Efficient VNF Placement

Introduction:

In 5G networks, slicing plays a major role in making a service-customized network as a one-size fits all approach becomes highly inefficient.

The throughput degrades from 12.36% to 50.30% as more VNFs are consolidated on the same server. It therefore becomes important to take VNF interference into consideration while deploying the VNFs.

Here we tackle the problem of placing the VNFs of a request in the topology considering VNF interference in order to maximize the total effective throughput and maximize the total number of accepted requests.

3 algorithms (Shortest Path Heuristic, Greedy on Used Servers, our new algorithm) are implemented and their performance compared.

Currently the Interference coefficients are taken by the understanding of the contention of different types of VNFs but we plan to perform real time testing to get realistic coefficients.

Description:

For all the three algorithms:

1. Sort all requests in descending order (based on the ratio of requested throughput and resource consumption of VNFs)
2. For every request:
    a. Check if there are any VNFs in the current request which are shareable
    b. For each of these VNFs check if any node in the graph already host these VNFs and have sufficient resources to accommodate this shareable VNF
    c. Apply the multi-stage algorithm considering the nodes which run a particular shareable VNF as a stage
    d. Compute the minimum delay paths between every 2 nodes of this multi-stage graph using the Dijkstra's algorithm

SPH:

1. **Try provisioning the VNFs close to the source preferentially and move towards the destination**
2. Calculate VNF delay, interference delay while placing the VNFs along the path, if it violates any of the existing requests already placed on the path or if it violates the end-end delay, then reject this request
3. After successfully placing the request, update the graphs and the nodes

GUS:

1. **Try provisioning the requests on the nodes of the path which are highly loaded**
2. Calculate VNF delay, interference delay while placing the VNFs along the path, if it violates any of the existing requests already placed on the path or if it violates the end-end delay, then reject this request
3. After successfully placing the request, update the graphs and the nodes

Our Algorithm:

1. **Try provisioning the VNF on the nodes with minimum interference with respect to this VNF**, if while placing the VNF existing requests get violated then we skip this node and try to place this VNF on the node which incurs the second minimum interference
2. If the end-end delay for the current request, after selecting the nodes for the requests VNFs, then we reject this request
3. After successfully placing the request, update the graphs and the nodes

Interference is calculated according to the below formula:

Interference_metric =
((Sum of cpu resources consumed by $type_i$) * coefficient of vnf type (with $type_i$)) / Total cpu resources of the node

Example:

When a X type of VNF comes into the node, the total interference of the system becomes as follows:

We assume there are 3 types of VNFs already running in the node.

There are 3 fractions in the metric dictated by three coefficients

X-CPU_TYPE INTERFERENCE_COEFF
X-MEM_TYPE INTERFERENCE_COEFF
X-IO_TYPE INTERFERENCE_COEFF

Here X can be CPU_TYPE OR MEM_TYPE OR IO_TYPE

Interference_metric_NUMERATOR =
 X-CPU INTERFERENCE_COEFF * CPU Resources consumed by CPU_TYPE
+X-MEM INTERFERENCE_COEFF * CPU Resources consumed by MEM_TYPE
+X-IO_TYPE INTERFERENCE_COEFF*CPU Resources consumed by IO_TYPE

Interference_metric_DENOMINATOR= Total CPU resources available in the node

Interference_metric =
Interference_metric_NUMERATOR/Interference_metric_DENOMINATOR

The Interference_metric is guaranteed to be less than 1 always.
The interference delay is calculated as, VNF-delay*interference-metric.

The VNF delay is fixed for VNF types.

The main idea behind creating the metric in this manner was that we can adjust the coefficients according to the VNF types based on their resource contentions

	Cpu-Intensive	**Memory-Intensive**	**I/O-Intensive**
Cpu-Intensive	X	0.5X	0.3X
Memory-Intensive	0.5Y	Y	0.3Y
I/O-Intensive	0.3Z	0.3Z	Z

Fig. Interference coefficients chosen (temporarily until real time experiments are complete)

<u>Shortcomings of the previous approaches:</u>

SPH:

Since we place the VNFs preferentially close to the source which generally connects to the EDGE NODES, the delay sensitive requests may not be satisfied since the resources at the edge get consumed quickly

GUS:

Since we place the VNFs on the maximum loaded servers, later requests may create more interference (which is dynamic) which could lead to the end to end delay of the existing requests to be violated.

<u>Resolution:</u>

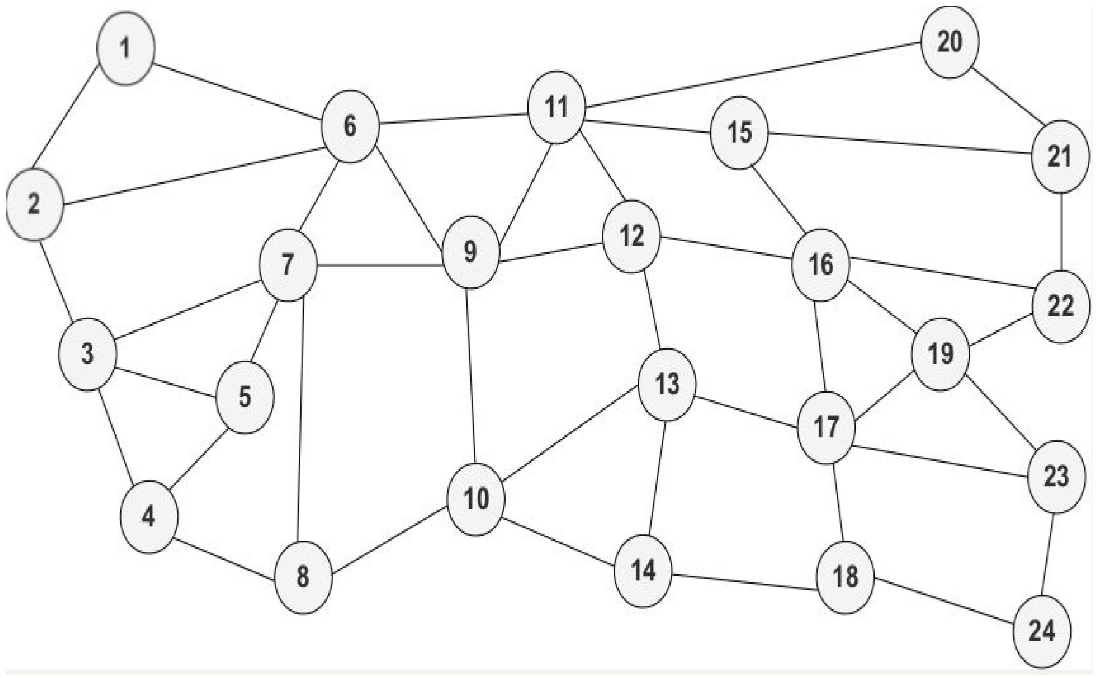
Our Algo:

- We place the VNFs of the request on the nodes which have the minimum interference while taking care that no existing request gets violated.

- Also since we use this metric while placing the request, we do not place the request at the EDGE NODES every time

Simulation and performance analysis:

The following topology was used for the simulation



USNET Topology

Core cloud is connected to 5,7,9,12,13,15,16,17,19

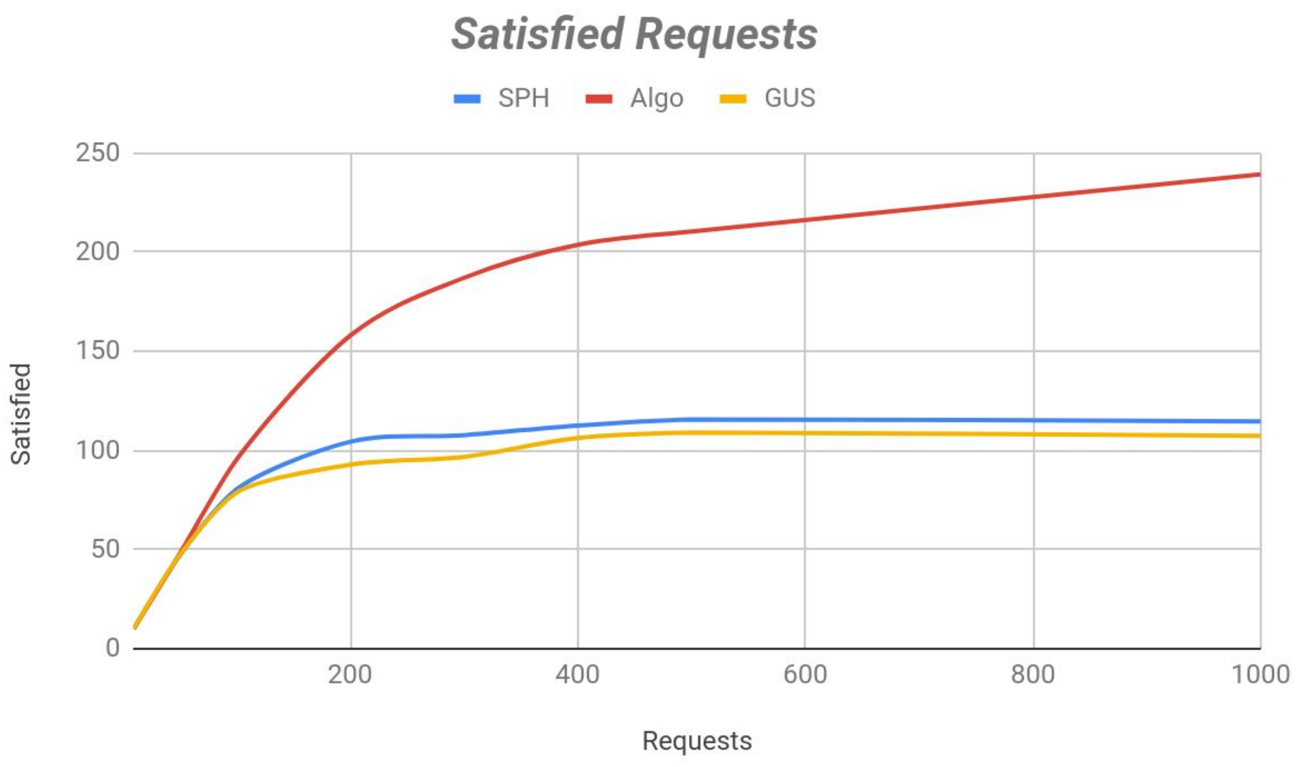
It is considered that more resources are available in the core cloud and less resources are available in the EDGE cloud. Also the link delays between edge-core cloud is more than that of edge-edge.

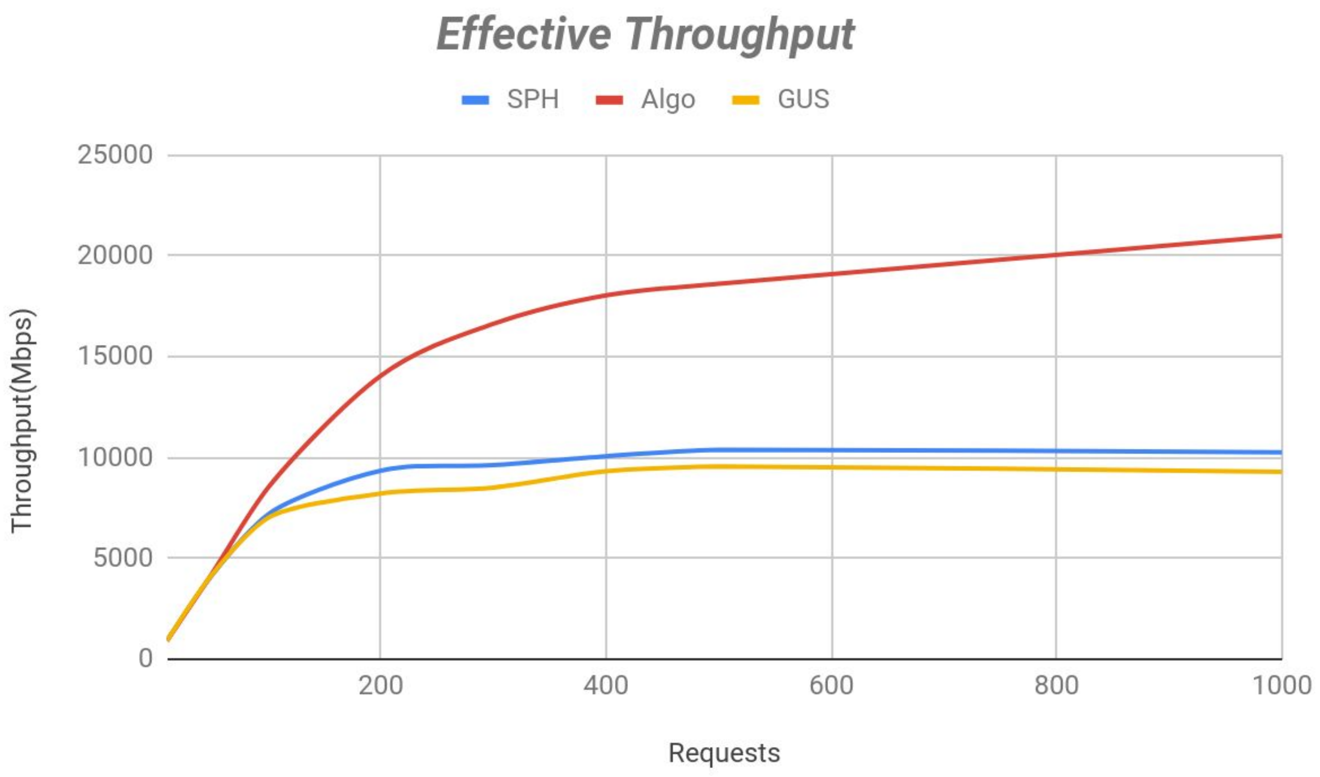
Topology	**Values**
Types of VNFs	4
Edge Node resources	20-40
Core Node resources	50-200
Link bandwidth	1 Gbps
VNF Resources	5-10
Edge-to-Edge delay	1-5 ms
Edge-to-Core delay	5-10 ms
Core-to-Core delay	10-15 ms

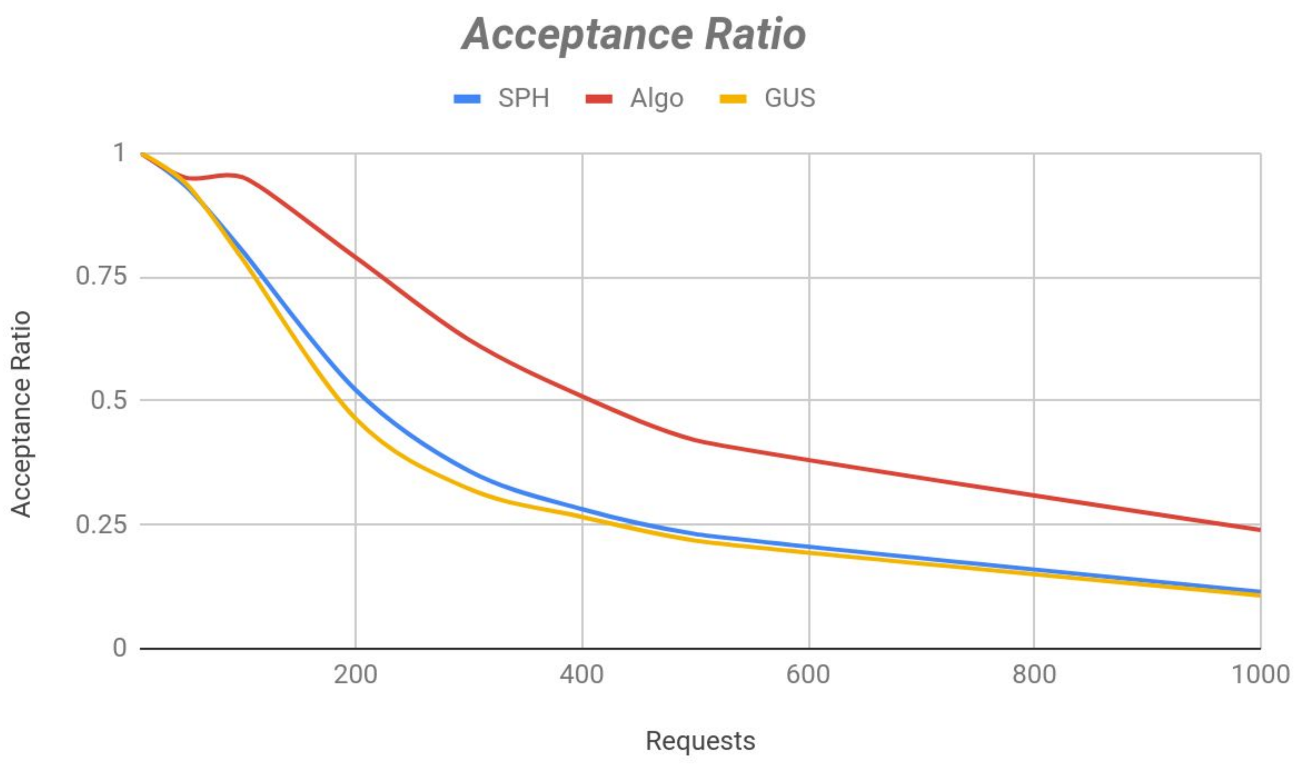
Table 1: Hyperparameters considered for the topology

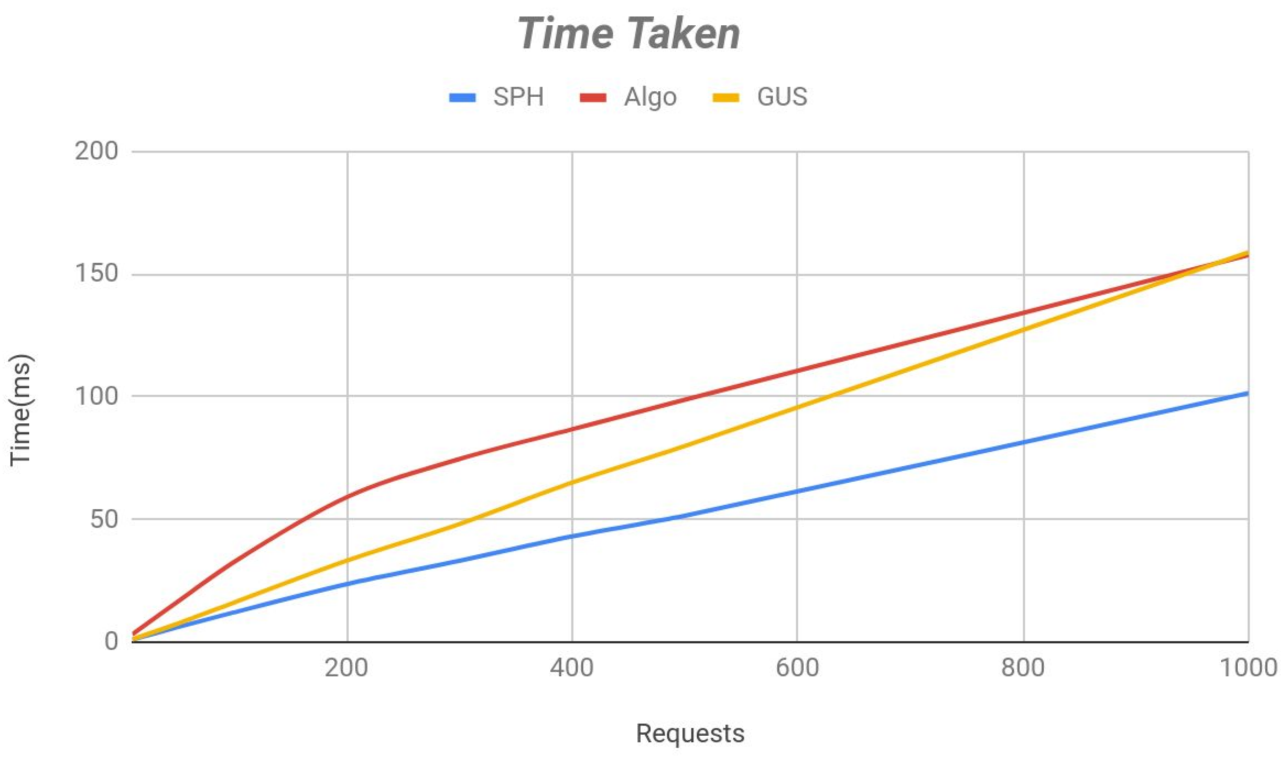
Request	**Values**
Chain Length	3
Requested resources	1-5
Requested throughput	80-100 Mbps
Requested delay	80-100 ms

Table 2: Hyperparameters considered for the request creation


Satisfied Requests
SPH
Algo
GUS
Satisfied
250
200
150
100
50
0
200
400
600
800
1000
Requests


Effective Throughput
SPH
Algo
GUS
Throughput(Mbps)
25000
20000
15000
10000
5000
0
200
400
600
800
1000
Requests


Acceptance Ratio
SPH
Algo
GUS
1
0.75
0.5
0.25
0
Acceptance Ratio
200
400
600
800
1000
Requests


Time Taken
SPH
Algo
GUS
200
150
100
50
0
Time(ms)
200
400
600
800
1000
Requests

It can be seen that our approach performs better than the existing algorithms.

These can be explained by the fact that we are taking VNF interference into account at the granularity of specific VNF types, this is important to note as not all types of VNFs interact in similarly with each other.

Few VNF types are independent in a manner that they cause very less interference and can coexist with each other on the same node.

Note: These results do not take VNF delay, interference delay, shareability of nodes for SPH, GUS and throughput degradation because of Interference into account

Conclusion & Future Work:

We have developed a heuristic based approach to maximize the total accepted throughput and the number of requests served. We do this by taking into consideration the interference incurred in a node because of the VNFs already placed there.

We plan to implement AIA and compare the performance of all the algorithms.

References:

1. Qixia Zhang, Fangming Liu and Chaobing Zeng (Huazhong University of Science and Technology, P.R. China) **Adaptive Interference-Aware VNF Placement for Service-Customized 5G Network Slices** [Infocom 2019].