Efficient VNF Placement

TWiN Project

Idea

- Given a SFC request, we want to maximize the total number of accepted requests(or TAT) and the end-end throughput
- We take into account 2 factors here:
    - Shareability of VNFs
    - Interference while deploying a particular VNF at a node with other VNFs already deployed there

Algorithm 1 - Shortest Path Heuristic (or SPH)

- Input: The SFC request, Graph topology G(V, E)
- Sort all requests in descending order (based on the ratio of requested throughput and resource consumption of VNFs)
- Select the shortest path from source to destination with respect to the link delay taking shareability of VNFs into account (using multi-stage approach)
- Then try to place as many VNFs near the source and go towards the destination
- A request is rejected either if
    - The shortest path is not able to meet the required delay or the nodes in the shortest path cannot be used to place a VNF in the SFC (insufficient resources)
    - By placing this vnf on the chosen node, already deployed requests get violated

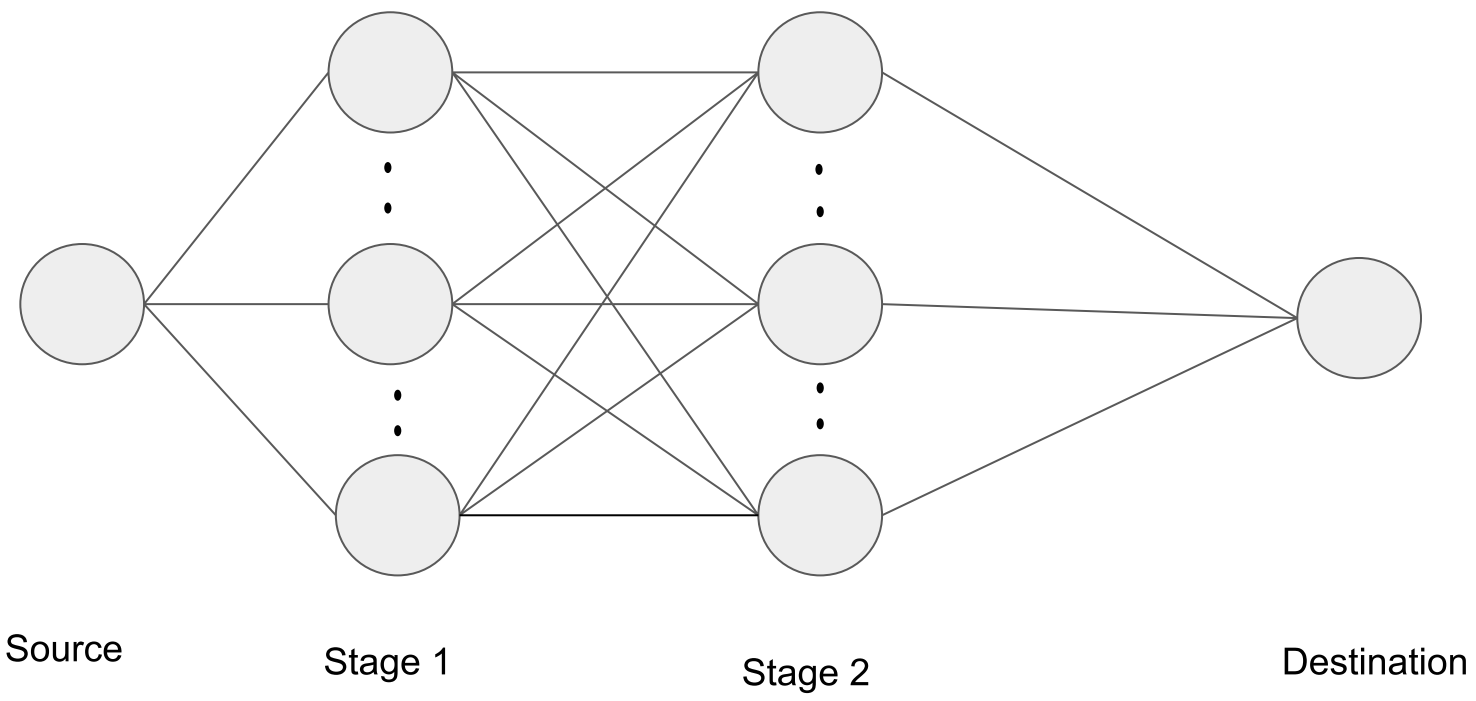
Algorithm 2 - Greedy on Used Servers (or GUS)

- Input: The SFC request, Graph topology G(V, E)
- Sort all requests in descending order (based on the ratio of requested throughput and resource consumption of VNFs)
- Select the shortest path from source to destination with respect to the link delay taking shareability of VNFs into account (using multi-stage approach)
- Then try to place as many VNFs on nodes which are the most loaded
- A request is rejected either if:
  - The path developed using the above approach is not able to meet the required delay or the nodes in the shortest path cannot be used to place a VNF in the SFC (insufficient resources)
  - By placing this vnf on the chosen node, already deployed requests get violated

Algorithm 3 - Custom Algorithm using Multi Stage

- Input: The SFC request, Graph topology G(V, E)
- Sort all requests in descending order (based on the ratio of requested throughput and resource consumption of VNFs)
- Select the shortest path from source to destination with respect to the link delay taking shareability of VNFs into account (using multi-stage approach)
- Then try to place as many VNFs preferentially on nodes with least interference with other VNFs in that node
- A request is rejected either if:
  - The path developed using the above approach is not able to meet the required delay or the nodes in the shortest path cannot be used to place a VNF in the SFC (insufficient resources)
- If while placing a VNF in a node, already existing request gets violated then we try to place this VNF on the node which incurs next minimum interference

Shareability (using Multi-Stage)



Interference

	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

Cross Interference coefficients considered while deploying a VNF on a node

Interference delay

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

- This Interference_metric is guaranteed to be less than 1, and gives us a good estimate of the interference created because of an incoming VNF in a node

Interference delay = Interference_metrtic * VNF_delay

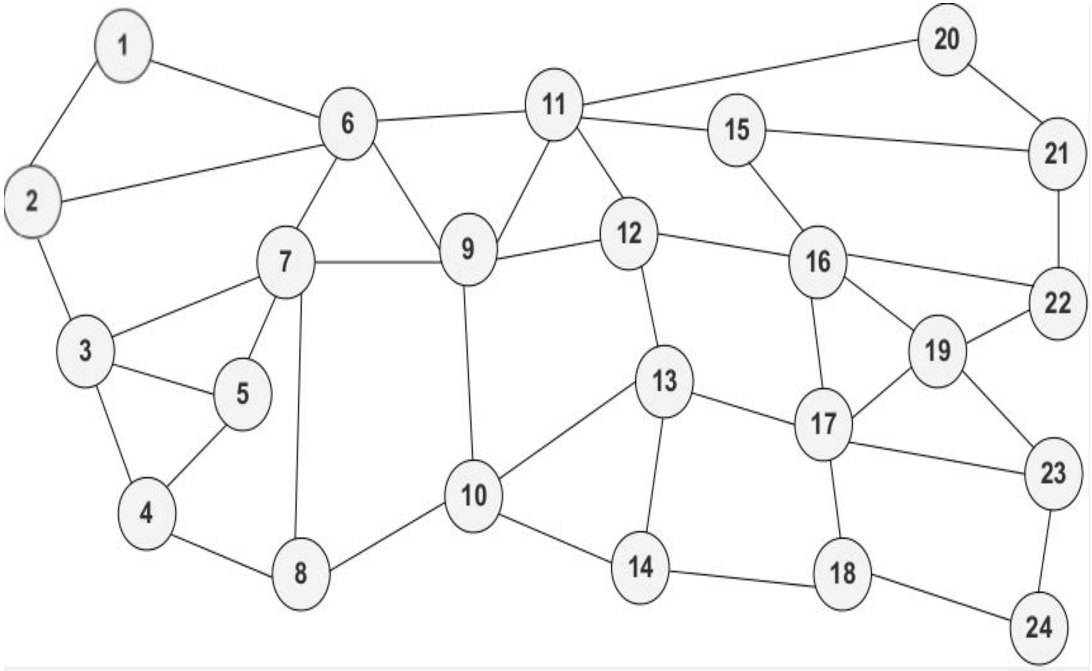
- The VNF delay is fixed for different kinds of VNFs which captures the amount of processing time typically required for a VNF to process the request

Hyper Parameters

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

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

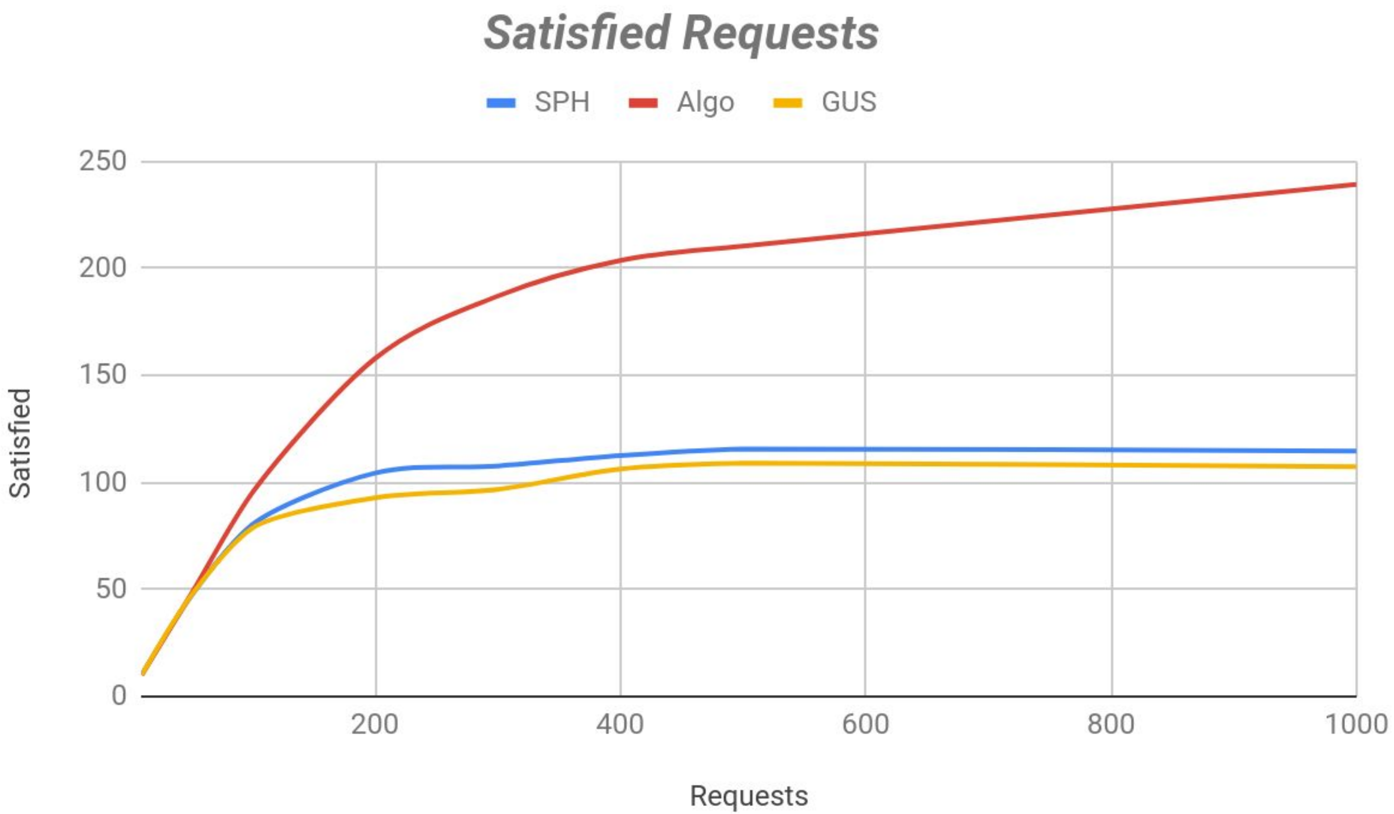
Graph topology



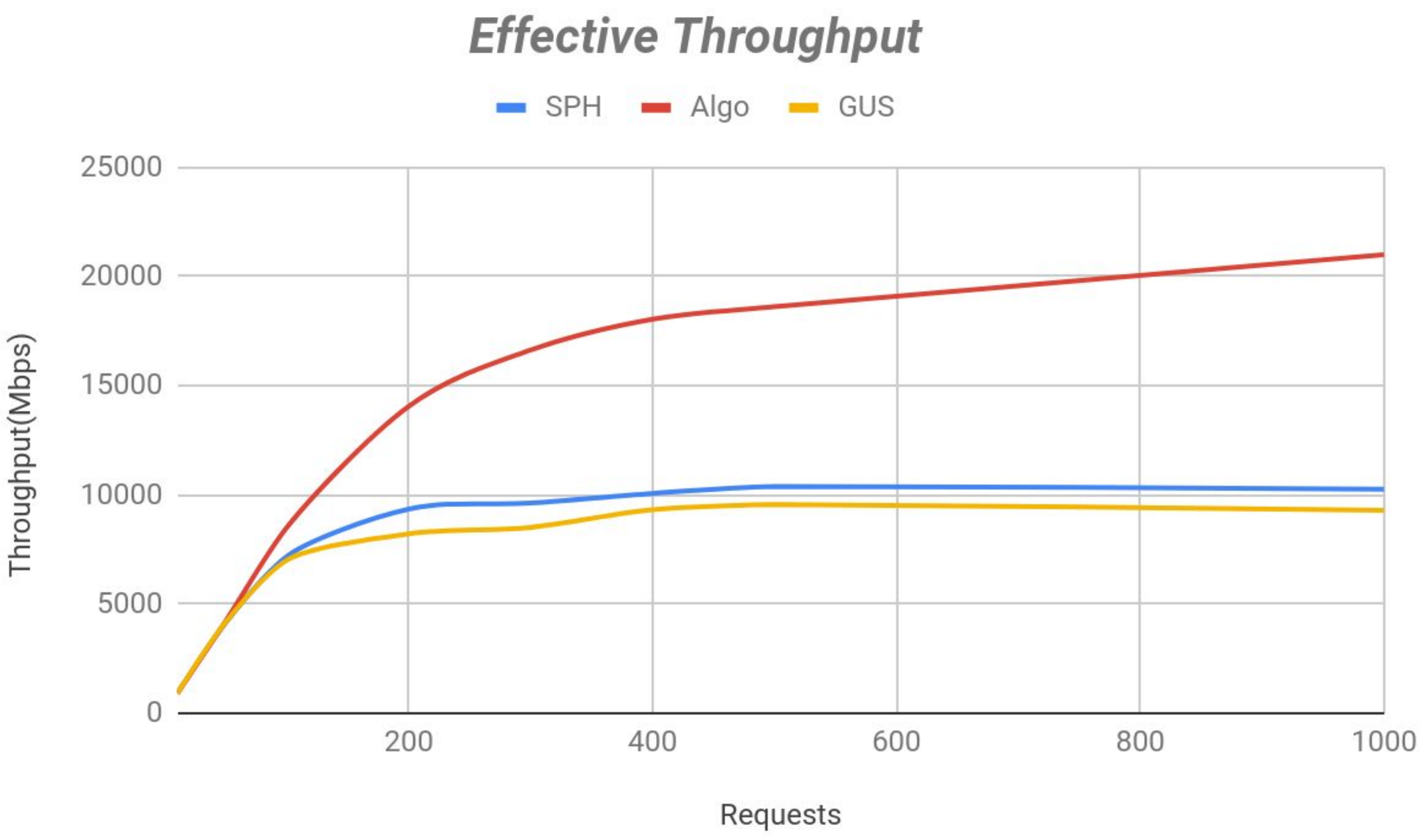
Results

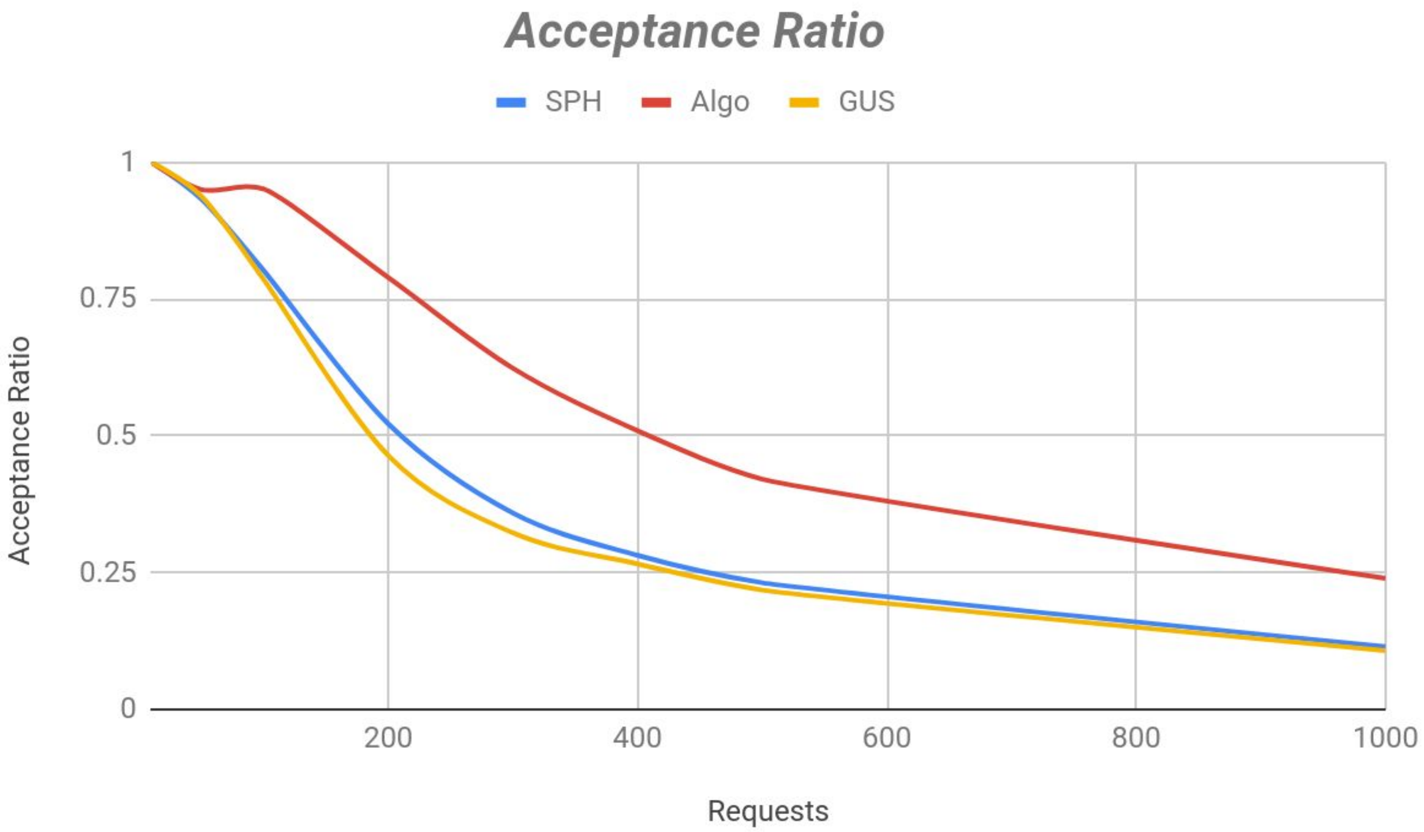
- Satisfied Requests vs Number of Requests
- Effective Throughput vs Number of Requests
- Acceptance Ratio vs Number of Requests
- Time taken vs Number of Requests

Note: 100 runs were averaged to plot a single point of the below graphs



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


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


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


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

Implementation

- I implemented the 3 algorithms mentioned, in a modular infrastructure where if needed more algorithms can be integrated easily and their performance compared
- A private repository is made where all the incremental changes are pushed
- GitHub Link : https://github.com/abhi1604/Efficient-VNF-Placement

Present Work

- Currently implementing the AIA algorithm

Future Work

- Comparing the results all the approaches

Motivation Expirements

Demystifying the Performance Interference of Co-Located Virtual Network Functions

Goal: Run 2 different types of NF on same machine and see what resources they are competing for (e.g. cache, CPU, Memory etc.). This experiment will help us for better placement of NFs.

Tools Used

- Network Functions
  - Snort (DPI)
  - Nginx (NAT)
  - Pktstat (Firewall)

- For Performance Metric
  - perf
  - dstat
  - Siege

Metrics Calculated

- Throughput
- Response Time
- Pkt loss (%)
- CPU Utilization
- Cache-references , cache-misses, cache-hits
- context-switches
- Memory Usage

Experimental Setup #1

Generate Traffic with packEth

Note : Arrow - Network Traffic

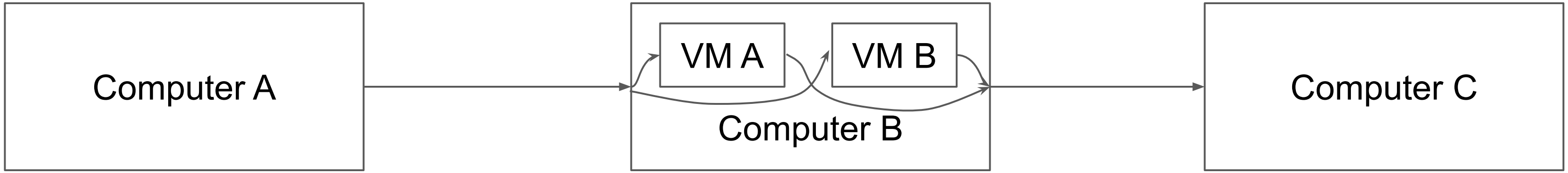
Problem with Experimental Setup #1

Used packEth for network traffic generation.

packEth generates traffic (raw packets) with maximum possible rate (or specified constant rate) without worrying about the dropped packets.

In this setup, we do not know if the packets that was received at VM were actually completely process by NF.

Experimental Setup #2



Generate Traffic with
iperf

In this setup, we can check at computer C if all the packets received by VM were actually processed by NF.

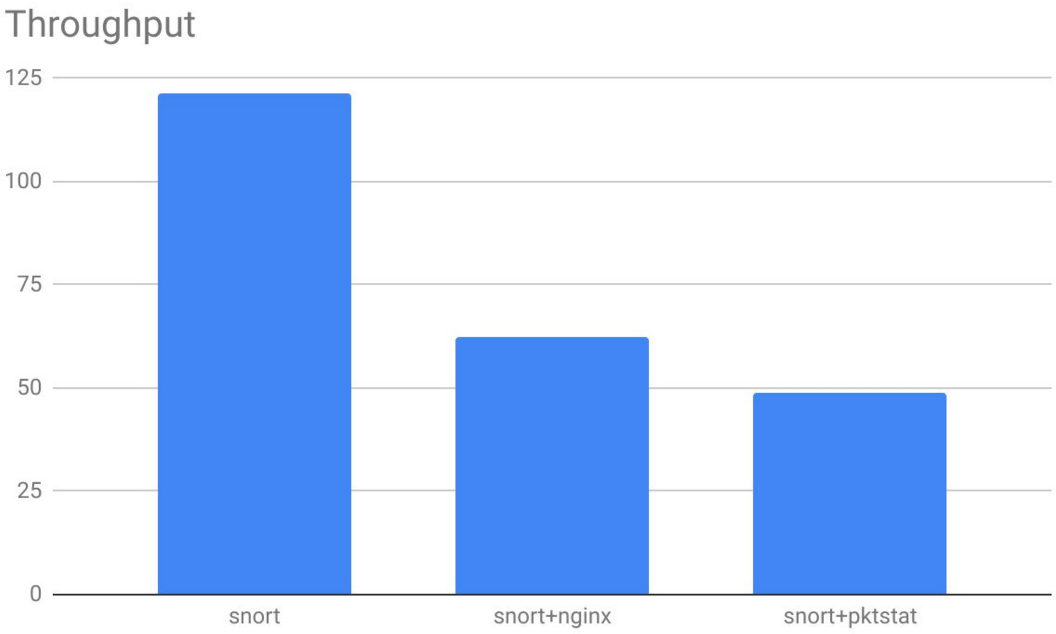
Note: Separate iperf command for each VM (NF).

Results

Experimental Setup #1

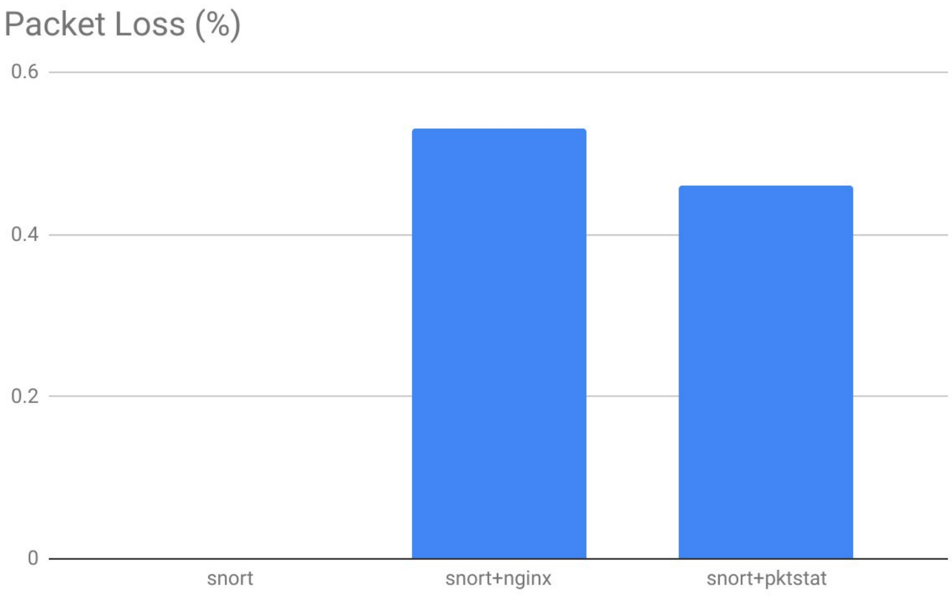
All the reported metrics are on VM on which snort is running

Throughput



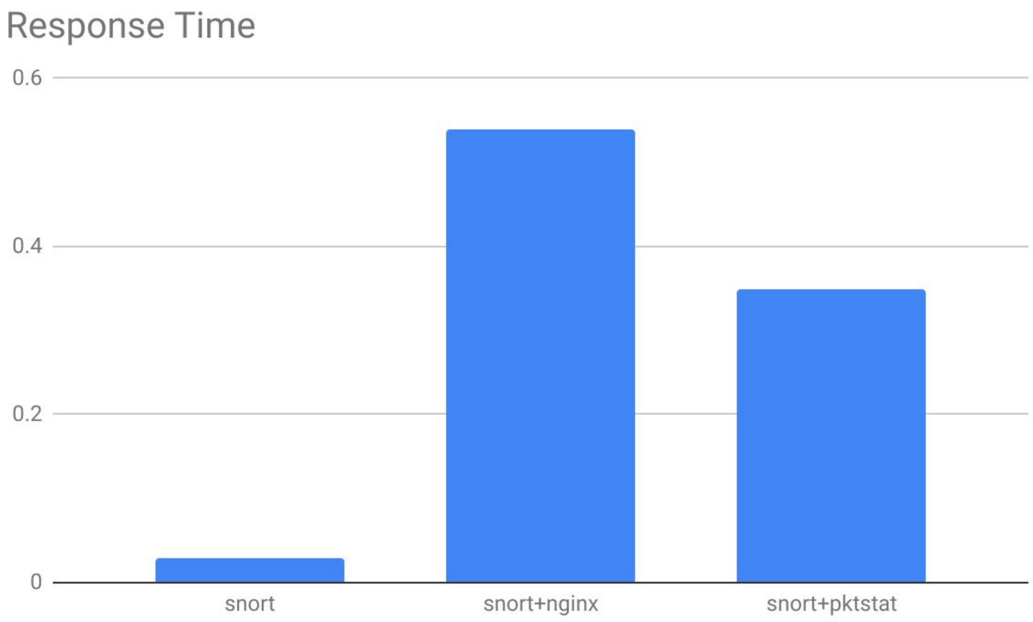
Throughput is the total incoming traffic at VM
Tool:dstat

Packet Loss



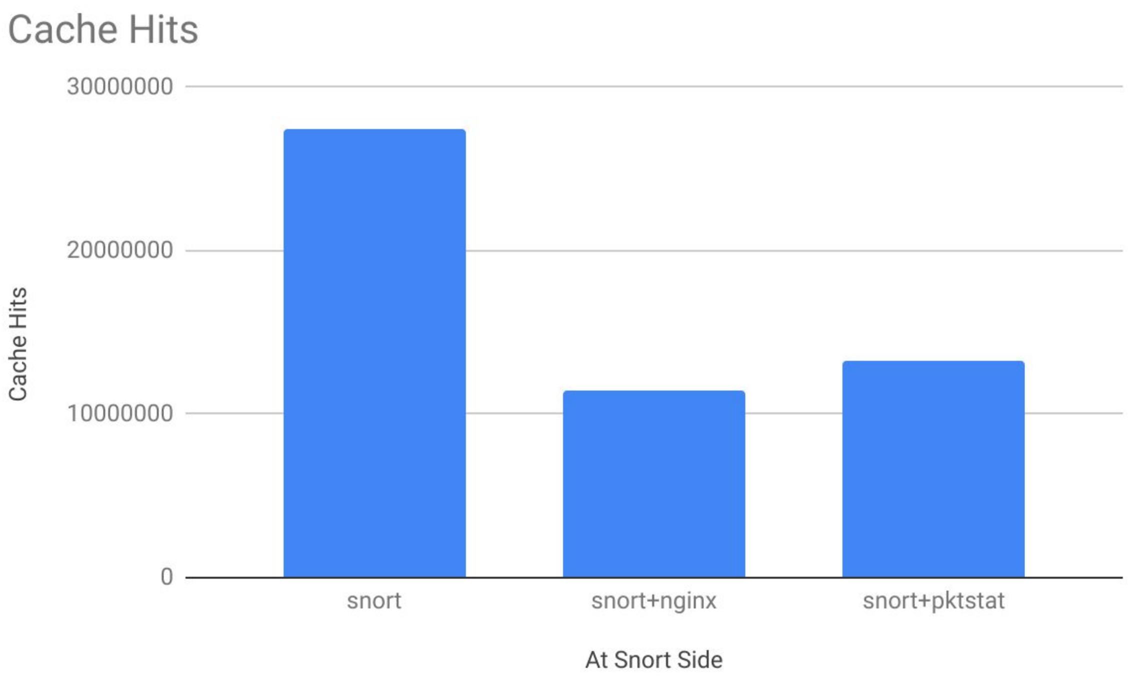
Sending ping packets at 10 ms interval for 10 sec and looking at how many ping packets got lost.

Response Time



Siege send some get index file queries. Response time is the time took by NF to respond to those queries.

Cache Hits


Cache Hits
30000000
20000000
10000000
0
Cache Hits
snort
snort+nginx
snort+pktstat
At Snort Side

Thank You