A summation of...

Machine Learning Driven Scaling and Placement of VNF at the Network Edges

Two - Fold Purposes of Paper

1. Effective use of using a machine learning based model to predict accurate amount of Virtual Network Functions (VNFs) to deploy based on the network traffic(auto scaling)

   This would take place in a distributed Mobile Edge Computing - Network Function Virtualization (MEC-NFV) environment

2. Use of Integer Linear Programming(ILP) to formulate the placement of Virtual Network Functions(VNFs) at the edge nodes to minimize latency from all users from their respective VNFs

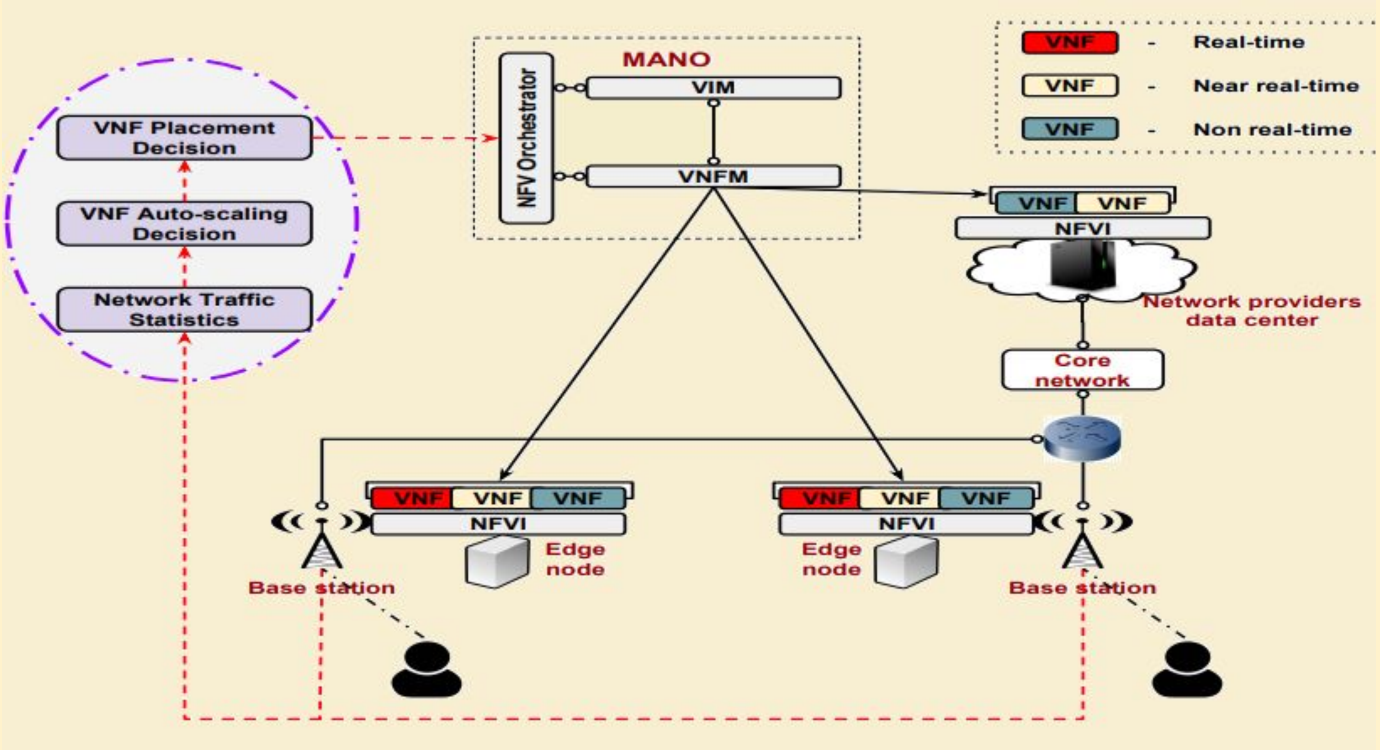


Fig. 1: A high-level distributed MEC-NFV System Architecture.

PART I

Auto Scaling + Machine Learning

Background Information

Autoscaling can be broken down into 2 categories

1. <u>Reactive mode</u>: thresholds can be **statically predefined** or **dynamically updated** and depending on the value you can scale (UP or DOWN).
   a. Main weakness is that it's still a reactive solution, cannot prepare itself
2. <u>Proactive mode</u>: Let the system 'learn and anticipate' future needs based on scaling decisions.
   a. Where ***machine learning*** techniques become very useful

Crash Course of machine learning.. Under the scope of this paper

1. Problem description
2. Multi-layer Perceptron(MLP)
3. Modeling MLP in Keras
4. Feature Engineering(4 parts)
5. Classification using Neural Networks
6. Model Evaluation

1. Problem Description

How to map the following

X (input) : traffic load statistics

Y (output): required number of VNFs to deploy without violating QoS SLA

X and Y will evolve over time with the main influences being

1. Mobile network traffic dynamics
2. Amount of **active** mobile users

X,Y will evolve together and can be modelled as a time series, and that can be best modelled with a **neural network** to estimate the parameters

2. MLP - Why a Neural Network?

- ➔ Its proven effectiveness in evaluation time series problem
- ➔ Ability to ***learn*** new patterns or customized features when there isn't a definite math function available to fit (eg: non linear activation functions)
- ➔ This paper utilizes a **Multilayer Perceptron**(MLP - feed forward neural network of 3 parts)
  - ◆ Input layer
  - ◆ Hidden layer (one or more)
  - ◆ Output layer
- ➔ **All** nodes are interconnected and fed forward to the next layer

3. Modeling MLP via Keras - Why?

- ➔ Specifically for neural network experimentation
- ➔ Can be run on top of other software (eg: Tensorflow, Theano, R, etc.
- ➔ Simple pre built functions
- ➔ Wide range of activation functions
- ➔ Has predefined layers
- ➔ Modular

4-2 Feature Engineering

Feature Extraction & Class Definition

a. $X_{default} + X_{constructed} = Y_{output}$
b. Default => current information that can be accessed or calculated
c. Constructed => how features will evolve over ***time*** (proactive scaling)

Measurable / 'Raw' data

Default features ($X_{default}$)
1. Base station ID.
2. Date.
3. Time-stamp t.
4. Average number of users between t and $t-1$ in each cell.
5. Maximum number of users between t and $t-1$ in each cell.
6. Average downlink user throughput in each cell.
7. Average uplink user throughput in each cell.
8. Traffic load measured in each cell at time t, given by $\lambda(t)$.

TABLE I: Default set of features available in the dataset.

Feature Transformations of data

Constructed features ($X_{constructed}$)
9. Traffic load measured in each cell at time $t-2$, given by $\lambda(t-2)$.
10. Traffic load measured in each cell at time $t-1$, given by $\lambda(t-1)$.
11. Traffic load measured in each cell at time $t+1$, given by $\lambda(t+1)$.
12. Traffic load measured in each cell at time $t+2$, given by $\lambda(t+2)$.
13. Change in traffic load in each cell from time $t-2$ to $t-1$.
14. Change in traffic load in each cell from time $t-1$ to t.
15. Change in traffic load in each cell from time t to $t+1$.
16. Change in traffic load in each cell from time $t+1$ to $t+2$.
17. Weekday or weekend.

TABLE II: Constructed set of features from the dataset.

4-1 Feature Engineering

Generally how data is collected, organized, cleaned/scrubbed, and separated

Data collection: **where** data will be collected

a. 6 LTE Base stations
b. 10 cells per base station
c. 8 day period, with a hourly collection rate

4-3 Feature Engineering

<u>Feature Subset Selection:</u> Help eliminate redundant or unnecessary features

Why?

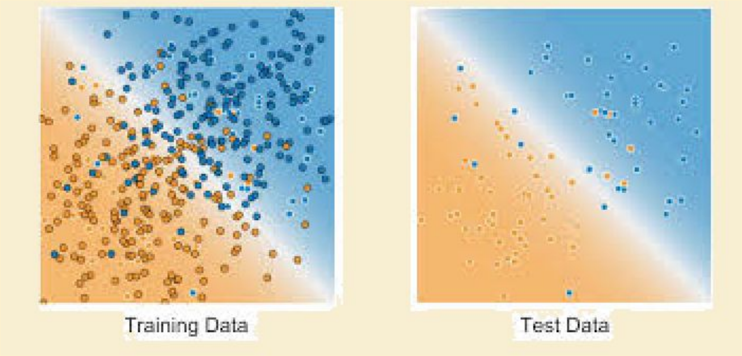
1. Reduce complexity
2. Reduce dimensionality of feature sets
3. Reduce computational overhead

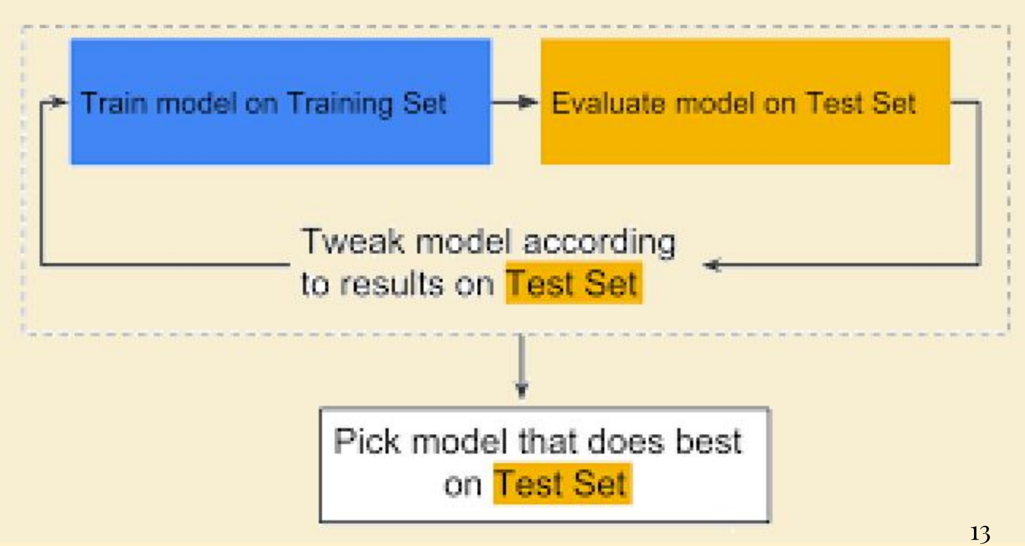
4-4 Feature Engineering

<u>Dataset Decomposition:</u> how data is ‘split’ into **test** & **training** sets

Typical split is (75% training, 25% test)

MLP model will try to find a relationship between its features and classes





5. Classification using Neural Networks

Hyperparameters: ***manually*** set parameters to get the estimations of regular parameters

Following methods to find their hyperparameters

1. **Babysitting Search** - Start with an initial value,watch the learning process and then manually tune it again, 100% manually done

2. **Grid Search** - grid with ‘n’ dimensions, and each dimension maps to a different hyper parameter, then define range of possible values, then go through all combinations and pick the best one

Experiment Breakdown

- 1 input layer with 12 nodes
- 3 hidden layers each with 12, 24, 12 nodes correspondingly
- 1 output layer with 10 nodes
- Regularization Parameter = 0.01
- Optimizer: Stochastic gradient
- Learning Rate = .001
- Batch Size = 100
- Number of epochs = 300

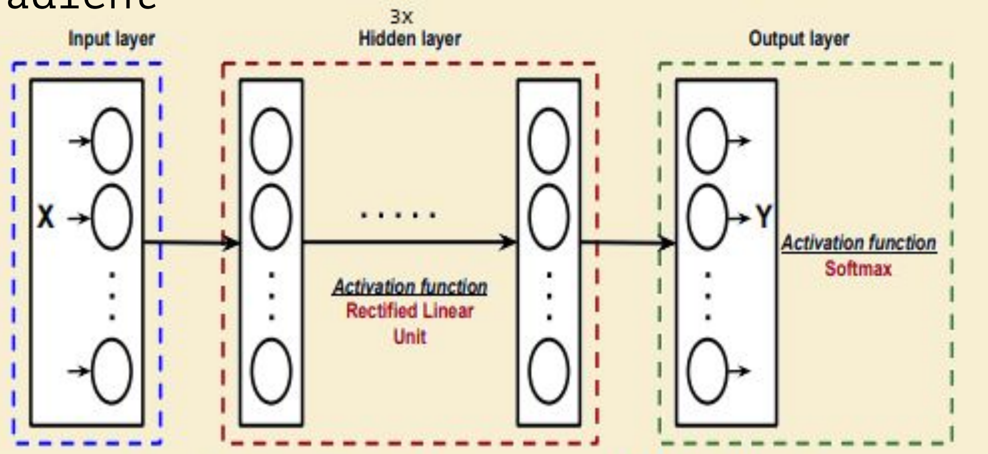


Fig. 2: Structure of the proposed MLP Classifier.

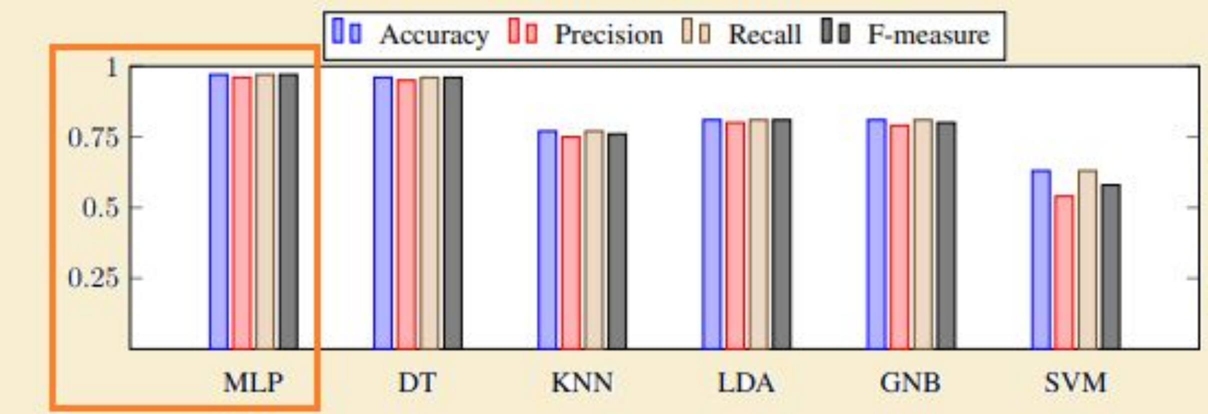
6. Model Evaluation/Results

Experiment’s Assumptions and Set-up

- **Bandwidth capacity** (per node) = 20 Gbps
- Per VNF can process 200 Mbps without QoS degradation
- Horizontal scaling: each node can host 100 VNFs
    - Maximum of 10 VNFs per cell
- **Setting**: mobile data network
- **Number of Base Stations**(LTE) = 6
- 10 cells per base station
- **Data collection duration**: 8 days at a hourly rate

The model was then trained with different types of *classification* algorithms

Performance Results



MLP - MultiLayer Perceptron
DT - Decision Tree
KNN - K-Nearest Neighbor
LDA - Linear Discriminant Analysis
GNB - General Naive Bayes
SVM - Support Vector Machine

Accuracy: total number of correct predictions with total number predictions

Precision: correct positive predictions to number of total positive predictions

Recall: actual number of positives that were caught by the model by labelling it as positive

F-Measure: weighted average of **both** precision and recall

OVERALL....

Multilayer perceptron model was the most accurate with:

97% accuracy   **97%** recall

96% precision   **97%** f-measure

Experiments weakness: Hyperparameter tuning, the process was all manually or iteratively done

Can be very time consuming

- Could be a pathway for future work and research!

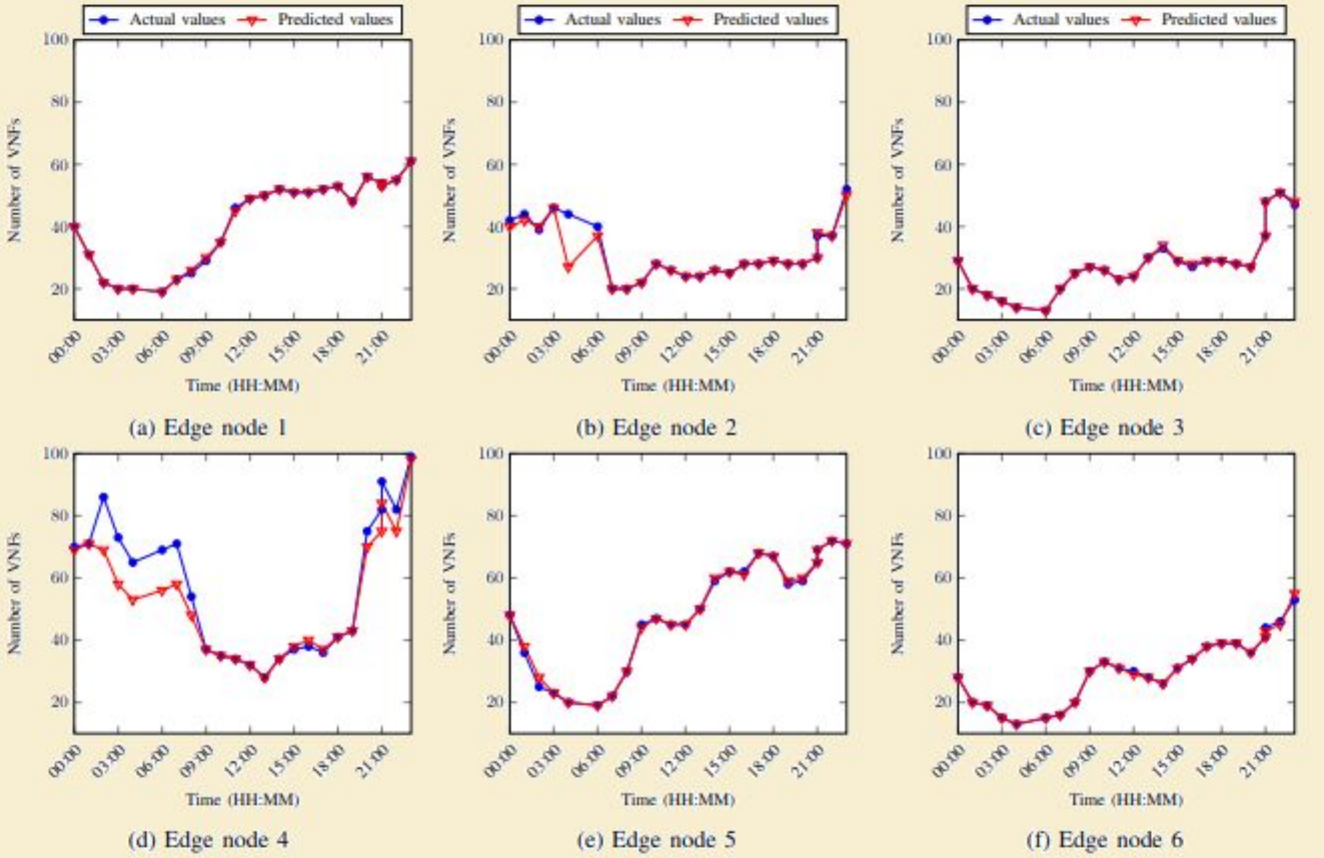


Fig. 4: Prediction results on the number of VNFs required at each edge node based on the proposed MLP model.

Figure displays MLP's prediction power , for all 6 nodes.

Blue: actual output

Red: Predicted VNF scaling decision

PART II
Integer Linear Programming(ILP) + Formulation

Latency Optimal VNF Placement Problem in MEC-NFV Environment

1. System Modeling
2. Problem Formulation
3. ILP Model Evaluation

Recall...

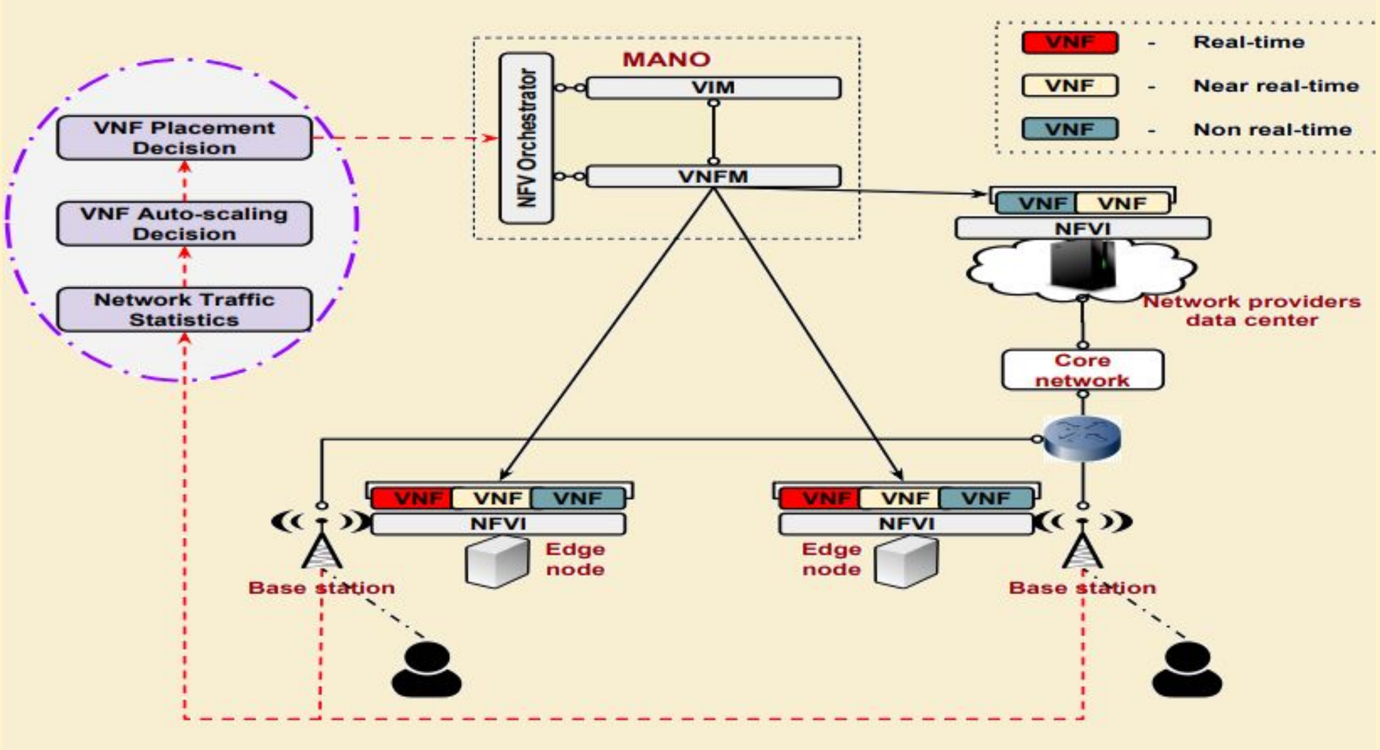


Fig. 1: A high-level distributed MEC-NFV System Architecture.

System Modeling

<u>**Goal**</u>: **Minimize** end to end latency by:

- placing VNFs on edge devices closes to end users
- Once VNFs run out of capacity *then* fall back to VNFs in the providers cloud data center

Table: defines all parameters used in the formulation

Notation	Definition
$G = (N, E, Z)$	Graph of the NFVI.
$N = \{n_1, n_2, ..., n_i\}$	Set of physical nodes (edge and distant cloud) within the network.
$E = \{e_1, e_2, ..., e_l\}$	Set of physical links in the network.
$Z = \{z_1, z_2, ... z_q\}$	Set of users associated with VNFs.
θ^i	Hardware capacity (CPU, memory, network) of the physical node $n_i \in N$.
δ^l	Capacity of the physical link $e_l \in E$.
d^l	Latency on the physical link $e_l \in E$.
$V = \{v_1^1, v_2^2, ..., v_j^q\}$	VNFs associated to users (e.g. $v_j^q \in V$ is associated to user $z_q \in Z$).
$P = \{p_1, p_2, ..., p_k\}$	All paths in the network.
ψ^j	Required capacity (CPU, memory, network) of the physical node to host VNF $v_j \in V$.
d^j_{max}	Maximum end-to-end latency threshold VNF $v_j \in V$ tolerates from its user.
X_{ijk}	Binary variable denoting if VNF $v_j \in V$ is hosted by physical node $n_i \in N$ using path $p_k \in P$.
b_{ijk}	Required bandwidth between VNF $v_j \in V$ to the user, if the VNF is hosted by physical node $n_i \in N$ using path $p_k \in P$.
d_{ijk}	Required latency between VNF $v_j \in V$ to the user, if the VNF is hosted by physical node $n_i \in N$ using path $p_k \in P$.

TABLE V: Key notations in our model.

System Modeling - Most important parameters

➔ Each VNF has its own: CPU, Memory, and Network requirements

➔ VNF has an end to end delay threshold (**d^j**) **AND** specifies a bandwidth requirement

➔ Latency from a user to a VNF(**d_{ijk}**)

➔ Decision variable(**X_{ijk}**) binary variable where 1 assign v_j to node n_i using path p_k

Problem Formulation

ILP model that takes in the following as **input**

- -Set of users(U)
- -Set of VNFs hosts(N)
- -Set VNFs (V)
- -Latency Array (d)

Then **outputs** optimal solution for VNF placement by minimizing the total end to end latency from all users

<u>Formulated Objective Function</u>

$$ILP: minimize \sum_{n_i \in N} \sum_{v_j \in V} \sum_{p_k \in P} X_{ijk} . d_{ijk}$$

Problem Formulation

Constraints of Optimization Objective: all help ensure the following:

$$\sum_{v_j^q \in V} \sum_{p_k \in P} X_{ijk}.\psi^j < \theta^i, \forall n_i \in N \quad (9)$$

$$\sum_{n_i \in N} \sum_{p_k \in P} X_{ijk}.d_{ijk} < d^j_{max}.\forall v_j^q \in V \quad (10)$$

$$\sum_{n_i \in N} X_{ijk} = 1, \forall v_j^q \in V, \forall p_k \in P \quad (11)$$

$$\sum_{n_i \in N} X_{ijk}.b_{ijk} < \delta^l, \forall e_l \in p_k, \forall p_k \in P \quad (12)$$

Constraint 9: ensures amount of hardware resources allocated to VNFs is within the available resources on the physical node

Constraint 10: end to end delay between user and VNF doesn't exceed the max delay

Constraint 11: each VNF is hosted by exactly one physical node

Constraint 12: none of the physical links becomes overloaded

ILP Model Evaluation

Model was evaluated using simulation experiments

Simulation Environment: Based on backbone network by a private Mobile Network Operator

- ➢ Edge nodes at all base stations and capable of hosting finite number of VNFs
- ➢ 1 Cloud data center capable of hosting several VNFs

ILP Model Evaluation

VNFs were categorized into 3 categories depending on latency tolerance levels

1. Real Time
2. Near Time
3. Non-real Time

Used **equal** number of VNFs in all 3 categories

ILP Model Evaluation

Using **IBM ILOG CPLEX:**

- 1st scenario: all VNFs are assigned to cloud data center
- 2nd scenario: VNFs assigned to edge nodes first, then to cloud data center once capacity runs out

Had a **fixed** latency of 5ms from user to edge nodes

Number of VNF hosted on each node = 40

Total edge capacity of network = 240 VNFS

-Once over 240; automatically gets assigned to cloud data center

ILP Model Evaluation

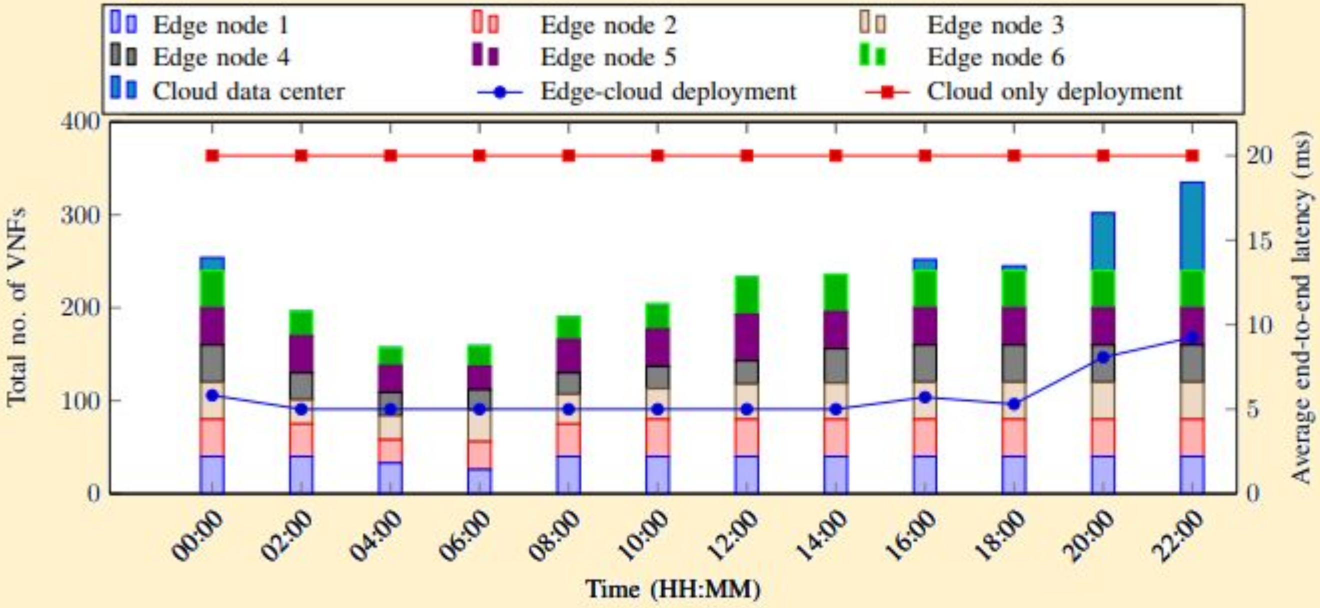


Fig. 5: Performance measure of the proposed system model.

ILP model took 6.25 seconds to place 335 VNFs to help minimize aggregated user to VNF end to end latency

RED: Cloud only deployment; **fixed** latency of ***20ms***
BLUE: Edge + Cloud; **lower** latency times(avg: **5ms**) increased when the edge nodes were exhausted

IN CONCLUSION..

- MLP was the most effective model in predicting amount of VNFs to deploy
  - Beneficial in **proactive** auto scaling
  - Helped minimize downtime and reduce operational costs
- Proposed a optimal placement model that carefully selects where to place VNFs to reduce user -> VNF latency
  - Results averaged 75% reduction in end to end latency when *all* VNFs were placed at the network edges
- Future work potentially with federated learning

Citation

T. Subramanya and R. Riggio, "Machine Learning-Driven Scaling and Placement of Virtual Network Functions at the Network Edges," *2019 IEEE Conference on Network Softwarization (NetSoft)*, Paris, France, 2019, pp. 414-422.

doi: 10.1109/NETSOFT.2019.8806631

keywords: {integer programming;learning (artificial intelligence);linear programming;neural nets;telecommunication traffic;virtualisation;virtual network functions;network operators;migrate VNFs;network edges;efficient VNF placement;continuously changing network dynamics;neural-network model;network traffic;commercial mobile network;Network Function Virtualization;machine learning-driven scaling;Predictive models;Cloud computing;Real-time systems;Data centers;Load modeling;Measurement;Base stations;Network Function Virtualization;Machine learning;Proactive Auto-scaling;Virtual Network Function Placement;Multi-access Edge Computing},

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8806631&isnumber=8806619