusion Detection/Prevention system	Monitoring, logging, reporting or blocking
Deep Packet Inspection	Packet classification and content inspection

Table-1

Due to the variety of requirements on network services, the number of middle-boxes is increasing constantly. Each middle-box appears as a solution for a specific purpose. For example, a middle-box of an IP (Internet Protocol) firewall is required, if a computer wants to be protected from certain threats. However, before applying them into use, these middle-boxes must be integrated into network infrastructure via a complex deployment process which requires not only a lot of manual work of the technically trained person but also a long deployment cycle. In this way, such

purpose-built middle-boxes are certain to cause many issues overall. Firstly, since middle-boxes are standalone and closed, they naturally introduce new failures when they crash, and the diagnosis for these failures and some misconfigurations would be rather complex. On one hand, the network operators need to pay for the purchase of new middle-boxes and the maintenance of old middle-boxes, which increase both the Capital expenditure and Operational expenditure. On the other hand, these new and old middle-boxes do not share the same underlying hardware even with enough available capabilities. Thus, coordinating these middle-boxes is another complicated process that takes a long time and excessive cost of employing a technically trained person.
Apart from the increasing number of middle-boxes, it is known that they are fixed to somewhere in the network and cannot be moved or shared easily. Hence, the network becomes increasingly permanent and inflexible. This situation is even worse with the increasing number of network services.

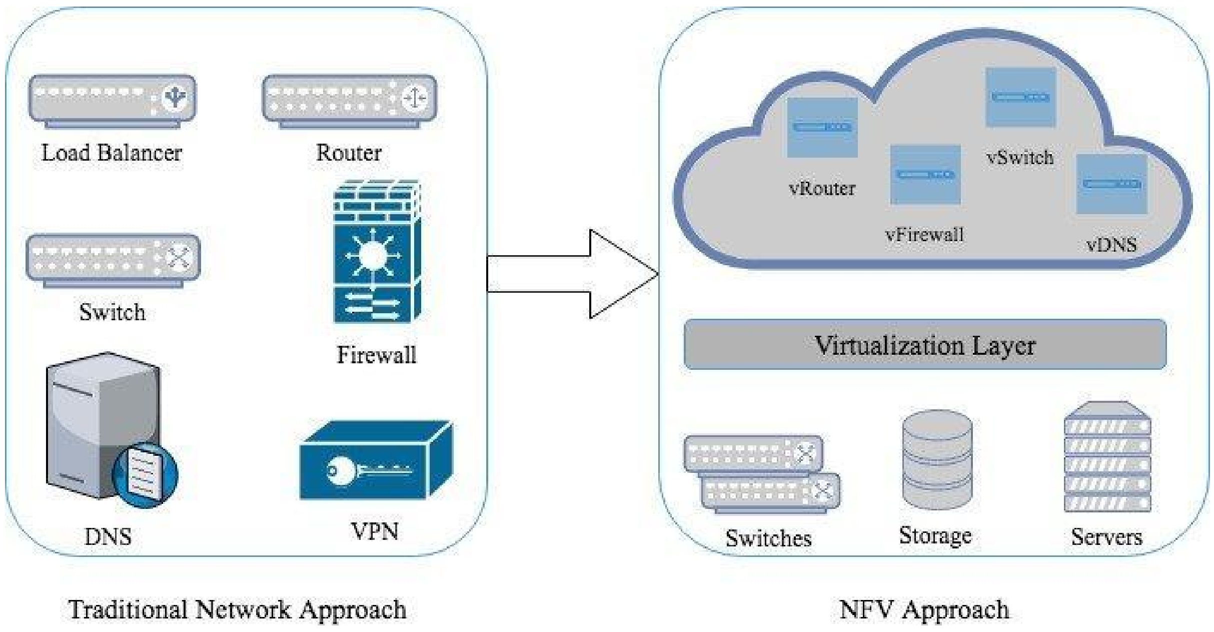


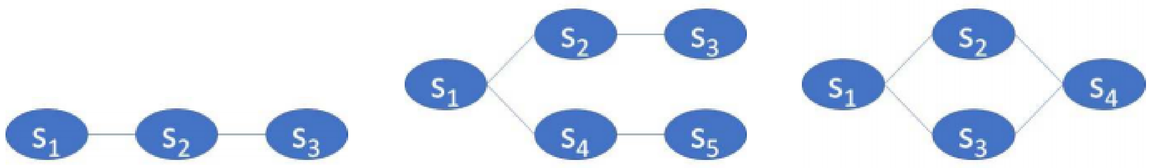
Figure-1

Related Work

Network function virtualization is intended to abstract the underlying physical network function and finally implement them in the form of software (VNF). VNFs can provide the network functionalities originally provided by proprietary network devices.
In a Network Functions Virtualization, a virtualized network function (VNF) is responsible for handling specific network functions that run in virtual machines on top of the hardware networking infrastructure, which means routers, switches, servers etc.
Individual virtualized network functions can be chained to deliver full-scale networking communication services. These chained VNFs can help increase network scalability and agility, while also offering better use of network IT infrastructure resources. Some other benefits include reducing power consumption and increasing security and physical space. This brings the concept of chaining of virtual network functions called as service Function chaining .

Service Function Chaining

We have seen there are so many network functions which are underlying on dedicated hardware. Most of these network functions can be virtualized. We should also understand that virtual network functions should execute in specific order in End-to-End network service. A network service is a complete end-to-end functionality offered to users by network operators. It usually consists of several Service functions (SFs) chained in order. For example, in a “Network protection” service, required service functions may be firewall, DPI and virus scanner. In a formal definition, each service function (also called network function), is responsible for specific processing on the received packets. By definition, a service function chain is a sequence of service functions that must be executed in a determined order as the result of classification or policy and Service function chaining (SFC) is the mechanism of building service function chains and forwarding packets/flows through them. A service function chain is the representation of a service consisting of an ordered set of network functions. The simplest service function chain is a linear sequence of service functions between endpoints. In other complicated cases, service functions can split network flows over different paths. Such service chains that can be modelled as directed graphs are named VNF Forwarding graphs (VNF-FG). A VNF-FG consists of network functions as nodes and the connections between them as edges as shown in figure-2.



(a) Topology 1 **(b)** Topology 2 **(c)** Topology 3

Figure 2. Different service chain topologies.

Stages of Service function chaining

In this section, we will discuss the four stages needed in practice for delivering a service function chain optimally. These stages are designed to address the issues arising in automatic deployment of service function chains in the Network.

The first issue is how to describe the functional and non-functional properties of a service function. A service description defines what a service provides and how it is used (Description). It includes service interfaces and the technical constraints for its usage. The second issue in deploying service function chains is the ordering constraints (Composition). Although service functions are typically independent, many service functions have a strict ordering that must be considered carefully. Therefore, a consistent way is needed to impose the ordering of the service functions in a service chain. On the other hand, the deployment of network services is coupled to the network topology. This implies some constraints on delivering service chains in terms of optimal resource usage. Therefore, the most critical issue in optimal service function chaining is efficient resource allocation. This problem determines on which physical nodes VNFs must be placed (Placement), and in which time they must be executed (Scheduling). According to the mentioned issues, the implementation of SFC in a network is carried out in four stages including Service chain Description (SC-D), Service chain Composition (SC-C), VNF-FG Placement (VNF-P) and VNF Scheduling (VNF-S).

Service Chain deployment at Edge and Cloud

NFV is often considered with Cloud Computing to provide networking functions to benefit customers and end-users remotely. Network function virtualization along with Cloud Computing has also started to be seen in the Internet of Things platforms to provide networking functions to the IoT (INTERNET OF THINGS) traffic. The fusion of IoT, NFV, and Cloud technologies, however, is still in its infancy creating a rich and open future research area. In this section we will discuss a novel approach to achieve the deployment of service chains in the Mobile Edge Computing infrastructure for tolerating mission-critical and delay-sensitive traffic. Our goal is to minimize the end-to-end communication latency delay while keeping the overall service chain deployment cost to a minimum. This proposed approach can significantly reduce the delay experienced while satisfying the Service Providers' goal of low deployment costs. Below we have an example figure-3 which shows how this novel approach will look like.

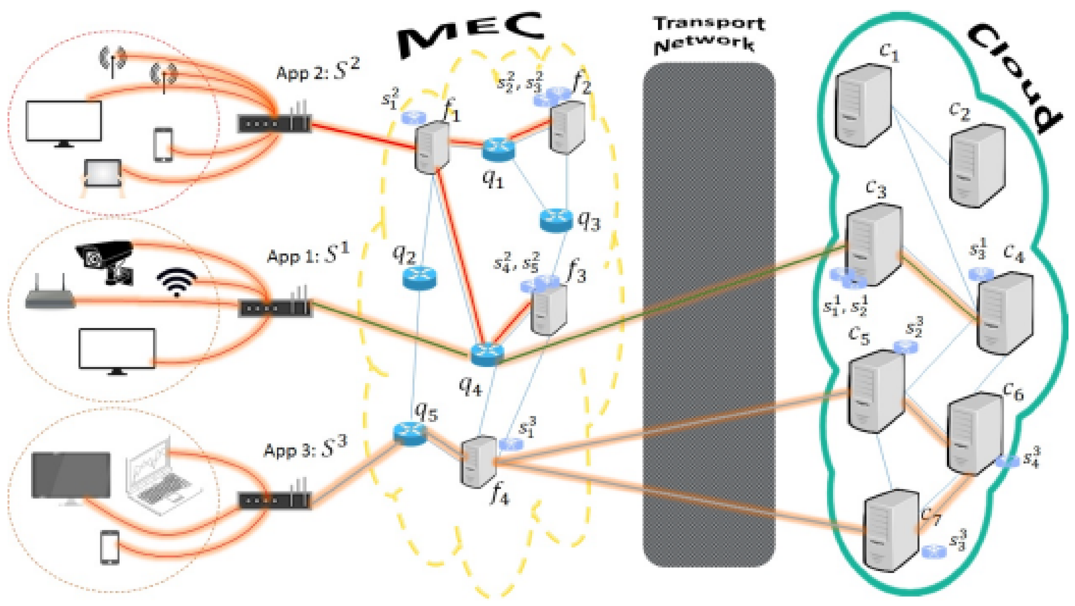


Figure 3. VNF Placement at the Edge and Cloud example.

Let us consider a scenario of cloud-based managed services for IoT applications. In this case, multiple IoT devices can produce substantial amounts of traffic. This traffic can be routed through suitable IoT gateways situated at the edge of the network to a cloud infrastructure comprising multiple data centres. Hence, according to the amount of data produced by the Internet of things devices, suitable service chains must be deployed in the Mobile Edge and Cloud infrastructures. At the same time, the data traffic has specific requirements in terms of bandwidth and overall delay. For

example, for mission critical IoT applications, service chains should be deployed in the MEC taking advantage of the proximity to the IoT layer and, thus, lowing the latency. However, more resource dependent applications can benefit from the large and reliable resources in the Cloud.
The deployment of service chains amounts to solving the problem of finding the dedicated physical resources to instantiate the various VNFs constituting the service chains. This problem is also known as the VNF Placement problem. One major challenge when solving the VNF Placement problem in such an environment is how to efficiently utilize the resource among the Mobile Edge and Cloud infrastructures to achieve the conflicting goals of latency performance and service chains deployment cost. This is a major concern even if the Cloud was the only infrastructure available. Service Providers need to balance the conflicting goals of minimizing the cloud resources used and minimizing the end-to-end delay for achieving pre-specified service-level agreements. However, reducing the number of servers can result in an over-utilization of the server resources which can have a serious effect on the network delay.
The existence of both the Mobile Edge Computing and Cloud creates an extra complexity when trying to balance the conflicting goals. And the utilization of resources of the Mobile Edge and Cloud can significantly reduce the perceived propagation delay or can say latency delay. However, the Mobile Edge and Cloud have a limited number of resources and we do not want over utilization of limited resources. On the other hand, the Cloud has abundant resources, but are usually away from the IoT devices which adds delay overhead. Furthermore, a subset of virtual network functions may come with location constraints. In this case, we must deploy some VNFs to a more reliable and resource powerful cloud computing layer. while others need to be deployed close to the end-users at the MEC (e.g., network address translation-NAT, firewall, traffic prioritization).
The topological characteristics of service chains can have some more limitations which means a service chain is a simple line sequentially connecting several VNFs. However, more realistic scenarios include topologies and graphs.

Proposed Work

Virtual network functions can help to increase network scalability and agility, while also enabling better use of network IT infrastructure resources. Other benefits of using VNFs include reducing power consumption and increasing security and physical space, since VNFs replace physical hardware. This also results in reduced service chains deployment cost for the service provider. But a problem appears, that is, from the consumer perspective we also need to reduce the total latency of service function so that we could not compromise with user experience. That means our goal is twofold now, first is to reduce the capital expenditure and expenses and reduce the total latency for better user experience. Now how to determine the positions for placing VNFs such that the service requirement and quality can be satisfied. Such a problem is called as the VNF Placement (VNF-P) problem which is proved to be NP-hard. In this regard, it is hard to find the optimal solution for VNF-P specifically in large scale network scenarios. To achieve the optimal solution for VNF-P, mathematical programming methods such as Integer Linear Programming (ILP) and Mixed ILP (MILP) are used. But Integer Linear programming or Mixed ILP solutions are much higher in time complexity. So, we work on a time-efficient heuristic based on affiliation-aware VNF placement.

Integer Linear Programming Formulation

Problem Description

The network topology is modelled as $G = (V, E)$, where V is the node set and E is the link set. For each node $v \in V$, the amount of IT resources is C_v. For each link $e \in E$, its bandwidth capacity is B_e. The set of VNFs that are supported in the network is M, i.e., $|M|$ types of VNFs can be instantiated on each node $v \in V$. For an $m \in M$ type VNF, it consumes c_m IT resources and can at most process b_m data traffic in terms of bandwidth units. Note that, although service chaining can enrich the functionality of NFV, the actual form of a Service Chain (i.e., the VNF sequence in it) usually is not arbitrary. We assume that the Service Chains can only take N forms, and the VNF sequence in an n type Service Chain is denoted as ψ_n. For instance, if $\psi_1 = \langle$VNF 1, VNF 3$\rangle$, all the $n = 1$ type Service Chains take the form $s \rightarrow$VNF1$\rightarrow$VNF3$\rightarrow d$, where s, $d \in V$ are the source and destination nodes. Here, we use $|\psi_n|$ to represent the number of VNFs in an n type Service Chain. A Service Chain request is denoted as $Rj = \{sj, dj, bj, nj\}$, where sj and dj are the source and destination nodes, bj is the bandwidth requirement, and nj is the Service Chain type. To realize a Service Chain deployment, we need 1) deploy the requested VNFs on nodes, and 2) connect the VNFs in sequence. Hence, the cost of an S Service Chain deployment consists of 1) the VNF placement cost, and 2) the bandwidth cost. To save the total Service Chain deployment cost, we consider to share VNFs among Service Chains and avoid bandwidth consumption by deploying the adjacent VNFs in a Service Chain on the same node since the communication between them becomes intra-node.

Notation:

- *L*: the upper bound number of any type of VNF that can be deployed in the network
- *P*: the precalculated path set that contain K shorted path between source and destination pair in graph

- α_m : the cost of IT resources used for deploying an m type VNF

- β: the cost of per bandwidth unit on a link
- $f_{l, n, m}$: the Boolean flag that equals 1 if the *l*-th VNF in an *n* type Service Chain is an *m* type VNF, and 0 otherwise.
- $X_v^{m, i}$: the Boolean variable that equals 1 if the *i*-th *m* type VNF is deployed on node *v*, and 0 otherwise.
- $Y_{j, l}^{m, i}$: the Boolean variable that equals 1 if the *l*-th VNF in the Service Chain of *Rj* uses the *i*-th *m* type VNF, and 0 otherwise
- $Z^p_{j, l}$: the Boolean variable that equals 1 if the *l*-th link in the Service Chain of *Rj* uses path $p \in P$, and 0 otherwise
- ξ_v : the total VNF placement cost
- ξ_b : the total bandwidth cost

Objective Function : $\text{Minimize} \quad (\xi_v + \xi_b), \qquad (1)$

$$\begin{aligned} \xi_v &= \sum_m \alpha_m \cdot \left(\sum_{i=1}^{L} \sum_v x_v^{m,i} \right), \\ \xi_b &= \sum_{j,p} \beta \cdot |p| \cdot \left(b_j \cdot \sum_l z_{j,l}^p \right). \end{aligned} \qquad (2)$$

Constraints:

$$\sum_v x_v^{m,i} \leq 1, \quad \forall m, \ i. \qquad (3)$$

Eq. (3) ensures that each *m* type VNF is deployed only once.

$$\sum_{m}\left(\hat{c}_m \cdot \sum_{i=1}^{\mathcal{I}} x_v^{m,i}\right) \le C_v, \quad \forall v \in V. \qquad (4)$$

Eq. (4) ensures that each node's IT resources would not be overused during VNF placement.

$$\sum_{i=1}^{\mathcal{I}} \sum_{l=1}^{|\psi_{n_j}|} y_{j,l}^{m,i} = \sum_{l=1}^{|\psi_{n_j}|} f_{l,n_j,m}, \quad \forall j,\ m. \qquad (5)$$

Eq. (5) ensures that we provision each requested VNF one and only one time when serving *Rj*.

$$\sum_{j}\left(b_j \cdot \sum_{l=1}^{|\psi_{n_j}|} y_{j,l}^{m,i}\right) \le \sum_{v} x_v^{m,i} \cdot \hat{b}_m, \quad \forall m,\ i. \qquad (6)$$

Eq. (6) ensures that each VNF's traffic processing capacity would not be overused.

$$\sum_{j,l} b_j \cdot \sum_{\{p:e\in p\}} z_{j,l}^{p} \le B_e, \quad \forall e \in E. \qquad (7)$$

Eq. (7) ensures that each link's bandwidth capacity would not be overused.

Service Chain Deployment with Affiliation-Aware vNF Placement

We propose a time-efficient heuristic solution. The main idea of which is to place VNFs considering their affiliation in requested SCs. The term affiliation aware means we need to find total VNF placement cost and total latency considering the source and destination server of every service chain in mind. Firstly, we need to see what the input is and in what format we are having it. So, in our test case, we are having several virtual network functions and several service chains. Then following we are having input of all the service chains, their source server, destination server and their corresponding bandwidth requirement. After that we are preparing a VNF graph with an adjacency matrix which is shown as below Figure-4(a).

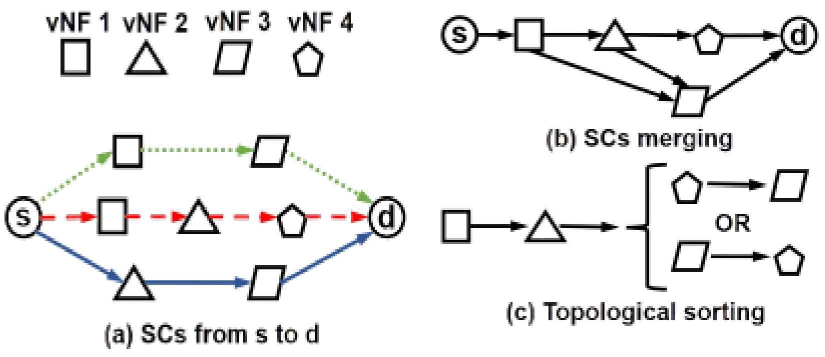


Figure-4

In Our all four solutions, we merge the SCs based on their source-destination pairs, and generate a VNF graph $\Psi_{s,d}$ to cover all the SCs whose source destination pair is *s*-*d*, where $B_{s,d}$ is the total bandwidth demand. Figs. 4(a) and (b) show an example on merging SCs to generate a VNF graph. As shown in Fig. 4(a), there are three SCs from *s* to *d*. To obtain their VNF graph, we merge the same type VNFs together in Fig. 4(b). Then we perform topological sorting on VNF graph $\Psi_{s,d}$ to order the VNFs as a sequence ψ, i.e., obtaining the VNFs' affiliation information. As shown in Fig. 4(c), topological sorting orders VNFs such that for each direct link VNF A→VNF B in the VNF graph generated in Fig. 4(b), VNF A comes before VNF B. Lets have an example of a test case and how we are doing it.
Assuming the number of VNFs is 5 and the number of service chains is 3. Then input of all the service will looks like N1(first service chain): 0->1->3->5->6,
N2: 0->1->2->4->6,
N3: 0->2->3->6
Here, '0' and '6' are the nodes of network topology and they are acting as servers. Below Figure-5 shows how VNF graph $_{0,6}$ will look like after merging the service chains whose source is '0' and destination is '6'.

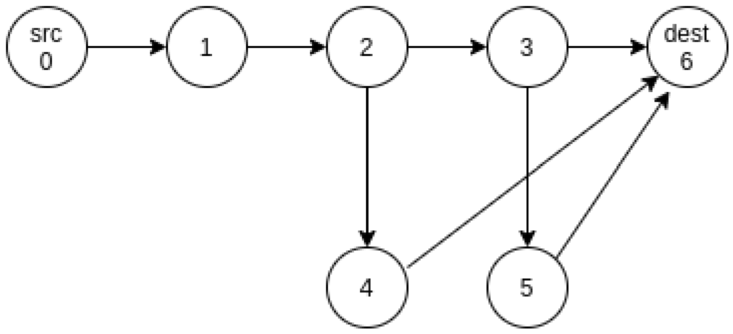


Figure -5

The next step is topological sort which will give the VNF sequence i.e. [0,1,2,4,3,5,6]. We have also taken the load capacity of each VNF as input.

That was all about VNF chain input and their requirements. The next is network topology G (*V, E)*. We have Savvis topology as our network topology. After having input, we are preparing a network graph using an adjacency matrix and performing Floyd Warshall algorithm so that we can access source to destination shortest latency easily. We are also having the IT resource capacity of each node as well and the bandwidth capacity of each link.
That is all about our input and their pre-processing. Then we can move to our solutions which will tell us how we can place the VNF sequence on network topology, so that we can minimize total latency delay and SCs deployment cost.

Approach 1: Using Simple Hill Climbing

In Simple Hill Climbing we examine the neighbouring nodes one by one and select the first neighbouring node which optimizes the current latency as the next node. Let us see the steps we are using to solve our problem.
Step 1: Evaluate the initial state and make the initial solution the current solution. It will be a random server path from source server to destination server of network topology G (V, E).
Ex. [0->1->3->10->6]
Step 2: Loop until the target solution state is found.
a) Mutating servers randomly other than source and destination server is the strategy which has been used to find the neighbour solutions.
Ex [0->1->3->10->6] to [0->8->3->10->6] or [0->9->3->10->6]
b) Perform these until a solution is found
i. If the current solution is a target solution, then stop and return to success. And can do that by checking If the last 5 iteration we are not improving then we are stuck in global minima or local minima and no further solution is possible from simple hill climbing.
ii. If the best solution among the neighbour solutions is better than the current solution, then make it the current solution, mutate the current solution, and then repeat step2.
iii. If the best solution among the neighbour solutions is not better than the current solution, then continue in the loop until a target solution is found.
To find if the current state is better or not, find the latency of the current state in our objective function. If latency is improving then will carry on the iteration otherwise check for the last 5 iterations. If the last 5 iterations are not improving then the solution is stuck in global minima or local minima and no further solution is possible from simple hill climbing.

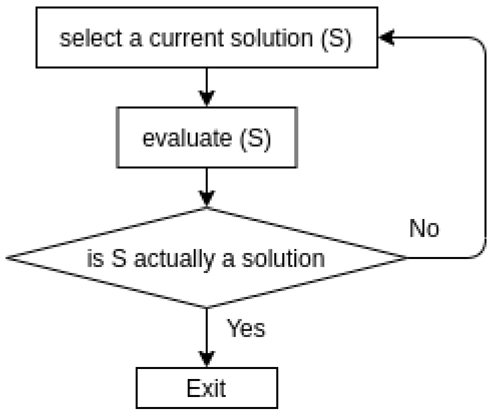
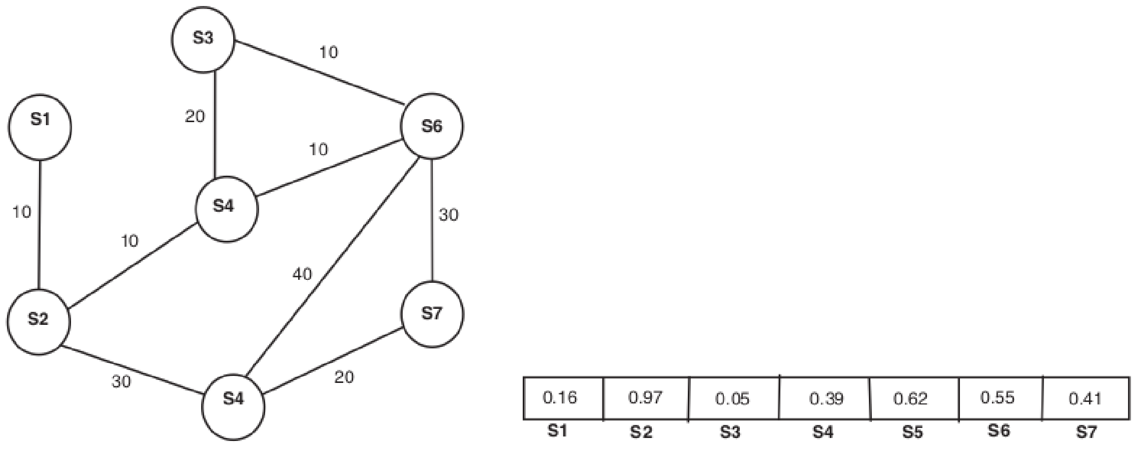


Figure-6

Approach 2: Using Particle Swarm Optimization

Eberhart and Kennedy7 introduced the PSO (Particle Swarm Optimization) algorithm in 1995. It is a nature-inspired population-based optimization algorithm. The goal of using particle swarm optimization is to minimize the total latency coming from all the service chains. Below there is an objective function which aims to check load and bandwidth constraint and perform latency calculation. For all optimization methods, a particle (denoted as X) refers to a probability vector for placing VNF at different servers. In our case, the particle is taken as a d-dimensional vector, where d is the total number of servers in the network which is equal to nodes of G (*V, E*). Moreover, the value in each dimension indicates the probability of the VNF to be placed at that server. E.g. There is a position vector as shown below with their probability's values and two VNF need to be placed on it. S2 and S5 have two maximum probabilities so VNFs should be placed on the S2 and S5 server.



(A) (B)

Figure-7

Objective Function *f*

```
Inputs:
VNF seq, position vector of the particle
     3. Create a map from VNF to server
     4. For every pair (v,s) belongs to map
     Check load on VNF should be less than resource capacity on server
     5. Perform depth first search on VNF graph $\Psi_{s,d}$ to check
        bandwidth requirement between two VNF should be less than
        bandwidth of link, Be
     6. For every VNF u:
For every VNF v which adjacent to u:
total latency+= get min latency from map[u] to map[v]
     7. Return max value if constraint fails to satisfy so that
        solution will not converge to that mapping again
     8. Otherwise return total latency
```

```
Pseudo code:
Inputs: Objective function (f), number of particles, iteration,
number of nodes in topology

     9. Initialize position of each N particle X1, X2..
     10. Initialize velocity of each N particle V1, V2..
     11. While termination criteria not meet:
     12. For i 1->N do:
          a. r1-> d dimensional vector having random value [0,1]
          b. r2-> d dimensional vector having random value [0,1]
          c. $Vi = w * Vi + c1 * r1 * (pbesti - Xi) + c2 * r2 * (gbest - Xi)$.
     13. End for
     14. For i 1->N do:
          d. Xi=Xi+Vi
     15. End for
     16. Calculate latency of each particle using f
     17. For i 1->N do:
          e. If latency of Xi is lower than the pbesti then:
               pbesti =Xi
          End if
     18. End for
     19. Gbest = get particle position having minimum latency
     20. End while
     21. Optimal placement = gbest
     22. Optimal latency = latency corresponding to gbest
     23. Return optimal placement and optimal latency
```

W is inertial constant, c1 is cognitive constant and c2 is social constant.

Approach 3: Using Tabu Search metaheuristic

We adopt the Tabu Search algorithm to model the VNF-P problem. Tabu Search is a meta-heuristic approach that applies local search to provide solutions close to optimality. In contrast with other heuristics and meta-heuristics, Tabu Search can avoid being trapped in a local optimum by incorporating a short-term memory, called Tabu List. In general, the Tabu Search algorithm has five components: (i) the Initial Solution; (ii) the Neighbor Solution; (iii) the Tabu List; (iv) the Aspiration Criterion; and (v) the Termination Criterion. These components are appropriately adapted for our problem and are described in detail as follows.

Step 1: We first construct an initial state. And this initial solution is again a random server path from source server to destination server of network topology G (V, E).

Step 2: Create a set of neighbouring solutions to the current solution. The solutions that are already in the Tabu List should be removed from the set of neighbouring solutions.

Step 3: Choose the best solution out of all neighbouring solutions and if that solution is better than the current best solution, update the current best solution.

Step 4: Update the Tabu List by removing all moves that have expired past the Tabu Tenure and add the new move to the Tabu List.

Step 5: If the Termination Criteria are met, then the tabu search stops or else it will move onto the next iteration. Termination Criteria is a max number of iterations.

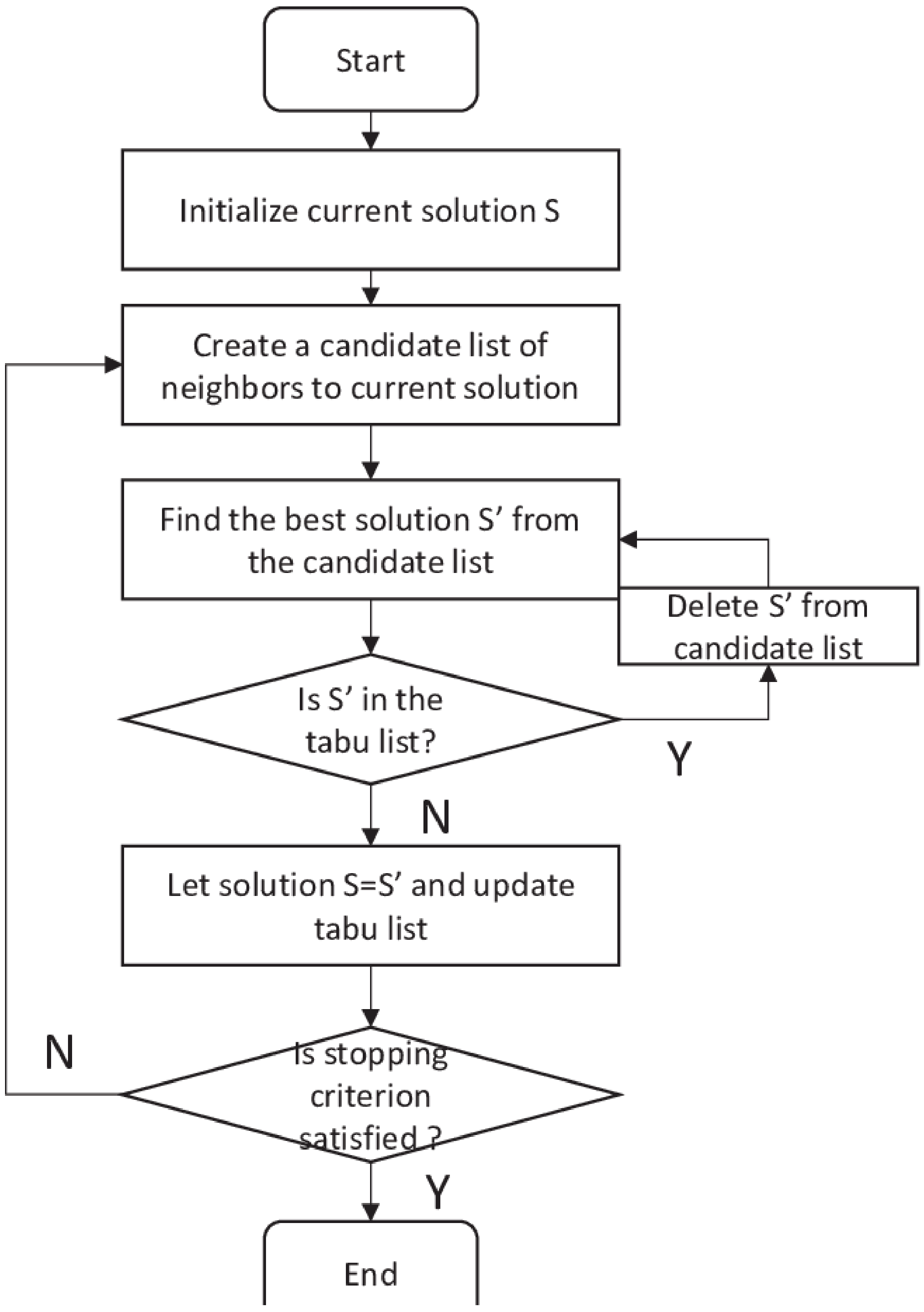

Start
Initialize current solution S
Create a candidate list of neighbors to current solution
Find the best solution S' from the candidate list
Delete S' from candidate list
Is S' in the tabu list?
Y
N
Let solution S=S' and update tabu list
N
Is stopping criterion satisfied ?
Y
End

Figure-8

Approach 4: Using Yen’s k shortest Path Algorithm

Yen's k-shortest path algorithm gives k shortest paths so that we can get the first shortest path, second shortest path and the third shortest path and so on. Assuming a scenario that we must travel from place C to place H and there are multiple routes available between place C and place H, we must find the shortest path and neglect all the paths that are less considered in terms of its time complexity to reach the destination.

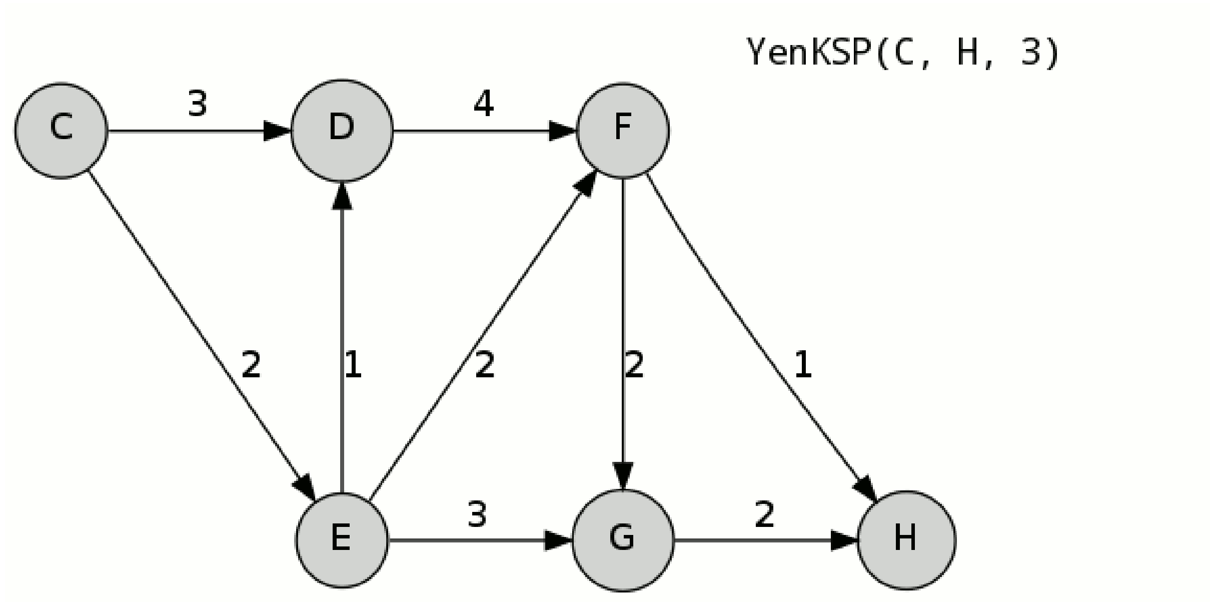


Figure-9

The three shortest paths from C->H will as follows C->E->F->H: Cost=5
C->E->G->H: Cost=7
C->D->E->F->H: Cost=7
In our problem we will use yen’s k shortest path algorithm for finding the pre-calculated path set that consists of K shortest paths between each source-destination pair in G (V, E) and will call it P_k.

```
Step1: For each $p_{s,d}P_k$ hat has enough bandwidth resources to satisfy
$B_{s,d}$ from the shortest do
    a. place/reuse VNFs in ψ in sequence greedily on v, and
       record the result temporarily.

Step2: If the VNF sequence can place a path $p_{s,d}$ then repeat step 1
for another source and destination pair, otherwise get the second
shortest path from $P_{s,d}$ and so on.
```

Performance Evaluation

We use a three small to medium scale test case and a Savvis topology with 19 nodes and 20 links. Here, each node has Cv = 100 units of IT resources, each link has Be = 200 Gbps bandwidth. In our first test case there are ｜M｜ = 5 types of VNFs considered, and the Service Chains can take N = 3 forms. Each type of VNF consumes IT resources within [0.4, 1] units. Each type of Service Chain includes 2 to 4 VNFs and the bandwidth demand is randomly selected within [5, 12] Gbps. We set β = 0.01 and αm ∈ [1, 1.2].
We then evaluate the performance of virtual network function placement solutions with simple hill climbing, PSO, Tabu Search and Yen's k shortest path for the Savvis topology that consists of 19 nodes and 20 links. In our second test case, we assume that there are ｜M｜ = 10 types of VNFs, and the SCs take N = 5 forms, each of which includes 2 to 5 VNFs. While another parameter like Cv, Be remains the same. In our third test case We assume that there are ｜M｜ = 15 types of VNFs, and the SCs take N = 10 forms, each of including 2 to 5 VNFs. Below are the results for our three test cases and four solution approaches.

No. of Service Chain Request	Total Service Chain Deployment Cost				Total Latency (Ms)			
	SHC	PSO	TABU	YEN	SHC	PSO	TABU	YEN
TEST CASE 1	11.192	11.192	9.592	8.692	3606.58	8013.51	1342.26	328.86
TEST CASE 2	21.605	21.605	18.905	18.705	10947.30	9410.62	5175.64	4162.25
TEST CASE 3	41.397	41.397	35.197	37.197	14199.56	10585.53	7470.06	6456.66

Table-1

To visualize better, we have plotted two bar charts and two count charts. The bar chart below illustrates how our Service Chain deployment cost is affecting the increasing number of service chain requests i.e., from test case-1 to test case and test case-2 to test case-3.

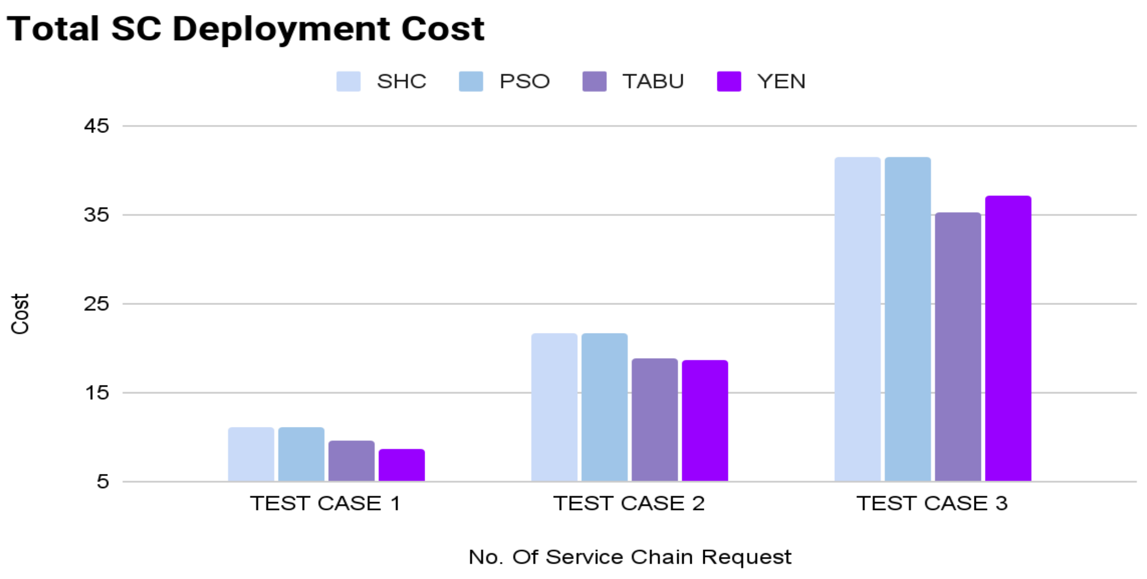


Figure-10

Similarly, the bar chart below illustrates how Total latency of service chains affects the increasing number of service chain requests.

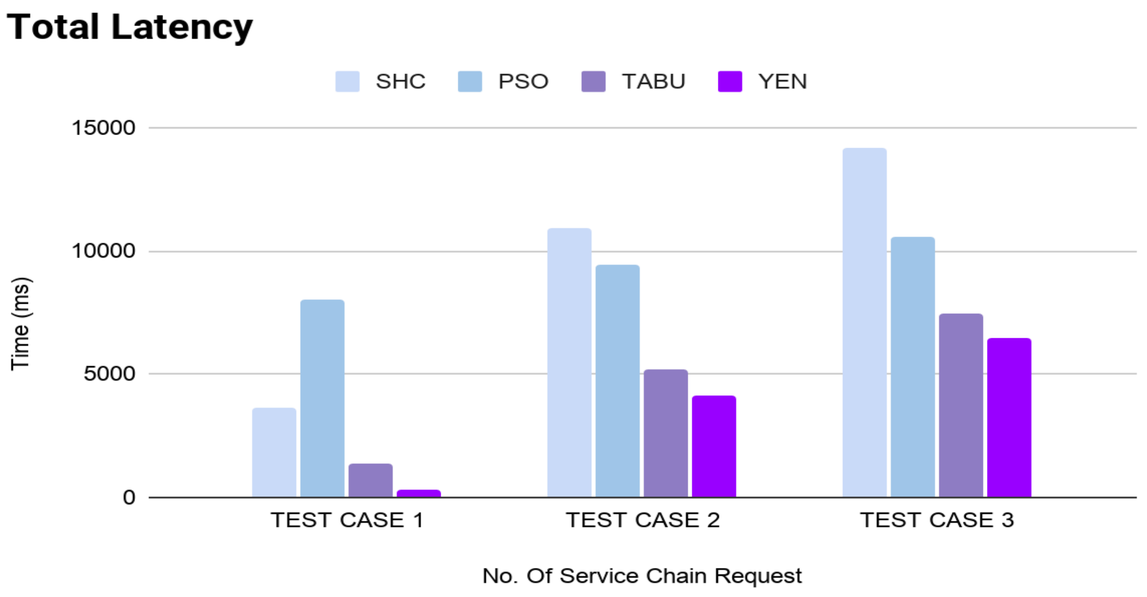


Figure-11

We have seen Total Latency or Cost vs number of service chain request comparison. If we consider our worst test case the only test case and then compare which approach is performing efficiently. Below are the two count charts showing individual algorithm's performance in terms of Total latency delay and service chain deployment cost.

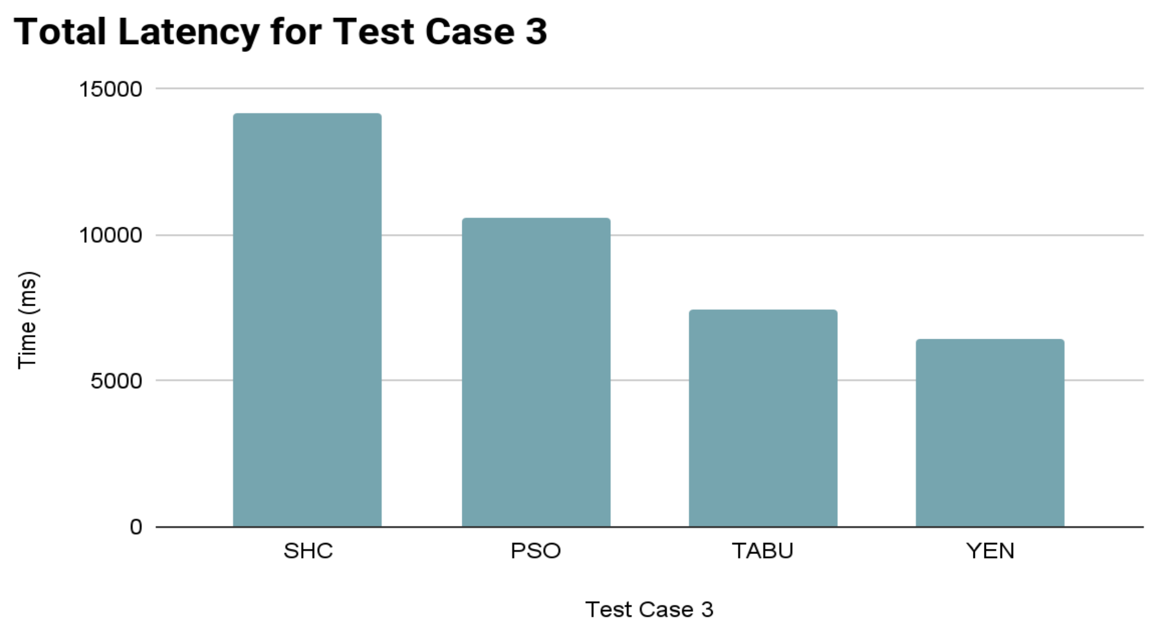


Figure-12

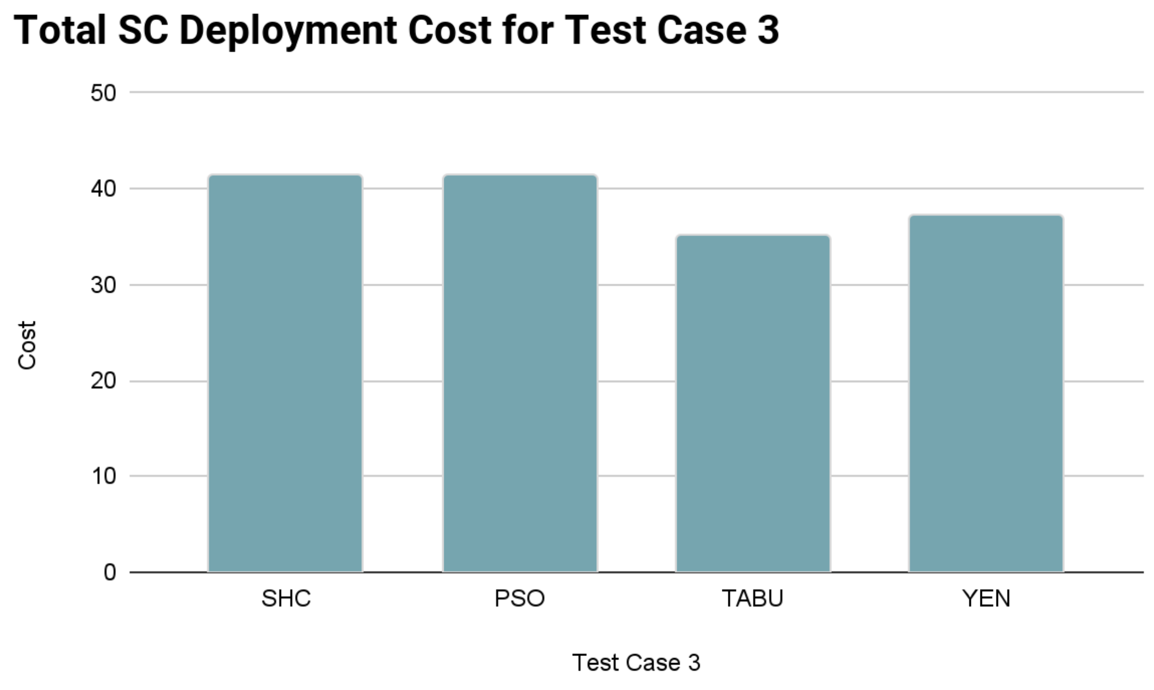


Figure-13

So, analysing all the above graphs, it is conclusive that on increasing the number of service chain requests, the service chain deployment cost and total latency will increase and it is also very much conclusive that the solution approach using Yen's algorithm performs the best in terms of latency delay. While in terms of service chain deployment cost the Tabu search solution is slightly better.

Conclusion

In our project work, we have successfully implemented our all four approaches with keeping in mind all the constraints and VNF affiliation awareness. We try to place virtual network functions (VNFs) on the network nodes and connect the VNFs in series through link mapping. We propose how Individual virtualized network functions can be chained or combined to deliver full-scale networking communication services. These chained Virtual network functions can help to increase network scalability and agility, while also enabling better use of network IT infrastructure resources. Other benefits of using VNFs include reducing power consumption and increasing security and physical space, since VNFs replace physical hardware. This also results in reduced service chains deployment cost for the service provider. This brings the concept of chaining of virtual network functions called as service Function chaining.

An integer linear programming (ILP) model is formulated to minimize the total Service Chain deployment cost. With the ILP, we prove that the problem is NP-hard and propose a time-efficient heuristic solution based on affiliation-aware VNF placement. The importance of the project is twofold. First is for the service provider it will reduce their deployment cost and for common users like us, it will reduce the network latency as well. The time-efficient heuristic algorithm can be used for that purpose. Our solution approaches are implemented with keeping in mind all the constraints and affiliation aware VNF placement.

We implemented four solution approaches using simple hill climbing, particle swarm optimization, tabu search and Yen’s algorithm. We have formulated our result in table-1 and visualize the comparison for solution approaches in figure10 to 13. So, analysing those charts, it is very much conclusive that the solution approach using Yen’s algorithm performs the best in terms of latency delay. While in terms of service chain deployment cost the Tabu search solution is slightly better.

Appendix A

Challenges with NFV

Individual virtualized network functions (VNFs) can be chained or combined in a building block-style fashion to deliver full-scale networking communication services. VNFs can help increase network scalability and agility, while also offering better use of network infrastructure IT resources. Some other benefits include reducing power consumption, increasing security and physical space, since VNFs replace physical hardware. This also benefits in reduced operational and capital expenses. Our Project work is focused on VNF placement challenges but following are the challenges with NFV.

VNF placement

VNFs can help increase network scalability and agility, while also offering better use of network infrastructure IT resources. Some other benefits include reducing power consumption, increasing security and physical space, since VNFs replace physical hardware. This also benefits in reduced operational and capital expenses. But a problem appears, that is, how to determine the positions for placing VNFs such that the service requirement and quality can be satisfied. Such a problem is referred to as the VNF Placement (VNF-P) problem which is proved to be NP-hard. In this regard, it is generally hard to find the optimal solution for VNF-P specifically in large scale network scenarios. To achieve the optimal solution for VNF-P, mathematical programming methods such as Integer Linear Programming (ILP) and Mixed ILP (MILP) are used.

VNF migration

The VNF Migration (VNF-M) problem refers to the process of migrating VNFs from one place to another due to specific requirements such as load balance and hardware maintenance. During the process of migration, the VNF related state (e.g., CPU interrupt and memory) must be migrated to the destination as well. However, the new VNF may ignore some malicious activities due to the lack of necessary information. In addition, VNFs are software that can be implemented in either VM (Virtual Machine) or container (a stand-alone and executable environment for software, e.g., Docker). Typically, VM is used to host VNFs in the research work.

VNF scheduling

The VNF Scheduling (VNF-S) problem is different from the VNF-P problem and VNF-S has two research points. In particular, the first research point is to schedule different VNF instances to serve and provide network services. Under this situation, the VNF-S problem is usually processed together with the VNF-P problem since they have the same goal, i.e., service provision.

VNF chaining

The VNF Chaining (VNF-C) problem is also called service function chaining problem which focuses on the mechanisms for chaining VNFs and controlling the corresponding traffic through these VNFs in order before reaching destinations. Targeting this problem, IETF (Internet Engineering Task Force) has specially established a working group to document novel approaches for VNF constituted service delivery and operation as well as the SFC (service function chain) architecture and algorithms for traffic steering, which proves the importance of SFC. To keep a unified style, the term VNF-C is used instead of service function chaining in this section.

Appendix B

Yen’s k shortest algorithm

Yen's Shortest Path algorithm calculates several shortest paths between two nodes. This algorithm is often referred to as Yen's k-Shortest Path algorithm, where k is the number of shortest paths to compute. The algorithm supports weighted graphs with positive relationship weights.
For k=1, the algorithm behaves like Dijkstra's shortest path algorithm and returns the shortest path. For k=2, the algorithm returns the shortest path and the second shortest path between the same source and target node. For k=K, the algorithm computes at most K paths which are discovered in the order of their total cost. Below is the pseudo code of Yen's algorithm.

```
function YenKSP(Graph, source, sink, K):
    A [0] = Dijkstra (Graph, source, sink);
    B = [];

    for k from 1 to K:
        for i from 0 to length (A [k − 1]) − 2:
            spurNode = A[k-1]. node(i);
            rootPath = A[k-1]. nodes (0, i);

            for each path p in A:
                if rootPath == p.nodes(0, i):
                 remove p.edge(i,i + 1) from Graph;

            for each node rootPathNode in rootPath except spurNode:
                 remove rootPathNode from Graph;

            spurPath = Dijkstra (Graph, spurNode, sink);
            totalPath = rootPath + spurPath;

            if (totalPath not in B):
                B.append(totalPath);

           restore edges to Graph;
           restore nodes in rootPath to Graph;

        if B is empty:
            break;
        B.sort();

      A[k] = B [0];
        B.pop();

    return A;
```

References

1. Future Internet 11(3) VNF placement at edge and cloud.
2. Varna-based optimization: a novel method for capacitated controller placement problem in SDN by Ashutosh Kumar SINGH, Saurabh MAURYA, Shashank SRIVASTAVA
3. Forecast-Assisted NFV Service Chain Deployment based on Affiliation-Aware VNF Placement by Quanying Sun, Ping Lu, Wei Lu, Zuqing Zhu
4. Design modelling lectures by National Programme on Technology Enhanced Learning [click here]
5. A Survey on the Placement of Virtual Resources and Virtual Network Functions by Abdelquoddouss Laghrissi and Tarik Taleb
6. Yen's k shortest routing algorithm [click here]