

Placement of Service Chain and Removal of bottlenecks in Edge-Fog-Cloud computing

Mrinal Aich

A Thesis Submitted to
Indian Institute of Technology Hyderabad
In Partial Fulfillment of the Requirements for
The Degree of Master of Technology



Department of Computer Science Engineering

June 2018

Declaration

I declare that this written submission represents my ideas in my own words, and where ideas or words of others have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.

Mrinal Aich

(Signature)

MRINAL AICH

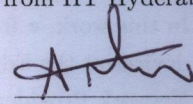
(Mrinal Aich)

CSIGMTECH11009

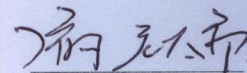
(Roll No.)

Approval Sheet

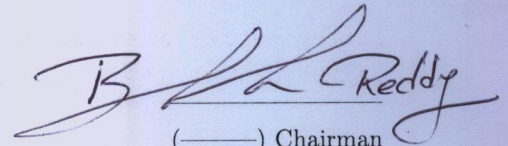
This Thesis entitled Placement of Service Chain and Removal of bottlenecks in Edge-Fog-Cloud computing by Mrinal Aich is approved for the degree of Master of Technology from IIT Hyderabad



(——) Examiner
Dept. of Computer Science Eng
IITH



(Dr. Kotaro Kataoka) Adviser
Dept. of Computer Science Eng
IITH



(——) Chairman
Dept. of Computer Science Eng
IITH

Abstract

Data management nowadays is being focused a lot in many areas in the industry, the reason for this is attributed to the speed we are obtaining the data, the volume of it and the new technologies. So, when we transfer this data through the network, we obviously need the most efficient way to send and retrieve data in our network with less response time, also known as Edge-Fog-Cloud computing. In this work, a framework is implemented in such a way that it provides minimum latency to User devices at the Edge of the network based on the SLA agreement between a Service Provider and the Network Provider.

It provides an algorithm for Service chain Placement and monitors the network for potential bottlenecks that may be created in future and removes it.

Contents

Declaration	ii
Approval Sheet	iii
Abstract	iv
Nomenclature	vi
1 Introduction	1
1.1 Overview	1
1.2 Challenges	1
1.3 Related Work	2
2 System Design	3
2.1 System Flow Chart	3
2.2 System Block Diagram	3
3 Placement of VNFs/SLA Service Chains	5
3.1 Overview	5
3.2 Placement Algorithm	5
3.3 Complexity	6
3.4 Working Illustration	6
4 Bottlenecks and its Detection	9
5 Bottleneck Removal Algorithm	11
5.1 Overview	11
5.2 Algorithm	11
5.3 Complexity	13
5.4 Working Illustration	14
5.4.1 Migration to Neighbor	14
5.4.2 Migration by creating resources at the Neighbor	14
6 Implementation	16
7 Evaluation Results	18
7.1 Overview	18
7.2 Cumulative Placement Time	19

7.2.1	Simulation	19
7.2.2	Emulation	19
7.3	Placement Ratio Over Edge-Fog Network	20
7.3.1	Simulation	20
7.3.2	Emulation	20
7.4	CDF of the Bottleneck Removal Algorithm	21
7.4.1	Simulation	21
8	Conclusion	23
	References	24

Chapter 1

Introduction

1.1 Overview

To tackle the high demand for Cloud Computing technologies researchers in both academia and industry are slowly coming to the conclusion that it's better to have smaller instances of data centers closer to the Users devices. This not only helps in reducing the response time from the Network but also decreases the load on the network. This is beneficial to the Users, the Service Providers and the Network Providers.

What this means to the end user is that now the users would get served by services closer to them than the Cloud. This would reduce the response time of the services and improve the experience for the Users. For the Service Providers, they can increase their profit by providing users with better services.

There will be situations where a particular service is popular and being heavily utilized by the Users. This may use high resources of the network and may affect other services. These type of scenarios should be detected and handled before hand to provide seamless services to the Users.

1.2 Challenges

The placement and readjustment of VNFs in the edge-fog-cloud network are extremely challenging, for the following reasons:

- **VNF interoperability.** VNF interoperability is interoperability between identical VNFs running in different computing environment like edge, fog, and cloud. For example, a VNF maybe deployed in a cloud, with provision for a copy to be executed in an edge to handle traffic peaks. The two VNFs must operate together. Data synchronization is a critical concern when VNFs in different clouds work together. High latency among the different computing environment makes synchronization difficult. The design of the communication protocol that enables the VNF interoperability requires that the interoperating VNFs share common process and data models.
- **Resource constraints.** Each SLA has node and link constraints, such as CPU and RAM resource on the nodes and link bandwidth that must be satisfied while VNFs is being served.

For example, a user may request the VNFs which demand 1 GHz CPU, 8 GB of RAM and 10 Mbps links between compute nodes and three different end-points of the network in the geographically distributed location that connect the users. Furthermore, the user may require additional constraints such as propagation delay. These restrictions on nodes and links make placement and readjustment of VNFs computationally hard.

- **Dynamic requests.** The VNF requests can be dynamic; it may come dynamically and stay in the network for an arbitrary time. The placement and readjustment of the VNFs algorithm must determine VNF location online. Online placement problems are typically computationally intractable.
- **Guaranteed resources.** Since the capacity edge and fog is limited, some VNF request may be denied, or some VNFs has to migrate to the cloud or apply scheduling techniques for sharing resources to provide the guaranteed resource availability.
- **Priority SLAs.** SLAs may have different priority depending upon the negotiation between the Service Provider and Network Provider. Some SLA requests would have high priority to be placed close to the End Users to provide low latency User service.
- **Handling busty high usage of VNFs.** Some VNFs may be highly utilized for certain interval of time. This would increase the Network load and indirectly effect other SLAs sharing the same node, or link, or both. For example, a Streaming service hosting an El-Classico football match would create high load onto the network. So, temporary movement or replication of these VNFs closer to the user devices would reduce the core Network load.
- **Detection and removal of potential bottlenecks.** to be filled later

1.3 Related Work

The Placement problem even for a single-layered network is NP-Hard as mentioned in [1], [2], [3] and [4]. Depending on the use cases and parameters considered, most work convert the Placement problem into a Mathematical model and solve with ILP-solvers using various Heuristic approaches. These provide sub-optimal solutions to the Placement problem which may require complete restructuring of the Network placement. Total restructuring of the network is both time-consuming and is not practical.

Chapter 2

System Design

The System can be considered as an Orchestrator which is built on top of a SDN Controller. The controller is used only to interact with the OVS switches. The Orchestrator interacts with both the Service Provider and geographically distributed Hypervisors.

2.1 System Flow Chart

Figure 2.1 shows the flowchart of the system. When an online request for a SLA is received, the placement algorithm 1 places the VNFs satisfying the SLA constraints. The network is continuously monitored and checks for a possible bottleneck. Once a possible bottleneck is detected, the Bottleneck Removal algorithm 2 removes it.

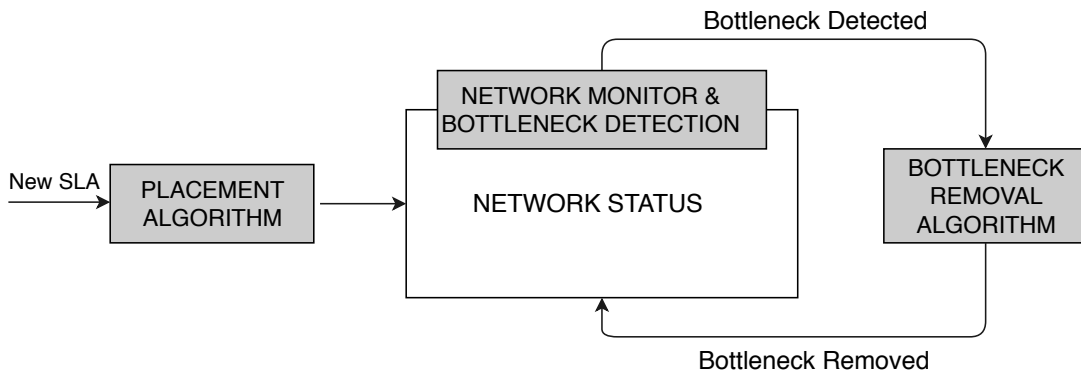


Figure 2.1: *System Flow chart*

2.2 System Block Diagram

Figure 2.2 shows the key components of the system. The system interacts with two external entities, the Service Provider and the underlying Network. The Service provider may provide Online SLA requests, and based on the agreement with the Network provider it is installed over the underlying

network. Client-Server protocol is used between the Service Provider and the System. The underlying network is composed of OVS-Switches and Hypervisors. Openflow v1.5 is used to communicate with the OVS-switches and Client-Server protocol is used between the system and the distributed Hypervisors. Next, all the components of the System are explained.

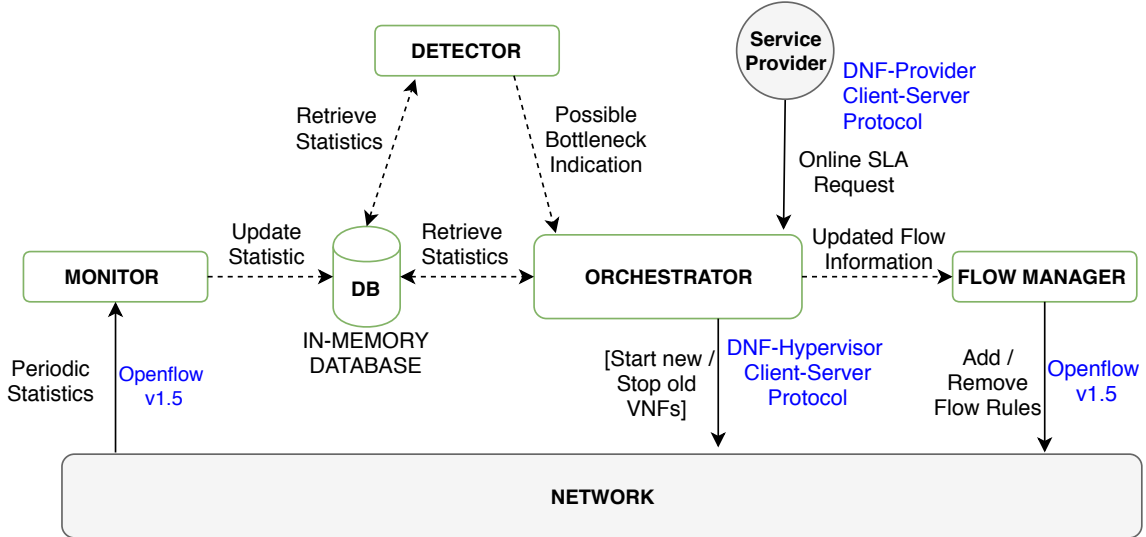


Figure 2.2: System Block Diagram

- **Orchestrator.** The Orchestrator is the controlling entity of the system. It processes online SLA requests and handles possible bottlenecks informed by Detector. It uses the information from In-memory Database and executes either Placement or Bottleneck Removal algorithms. Finally, it indicates the Flow Manager about the necessary actions to be taken. Also, it may setup new or remove old instances of VNFs on Hypervisor, if required.
- **Monitor.** This collects periodic statistics of the network and updates in the in-memory Database. It collects switch statistics from OVS-switch, measures link latency (Cite) and Hypervisor CPU utilization using the client-server protocol mentioned above.
- **Flow Manager.** The Flow Manager on receiving indication from the Orchestrator adds or removes flow rules from the underlying OVS-switches.
- **Detector.** This continuously detects any possible bottlenecks in the network. This is done by periodically retrieving statistics from the Database. On detecting a possible bottleneck, it informs the Orchestrator.
- **Database.** Periodic run-time information of the Network is maintained in an In-memory database by the Monitor module. The information is used by both the Orchestrator and Detector.

Chapter 3

Placement of VNFs/SLA Service Chains

3.1 Overview

The Network provider places the VNFs of a SLA by utilizing the resources efficiently. Also, with the advantages of Edge-Fog-Cloud Computing, the placement of VNF(s) at the Edge or Fog layer would reduce the response time and the Network load at the upper layers. If no such option is possible, then the VNF(s) are placed at the Cloud.

The algorithm accepts online SLA requests and performs VNF Service Chain placement satisfying the SLA constraints. It provides placement of VNFs closer to the User devices for low network latency for User traffic generated from any User end-point as mentioned in the SLA.

3.2 Placement Algorithm

The proposed algorithm uses an iterative version of Dijkstra algorithm along with subsequent greedy approach for placement. Algorithm 1 presents the pseudocode of the Placement Algorithm and Table 3.1 shows the Notations used in the pseudocode.

An iterative version of Dijkstra's algorithm is used to find the node where the first VNF of the Service Chain is placed (line 2-23). If network resources at the Edge or Fog layer are insufficient then the SLA is placed at the Cloud (line 24,25). If placement is at a node at the Edge-Fog layer, then the difference between the Maximum Delay tolerated by the SLA and maximum latency incurred from an entry-exit end-point to that node is the Latency Buffer (line 28). Remaining VNFs of the Service chain may be placed over the links within the range of this Delay Buffer.

After placement of the first VNF of the Service Chain, rest of the VNF(s) are placed iteratively using a Greedy approach (line 30-45). In each iteration, the same node (as in the previous iteration) along with all its neighbor nodes satisfying the SLA constraints and within the range of the Latency Buffer are considered for placement (line 32). Finally, the VNF is placed at the node with minimum latency from the current Hypervisor(line 38). If any of the VNFs are not being able to place either at the Edge or Fog layer, then all the VNFs of the SLA are placed at the Cloud (line 42).

Table 3.1: Notations for Algorithm 1

Notation	Description
G	Network Graph
U	List of Entry Points
L_{tol}	Latency tolerated as per the SLA
BW_{req}	Bandwidth requirement of SLA
$BW_{u,v}$	Available Bandwidth over physical link (u,v)
RAM_{req}	Hypervisor RAM requirement
$Seen_x$	List of entry-point nodes that have visited node-x
$d_u[v]$	Delay incurred till node-v from starting from entry-point-u
$L_{u,v}$	Latency incurred over the link (u,v)
$Start$	Node where 1st VNF of Service Chain is placed
DB	Delay Buffer of the SLA
$Prev$	Node where previous VNF was placed
$minDelayNbr$	Neighbor with minimum Latency
$Place(x, y)$	Placement of VNF-x at node-y
SC	Service Chain of the SLA

3.3 Complexity

Consider a network having V - Edge or Fog nodes and E - links between them. K be the length of Service Chain of a SLA. To find the center of Edge-Fog Network, we use Iterative Dijkstra algorithm w.r.t. all the User entry points. The complexity for this part is $V(V \cdot \log V + E \cdot \log V)$. The rest of the VNFs of the Service Chain are placed using Greedy approach either at the same node where the previous VNF was placed or any of its neighbor node. The complexity for this is $K \cdot V$. So, the complexity of the algorithm is $V(V \cdot \log V + E \cdot \log V) + K \cdot V$, as $E \geq V$ and $K \ll V$, then

$$O(V^2 \log V)$$

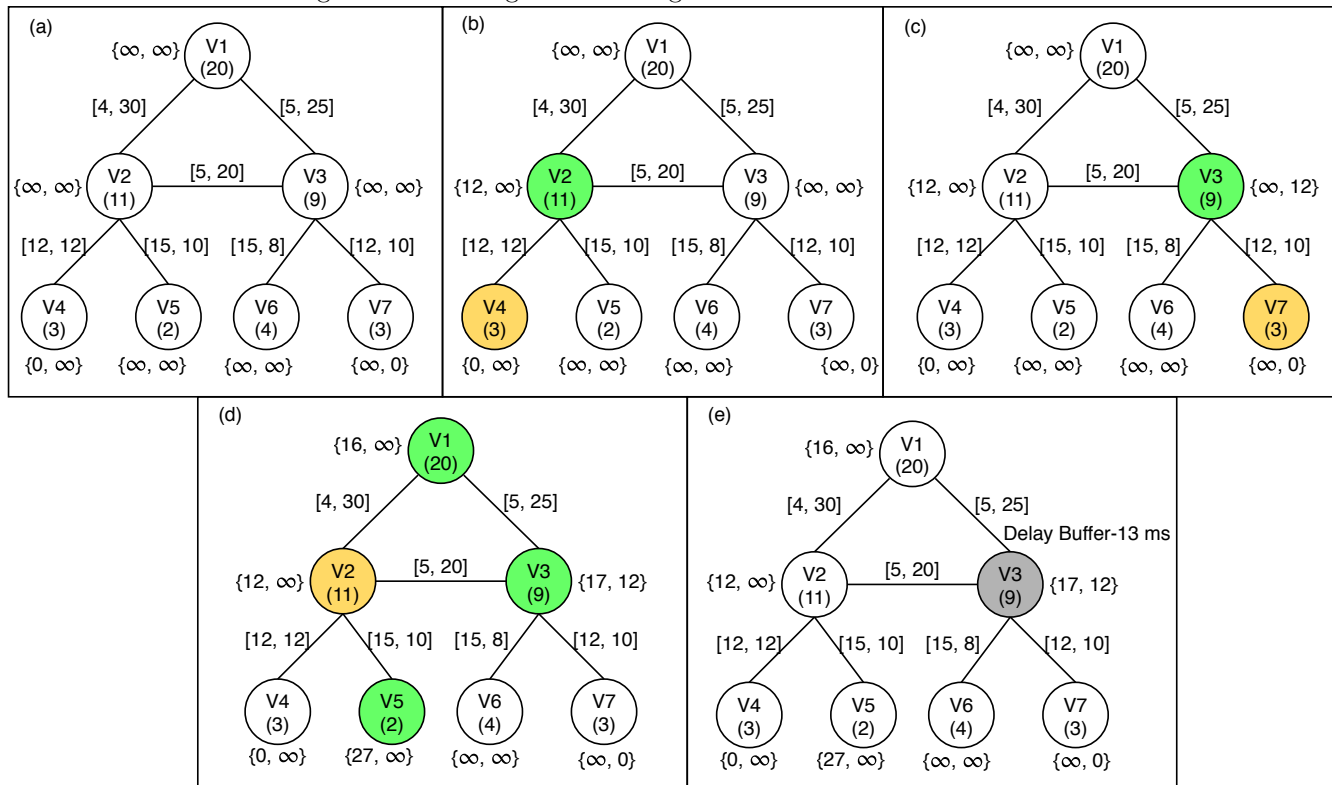
3.4 Working Illustration

The Figure 3.1 shows a working illustration of the Placement Algorithm (line 2-23). Consider a SLA agreement as shown in the Table 3.2 to be placed over the network. Fig 3.1(a) shows the network status where each node represents a Hypervisor and the edges are the interconnections between them. The three-layered architecture represents the nodes at the Edge, Fog and Cloud with resources allocated relatively. Also, it shows the data-structures used by the algorithm (line 2-6). Fig 3.1(b) & Fig 3.1(c) shows the 1st and 2nd iteration of the algorithm from the two entry-points $V4$ & $V7$ respectively and the updated data-structures. In the next iteration Fig 3.1(d), the node with the minimum latency from entry-point $V4$ is selected (line 9) and its neighbor nodes $V1$, $V3$ and $V5$ are considered as they satisfy the SLA constraints (line 10-14). As $V3$ has been visited by all the entry-points it is chosen for placement of the first VNF of the Service Chain (line 15-17) as shown in Fig. 3.1(e). Finally, the Latency Buffer of 13 ms is calculated.

Table 3.2: SLA Agreement

Parameter	Value
Maximum Delay tolerated	30 ms
Minimum Throughput Required	10 Gbps
Hypervisor RAM Required	2 GB
Hypervisor CPU Cores	1
List of Entry-Exit Points	V4, V7
Service Chain	Firewall

Figure 3.1: Finding centre during SLA Placement



[x, y] - x ms Link Latency, y Gbps Available Link Bandwidth, {a, b} - Latency of 'a' and 'b' from the End-points 'V4' and 'V7' respectively, Available CPU RAM on Hypervisors is represented by (z) GB.

Algorithm 1 Placement of VNF

```
1: procedure PLACEMENT( $G, U, L_{tol}, BW_{req}, RAM_{req}$ )
2:   for  $v \in G(v)$  do
3:      $Seen_v \leftarrow \emptyset$ 
4:   end for
5:   for  $u \in U$  do
6:      $d_u[u] \leftarrow 0$ 
7:      $Seen_u \leftarrow \{u\}$ 
8:     Enqueue( $Q_u, u$ )
9:   end for
10:  while  $Q_u$  is not Empty for each  $u \in U$  do
11:     $v \leftarrow ExtractMin(Q_u)$ 
12:    for  $x \in Adj(v)$  do
13:      if  $d_u[v] + L_{v,x} \leq L_{tol}$  AND  $BW_{v,x} \geq BW_{req}$  AND  $RAM_x \geq RAM_{req}$  then
14:         $d_u[x] \leftarrow d_u[v] + L_{v,x}$ 
15:        Enqueue( $Q_u, x$ )
16:         $Seen_x \leftarrow Seen_x \cup \{u\}$ 
17:        if  $|Seen_x| \equiv |U|$  then
18:           $Start \leftarrow x$ 
19:          break
20:        end if
21:      end if
22:    end for
23:  end while
24:  if  $Start$  not found then:
25:     $Place(v, Cloud) \forall v \in SC$ 
26:  else
27:     $Place(SC\{1\}, Start)$ 
28:     $DB \leftarrow L_{tol} - max(L_{u, Start})$ 
29:     $Prev \leftarrow Start$ 
30:    for  $v \in SC - SC\{1\}$  do
31:       $minDelayNbr \leftarrow \emptyset$ 
32:      for  $y \in Adj(Prev) \cup \{Prev\}$  do
33:        if  $2 * L_{x,y} \leq DB$  AND  $BW_{x,y} \geq BW_{req}$  AND  $RAM_y \geq RAM_{req}$  then
34:           $minDelayNbr \leftarrow min(minDelayNbr, 2 * L_{prev, minDelayNbr})$ 
35:        end if
36:      end for
37:      if  $minDelayNbr \neq \emptyset$  then
38:         $Place(v, minDelayNbr)$ 
39:         $DB \leftarrow DB - 2 * L_{x,y}$ 
40:         $Prev \leftarrow y$ 
41:      else
42:         $Place(v, Cloud) \forall v \in SC$ 
43:      return
44:    end if
45:  end for
46:  end if
47: end procedure
```

Chapter 4

Bottlenecks and its Detection

Some Network resources may be over-utilized or under-utilized depending on the placement of VNFs. Over utilization may affect the placement of future SLA requests. In the worst case, even if the resources are available at the Edge or Fog layer, placement of VNFs may not be possible because of some SLA constraint violation. This would result in the placement of VNFs at the Cloud, thus not utilizing the benefits of Edge or Fog layers. Our algorithm does not focus on an optimal use of Network resources, but aims at the placement of VNFs in the proximity of the User devices. A Hypervisor can be interpreted as a node, hence both are used interchangeably during explanation.

Suppose there is a tremendous increase in the number of requests for particular VNFs resulting in high CPU utilization at the node and increased load over the links from the user to the VNF. This may affect the other VNFs either utilizing some of those links or placed at the same node which may cause violation of some SLA constraint. Thus, a bottleneck may be created due to high-utilization of any of the network resources which may lead to the violation of one or more SLA(s). Bottleneck may be either over a link or in the node.

To avoid bottlenecks, appropriate measures needs to be taken which could be either proactive or reactive. No SLA agreement should be violated at any point of time. Taking reactive measures after SLA violation is thus not a choice. Hence, proactive measure would require detection of a possible bottleneck in the future using a Threshold-based detection. The cost of having threshold is the under-utilization of the resources but it would not lead to an SLA violation. Table 4.1 shows the classification of bottlenecks and possible actions for their removal.

We consider node bottlenecks due to high CPU utilization of the Hypervisor and perform migration of one or more VNFs from that Hypervisor to reduce the CPU load.

Table 4.1: Classification of Bottlenecks

Bottleneck	Cause	Action
High CPU Utilization of Hypervisor	Some VNFs in the Hypervisor may get heavily utilized increasing the CPU utilization at the Hypervisor thus impacting the performance of other VNFs.	Migration of one or more VNFs in the Hypervisor to reduce the CPU Load.
High Link Throughput	Increase in traffic for a particular SLA may lead to high load over the links of the VNFs. This is in addition to the other Service Chains utilizing the same links.	Select VNFs of the Service Chain in decreasing order of their utilization of the link. Create copies of the subsequent VNFs on both sides of the link.
Increase in Delay over a Link	Same as above. Due to congestion, the delay over a link may increase which may lead to violation of SLAs.	Same as above.

Chapter 5

Bottleneck Removal Algorithm

5.1 Overview

The algorithm removes potential node bottlenecks by migrating VNFs from the bottlenecked node. It also avoids ping-pong effect of traffic in the network because of the Service Chain. The algorithm takes as input the node detected for a possible bottleneck. It reduces the CPU load by migrating one or more VNFs in the network.

The algorithm takes into account multiple cases during migration without the violation of the SLA constraints. Also, to avoid looping, it considers migrating part of Service Chain of VNFs, if possible to the neighbor node hosting either the predecessor or successor of that part of Service chain.

5.2 Algorithm

The proposed algorithm considers three cases of migration which takes place one after the other, if required. The first case aims to migrate VNF to a neighbor node having both sufficient resources and satisfying the SLA constraints. If the resources are not available at the neighbor, then a VNF is migrated from the neighbor node to create resources at the neighbor. Multiple VNFs are thus migrated from both the bottlenecked node and its neighbor node. Finally, if migration is not possible using the above two cases then a VNF along with all the other VNFs of its SLA are moved to the Cloud.

Choice of Node to Migrate - Priority Score

Choosing the victim VNF to be migrated and the node to which the migration would take place is calculated using Priority Score. This takes into account the link characteristics between the bottlenecked node and its neighbor node along with the node characteristics of the neighbor. The formula for Priority Score for migrating VNF v from node r to s is

$$PS(v, r, s) = \alpha.R_r + \beta.BW(r, s) + \gamma.L(r, s)$$

where R_r is the available RAM on the node r , $BW(r, s)$ and $L(r, s)$ are the available link bandwidth and link latency between r and s respectively. α , β and γ are the normalization constants.

For normalizing different parameters, the parameter is divided by its maximum value achieved. For example, the maximum values for R_r and $BW(r, s)$ are known initially. Higher the available RAM at the node and high link bandwidth increases the chances for migration but the link latency decreases it. So, higher the Priority score, more feasible is the migration.

Avoid Looping of Traffic due to Service Chain

In cases, where a part of Service chain is installed on the same node, then preference is given to the VNFs at the two ends of this part of the Service chain to avoid looping of traffic as shown in Fig.5.1. Also, nodes hosting the predecessor or successor of this Service chain is more preferred, shown in Fig.5.2.

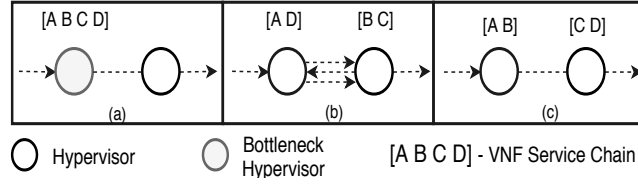


Figure 5.1: Migration of successive VNFs of Service Chain

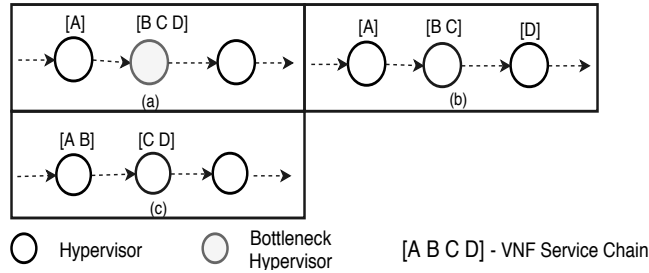


Figure 5.2: Migration to nodes containing part of Service Chain

Pseudocode

The Algorithm 2 presents the pseudocode of the Bottleneck Removal Algorithm and the Table-No. shows the Notations used in the pseudocode. Choosing a victim VNF and a neighbor node is shown in line(2-8). If the migration does not violate any SLA constraint then Priority Score is maintained line(8). If no migration is possible due to insufficient resources at the neighbor nodes, then resources are created by migrating VNF from the neighbor node. So, victim VNF and neighbor nodes are considered at both the bottlenecked node and its neighbor nodes (line 9-16). The Priority score for this multiple migration is updated (line15). After this, if there is no choice of migration, then the network is assumed to be fully utilized and hence a victim VNF along with all other VNFs of its SLA are moved to the cloud.

Algorithm 2 Bottleneck Removal

```
1: procedure BOTTLENECK_REMOVAL(Hypervisor r)
2:   for each VNF  $v \in \text{VNFs}(r)$ :
3:     for Hypervisor  $s \in \text{Adj}(r)$  do
4:       if  $s$  satisfies the Constraints of moving ' $v$ ' then
5:         1. Available bandwidth along link  $r$  to  $s$ 
6:         2. Within Latency Buffer
7:         3. New CPU utilization  $\leq$  Threshold
8:         Score[ $v, r, s$ ]  $\leftarrow \zeta_{v,r,s}$ 
9:       end if
10:    end for
11:    If No Score:
12:    for for each VNF  $v \in \text{VNFs}(r)$  do
13:      for Hypervisor  $s \in \text{Adj}(r)$  do
14:        for for each VNF  $x \in \text{VNFs}(s)$  do
15:          for Hypervisor  $p \in \text{Adj}(s)$  do
16:            if  $p$  satisfies the Constraints of moving ' $x$ ' AND  $s$  satisfies the Constraints of
            moving ' $v$ ' then
17:              PS[ $v,r,s$ ]  $\leftarrow \max(\text{PS}[v,r,s], \zeta_{v,r,s} + \zeta_{x,s,p})$ 
18:            end if
19:          end for
20:        end for
21:      end for
22:    end for
23:    If atleast one Score
24:    Return solution with the maximum Score
25:    Else
26:    Return 'Move any SLA to the Cloud'
27: end procedure
```

5.3 Complexity

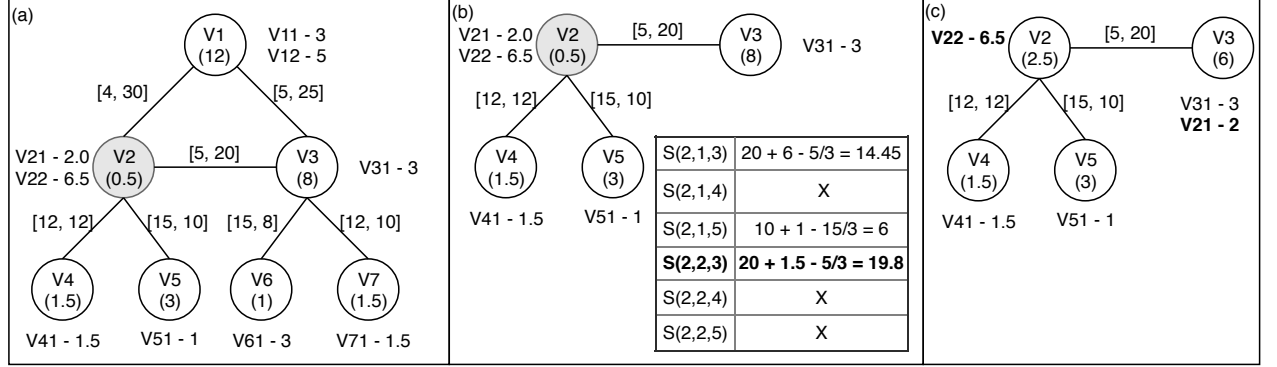
For a network having V - Edge or Fog nodes and E - links between them. Also, N is the number of VNFs installed on every node. Migration to a neighbor node would require selecting a victim VNF and a neighbor node, $O(N.E)$. For migration to neighbor by creating resources at the neighbor would lead to multiple VNF migration at different nodes. Hence, selecting the victim VNF and the neighbor at multiple place would involve a complexity of $O(N^2.V^2)$. Finally, migrating a VNF to cloud is about selecting a victim VNF, $O(V)$. Complexity of the algorithm is

$$O(N^2.V^2)$$

5.4 Working Illustration

5.4.1 Migration to Neighbor

A working illustration for Case-1 of the Migration algorithm is shown in Fig 5.3. The Fig5.3(a) shows 4 Edge nodes, 2 Fog nodes and a cloud node and the current network status, also a possible bottleneck is detected at node- V2. Fig5.3(b) shows the calculation of priority scores for moving a VNF from V2 to its neighbor nodes V3, V4 and V5. The table shows the calculation of the priority scores. For simplicity, the values of α , β and γ are +1, +1 and $-1/3$ respectively. The result of migration is based on the solution having the highest priority score and is shown in Fig.5.3(c).

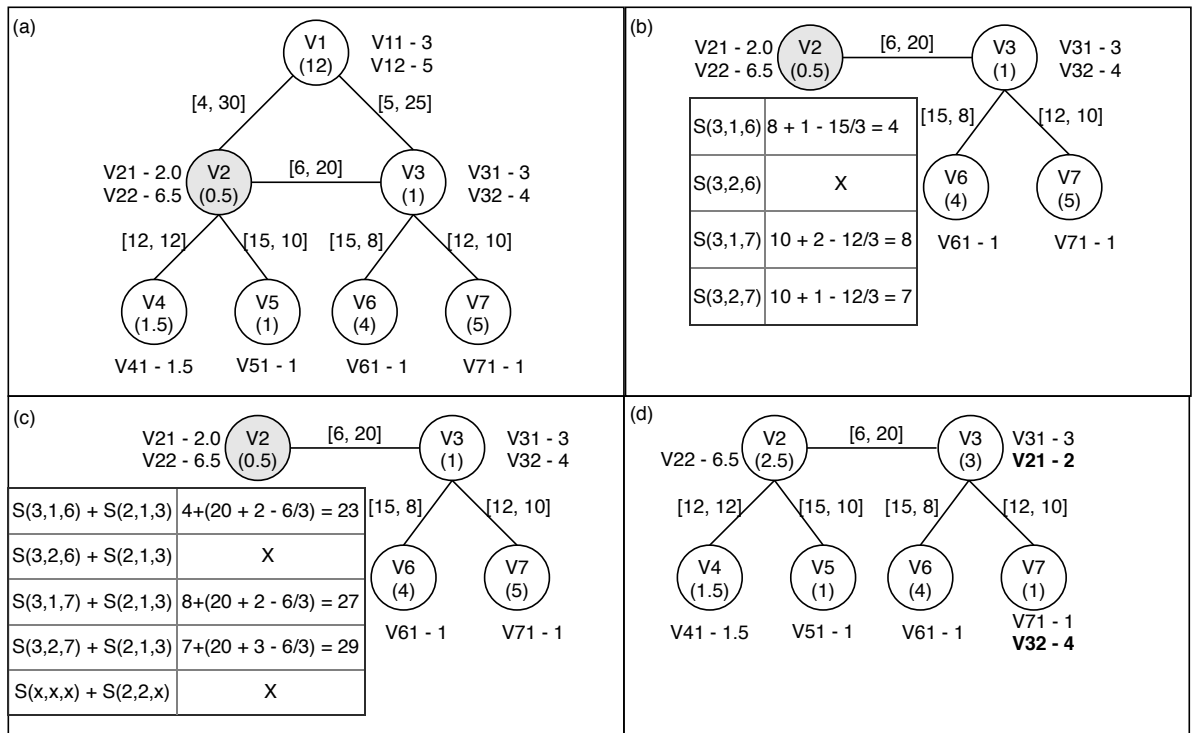


[x, y] - x ms Link Latency, y Gbps Available Link Bandwidth, Available CPU RAM on Hypervisors is represented by (z) GB, Vpq - Hypervisor 'p' hosting VNF 'q', S(a,b,c) - Priority Score of moving VNF-'b' at Hypervisor-'a' to Hypervisor-'c'.

Figure 5.3: Migration Algorithm - Case 1

5.4.2 Migration by creating resources at the Neighbor

Fig.5.4 shows the Case-2 of the Migration algorithm. The topology of the network is same as Fig.5.3 but the current network status is changed. No VNF can be migrated to a neighbor node of V2 Fig.5.4(a). Hence, resources are created at neighbor nodes V3, V4 and V5. Fig.5.4(b) shows the priority score of migrating a VNF from node V3 to its neighbor nodes. Nodes V4 and V5 are not considered as they do not have any other neighbors. As a VNF is migrated from node V3, resources are created at this node, which is considered for migrating a VNF from the bottlenecked node-V2. Fig.5.4(c) shows the possible solutions and their priority scores. The solution having the highest priority score is applied to the network Fig.5.4(d).



[x, y] - x ms Link Latency, y Gbps Available Link Bandwidth, Available CPU RAM on Hypervisors is represented by (z) GB, Vpq - Hypervisor 'p' hosting VNF 'q', S(a,b,c) - Priority Score of moving VNF-'b' at Hypervisor-'a' to Hypervisor-'c'.

Figure 5.4: Migration Algorithm - Case 2

Chapter 6

Implementation

The implementation is a proof-of-concept Emulated System of DNF using Ryu controller [5] and Ubuntu 16.04 VMs as Hypervisors on a Ubuntu 16.04 Host Machine with 32 GB RAM. OVS-Switches [6] are used for interconnection between the Hypervisors and are connected to the controller. Both VNF(s) and Users are implemented as Docker containers [7] to be installed on the Hypervisors. No VNFs of the SLA are placed on the same Hypervisor hosting the User Container. The Docker containers used as VNF are either a Centos container [8] or Kali-Linux container [9]. The Users are Iperf Containers [10] to generate Traffic. SLAs of different Service Chain length are created randomly with Users at different Edge Hypervisors. A Hypervisor and a OVS-switch is connected on a one-to-one basis. Interconnections between the Hypervisors are actually interconnections between the their respective OVS-switches.

Figure 6.1 and Table 6.1 shows the implementation topology and setup parameters for the proof-of-concept.

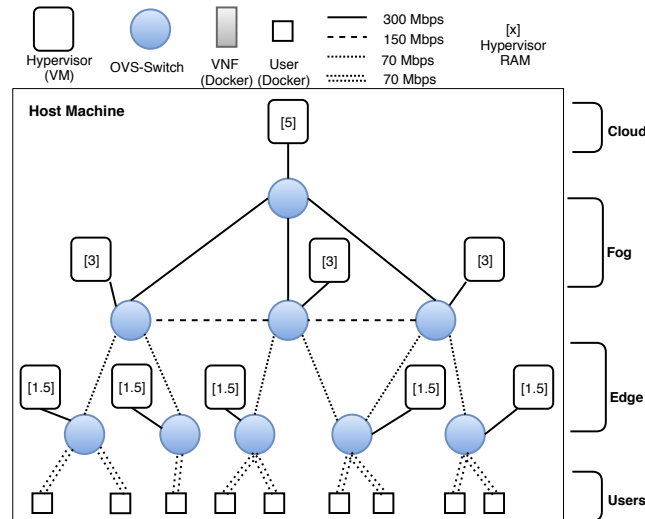


Figure 6.1: *Implementation Topology*

Table 6.1: Emulation Setup

Parameter	Value
Number of Edge Hypervisors	5
Number of Fog Hypervisors	3
Number of Cloud Hypervisors	1
Number of OVS-Switches	9
Number of Hypervisors per OVS-switch	1
Host Machine RAM	32 GB
Cloud Hypervisor RAM	5 GB
Fog Hypervisor RAM	3 GB
Edge Hypervisor RAM	1.5 GB
Fog-Cloud Link Bandwidth	300 Mbps
Fog-Fog Link Bandwidth	150 Mbps
Edge-Fog Link Bandwidth	70 Mbps
Traffic generated by Users	50 Mbps
User Docker Container	IPerf or Centos
VNF Docker Container	Centos or Kali-Linux
Hypervisor RAM allocated to VNF Container	512 MB

Chapter 7

Evaluation Results

This chapter shows the evaluation of the performance of both Placement and Migration Algorithms in both simulation and emulation test bed.

7.1 Overview

Table 7.1 shows the steps involved during the Placement Procedure. Total time is the interval from the input of an online SLA request till all the VNFs along with their flow rules are installed.

Table 7.1: Time distribution during Placement Algorithm

Name	Partition
T1	Algorithm Run-time
T2	VNF Deployment Time
T3	Flow Rules installation Time
Total Time	T1 + T2 + T3

Table 7.2 shows the steps involved during the Bottleneck Removal Procedure. Migration time would consider the first three steps as the bottleneck is removed by that time. Subsequent steps may be done later.

Table 7.2: Time distribution during Migration Algorithm

Name	Partition
T1	Algorithm Run-time
T2	New VNF Deployment Time
T3	New Flow Rules installation Time
T4	Stopping of Old VNF(s)
T5	Removal of Old Flow Rules Time
Total Time	T1 + T2 + T3

Due to the resource constraints of the Host machine, limited number of SLAs could be deployed. To evaluate the algorithms on a large scale, simulation was performed on a Ubuntu 16.04 with 32 GB RAM. Neither VNFs were deployed nor were Flow rules installed. Hypervisors were considered as nodes with different characteristics as per the Edge-Fog-Cloud architecture and their interconnections were edges. The implementation topology remains the same as in Emulation setup.

7.2 Cumulative Placement Time

In this experiment, we plot the cumulative Placement time of different number of SLAs for specific lengths of Service Chain.

7.2.1 Simulation

Figure 7.1 shows with increase in the number of SLAs, the Placement time increases. Also, as the Service Chain length increases, the Placement time also increases. This increase is due to network resources getting utilized as more VNFs/SLAs are installed over the network. Hence, subsequent placement are placed at the upper layers and require more computation. Table 7.3 shows the simulation parameters for this experiment.

Table 7.3: Simulation Setup

Parameter	Value
Cloud Hypervisor RAM	512 GB
Fog Hypervisor RAM	64 GB
Edge Hypervisor RAM	16 GB
Cloud Hypervisor CPU Cores	1024
Fog Hypervisor CPU Cores	32
Edge Hypervisor CPU Cores	8
VNF RAM Requirement	2 GB
VNF CPU Utilization	2 %
VNF CPU Cores Requirement	1

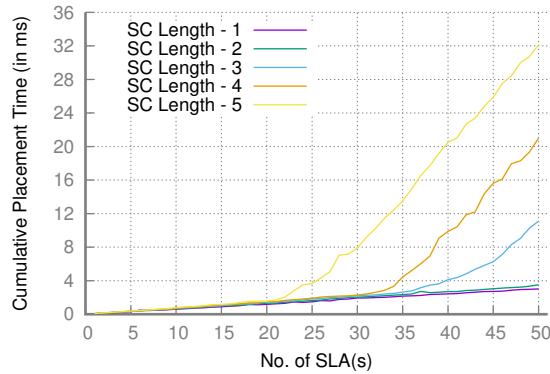


Figure 7.1: Cumulative Placement Time of SLAs in Simulation Setup

7.2.2 Emulation

Figure 7.2 and Figure 7.3 shows the cumulative Placement time in the Emulated setup for SLA hosts placed at Dedicated and Random Edge Hypervisors. With dedicated user entry points, the placement algorithm would converge at the same point and would require comparison of the same conditions for every SLA, hence uniformly increase in placement time. The other conclusions remain the same as in the case with simulation.

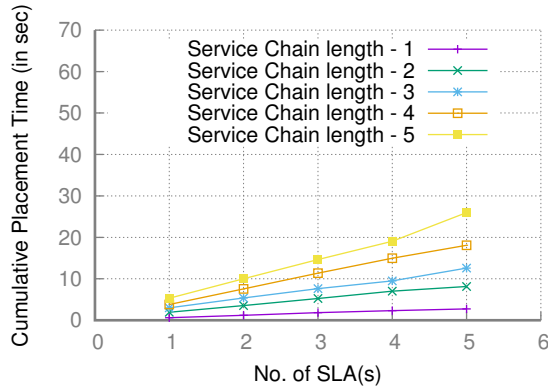


Figure 7.2: *Cumulative Placement Time of SLAs in Emulation Setup for Dedicated User-End points in SLA*

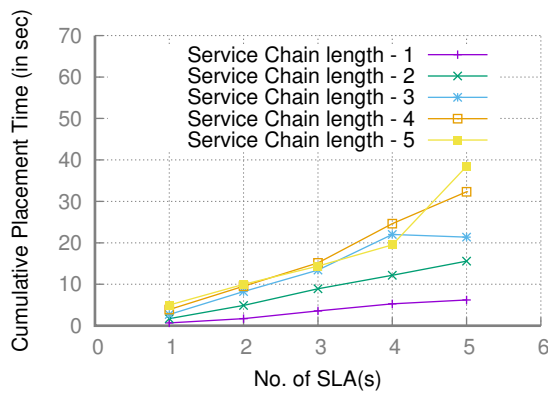


Figure 7.3: *Cumulative Placement Time of SLAs in Emulation Setup for Random User-End points in SLA*

7.3 Placement Ratio Over Edge-Fog Network

In this experiment, we plot the ratio of SLAs that are placed over Hypervisors at the Edge and Fog layers with the total number of SLAs to be placed for different Service Chain lengths.

7.3.1 Simulation

Figure 7.4 shows with the increase in the number of SLAs, lesser number of SLAs get placed at the Edge-Fog layer. Also, increasing the Service Chain length would decrease the SLAs placed at the Edge-Fog layer. Initially, as VNFs/SLAs get placed the network resources at the Edge and Fog layers get utilized. For subsequent VNFs/SLAs, the resources at the Edge and Fog layers get exhausted and thus are placed at the Cloud. The simulation setup remains the same as in Table 7.3.

7.3.2 Emulation

Figure 7.2 and Fig. 7.3 shows the Ratio of SLAs placed over the Edge and Fog Layers in Emulated System for SLA hosts placed at Dedicated and Random Edge Hypervisors.

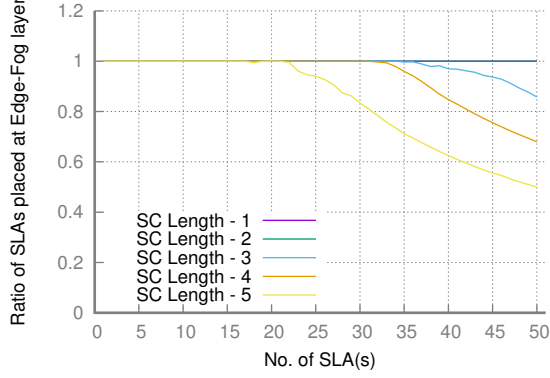


Figure 7.4: *Simulation - Ratios of SLAs placed over Edge&Fog Layers*

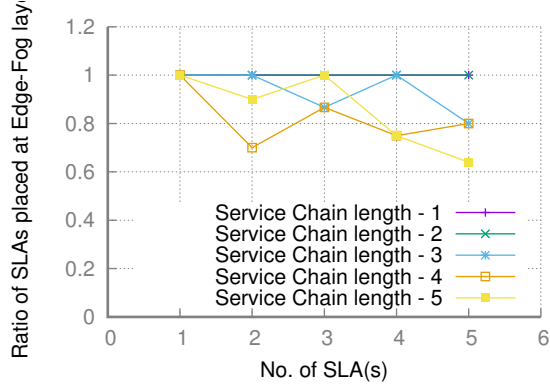


Figure 7.5: *Ratios of SLAs placed over Edge and Fog Layers for Dedicated User entry points*

7.4 CDF of the Bottleneck Removal Algorithm

In this experiment, we evaluate the Bottleneck removal time for each Bottleneck in the network. Also, the CDF shows the frequency of Bottlenecks getting handled by the different cases as mentioned in Migration Algorithm.

7.4.1 Simulation

Large number of SLAs of Service length 3 were placed over the network. Later every node in the Network was checked for possible bottleneck. The detection was based on violation of the node constraints. Table 7.4 shows the simulation parameters for this experiment.

Figure 7.7 shows that for lower number of SLAs installed over the network, the frequency of Bottlenecks is less and are handled by Case-1 (Migrating a VNF at a Neighbor). When more SLAs are installed, more Bottlenecks are created which require more time and are removed by Case-3 (Migrating an entire SLA to the Cloud). This is because as more SLAs are installed over the network, resources are utilized at the Edge and Fog layers. So, when a bottleneck is detected, no VNF at the Bottlenecked Hypervisor can be migrated(both Case-1 and Case-2 are not applicable). Hence, a VNF at that Hypervisor along with all other VNFs of the Service Chain are migrated to the Cloud.

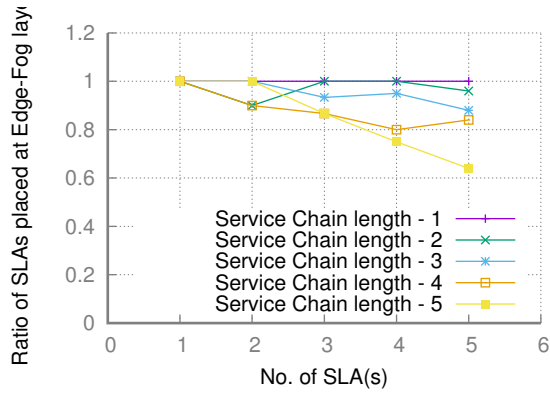


Figure 7.6: Ratios of SLAs placed over Edge and Fog Layers for Random User entry points

Table 7.4: Simulation Setup

Parameter	Value
Cloud Hypervisor RAM	512 GB
Fog Hypervisor RAM	64 GB
Edge Hypervisor RAM	32 GB
Cloud Hypervisor CPU Cores	1024
Fog Hypervisor CPU Cores	64
Edge Hypervisor CPU Cores	32
VNF RAM Requirement	2 GB
VNF CPU Utilization	2 %
VNF CPU Cores Requirement	1

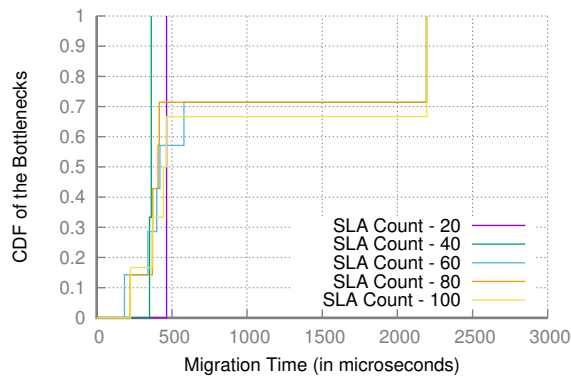


Figure 7.7: CDF of the Bottlenecks with Bottleneck Removal Time

Chapter 8

Conclusion

In this work, we have explored the handling of online SLA requests from Service Providers by placing the VNFs closer to the User devices which reduces both the Network latency and the load on the network. Also, there may be situations where bottlenecks may be created in the network. Such scenarios are detected proactively and measures were taken to handle it so as not to violate any SLA agreement.

The Placement algorithm placed the VNFs closer to the user devices by placing VNFs of smaller instances at the Edge and the larger or heavy instances at the upper layers. On detecting a possible Node bottleneck based on a Threshold based system, they were removed using the Bottleneck Removal algorithm by migration of VNFs. The framework for this work can handle multiple Hypervisors which can be geographically distributed.

References

- [1] P. Naik, D. K. Shaw, and M. Vutukuru. NFVPerf: Online performance monitoring and bottleneck detection for NFV. *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)* 154–160.
- [2] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gasparry. Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions. *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)* 98–106.
- [3] S. Ahvar, H. Pann Phyu, S. Buddhacharya, E. Ahvar, N. Crespi, and R. Glitho. CCVP: Cost-efficient Centrality-based VNF Placement and chaining algorithm for network service provisioning .
- [4] R. Cohen, L. Lewin-Eytan, J. Seffi Naor, and D. Raz. Near optimal placement of virtual network functions 1346–1354.
- [5] Ryu SDN Controller. <https://osrg.github.io/ryu/>. [Online].
- [6] OVS Switch. <https://www.openvswitch.org/>. [Online].
- [7] Docker container. <https://www.docker.com>. [Online].
- [8] Centos Docker container. https://hub.docker.com/_/centos/. [Online].
- [9] Kali-Linux Docker container. <https://hub.docker.com/r/kalilinux/kali-linux-docker/>. [Online].
- [10] Iperf container. <https://hub.docker.com/r/networkstatic/iperf3/>. [Online].