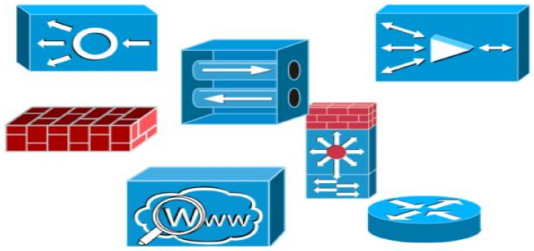
Tackling Roadblocks of Network Function Parallelization for Efficient Service Chaining

భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Network Functions Virtualization (NFV)

- Hardware (middlebox) → Software (VNFs)



Classic approach

NFV approach

Low Cost

Flexibility

Scalability

.....

Low Throughput

High Latency

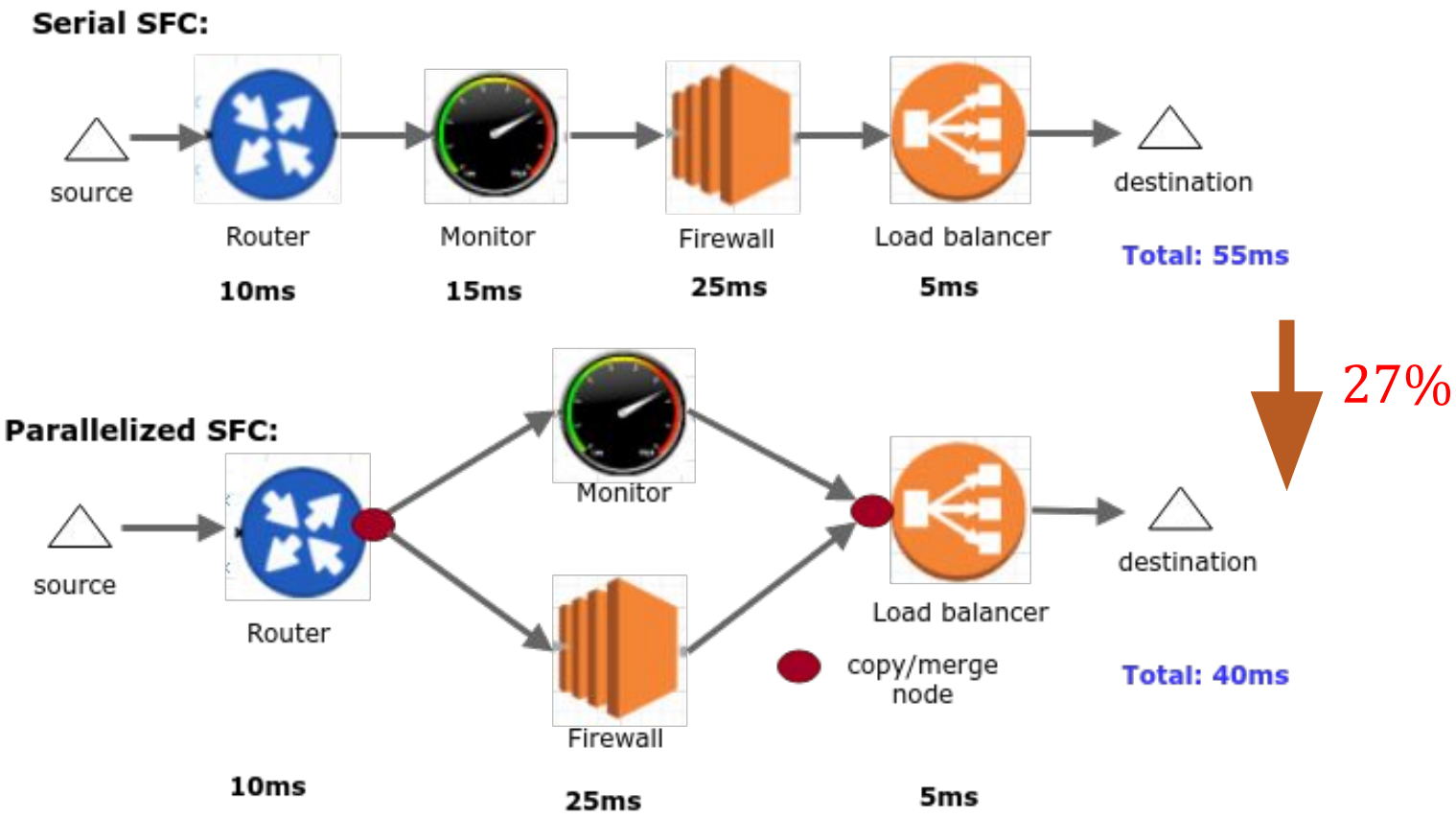
Recent Research on Reducing NFV Latency

- NFV introduces extra overhead because of virtualization layer that results in performance degradation.
- Introduced Data Plane Development Kit (DPDK) and Direct Data I/O (DDIO) technologies to reduce delay.
- Still, VNF processing is **6.4 times** slower than corresponding hardware-based implementation on average[1].
- Service Function Chain (SFC):

- SFC delay increases linearly with the SFC length.
- Network Function Parallelism is exploited to reduce NFV latency[2] .
- **53.8% network function (NF) pairs can work in parallel[2].**

[1] Sun, Chen, et al. "HYPER: A hybrid high-performance framework for network function virtualization." *IEEE Journal on Selected Areas in Communications (JSAC)*, 2017.
[2] Sun, Chen, et al. "NFP: Enabling network function parallelism in NFV.", in *Proc. of ACM SIGCOMM*, 2017.

Illustration of Serial and Parallel SFC Processing



Overheads Introduced by VNF Parallelization

1. Packet duplication/merging overhead

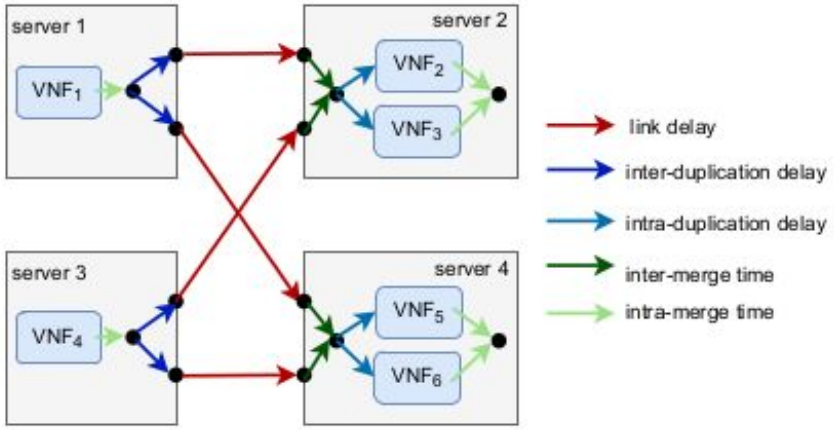


Illustration of Intra/inter-packet duplication/merging overheads

2. Packet deposition overhead

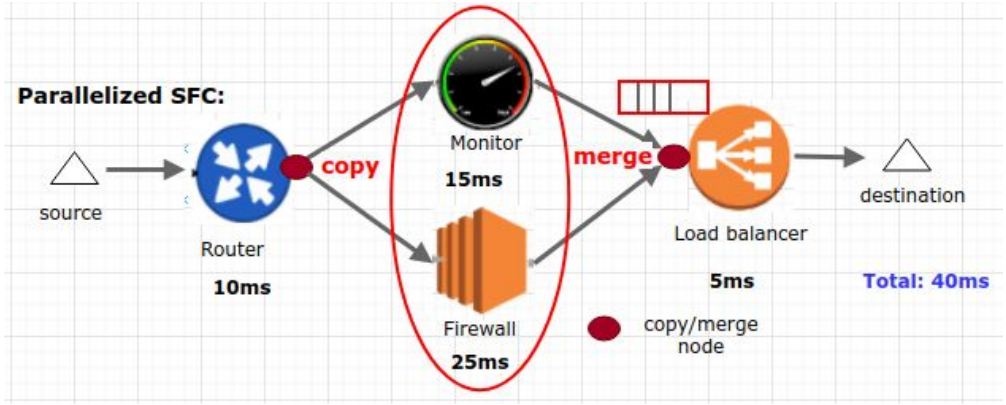
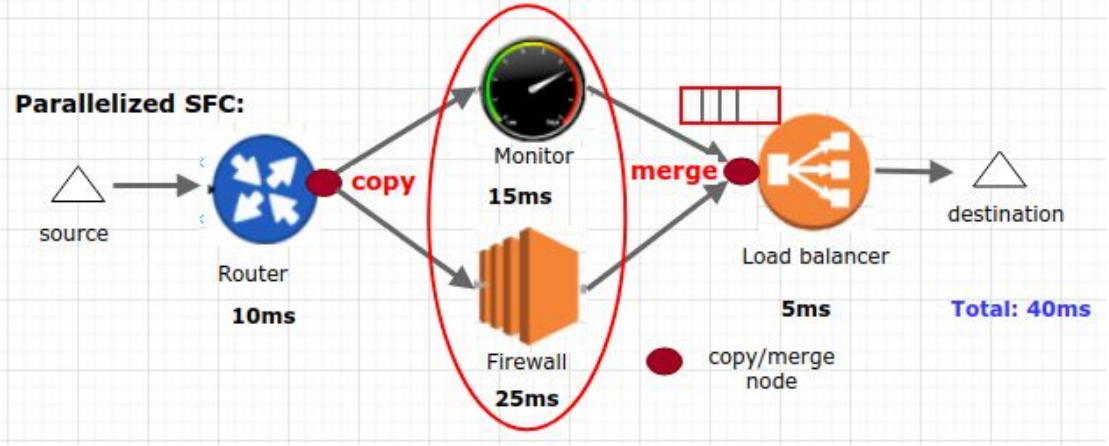


Illustration of packet deposition overheads

Limitations of Existing Studies

- Placing parallel VNFs in the same server have its limitations that server might not have sufficient capacity to accomodate.
- Ignoring overhead of buffer at merging point because some packets may arrive early. This will cause the **packet deposition problem**.



Recent Studies on Parallelized SFC Placement

- **Toward Optimal Partial Parallelism in Service Function Chaining[1]:**
  - They brought in the concept of packet copy/merge overhead but do not consider the packet deposition problem. Their algorithm strictly focuses on minimising end-to-end latency.
  - They also propose an algorithm which is exponential in complexity.

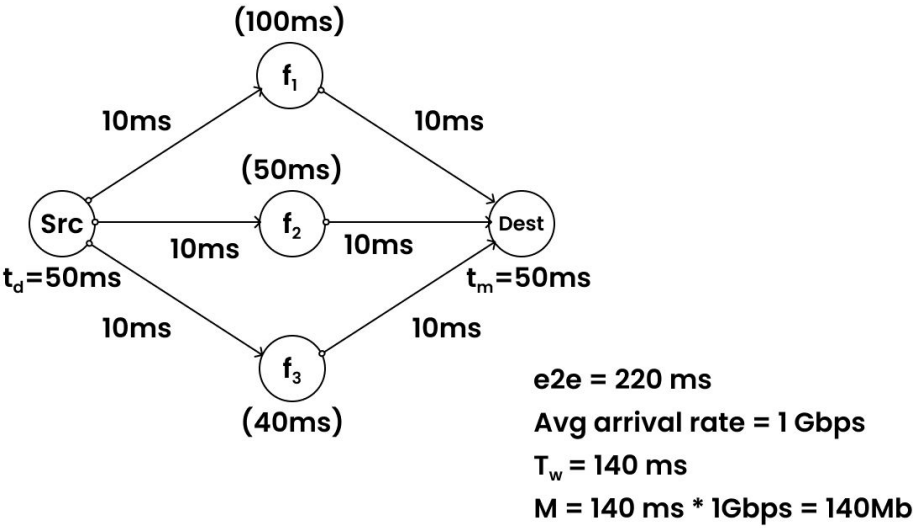
- **APPM[2]:**
  - This paper talks about deploying VNFs in a network using RL approaches. It talks about the packet deposition problem but does not take any steps to minimise the overhead..

However, none of these existing works considered the **packet deposition** and **packet copy/merge overheads jointly** while selecting parallel VNFs from the network.

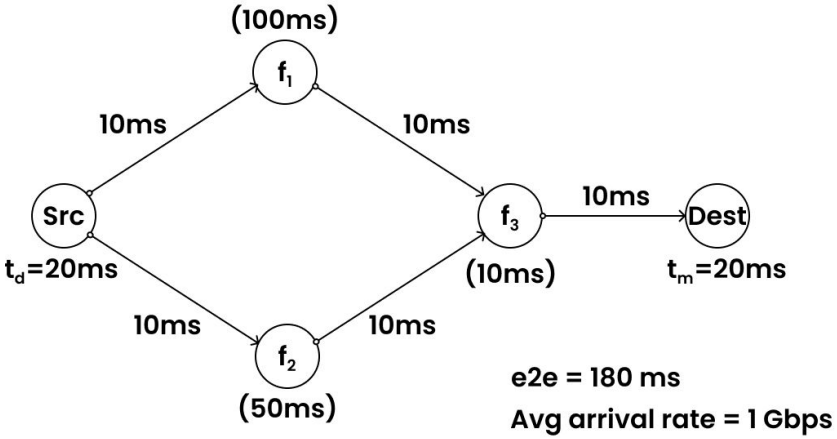
[1] I. -C. Lin, Y. -H. Yeh and K. C. -J. Lin, "Toward Optimal Partial Parallelization for Service Function Chaining," in *IEEE/ACM Transactions on Networking*, vol. 29, no. 5, pp. 2033-2044, Oct. 2021, doi: 10.1109/TNET.2021.3075709.

[2] J. Cai, Z. Huang, L. Liao, J. Luo and W. -X. Liu, "APPM: Adaptive Parallel Processing Mechanism for Service Function Chains," in *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1540-1555, June 2021, doi: 10.1109/TNSM.2021.3052223.

Motivation



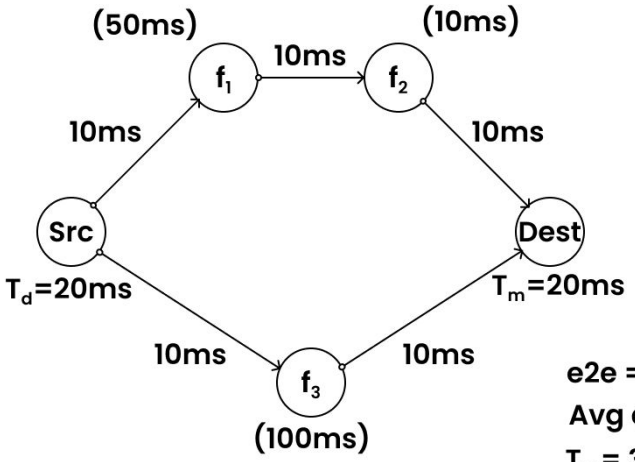
e2e = 220 ms
Avg arrival rate = 1 Gbps
T_w = 140 ms
M = 140 ms * 1Gbps = 140Mb



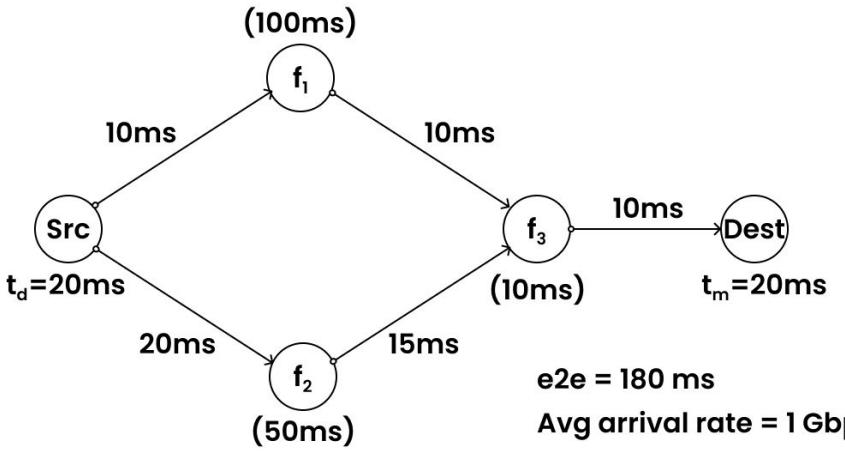
e2e = 180 ms
Avg arrival rate = 1 Gbps
T_w = 50 ms
M = 50Mb

- In the case of full parallelism, it can be seen that the e2e latency and delay difference is high. This is due to the fact that three copies had to be made which significantly increased the latency. This is an overhead that cannot be ignored.
- in the second case, which recent works have proposed to do is to shift to partial parallelism, although this reduced the packet duplication/merge cost, we can still do better!

Motivation



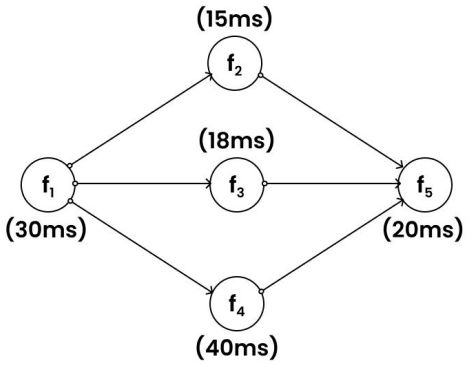
e2e = 160 ms
Avg arrival rate = 1 Gbps
T_w = 30 ms
M = 30 Mb

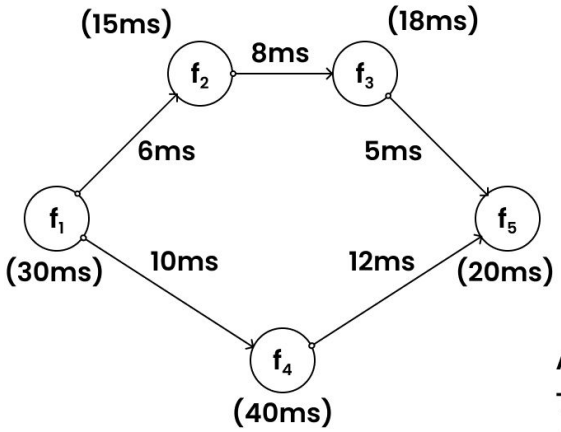


e2e = 180 ms
Avg arrival rate = 1 Gbps
T_w = 35 ms
M = 35Mb

- As you can see, with this new construction we can reduce the packet duplication cost and also the overall e2e latency as we are pushing more functions into a non critical branch instead of processing the function later as seen in the previous example. We also notice a drop in the delay difference between branches.
- The other example shows a scenario exactly similar to the previous example only that the non critical branches embed longer paths to reduce the delay difference while achieving same e2e latency. This is a key observation to make.

Motivation





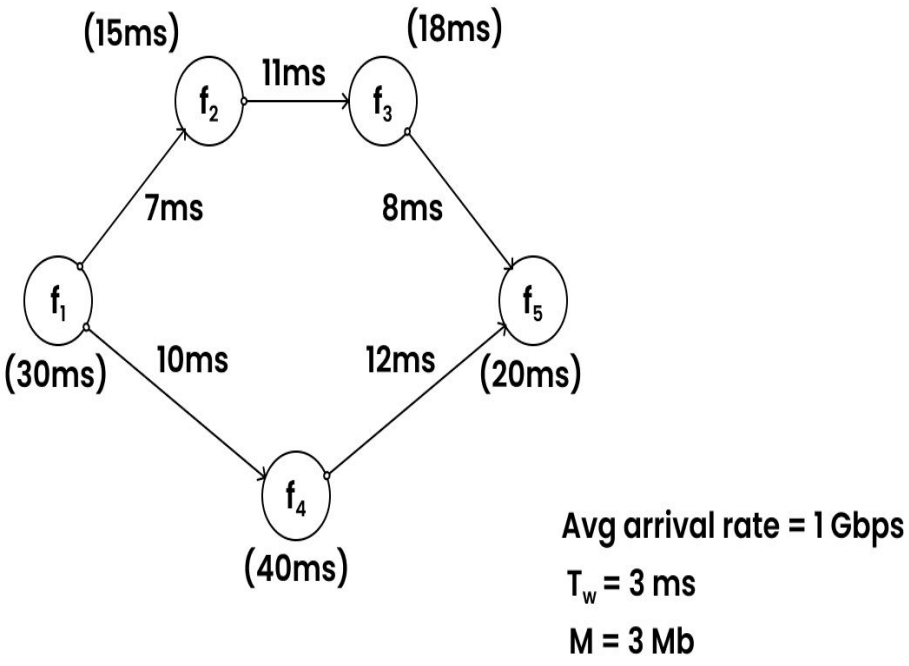
Avg arrival rate = 1 Gbps

T_w = 10 ms

M = 10 Mb

- APPM does bin packing of the SFC to address the packet deposition problem but does not look at the topology when doing so. It only tries to match the critical branch's function delays to the non-critical branch's function delays.
- After which they embed the links by traversing the shortest paths from one node to another.
- This might seem like a logical thing to do for the critical branches to reduce e2e latency, but seems unnecessary for the non critical branches.

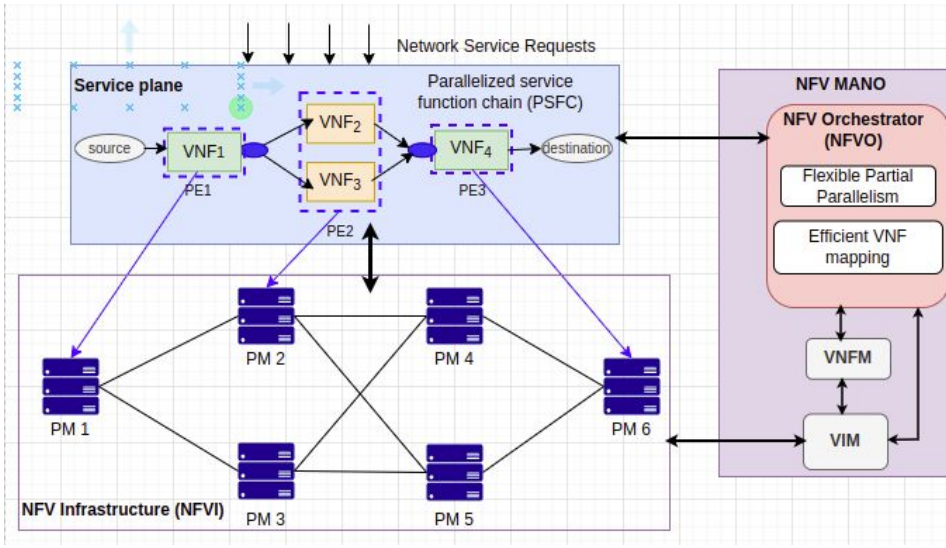
Motivation



- In this example, we can clearly see that although the SFC structure is the same and the e2e latency is same since there is no changes in the links of the critical branch, there is significant reduction in the delay difference between branches by simply traversing a longer path.
- Through this we made the observation that always picking shortest paths to traverse from one node to another is actually inhibiting the benefits of parallelism.

Problem Statement

- **Given:** a physical network topology G with a set of VNFs running in each node and a set of parallized SFC (PSFC) requests M , for each PSFC request
- **Determine:**
  - 1) How to pick nodes to run the functions of the PSFC and route the traffic between nodes efficiently
  - 2) How to pick paths for non critical branches so as to minimize delay difference with the critical branch
- **Minimize** overheads such as packet deposition and packet copy/merge cost
- **Ensure** end-to-end delay requirements of the PSFC requests are met



Parallelized SFC mapping in NFV framework

VNF shareability

- Each server V has a fixed number of VMs running on it and each VM can run one VNF at a time.
- Each VM has a fixed processing capacity of λ packets per second to allow the shareability of VNFs among PSFC requests. Currently, we assume that all VMs running on different servers have the same processing capacity.
- For example, if three PSFC requests have an arrival rate of μ_1, μ_2, and μ_3 respectively, and must utilize a function f running in on a VM of a server V then it must obey the constraint $\lambda \geq \mu_1 + \mu_2 + \mu_3$.

Proposed Heuristic Procedure

The proposed solution has two stages to address the overheads caused by parallelism.

- Bin packing algorithm to find the pSFC structure.
- The SFC placement algorithm which made use of layer graphs and Yen's K shortest path algorithm to efficiently route parallel VNFs such that they have minimum packet deposition

Stage 1: Bin packing

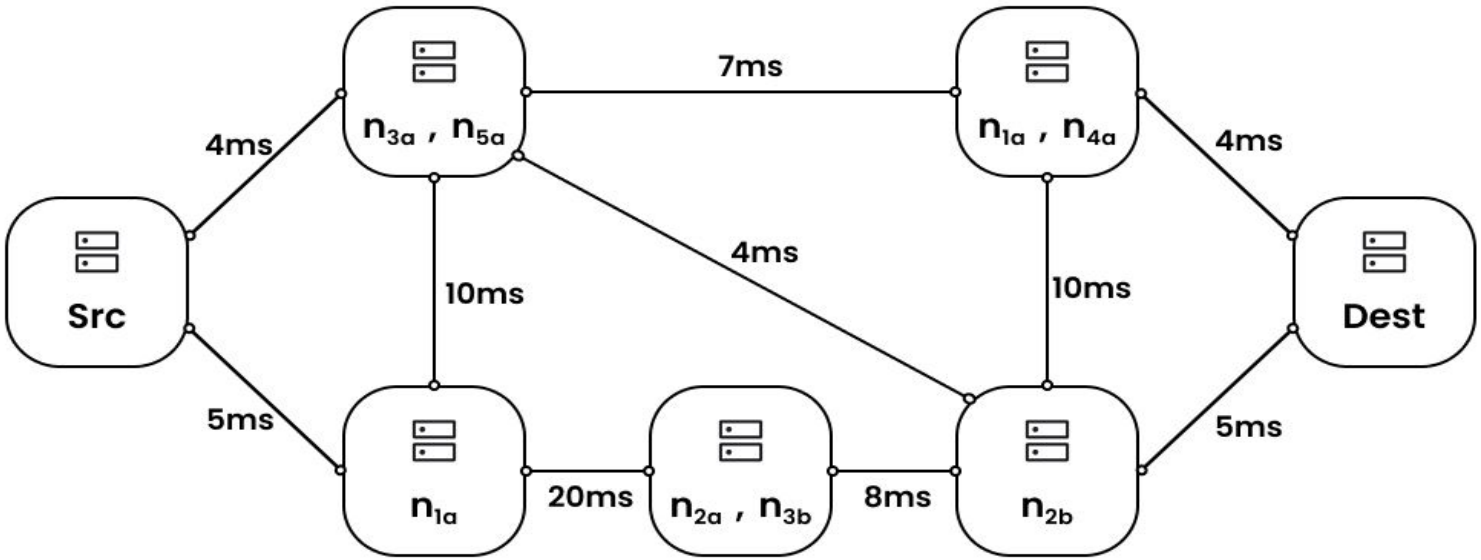
- This is a simple greedy first fit bin packing algorithm.
- It picks the maximum function from the parallel entity and then tries to merge as many remaining functions into a branch such that the sum of processing time of the functions is less than the maximum function.
- This algorithm helps in reducing the pSFC from full to partial parallelism as well as tries to minimize packet deposition.

NOTE: We still haven't taken the links into consideration. This is will come in the next part of the algorithm.

Stage 2: SFC placement algorithm

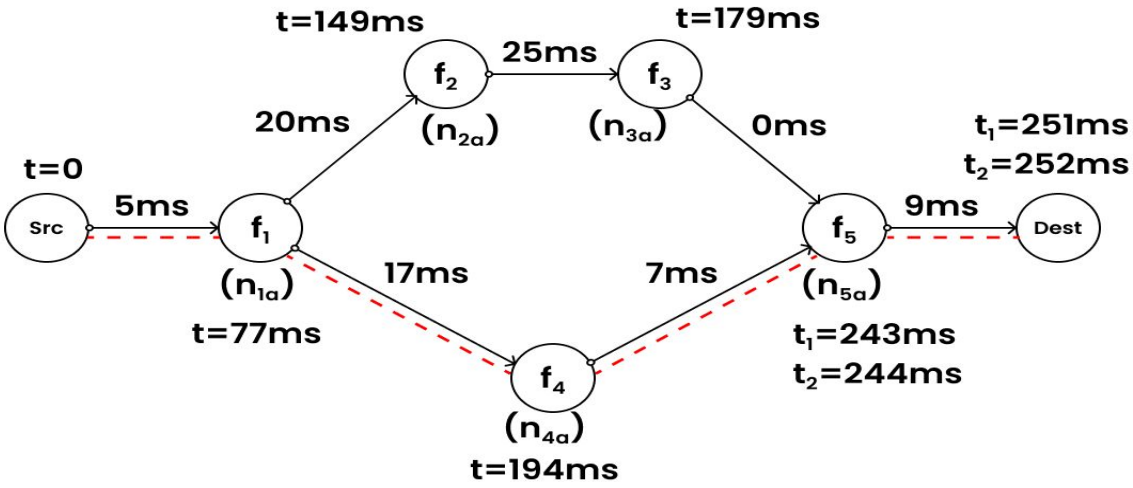
- After running the bin packing algorithm, we have the structure of the PSFC, but we still must decide what server instances to pick and how to efficiently route the parallel traffic.
- To address this, we developed an SFC embedding algorithm that meets end-to-end latency constraints at the same time minimizes the packet deposition problem by routing packets of non-critical branches such that the delay difference in the arrival of packets with respect to the critical branch is almost the same.
- Let us understand the algorithm through a simple illustration in the following slide.

An Illustrative Example



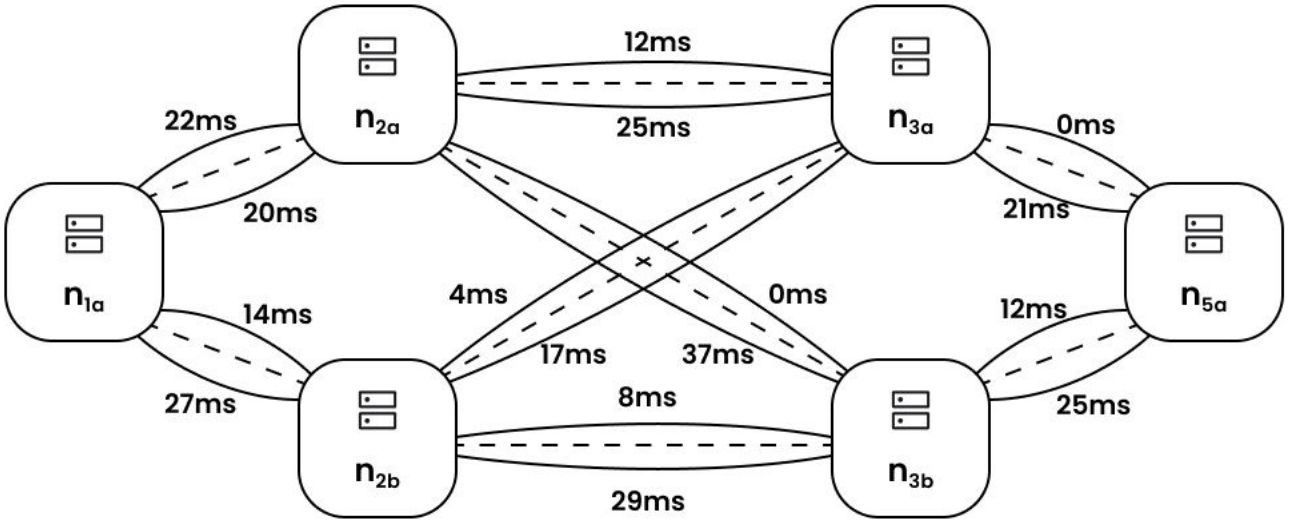
- Assume this to be the topology of the network with VNFs already deployed and running. Each node clearly mentions which VNFs it is running. There are 2 instances of function f_1, f_2, and f_3 and 1 instance of f_4 and f_5.

An Illustrative Example



- The first step in the algorithm is to pick the nodes of the critical branches of all parallel entities in the PSFC. Picking nodes for the critical branches should always be in the shortest path so as to minimize the e2e latency. Obviously the paths and nodes picked should have sufficient resources.
- By doing this step, we have embedded all functions of the critical branches. Now all that is left is to pick the nodes for non critical branches and the nodes and paths should be picked in such a way that delay difference with the critical branch is minimum.
- This is done by creating a layer graph of the functions in the non critical branch.

An Illustrative Example

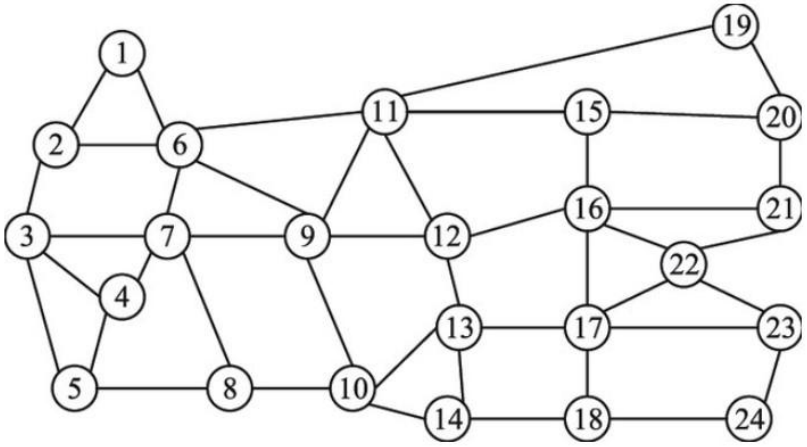


- The above is the layer graph of the VNFs in the non critical branch. We connect each node in this graph by K links which represent the K shortest paths between those nodes. Note that each node in this layer graph is an actual node in the topology that has the particular VNF running on it.
- Finally we pick the path from this graph which has the least delay difference and select those nodes to run the VNFs of the non critical branch.

Performance Evaluation

Simulation setup

- The entire simulation was run in c++ environment.
- The input given was the USPNET topology where the links were randomly generated within a certain range and the VNFs were randomly placed in the topology. The topology itself consists of 24 nodes and 43 edges.
- A batch of parallel SFCs were randomly generated and placed.



Performance Evaluation

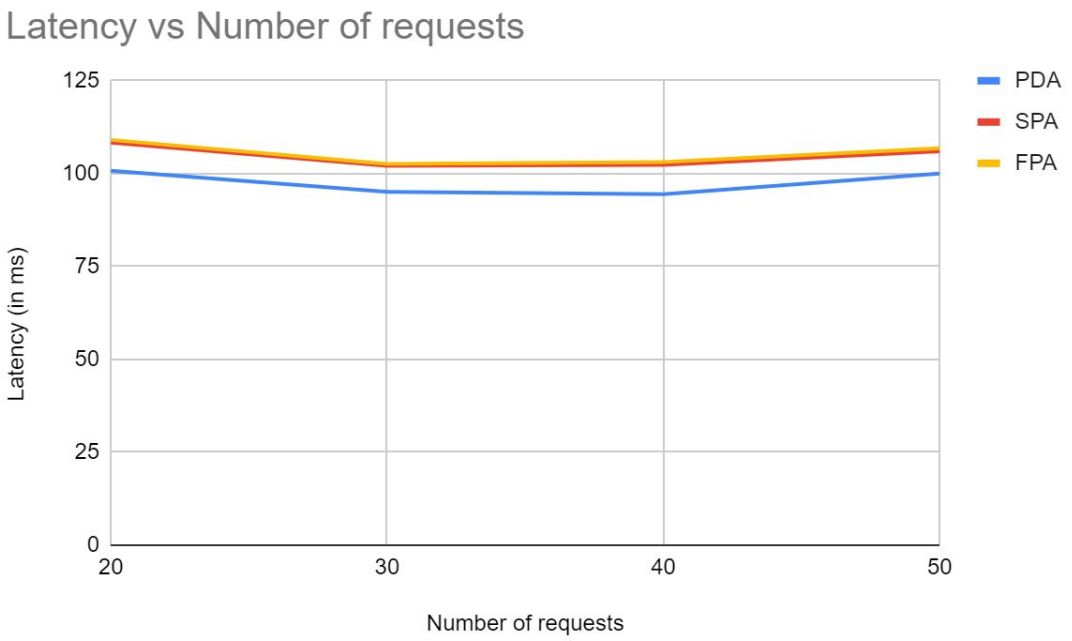
- There were total three algorithms that were tested on the same setup for comparison purposes.
  - PDA: This is my proposed solution that is packet deposition aware.
  - SPA: This algorithm tries to go from one node to another in the shortest possible path always. (most existing solutions do this)
  - FPA: This algorithm abuses full parallelism as well as always travels in the shortest path from one node to another.

Performance Evaluation

Parameters:

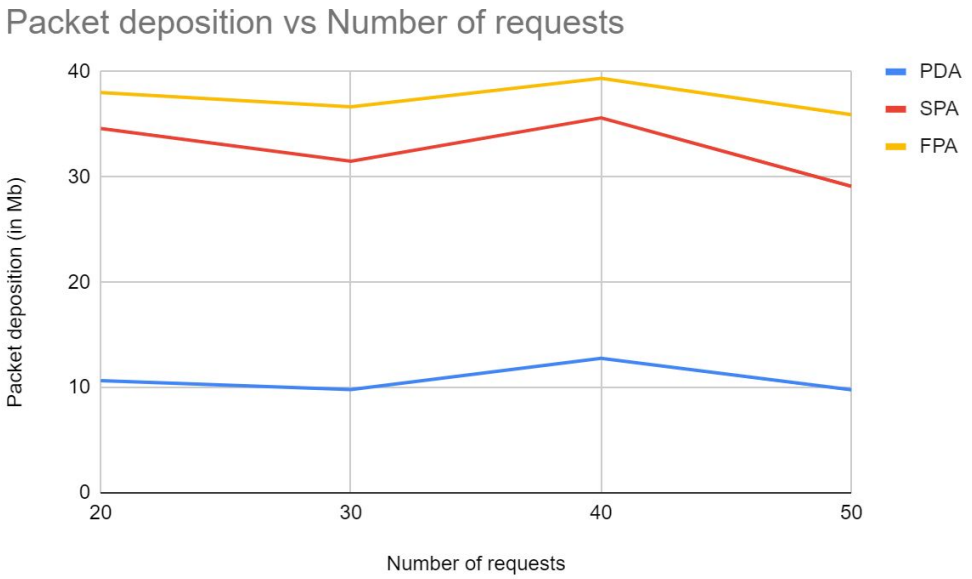
- link delays: [4,8] ms
- Number of VNFs: 10
- link BW: [20,30] Gbps
- VM capacity: 5 Gbps
- Node capacity: 4
- SFC length: [4-8]
- VNF delay: [1,20] ms
- Only one parallel entity per SFC request
- SFC arrival rate: [1-1.5] Gbps
- e2e latency requirements - [140-180] ms
- SFC length was taken to be 6 and K to be 3 unless mentioned otherwise.
- All results were collected after running the simulation for 10 times and taking the mean of them.

Latency vs Number of requests



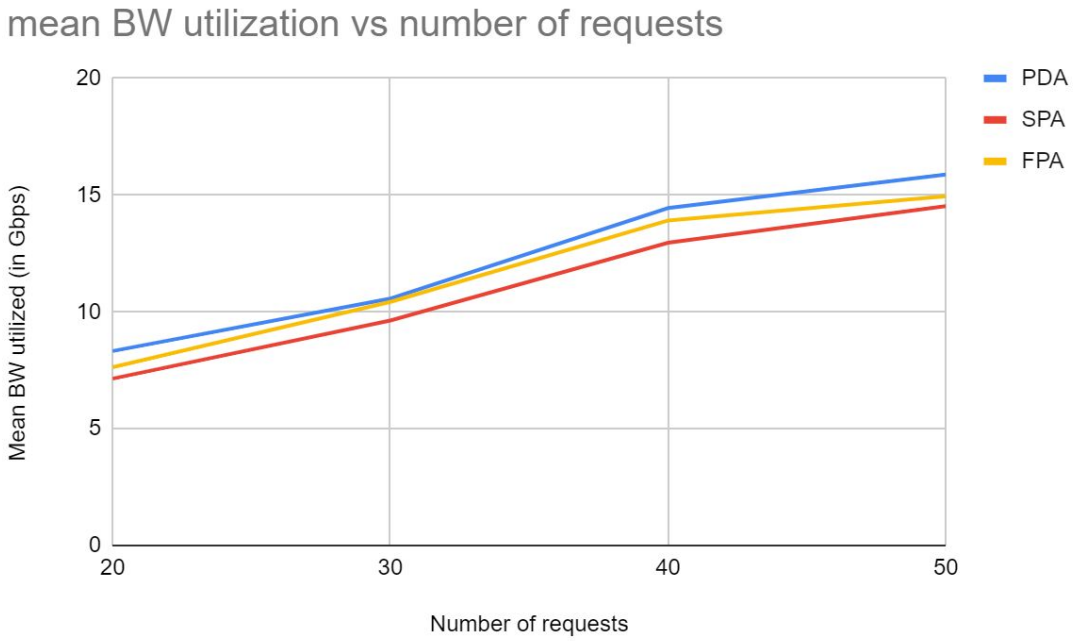
- We can see that our proposed algorithm does better than FPA which abuses full parallelism. This shows how packet copy/merge cost can increase the e2e latency.
- It also does better than SPA mainly because of using longer paths for non critical branches and saving shortest path bandwidth for critical branches.

Packet deposition vs Number of requests



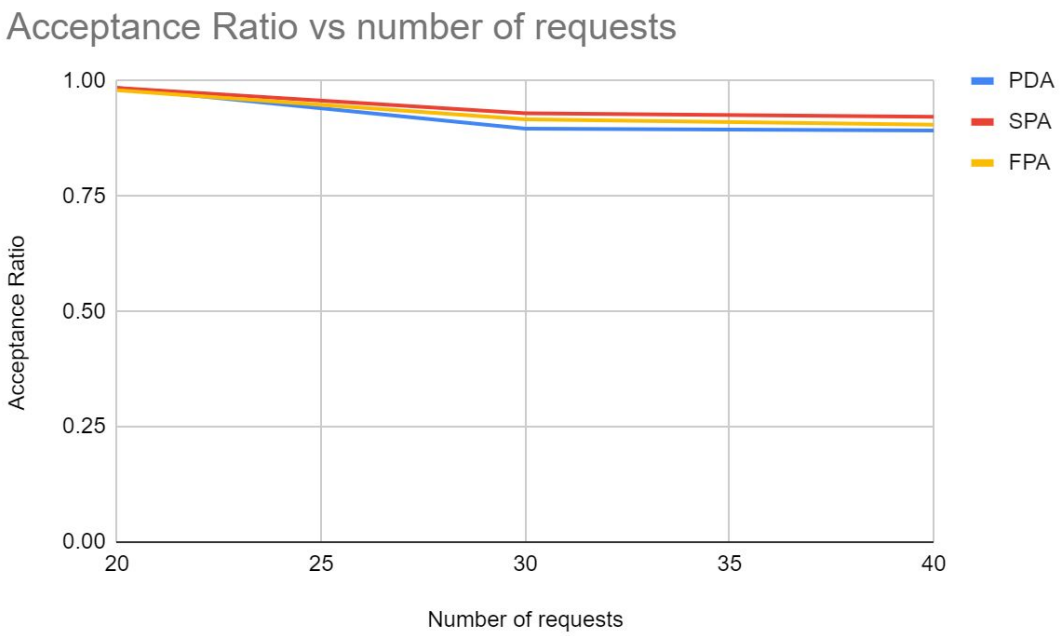
- We can see that our proposed algorithm does significantly better in terms of memory used for caching waiting packets thus minimizing the packet deposition problem.

BW utilization vs Number of requests



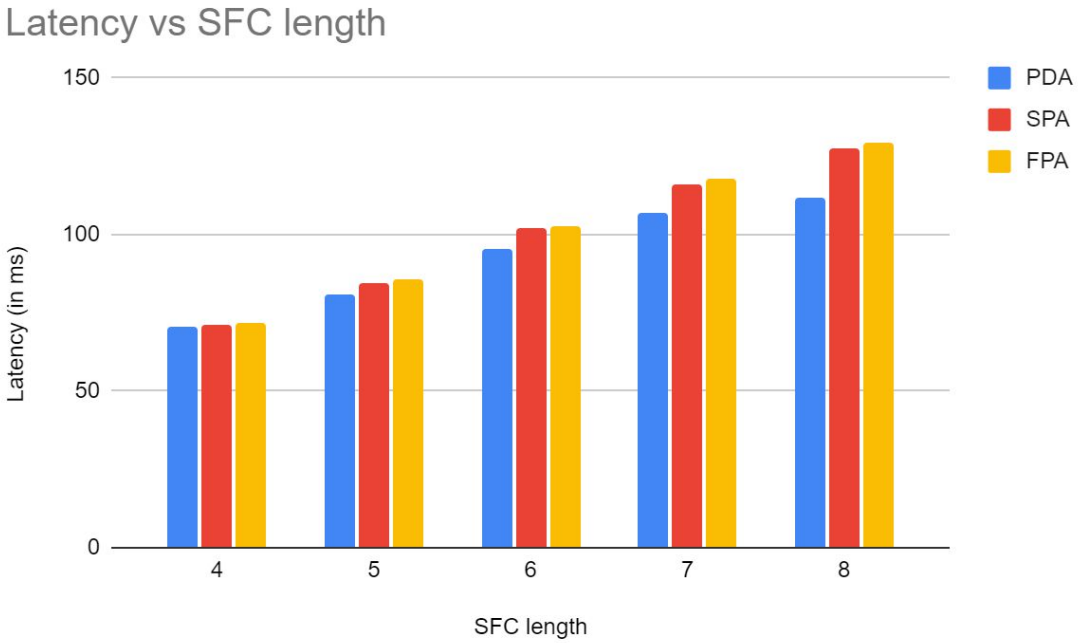
- Our proposed algorithm uses slightly more link bandwidth compared to the rest because of using longer paths to reduce delay difference between parallel branches.
- Also bandwidth utilization increases with increase in batch size. This is self explanatory.

Acceptance Ratio vs Number of requests



- The acceptance ratio are almost similar. Our proposed algorithm has slightly lower acceptance mainly because of using more bandwidth but this difference is very minimal.
- Acceptance ratio also reduces with increase in batch size as the network might not have the resources to fit bigger size batches.

Latency vs SFC length

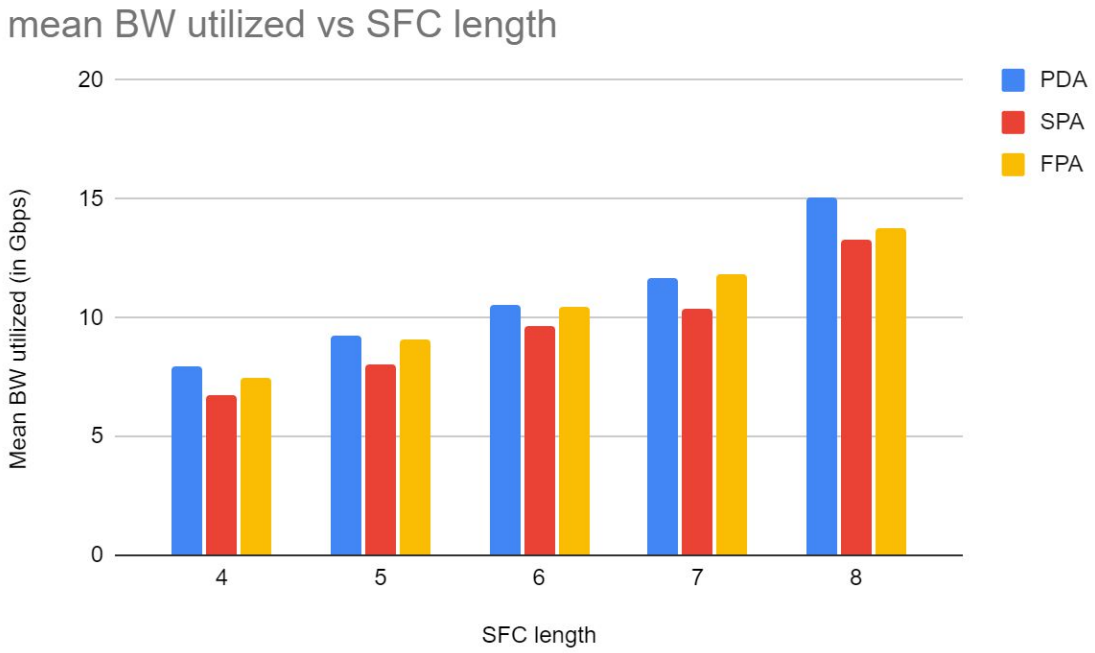


- We can see that as we increase the length of SFC the mean latency also increases.
- This is because more functions have to processed as the length of the chain increases.

Packet deposition vs SFC length

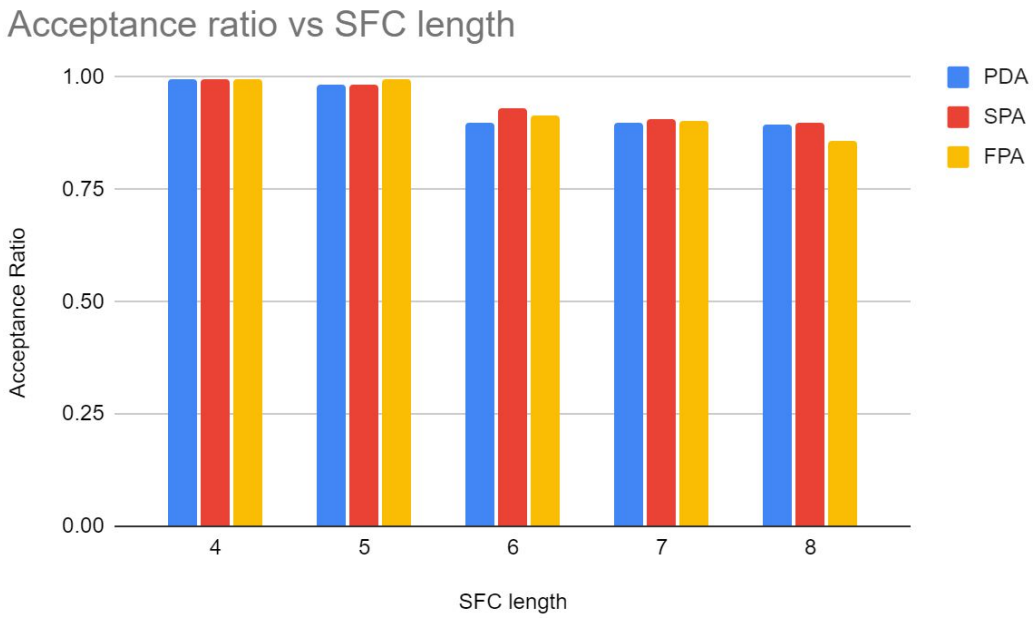
- Expected trends are observed for packet deposition as well.
- We must note that there is a significant increase in the packet deposition in the case of SPA and FPA but our proposed algorithm spike isn't that high and always does better than the rest.

BW utilization vs SFC length



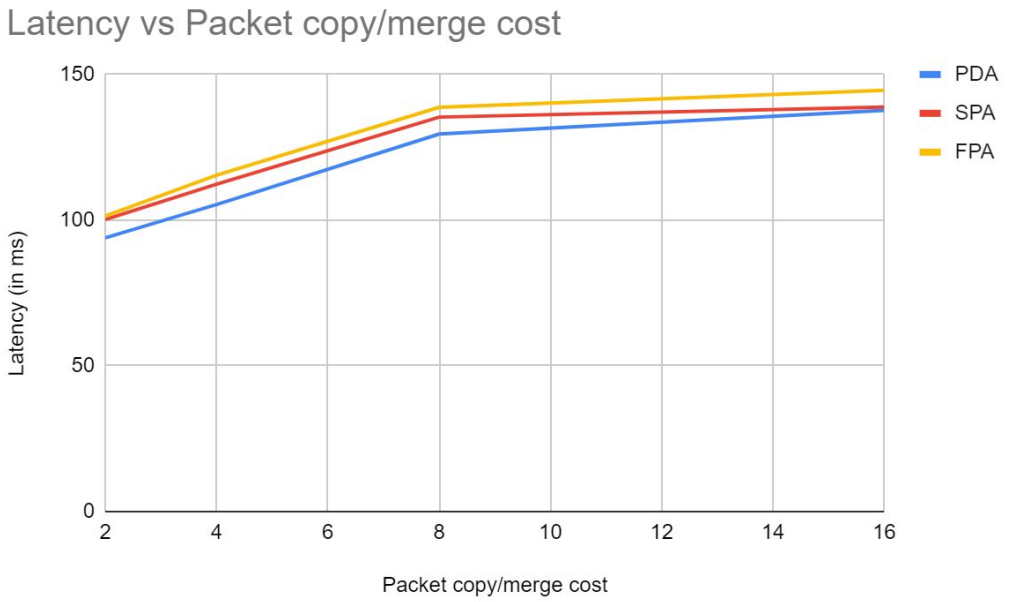
- As the SFC length increases more bandwidth is utilized because we have to embed more VNFs which will inturn cause the utilization of links more.
- Our proposed solution uses more bandwidth compared to the rest but the difference isn't that high.

Acceptance Ratio vs SFC length



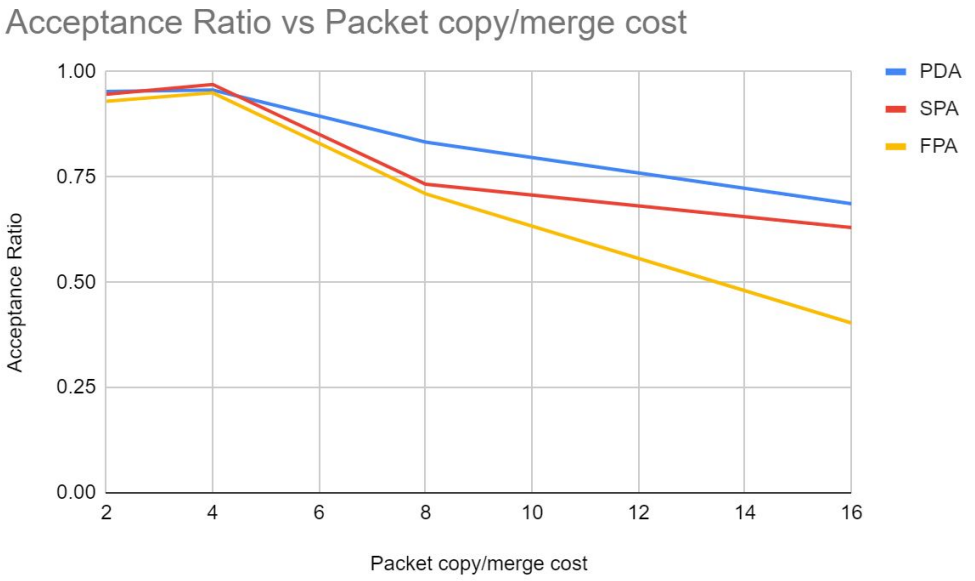
- As the length of chain increases the acceptance ratio decreases and this can be correlated to both the increase in bandwidth and latency. As seen bandwidth and latency increases as chain length increases.
- Using more bandwidth may cause rejection of chains more due to unavailability of network resources. Higher latencies may cause rejections due to failing to meet e2e requirements.

Latency vs Packet copy/merge cost



- The latency increases as we increase the packet copy/merge cost (this is done by increasing the packet size).
- There is a significant spike in the FPA algorithm since it abuses full parallelism leading to max possible packet duplication/merge.

Acceptance Ratio vs Packet copy/merge cost



- Acceptance ratio significantly falls as we increase the packet copy/merge cost since latencies increases and thus may cause higher rejections due to failing to meet e2e requirements.
- Compared to our solution and SPA, FPA has a significant drop in the acceptance ratio due to the fact that more packet duplications/merges cause a significant increase in the e2e latency.

Work in Progress

- I have formulated an integer program to solve the above problem and I am yet to implement it and compare it with the heuristic algorithm.
- We are yet to collect results for a large topology.

భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

THANK YOU!

Queries