Artificial Intelligence in Computer Networks: Delay Estimation, Fault Detection, and Network Automation

Shady Mohammed

Thesis submitted to the University of Ottawa
in partial Fulfillment of the requirements for the
Ph.D. degree in Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Computer network complexity has increased in the last decades due to the introduction of various concepts, leaving network maintainers in hardship to manage such huge and tangled networks.

In this study, we aim to aid service providers to optimize and automate their networks. Currently, network maintainers perform a vast number of explicit measurements, which has a negative effect on the network's health and stability. Depending on the service's nature, measurements are either made at service initiation as in the case of server-client selection or continuously done to monitor the quality of service as in the case of quality assurance applications. We intend to apply artificial intelligence to minimize the dependency on such explicit measurements and hence, optimize the network with minimal cost. From the two types of applications, we focus on distributed delay measurements for Esports server-client selection problem as well as network automation and failure mitigation task done by Internet service providers.

In large-scale networks, it is impractical to measure the delay between every node explicitly. As a result, we propose an AI-based delay measurement estimator system. The system's inputs are just the source and destination nodes' IP-addresses.

Network maintainers continuously monitor their network status to detect any sudden change in the network and take suitable action(s) to keep the network in the best conditions. We propose an ML-based action recommender engine that is able to identify the current network status and suggest a set of actions that restore the network to its optimum state.

Acknowledgements

First, I would like to express my sincere gratitude and appreciation to my supervisor, *Prof. Shervin Shirmohammadi*, for his patience, unrelenting support, and constructive guidance. Without your encouragement and continuous motivation, I would not have been able to complete this work.

Moreover, I would like to thank Mitacs for sponsoring my work and giving me the opportunity to meet such a wonderful person and mentor as *David Côté*. David! Thank you for all of your support, understanding of our family situation during COVID-19, and constructive and insightful feedback.

Also, I would like to express my sincere gratitude to my mother-in-law, *Hatice Sulaoğlu*, for all the help, she provided during COVID-19.

"Anneciğim, tezimi sana da adadım. Senin fedakârlıkların ve yardımların olmadan doktoramı bitiremezdim."

Finally, I would like to thank my grandfather *Mokhtar El Refaie* for leaving his entire world and moving in to help me to be what I am today.

To my wife

Rumeysa

for being an amazing partner and carrying me during my research by taking care of everything else

To my Sons

Yusuf & Selim

Cheer up, more outdoor activities to come!

To my parents

Taghrid & Ashraf

for sacrificing everything possible to get me where I am today

This would not have been possible without you all.

Table of Contents

List of Figures

List of Tables

List of Definitions

ARE	Action Recommender Engine
AS	Autonomous System
BP-UAA	Blue Planet Assurance and Analytics
CDF	Cumulative Distribution Function
CM	Confusion Matrix
CNN	Convolutional Neural Network
DApp	Distributed Application
DASH	Dynamic Adaptive Streaming over HTTP
Deme	Delay Measurement Estimator
DL	Deep Learning
DMF	Decentralized matrix factorization
DPI	Deep Packet Inspection
DT	Decision Tree
FFNN	Feed Forward Neural Network
GB	Gradient Boosting
GNN	Graph Neural Network
GNP	Global Network Positioning
IDES	Internet Distance Estimation Service
IDMaps	Internet Distance Map Service
ISP	Internet Service provider
KNN	K-Nearest Neighbor
MDN	Multimodal Deep Learning Network
MF	Matrix factorization
ML	Machine Learning
MLOps	Machine Learning Operations
MLP	Multilayer Perceptron
NAMT	Network Assurance and Management Tools
NCS	Network Coordinates System

NFV	Network Function Virtualization
NOC	Network Operations Center
NPS	Network Positioning System
OPEX	Operational Costs
p2p	Peer-to-Peer
QoE	Quality of Experience
QoS	Quality of Service
ROA	Route Optimization, Analysis
SDN	Software-defined Networking
SLA	Service Level Agreement
SVM	Support Vector Machine
TE	Traffic Engineering
XGB	Extreme Gradient Boosting

Chapter 1. Introduction

1.1 Motivation

With the rapid advancement of the Internet of Things (IoT), Network Function Virtualization (NFV), and Software-defined Networking (SDN), computer networks have remarkably become more diverse and complex [1]–[3]. Moreover, computer networks have become service-oriented [4], [5]. Based on the service, network maintainers take vital decisions to optimize their network to fulfill the application constraints. Accordingly, a monumental number of measurements need to be performed periodically in a short time. These measurements provide network maintainers with information regarding instant network behaviour and enable them to interact. Two types of applications heavily depend on network measurements; Distributed Applications (DApps) and Network Assurance and Management Tools (NAMT). DApps are an example of the first type of application that requires initial measurements to start its service [6]. Realtime DApps such as Esports, media streaming, and peer-to-peer (p2p) file-sharing (e.g. Bit-Torrent clients) perform a significant amount of measurements to identify each client's delay-bandwidth profile [7]–[9]. Based on the collected measurements, a network slice is created to satisfy service constraints and connect each client to the best available server and to peers who share a similar delay-bandwidth profile [6]. While on the other hand, NAMT collects measurements continuously to have an updated insight into the network devices' status [10]–[16] and hence, the network's health. Internet Service providers (ISPs) adopt various NAMT such as Network automation, network quality assurance, and network failure management to closely monitor their network status. Such

monitoring processes enable ISPs to meet Service Level Agreements (SLA), demands of customers and, in the meantime, reduce their operational costs, i.e., maximize revenue.

This study focuses only on delay measurements related to DApps and Network Automation and Failure Mitigation for ISPs.

1.2 Challenges and research problem

In this section, we present the challenges addressed in this study. First, we discuss challenges facing distributed delay measurements then the challenges facing network automation.

For Large-scale distributed networks with N-nodes, it is impractical to perform explicit end-to-end delay measurements between all network pairs in the network. Such measurements need $O(N^2)$ operations [17]. These monumental operations are not only considered as computational overhead on each network node but also create a tremendous network traffic overhead, which can lead to the network's failure. The solution is to predict the delay between the nodes instead of explicitly measuring it. However, the available prediction methods are not accurate enough, require non-trivial time to reach a steady-state, and some of the algorithms are not proven to converge. Hence, a new system should consider the measurements overhead as well as minimize the steady-state time.

Network maintainers continuously monitor their network status to detect any sudden change in network behaviour and take suitable action(s) to keep the network in the best conditions. One traditional way of maintaining such services is to provide network maintenance manually. For instance, a network technician is always required to be an in-person service call to identify the network problem, i.e. network status. Then, this person takes suitable action(s) to either reroute end-users' packets or to fix any physical issues such as router

failures or link disconnections occurring in the network. This process is time-consuming, and it negatively affects the quality of service. Hence this leads to degradation of the quality of experience, which causes a decrease in customer satisfaction. Another traditional way of detecting network status is to use Deep Packet Inspection (DPI), a packet filtering technique for inspecting packets and extracting information to classify network traffic. However, this method fails to work on encrypted packets that form most of the Internet's data packets[18]. Furthermore, on some occasions, DPI causes packet delay due to additional computations. As a result, DPI may lead to an inaccurate performance in terms of network status, hence inappropriate action(s), which comprises devastating impacts on end-users. To cope with the rapidly changing nature of computer networks, a new system is required to rapidly process data in realtime.

1.3 Research Approach

This thesis focuses on the above-mentioned challenges and tries to address the following technical questions:

1. How to accurately predict latency in a large-scale network as well as minimizing traffic and computational overhead?
2. How to predict network status without much processing and in realtime?
3. After identifying the network state, in case of any abnormalities, what can be the recommended actions to restore the network to the original state?

The approach to addressing the above technical challenges is as follows:

Although there are several methods for estimating the end-to-end latency in large-scale distributed networks [17], they suffer from

inaccuracy, high settling time, and they still require sets of active measurements to adjust and calibrate themselves. We are interested in methods that work passively and do not impose any further communication burden on the network. To this end, we take advantage of Machine Learning (ML), aiming to find simple, lightweight, and instantaneous prediction schemes for the above-mentioned challenges. Machine learning tries to construct algorithms and models that can learn to make decisions directly from data without following pre-defined rules. Besides, ML does not require any settling time other than the training time. However, ML may need to be periodically fine-tuned to cope with any change in the network conditions and traffic behaviour.

For network automation and failure mitigation, utilizing ML in such issues is becoming a de facto in the network industry [19]–[21]. Tools like Ciena's Blue Planet Assurance and Analytics (BP-UAA) [22] Route Optimization, Analysis (ROA) [23], or Cisco's DNA Analytics and Assurance [24] are examples of the industry moving towards this direction. These tools use ML to help network maintainers gain deep insights into the network to make intelligent data-driven decisions that lead to improved efficiency, lowered costs, and providing more personalized services. To follow this stream, we employ an ML-based system to predict network status and then recommend an action to self-heal the network.

1.4 Intended Contributions

The intended contributions of this thesis are as follows:

- We propose a multi-modal Deep Learning (DL) based distributed network latency measurement system. The proposed system would estimate the latency between any arbitrary network pairs in near-realtime, even if there were

no prior contact between them. The only inputs of the system are the two IP addresses of the network pairs.

- We propose an ML-based system to predict the network's state, i.e., issue-free, congested, or link disconnected and recommend a suitable action, i.e., reroute specific traffic or fix a given link, to restore the network of interest to a normal state. The proposed system nearly operates in real-time. The proposed system inputs are a combination of Quality-of-Service metrics extracted from Cisco's Service-Level agreement and Quality of Experience (QoE) collected from end-users.

1.5 Research Publications

1.5.1 Journals

1. Shady A. Mohammed, Shervin Shirmohammadi, and Sa'di Altamimi. "A Multi-modal Deep Learning-Based Distributed Network Latency Measurement System." IEEE Transactions on Instrumentation and Measurement, 69.5 (2020): 2487-2494.
2. Shady A. Mohammed, Ayşe Rumeysa Mohammed, David Côté, and Shervin Shirmohammadi. "A Machine-Learning-Based Action Recommender for Network Operation Centers" 2021 I EEE Transactions on Network and Service Management (IEEE TNSM).
3. Ayşe Rumeysa Mohammed, Shady A. Mohammed, David Côté, and Shervin Shirmohammadi. "Machine Learning Based Network Status Detection and Fault Localization." 2021 IEEE Transactions on Instrumentation and Measurement, 70, (2021):1-10.

1.5.2 Refereed Conferences

1. Shady A. Mohammed, Shervin Shirmohammadi, and Sa'di Altamimi. "Artificial intelligence-based distributed network latency measurement." 2019 IEEE International Instrumentation and Measurement Technology Conference (I2MTC). IEEE, 2019.
2. Ayşe Rumeysa Mohammed, Shady A. Mohammed, and Shervin Shirmohammadi. "Machine learning and deep learning-based traffic classification and prediction in software defined networking." 2019 IEEE International Symposium on Measurements & Networking (M&N). IEEE, 2019.
3. Alchalabi, Alaa Eddin, Shervin Shirmohammadi, and Shady A. Mohammed. "QNetwork: AI-Assisted Networking for Hybrid Cloud Gaming." 2019 IEEE International Symposium on Measurements & Networking (M&N). IEEE, 2019.

1.5.3 Refereed Magazines

1. Shady A. Mohammed, Shervin Shirmohammadi, Alaa Eddin Alchalabi, "Measurement Prediction of Network Delay with Deep Learning: From Lab to the Real World", IEEE Instrumentation and Measurement Magazine (submitted)

1.5.4 Patents

1. Karthigesu Vijayasuganthan, Sorin Stoian, Shervin Shirmohammadi, Shady A. Mohammed, and Alaa Eddin Alchalabi "Methods and apparatus for network delay and distance estimation, computing resource selection, and

related techniques." U.S. Patent No. 11,063,881. 13 July 2021.

2. David Côté, Ayşe Rumeysa Mohammed, Shady A. Mohammed, and Shervin Shirmohammadi. “Action Recommendation Engine based on Machine Learning to support closed-loop applications for telecommunications networks”, Attorney Docket Number: 10.2767 – **Pending.**
3. David Côté, Shervin Shirmohammadi, Shady A. Mohammed, Sa'di Altamimi, and Basel Altamimi. “Action Recommendation Engine (ARE) for Network Operation Center (NOC) solely from raw un-labeled data”, Attorney Docket Number: 10.2846 – **Pending.**

1.5.5 Awards

1 “Action Recommendation Engine based on Machine Learning to support closed-loop applications for telecommunications networks” – Best idea in 2020 Ciena’s Think Tank Forum.

1.6 Thesis road map

The road map for the rest of this thesis is outlined below:

Chapter 2 – Background presents background information on Machine learning and deep learning algorithms.

Chapter 3 – Related works presents an extensive review of the existing works in the areas of distributed latency estimation as well as network automation and failure mitigation.

Chapter 4 – Proposed Delay measurement Estimator System presents the proposed system and explains its components. We present the KING dataset, which is used in both training and testing, data wrangling techniques used to prepare our data for training, and the multi-modal deep learning predictor.

This chapter reuses material from journal No. 1.5.1-1 and paper No. 1.5.2-1 with permission.

Chapter 5 – Machine Learning Operation for Deme presents a real-life implementation of Deme. It focuses on the downsides of any deployed machine learning model in general and the potential DevOps solution.

This chapter reuses material from magazine No. 1.5.3-1 with permission.

Chapter 6 – Proposed Network Automation System presents the proposed system and explains its components. We also present the simulation setup, the implemented scenarios used in data collection. Furthermore, we present the collected dataset and the potential ML-models which can be suitable for our dataset.

This chapter reuses material from journal No. 1.5.1-2 and 1.5.1-3 with permission.

Chapter 7 – Testing and Evaluation is divided into two sub-chapters. The first part presents the preliminary results of the delay measurement estimator system and a comparison to the state of the art of delay estimation domain. The second part introduces Network Automation and Failure Mitigation preliminary results and a cost comparison with Network operation center rules.
This chapter reuses material from journal No. 1.5.1-1, 1.5.1-2, 1.5.1-3, paper No. 1.5.2-1, and magazine No. 1.5.3-1 with permission.

Chapter 8 – Conclusion and Further Research work discusses our future research plans on the proposed Delay measurement estimator system as well as for Network Automation and Failure Mitigation.

Chapter 2. Background

2.1 Machine Learning

Machine Learning (ML) is the terminology to describe the mathematical formula that finds the correlation between a set of data so that when new inputs are introduced, it can drive similar statistical distribution [25].

2.1.1 Types of Machine Learning

Machine learning can be unsupervised, semi-supervised, and supervised. Unsupervised learning is also referred as pattern recognition. Without prior knowledge or labels, it groups the data through identifying similarities and discovering knowledge among every instance in the dataset [26]. On the other hand, supervised learning is a learning paradigm that takes labelled data as input to extract features that carry the necessary information to deduct the data label [26].

2.1.2 Ensemble Learning

Ensemble learning adopts a different approach than any other machine learning algorithms. Instead of training one accurate model, it trains numerous weak learning models and then aggregates them into an accurate hybrid model [26]. Weak learning models cannot learn higher-level complex models, but they produce fast results. Oftentimes, the choice for a weak learner is the Decision Tree. It splits the dataset into small subsets while incrementally building a tree-like structure with decision nodes and leaf nodes. These resultant trees are not very powerful but once combined, produce a better guessing algorithm [27]. With the introduction of

every new data point, each tree votes an output. Then through weighted voting, the prediction for that input is determined.

There are two main ensemble learning concepts: bagging and boosting. Bagging duplicates many copies of the training data, applies weak learners to those copies in parallel, independently from each other, and then it averages them since its focus is to produce an ensemble model with the least variance. On the other hand, boosting trains the weak learners sequentially; each model learns from the mistakes of the previous model and builds on top of its predecessor to create a stronger model with the least bias. Gradient Boosting (GB) and Extreme Gradient Boosting (XGB) are two commonly practiced boosting models. GB trains a weak learner then computes the residual errors of the trained model [28]. The labels in the training dataset are replaced with the residuals. This modified dataset is used to train the next model in the sequence. Each iteration finds the errors in the previous model, fixes its shortcomings and updates to a better version, the current model. However, due to its sequential model training, GB is slow in deployment and introduces scalability issues in the presence of large datasets. On the other hand, XGB is an optimized version of GB in terms of computational speed and model performance through parallelization of the tree construction process and distributed computing [29].

2.2 Deep Learning

Deep learning (DL) is a particular kind of machine learning (ML) inspired by the human brain's structure and functionality. It achieves high effectiveness by processing data through hierarchical concepts called layers. In general, a DL architecture consists of three layers: Input, Hidden, and Output. Each layer comprises a pre-

defined number of neurons that extract features and find patterns [30].

The simplest representation of a DL model can be a combination of a single perceptron. Unification of perceptrons forms Multilayer Perceptrons (MLPs). MLPs can be considered as the simplest representation of a DL model. They usually have a primary architecture, including one input, one hidden, and one output layer. All perceptrons, i.e. neurons, apply a linear activation function that takes a decision depending on the activation [31]. For MLPs, these activation functions are usually sigmoid function.

Different DL models are built by taking MLPs as their foundational blocks. Each DL model evolved into an advanced version of simple MLPs and specialized in handling various tasks such as decision making, speech recognition, language translation, image classification, object detection, fraud detection, etc.

In our experiments, we employed feedforward neural networks, convolutional neural networks, and multi-modal neural networks. In the next sections, we will explain them in detail.

2.2.1 Feed Forward Neural Networks

Feed Forward Neural Networks (FFNNs) or MLPs are a type of neural network where information moves forward from the input layer passing through hidden layer(s) and to output later, [32]. Its single mechanism doesn't include loops, recursive feeds, feedback connections, or backward information flow. See Figure 2-1. These networks are generally used for supervised learning tasks where the labels and the target function are already known. Hence the fundamental aim of FFNNs is to approximate some function that is close to the target function.

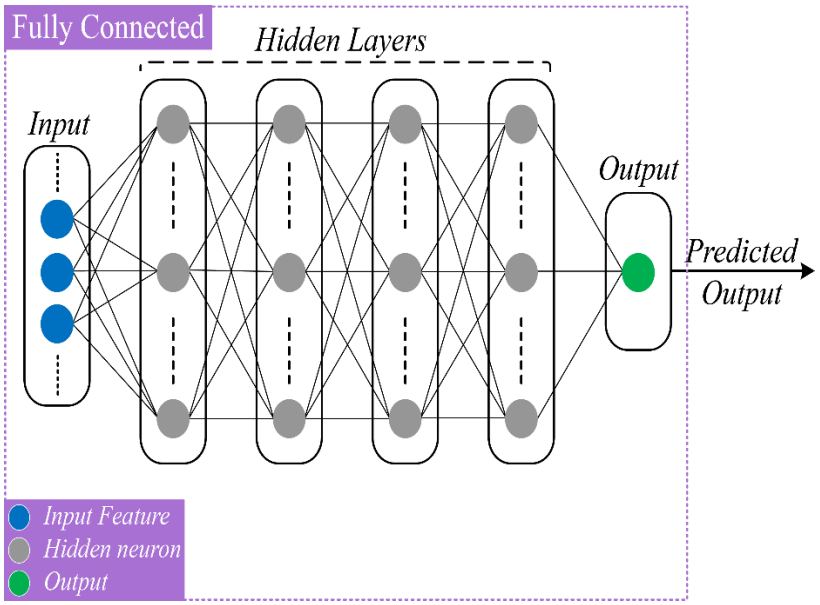


Figure 2-1 Fully Connected Layers with Single Output.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) is a type of neural networks that are specialized to process data with a spatial correlation between its neighbouring data points such as time series and image data [33]. CNNs mainly have two components (i) hidden layers where feature extraction takes part through convolutions and pooling operations and (ii) classification where classifiers are built on top of the extracted features and predicts probabilities for the output classes, see Figure 2-2.

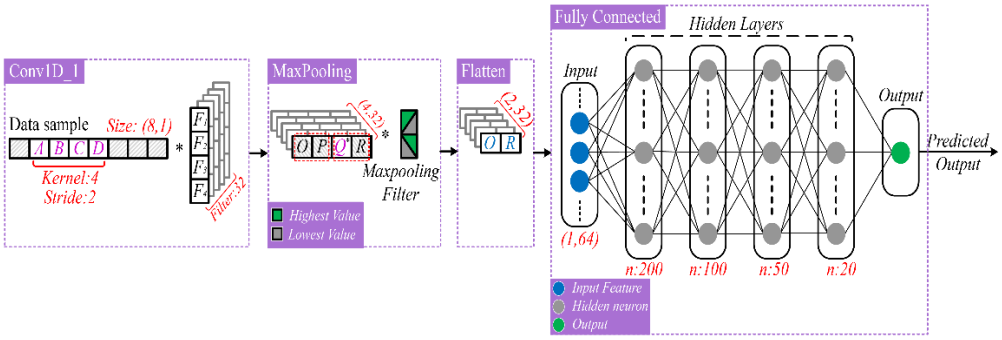


Figure 2-2 Convolution Layers Followed by Fully Connected Layers.

In hidden layers, CNNs deploy a linear mathematical operation called convolution in at least one of their layers. Convolution is the mathematical definition of combining two entities to yield a third function. Convolution is executed on the input data with a kernel (or filter), which then produces a feature map, see Equation 2-1. Convolution operation is executed via sliding the kernel with stride (the step that the kernel slides at time t) size over the input data while at every slide, a matrix multiplication takes places and the resultant sum is assigned as the feature map. 2D representation of a convolution operation is shown in Figure 2-3. In real-life applications, convolutions are operated in 3D matrices. For example, convolutions for an image would produce a 3D matrix with the dimensions being width, height and depth, where depth refers to the colour channels RGB. After a convolutional layer, usually, a pooling layer is added for dimensionality reduction purposes [34]. A general choice for a pooling layer is **max pooling**, which takes the feature map and returns its maximum values, see Figure 2-4.

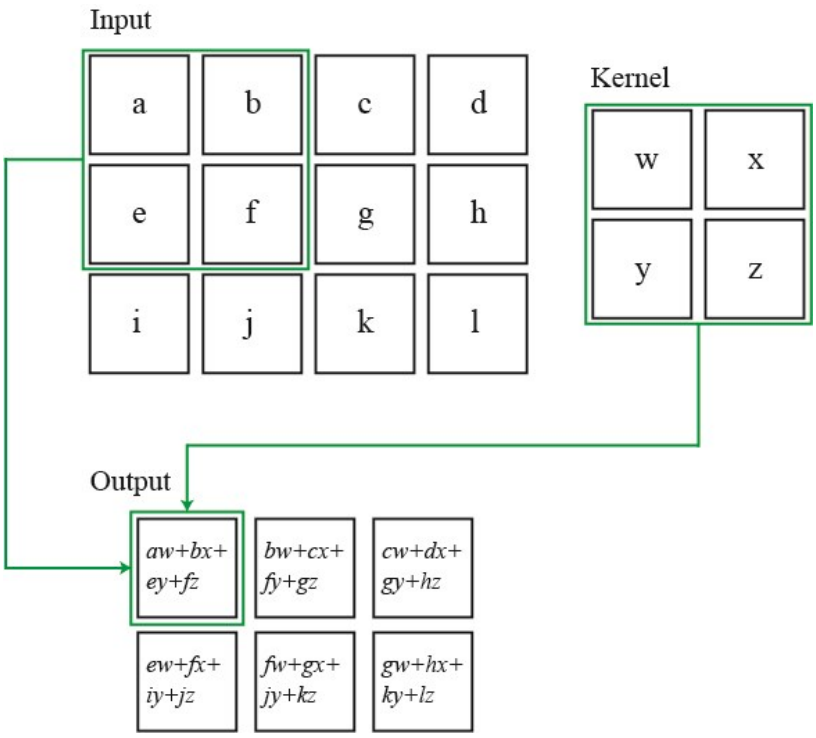


Figure 2-3 An Example of 2-D Convolution Without Kernel-Flipping.

$$Convolution(t) = (input * kernel)(\mathrm{t}) \qquad \textbf{Equation 2-1}$$

Performing convolutions over the input data result in creating distinct feature maps. These feature maps are combined and presented as the output of the convolutional layer of the CNN model. Then an activation function, generally ReLU, is applied to introduce non-linearity.

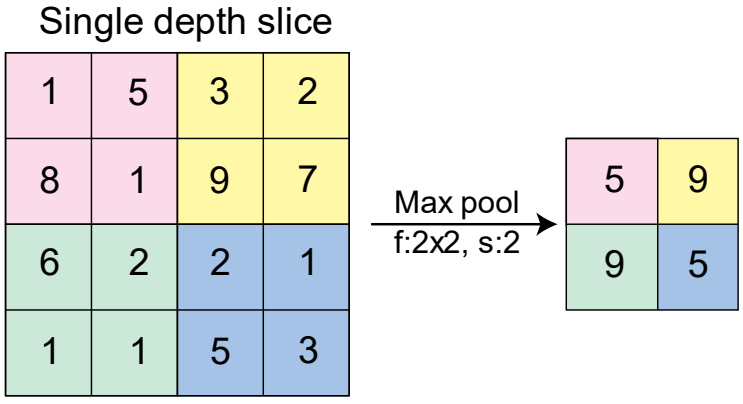


Figure 2-4 An illustration for Max-pooling Operation.

After the convolutional and pooling layers, the produced output is given to the classification. Classification consists of several fully connected layers. The last layer of a classification component yields the probabilities of the predictions for the output classes.

2.2.3 Multi-modal Deep Learning

The architecture of neural networks is inspired by the functionality of human brains, whereas Multimodal Deep Learning Networks (MDNs) are inspired by the way humans experience the outside world [35]. Humans understand through their senses; touch textures, see objects, hear sounds, smell fragrance, and taste flavours to comprehend the world around them. This modality is then combined in our brains to conclude a decision. In that vein, a multi-modal deep learning approach is emerged to interpret a variety of data to draw a conclusion. For example, videos usually consist of images, sounds, and texts. To interpret such various data structures, we need modality. A frequent way to implement this is to apply different sub-networks on each raw input and then integrate them into a multi-modal neural network.

Another implementation of MDNs is when the data is of the same type but significant variance. The dataset is grouped based on the feature characteristics of its instances. According to the task, a specialized neural network is chosen and trained on each dataset group separately. Hence, each model learns the features of the segment they are trained in. After fine-tuning by removing or adding extra layers to the models, the versions that yield the best results are chosen. The extra layers are created for the multi-modal neural network. The outputs of the sub-models are taken to be fed to the multi-modal network.

2.3 Machine Learning Operations

2.3.1 Machine Learning Design and Operation

According to Gartner [36], 85% of Machine Learning projects fail. Worse yet, the trend is predicted to continue through 2022. Data scientists and machine learning engineers train and implement their machine learning model with a satisfiable offline predictive performance. However, the trained model loses its high performance once it moves to production. The reason behind the performance degradation is that the training dataset is statistically different from the incoming inference data. This is due to the varying nature of real-world problems.

The real challenge besides designing a high-performance machine learning model is to maintain its performance all the time and in every situation. The solution is to integrate the machine learning model with a system that continuously operates and maintains it, see Figure 2-5. In the figure, the system consists of automation, data collection and verification, testing and debugging, resource management, model analysis, serving infrastructure, monitoring, etc. This system is known as *Machine Learning Operations (MLOps)*. MLOps is treated machine learning model as a software program that is subjected to all the DevOps practices (a.k.a. pipelines) such as Continuous Integration (CI) and Continuous Development (CD). Moreover, MLOps adds another practice which is Continuous Training (CT). CT means that the machine learning model is monitored and its performance is always compared against a baseline threshold. In case of any performance degradation below the threshold, the CT pipeline triggers a fine-tune task or even retraining trying to restore the model to its high performance.

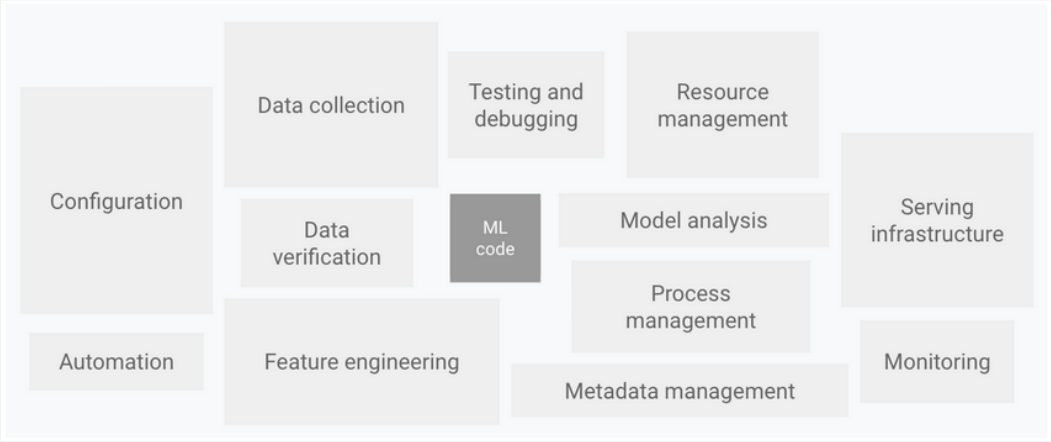


Figure 2-5 Elements for ML systems [37].

2.3.2 MLOps Levels

The level of automation defines the level of maturity of the MLOps process. In other words, this shows how fast the machine learning model can be trained and put back in production.

2.3.2.1 MLOps level 0: Manual process

This is the most basic level of the MLOps implementation. Data scientists and machine learning engineers perform all the designing and maintaining tasks manually. At this level, there is not any kind of automation or monitoring. Moreover, deployment, in this stage, means prediction service.

2.3.2.2 MLOps level 1: ML pipeline automation

At this level, the goal is to perform continuous training. This is done by automating the process of data ingestion, processing pipelines and deploying retrained models to production, i.e., continuous delivery. However, there is no model monitoring and continuous integration.

2.3.2.3 MLOps level 2: CI/CD pipeline automation

This is the most advanced level the MLOps can reach. At this level, rapid and robust fully automated CI/CD/CT pipelines are established. Also, the model performance is continuously monitored and always compared against a threshold.

2.4 Data Analysis (Performance Analysis)

2.4.1 Confusion Matrix

Confusion Matrix (CM) is one of the performance measurements for ML classification problems of 2 or more classes [38]. It is a table displaying predicted and actual values of the classes. In Figure 2-6, a CM for binary classification is presented. Let us assume that we have two classes named as Positive and Negative. From the provided values in the CM, we can define the following terminologies:

- True Positive (TP): prediction and label are positive
- False Positive (FP): prediction is positive, but the label is negative
- False Negative (FN): prediction is negative, but the label is positive
- True Negative (TN): prediction and label are negative

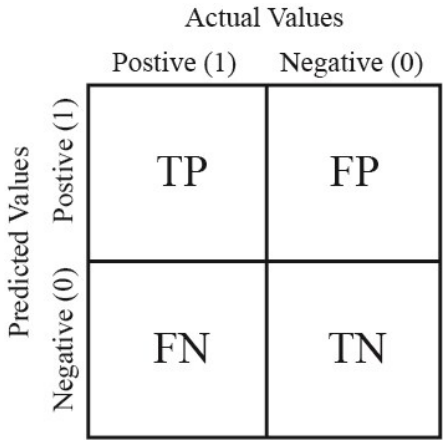


Figure 2-6 An Illustration for Confusion Matrix Calculations.

Now, we can derive several equations for analysis:

- Recall: shows out of all the Positive class, how many of them we predicted correctly. The higher the recall value, the better the model is.

$$Recall = \frac{TP}{TP + FN} \qquad \textbf{Equation 2-2}$$

- Precision: shows out of all the positive class predictions, how many of them are Positive. The higher the precision value, the better the model is.

$$Precision = \frac{TP}{TP + FP} \quad \text{Equation 2-3}$$

- Accuracy: shows out of all the classes, how many of them we predicted correctly. However, taking the proportion of correct over total predictions does not clearly represent the model's efficiency due to (i) if there are more than two classes in the dataset, which class the model predicted well will not be clearly observed.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad \text{Equation 2-4}$$

- F-measure: also known as an F1 score, is developed to compare the performance of models with high recall and low precision by taking the harmonic mean.

$$F - measure = \frac{2 * Recall * Precision}{Recall + Precision} \quad \text{Equation 2-5}$$

2.4.2 Relative Error

Relative Error (RE) is the measurement of precision that takes the ratio of absolute error to the measured value. RE is very useful when the comparison of the measured entities is in different units. For example, when comparing the height and the weight of an object, RE show relatively which one is more precisely measured.

$$RE = \frac{|Actual - Predicted|}{\min(Actual,\ Predicted)} \quad \text{Equation 2-6}$$

2.4.3 Monte-Carlo Dropout

Monte-Carlo Dropout is a technique of dropping a certain percentage of neurons during training and during runtime [39], [40]. Applying

Monte-Carlo Dropout gives an insight into how the network behaves with any arbitrary data with a statistical skew. In other words, Monte-Carlo Dropout measures the deep learning network's uncertainty.

Chapter 3. Related Works

In this chapter, we present related works for distributed delay measurements as well as network automation.

3.1 Distributed Delay measurements

Network delay can be either explicitly measured or predicted, depending on the size of the network. For a considerable network with N-hosts, it is impractical to have explicit end-to-end delay measurements between all network pairs. Such measurements need $O(N^2)$ operations. These massive operations are not only considered as computational overhead on each node in the network but also create a tremendous network traffic overhead, which can lead to the network's failure. These problems have led to the development of network delay prediction systems.

Network coordinates systems (NCS) is the most popular delay prediction systems. NCS requires a few sets of explicit delay measurements, and then it predicts the network delays between any pairs of network nodes even without any prior RTT measurement between them [17]. There are two major models for implementing the NCS: Euclidean Distance Model and Matrix Factorization Model.

3.1.1 Euclidean Distance Model

Euclidean distance-based NCS virtually maps network nodes to a set of finite-dimensional coordinates, usually 2 or 3 dimensions. In the beginning, each node performs a set of few specific delay measurements to calculate its coordinates, such that the Euclidean distance between any node pairs represents the delay between them. After calculating nodes' coordinates, delay prediction can be achieved by calculating the Euclidean distance between any node pairs even

without any prior contact. Equation 3-1 shows the calculation of the distance between hosts N and M.

$$E_{N,M} = \|(N - M\| = \sqrt{(N_x - M_x)^2 + \left(N_y - M_y\right)^2 + \cdots} \qquad \text{Equation 3-1}$$

NCS is either centralized or decentralized. Centralized NCS requires a fixed number of well-known nodes called "Landmarks". Landmarks act as reference points in coordinates calculation for all network nodes. On the other hand, decentralized NCS does not require any reference in coordinates calculations. Decentralized NCS is a lightweight and scalable architecture that fits well with large-scale networks.

3.1.1.1 Centralized NCS

Internet Distance Map Service (IDMaps) was the first system to predict network delay between network nodes [41]. IDMaps uses Internet Autonomous Systems (AS) as the smallest unit in the network. Within the Internet network, IDMaps has distributed landmarks called "Tracers". IDMaps performs delay prediction by first identifying the AS of the two endpoints. Then, each of the two endpoints nodes calculates the distance to the nearest tracer. The distance between the nearest two tracers is simply the summation of the distance between intermediate tracers in between them.

This approach is not easily feasible since it requires the knowledge of all AS within the Internet to distribute its Tracers accurately. Moreover, delay prediction suffers from cumulative error propagation between Tracers. Finally, Tracers are network bottlenecks since every tracer is required to reply to a vast number of requests that might lead to its failure.

Global Network Positioning (GNP) is considered the first P2P architecture [42]. GNP has two phases; one for landmarks

coordinates calculations, while the other is for ordinary users' coordinates calculations. In the first phase, each landmark (L) calculates its own coordinates by performing a few sets of explicit delay measurements to other landmarks, as shown in Figure 3-1a. In the second phase, regular users calculate their own coordinates by actively probing all the available landmarks.

The challenge in this approach is in landmarks selection. First, landmarks nodes need to be reliable since each of them is required to reply to all network requests, which creates network bottleneck. Second, landmarks need to be distributed among the network to represent the network size and depth.

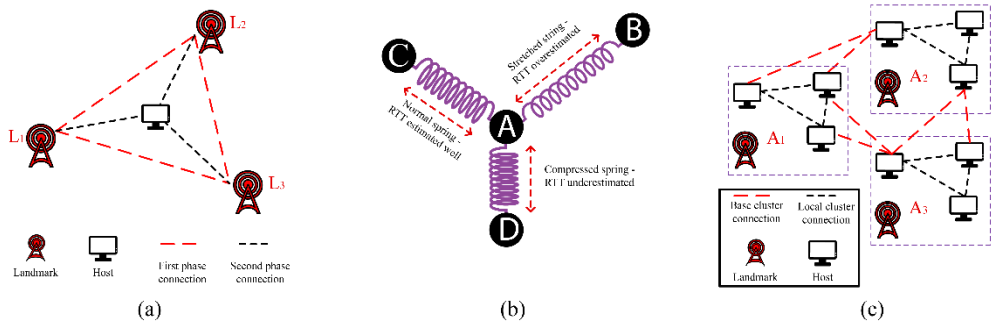


Figure 3-1 Network Coordinates Systems (a) GNP (b) Vivaldi's spring mode. (c) Pharos.

Lighthouses authors modified GNP's criteria for choosing landmarks [43]. GNP chose the same well-known landmarks as references in other nodes coordinates calculation. Instead, Lighthouses allows any node to depend on a set of nodes chosen randomly. By allowing nodes to do so, Lighthouses reduces the probability of network failure by eliminating network bottlenecks. However, this led to performance degradation since the randomly chosen nodes do not have any confidence measurements. In other words, the randomly chosen nodes may have a higher error that leads to inaccurate coordinate calculations.

Network Positioning System (NPS) is based on GNP and applied lighthouses modification but in a hierarchical approach [44]. NPS

allows any node to act as a reference landmark to other nodes in the network. NIPS's network is divided into layers. The first layer, layer 0, has well-chosen and fixed landmarks. The other layers are formed by each layer probing its previous layer. The advantage of this approach is that new nodes do not have to probe the well-known landmarks, and in case of any landmark failure, the system does not affect due to the hierarchical approach.

3.1.1.2 Decentralized NCS

Vivaldi is the most popular and widely used decentralized algorithm because it does not require any infrastructure to run [45]. Vivaldi maps each node to a virtual N-dimensional Euclidean model space. Vivaldi's error model is based on Hooke's Law. Figure 3-1b shows the three possible spring scenarios: normal ($E = A$), stretched ($E > A$), and compressed ($E < A$). Vivaldi computes a confidence weight for each node to handle significant relative errors. Moreover, it adopts an adaptive timestep that helps nodes with high error in coordinate calculations move larger steps in the correct direction compared with the nodes with smaller errors to avoid system oscillation [46]. It is reported that Vivaldi fails to maintain high accuracy between nodes with quite a small delay. This problem is known as the short-link distance prediction [17].

Pharos, Figure 3-1c, solves the short-link distance prediction problem by adopting the clustering approach and applying Vivaldi within the cluster [47]. Each cluster has a delegated node called "Anchor". The anchor's responsibility is to reply to ping requests coming from news nodes joining the network. Each node in the system has two sets of coordinates local and base coordinates. Local coordinates are for short-link distance, i.e. intra-cluster communications. In contrast, base coordinates are for medium-link to long-link distances, i.e. inter-cluster communications. When a new

node joins the system, it actively probes to all the anchors and joins the nearest cluster. After joining the cluster, the new node starts applying Vivaldi's algorithm to calculate the two sets of coordinates. Some other methods refine or improve Vivaldi, for example, authors in [48] presented a fully decentralized system inspired by Vivaldi, with the improvement that unlike the original Vivaldi, the authors' proposed system can be proven mathematically to converge.

3.1.2 Matrix Factorization

Matrix factorization (MF) takes advantage of classifying the Internet into ASes [49]. Nearby nodes, nodes within the same AS, have approximately the same delay to all remote nodes, i.e. nodes that belong to other ASes. Figure 3-2 shows a BGP example of 4 ASes. In AS1, node (A) and (B) have almost equal delay measurement to any node belongs to any different AS, nodes (F), for example.

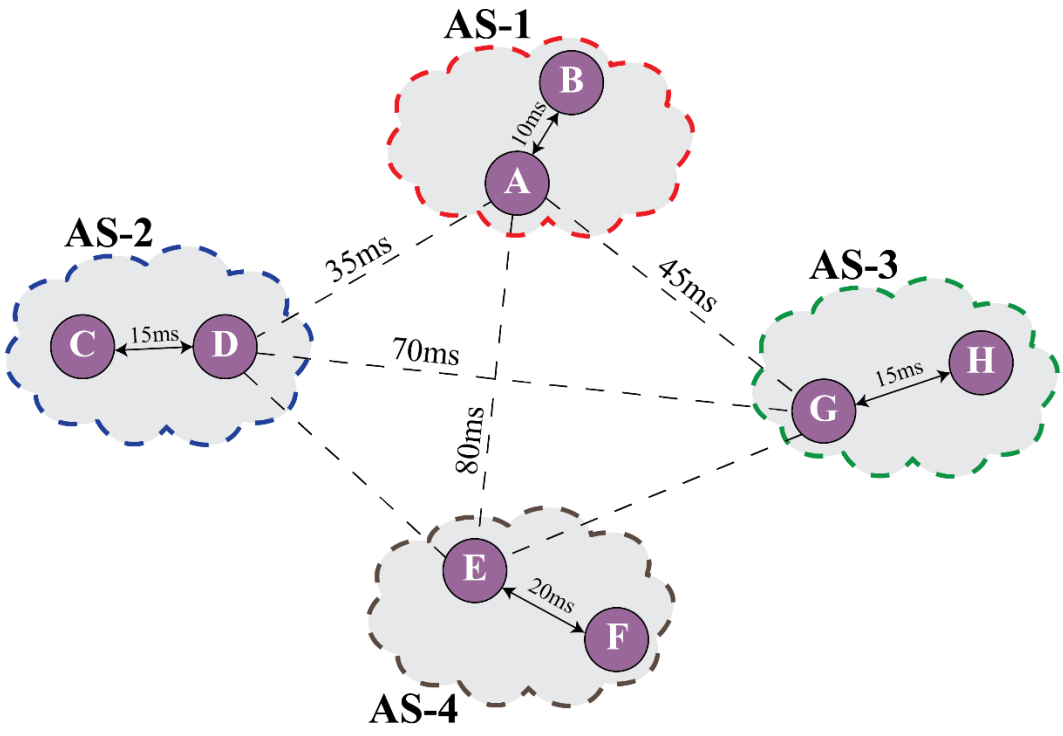


Figure 3-2 BGP Example of 4 ASes.

Based on the above assumption, the network matrix (M) of a large-scale network has almost typical rows for nearby nodes. Moreover, several rows could be expressed as the linear combination of the other rows.

MF tends to factorize the network matrix (M) to two lower lever matrices: the incoming (X) and outgoing (Y) (Y) matrices where $M = X.Y^T$. Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF) are the two well-known methods used for decomposing the network matrix (M). MF is also divided into two main categories; centralized or decentralized [49]–[51].

3.1.2.1 Centralized MF

Internet Distance Estimation Service (IDES) is the first matrix factorization system introduced [52]. IDES is very similar to GNP in both architecture and two-phase operation. IDES utilizes the idea of well-known Landmarks (L), but it has an extra node called "Information Server". In the beginning, all landmarks measure the distance between each other in the network. Then, each landmark reports a matrix of dimension 1xL to the Information server. The information server combines the reported matrices to form a network delay matrix of $D = LxL$. After that, it applies the SVD algorithm on the distance matrix to decompose it into an incoming matrix (X) and an outgoing matrix (Y) that will be used by any new host joining the system. When a new node wants to join the system, It contacts the Information server to get the landmarks' incoming and outcoming matrices and then start calculating its coordinates. The main drawback of IDES is that all the new nodes need to measure the distance to all landmarks, which limits the algorithm scalability. Besides, the SVD algorithm applied to the dot product of the incoming and outcoming matrices may produce a negative distance prediction, which is impossible to happen in the real world.

3.1.2.2 Decentralized MF

Decentralized matrix factorization (DMF) solved the IDES's scalability issues by allowing any nodes with calculated coordinates (matrices) acts as a landmark to any newly joined node [53]. Besides, DMF uses the NMF algorithm in decomposing the network delay matrix (D). NMF assumes that the network's measured distance has non-negative distances, which produces a non-negative distance prediction that reflects the real-world network. DMF suffers from prediction error propagation. DMF treats all landmarks equally regardless of their coordinates' accuracy. If a new node uses non-accurate nodes as its landmark, the resulting coordinates will be inaccurate and so on. This error propagation leads to a whole system performance degradation.

Phoenix is a weight-based MF system [54]. It assigns higher weights to accurate nodes and lower weights to inaccurate nodes. The weight-based method allows newly joined nodes to distinguish between nodes in accuracy and choose accurate ones to act as landmarks. Consequently, it stops error propagation throughout the system. However, it is reported by [55] that Phoenix suffers from short-link issues, i.e., Phoenix fails to sustain high performance between nodes with quite a small delay.

Pancake typically did the same what Pharos did with Vivaldi [55]. Pancake is a two-level hierarchical MF system. Pancake divides the network into two clusters, and each node within the network has a set of two coordinates: local and global coordinates. Nodes within the same cluster use their local coordinates in distance prediction. Clusterization solved the short-link issue that Phoenix suffered.

3.1.3 Summary

Despite the fact that NCS solved the network scalability issue, both of the NCS types still suffer from drawbacks. The Euclidean Distance-based Model did not consider that the network is asymmetric and the distance between any node pairs is different depending on the route the packets use [51]. On the other side, Matrix Factorization Models do not surpass the Euclidean Distance Models due to the ignores the geographical distances between hosts that introduce constant propagation delays. Besides, it is practically challenging to identify the exact rank of the latency matrix due to the inaccurate measurements. Finally, both types require an initial time to converge, which is proportional to the number of network nodes. This time can be long and is not trivial in practical situations.

3.2 Network Automation

The related works for network automation are in manifolds, network state classification and network action recommendation.

3.2.1 Network State Classification

Although many research studied ML-based traffic classification, which can be congested and non-congested, only very few have been carried out addressing ML-based network status classification, which could be normal, congested, or some other classes. In the literature, we were able to find only three research [56]–[58] with the ML-based network status classification. In [56], Decision Tree (DT), Support Vector Machine (SVM), and K-Nearest Neighbor (KNN) ML algorithms are applied on 2 datasets with time periods in the range of 5-60 seconds to classify software-defined wireless mesh-network status into two classes: congested and non-congested. DT, SVM, and KNN indicate 90% to 98% accuracy, KNN giving the top accuracy on

average – 98%. In [57], the authors used Naïve Bayesian Classifier to categorize the network status as faulty or non-faulty. In[58] Gupta et al. presented a hybrid framework of ML and DL to classify the network status for NFV systems. In addition, the system can decide if the network issue has already occurred or is predicted to happen with certain probability. The system first identifies the network status in terms of faulty or non-faulty. If the state is faulty, the system identifies the root cause, while for the predicted faults it localizes them and determines the severity. Their SVM method achieved 95.4% accuracy in the detection of the network status as faulty or normal, and 89.7% and 95.1% in classifying the faults as manifested or impending, respectively. Manifested problems are identified by multi-class classification with SVM. Localization of impending faults and prediction of their severity are done by stacked sparse autoencoder reaching 92% accuracy.

Two other works studied non-ML based network status classification. Park and Kim [59] aimed to classify network status via an analytical approach rather than an ML algorithm. In that work, RTT values are analyzed and the frequency of the outlier values are used as a distinguisher for the stability of the network status classes: stable strong, stable weak, stable transient, stable biased, and stable not. In [60], the SDN controller collected statistical information such as status of current topology connection, port status, type of packet counters, packet drop counters, link bandwidths, and latency measurements. Then such data is exploited to reach a diagnosis for the network status being correct or faulty.

In general, beforementioned works apply classification on the network status to differentiate the states as problematic or non-problematic. However, this binary separation hinders many information that might be represented in the problematic state since

it combines all of them into one class. On the other hand, we approach to network status classification problem in a similar yet different way than the above-mentioned works. Besides Congestion and Normal network status, we proposed a new class named Network Issue which could be the status of the network that causes interruption in the network such as router failures, link failures, security attacks. Whereas, Congestion state refers to the QoE degradation due to increased traffic flow and Normal state is when network does not face any issues.

3.2.2 Network Action Recommendation

In case of network failure, network operators choose a suitable action to restore the network to its optimum state. The potential actions are either fixing physical devices or traffic engineering (TE) i.e., rerouting, the affected customers to a non-faulty infrastructure [61]. There is not much work available demonstrating the way of fixing physical problems. However, several works are done in the field of TE. TE is very popular in programmable wired networks i.e., Software Defined Networking (SDN), as well as wireless communication systems.

In wireless networks, path-planning is essential when abnormal events are present [62] or when energy efficiency is crucial to meet resource restrictions [63]–[65]. These works study application-specific path decision which improves network conditions. However, application-level methods fail to generalize to other scenarios. On the other hand, network-level path-planning overcomes this generalization problem. In [61], the authors follow the latter approach and redefine the path planning problem as learning the sequence of nodes from the historical network traffic data. Also, they introduce an attention mechanism to enhance network performance.

In another work, Tang et al. [66]designed an algorithm that either improves the current routing strategy or creates a new routing strategy in wireless networks in an attempt to learn previous experiences from the network traffic data.

SDN is the paradigm that decouples the control and data plane while introducing a high-level controller that has global knowledge of the overall network. In [67], Azzouni et al. introduced a controller independent routing framework - NeuRoute. It optimizes the network flow via traffic prediction followed by traffic classification to learn the routing strategy. Other works in SDN either utilized a mathematical approach [68], a supervised learning approach [69], or supervised learning combined with reinforcement learning [70] to determine the optimum routing path in SDN. In [71], online unicast and multicast requests are analyzed when resource limitations pose demanding throughput constraints. To maximize the bandwidth requests of unicast and multicast admissions in SDN, both node costs and link resources are examined. Geyer and Carle [72] took graph representation of network topology as input and fed it to Graph-Query Neural Network to output the optimum path in the network by applying the shortest path and max-min routing approaches. In [73], Zhang et al. predict dropped packets' probability to understand the incoming traffic, network status, and link conditions. Then they apply an artificial neural network for resource allocation problems while preserving fairness among end-users.

Chapter 4. Delay Measurement Estimator System

4.1 Proposed Architecture Overview

In this section, we propose Delay Measurement Estimator (Deme, pronounced deem). Deme is an AI-based system trained with a dataset with hundreds of millions of previous explicit delay measurements, KING dataset, to create an estimation model that learns to estimate the delay between any node pair. Deme requires neither any explicit latency measurements during its operation nor any convergence time. It requires a one-time training, after which it can estimate, in a fragment of a second, the network delay between any pair of nodes. The source and the destination IP-addresses are the only two inputs to Deme. However, IP addresses alone are not trustworthy. The Internet Assigned Numbers Authority (IANA) divides the IP address space into IP blocks. Each block is assigned such that it represents a specific region or country. Nevertheless, multi-branch companies assign their branches to spread all over the globe with IP addresses within the same IP block, which contradicts the idea of the IP-address block and region relation. As a result, Deme uses the source and destination nodes' IP addresses to extract more accurate and reliable features such as nodes' geolocation and the ASes. Then, the extracted features are pre-processed before training the deep learning model. The overall architecture of Deme is shown in Figure 4-1.

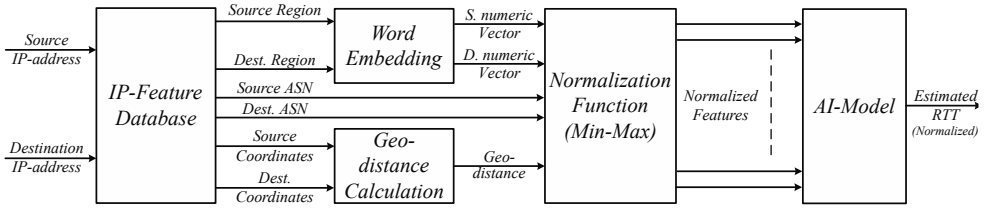


Figure 4-1 A high-level view of the proposed system.

Deme has two phases of development; choosing the suitable AI architecture and comparing it against the state-of-the-art algorithms. On one hand, the Ubisoft dataset is used to perform a quick comparison between various AI algorithms such as linear regression, support vector machine, and deep learning. On the other hand, the KING dataset is used for comparing the best-performed AI model against the state-of-the-art algorithms.

4.2 Datasets and Data preparation

4.2.1 Ubisoft Dataset

iConnect-Ubisoft is used for training different AI models. The Ubique dataset is provided by Ubique Networks Inc., a company specialized in lagless network solutions for online multiplayer games, while iConnect-Ubisoft is ”a project which has the goal of mapping the topology of the Internet as well as to measure, as accurately as possible, the latency between any two endpoints of the Internet where users could be via crowdsourcing” [74].

Figure 4-2 depicts the cumulative distribution for Ubisoft's RTT. The average measured latency of the Ubisoft dataset is 0.2 seconds. Figure 4-3 shows the probability mass function of the measured latencies in the Ubisoft dataset. The distribution does not have a single concentration. Moreover, the distribution has a long tail that adds quite a challenge to any AI learning algorithm since AI models consider such data as outliers and data scientists tend to clip such tails. However, the tail represents delay measurements between two far nodes in the network and cannot be ignored. As such, it is important that in the design of our system, the AI subsystem does

not cut the tail of the data, despite the usual practice of doing so in other AI-based systems.

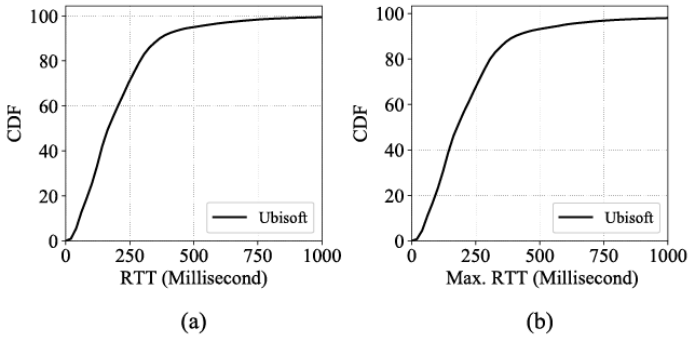


Figure 4-2 RTT distribution in the Ubisoft dataset. a) CDF of average RTTs, and b) CDF of maximum RTTs, for each pair of nodes.

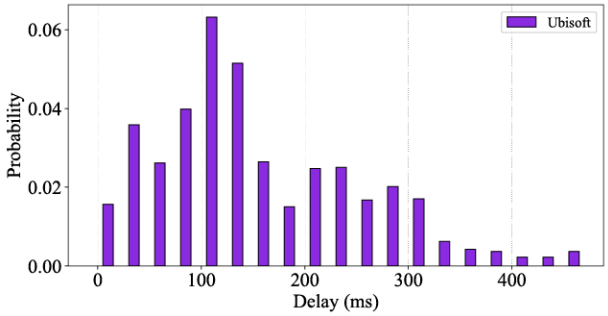


Figure 4-3 Probability mass function of measured latencies in the dataset.

4.2.2 KING Dataset

KING dataset is built using the king method described in [75] to collect the full matrix of RTTs. It has 1740 Internet DNS servers. The distance between any DNS servers is done in two steps. A host node measures the RTT to the first server. Then the first server is asked to resolve the DNS domain served by the other server. The difference between the two measurements is the actual distance between the two servers. Pairwise RTTs were continuously measured at random intervals over a time interval of a week to avoid congestion and to ensure link stabilization. The collected dataset size is around 100 million pairwise RTTs in total.

It was found that some of the measurements in KING are over 3 seconds. These measurements are the result of unreachable nodes at the instant the measurement was taken. Nevertheless, the same nodes were reachable later with latency measurements that were very small compared to the 3 seconds. As a result, we considered the 3-second measurements to be outliers, and around 10% of the dataset was eliminated to get rid of these data outliers. The rest of the dataset is under 1 second. Moreover, the average measured latencies are 0.28 seconds. Figure 4-4 depicts the cumulative distribution function (CDF) of the measured latencies.

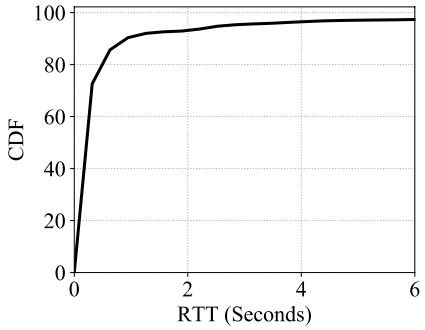


Figure 4-4 RTT distribution in the KING datasets for each pair nodes.

Figure 4-5 the probability mass function of the measured latencies of KING. The distribution does not have a single density. Besides, even after removing the outliers, it has a long tail that is challenging for DL algorithms since they consider such tails as additional data outliers that either need to be clipped or removed. However, in KING's case, the tail measurements represent actual delay measurements between different continents, i.e. two remote nodes in the network. These measurements are therefore valid and can neither be clipped nor removed.

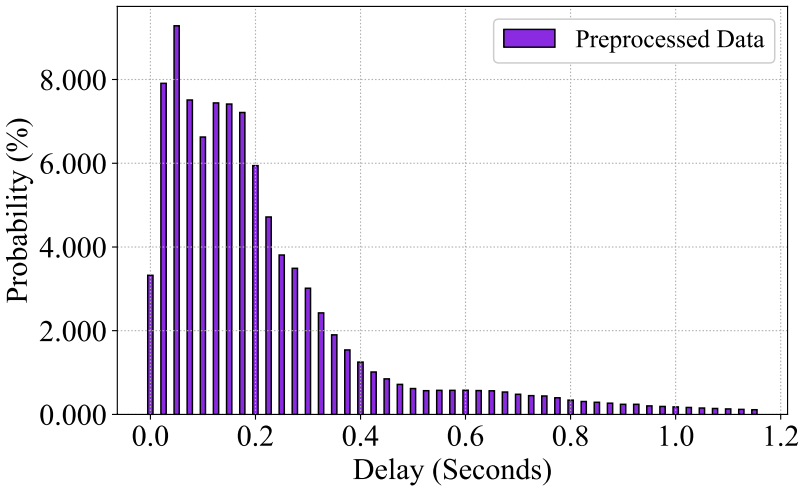


Figure 4-5 Probability mass function of measured latencies in the KING dataset.

4.2.3 Ubisoft Vs KING

Both datasets have comparable statistics. However, Ubisoft is relatively smaller than the KING dataset in size. Moreover, the Ubisoft dataset is collected with a concentration on North America. On the other hand, the KING dataset is balanced towards the different continents and regions. As a result, we used the Ubisoft dataset to train and compare different AI models. Then, the model with the highest performance is set to be our base regressor which is again trained on the KING dataset and compared against the state-of-the-art algorithms.

4.2.4 Data Analysis

Before training any machine learning model, the last step is to apply some data pre-processing techniques to enhance the system's performance and obtain the best possible delay estimation. Afterwards, both Ubisoft and KING dataset was split into three parts and used for training (60%), validation (10%), and testing (30%).

IP-to-features: As mentioned earlier, the IP address alone is not reliable to build an accurate predictor. As a result, more reliable features such as geographical location, autonomous system, DNS,

VPN, continent name, and country name are extracted from nodes' IP addresses. Such features are offered by many free-of-charge providers such as KeyCDN [75].

Word Embedding: Some of the given features are non-numeric and they cannot be fed to the DL model since they will not be understood. Word embedding is a technique that encodes non-numeric features into unique vectors such that each word keeps its correlation to other words [6]. For example, the word "US", in the country feature, is more linked to "CA" than to "UK" since both "US" and "CA" are in North America while "UK" is in Europe. It is necessary to maintain this correlation. As a result, we constructed three vectors for the three countries such that the distance between the "US" vector and the "CA" vector is shorter than the distance to the "UK" vector. In this way, we transform non-numerical features into numeric ones without losing their correlations.

Although one-hot encoding can do the same job as word embedding. We preferred using word embedding, word2vec, in order not to be limited to the dictionary of words provided during training. We utilized a pre-trained word2vec Google's word2vec model which contains 3 million words and of course contains most, if not all, countries and cities of the world [77].

Synthetic Featuring: Synthetic feature is the outcome of the mathematical operation(s) performed on one or more raw features [77]. The synthetic feature is a higher-level representation of the raw dataset because it concentrates all the information of the raw feature(s) used to produce it [78]. Producing such features via a complicated formula aids DL models to easily correlate these features with the output. For instance, instead of feeding directly redundant features such as the nodes' longitudes and latitudes to the

model, it is better to construct a new synthetic feature, "distance", which is based on the "Haversine" formula [79] presented below.

$$d = 2r \,.\arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_1 - \varphi_2}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin\ \hat{}2\left(\frac{\lambda_1 - \lambda_2}{2}\right)}\right)$$

4.3 Proposed AI Architectures

4.3.1 Different AI model

As mentioned earlier, the Ubisoft dataset is used to train and test different AI models. We focus only on comparing *Linear Regression (LR)*, *Support Vector Machine (SVM)*, fully connected *MLP (DNN)*, and *Convolutional Neural Networks (CNN)* models.

Figure 4-6 shows the cumulative distribution of relative errors of all the aforementioned models. From the figure, the CNN model outperforms all the other algorithms. CNN's ninetieth percentile is around 0.55. While the closest is around 0.8. Thus, we utilized CNN model as our base regressor in the next development, *a multi-model deep learning network*.

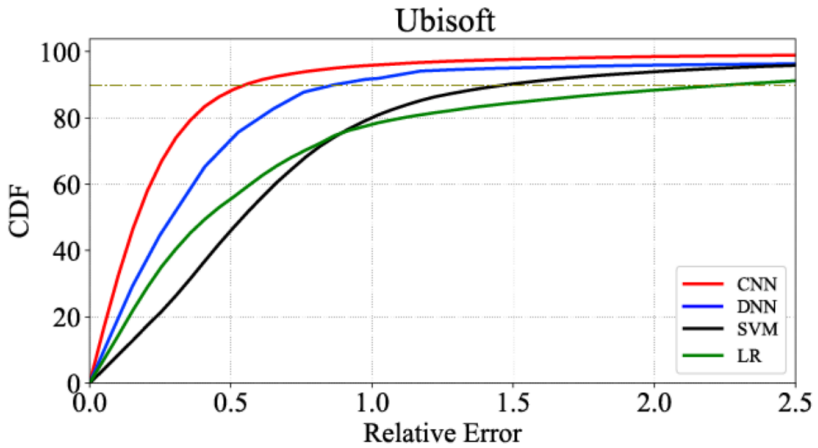


Figure 4-6 Cumulative distribution function (CDF) of relative errors (RE).

4.3.2 Multimodal Deep Learning Networks

Deme uses Multimodal Deep Learning Networks (MDN). MDN combines two CNN deep learning models to achieve the best possible

performance [80]. The KING dataset is split into two subsets according to nodes' location. The first subset is where all the measurements between any node pairs lie in the same continent. This subset is called the "Local" dataset. While the other subset contains all the measurements between node pairs belonging to a different continent and is named "Continental". Each of these subsets is used to train a vanilla CNN model followed by fully connected layers independently. Each of these models learns according to the slice trained with, as shown in Figure 4-7. Afterwards, several layers are removed, by trial and error to get the best result, from the trained models. Then, the trained models are treated as constants, frozen, and their outputs are concatenated. Finally, the concatenated outputs are followed by fully connected layers to generate the output. The two CNN models' specifications are presented in Table 4-1 and Table 4-2.

It is worth mentioning that further division to the "Local" subset into different extra subsets representing the five continents, but the predictor suffered from performance degradation due to the fact that the KING dataset is an imbalanced dataset. In other words, most of the data points collected are from nodes belonging to the Americas, Europe, or Asia and a very limited number of data points from Africa and Oceania. This limitation can lead to further improvement in Deme's performance in the very near future if a more diverse dataset is available.

Table 4-1 Local CNN models specifications.

Layer Name	Specification
Conv1D_1	Filter = 32, Kernel = 4, Strides = 2
Conv1D_2	Filter = 32, Kernel = 4, Strides = 1
MaxPooling	Chooses the highest number within window.
Flatten	Changes data shape from 2D to 1D.
Hidden layer 1	Neuron = 512, Relu.

Hidden layer 2	Neuron = 64, Relu.
Hidden layer 3	Neuron = 32, Relu.
Hidden layer 4	Neuron = 32, Relu.
Hidden layer 5	Neuron = 16, Relu.
Hidden layer 6	Neuron = 5, Relu.
Hidden layer 7	Neuron = 2, Relu.
Output	Neuron = 1

Table 4-2 Continental CNN models specifications.

Layer Name	**Specification**
Conv1D 1	Filter = 64 Kernel = 4, Strides = 2
MaxPooling	Chooses the highest number within window.
Conv1D 2	Filter = 32, Kernel = 4, Strides = 1
MaxPooling	Chooses the highest number within window.
Flatten	Changes data shape from 2D to 1D.
Hidden layer 1	Neuron = 64, Relu.
Hidden layer 2	Neuron = 32, Relu.
Hidden layer 3	Neuron = 32, Relu.
Hidden layer 4	Neuron = 8, Relu.
Hidden layer 5	Neuron = 4, Relu.
Hidden layer 6	Neuron = 2, Relu.
Output Layer	Neuron = 1, Relu.

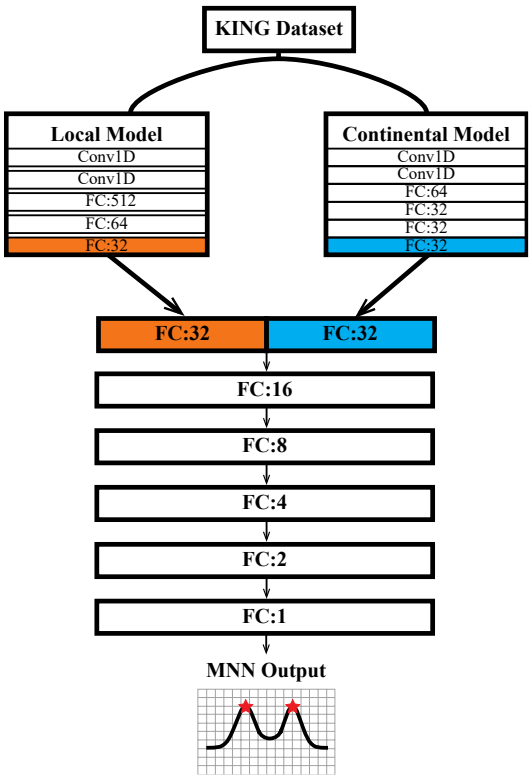


Figure 4-7 KING Dataset is split into two sets and fed into two different CNN models.

4.4 Proposed Architecture Limitations

Deme suffers from three main drawbacks: IP spoofing, IP-to-feature extraction overhead, and being quickly outdated.

Location spoofing is when the end-user establishes a Virtual Private Network (VPN) connection. VPN hides the user's original IP address and spoofs it with another one which in most cases belongs to a different location or region. In this essence, Deme uses the spoofed IP address, which results in the wrong location and ASN extractions. As a consequence, this leads to inaccurate measurements. It is worth mentioning that the state-of-art methods also suffer from the same problem but a different perspective. The node with VPN coordinates will depend on the summation of the delay between the original node, its VPN instance, and the reference node, this leads to inaccurate coordinates calculations. Moreover, if the user decided to disconnect the VPN connection. The node's coordinates will be outdated and require an essential refresh which may lead to an oscillation.

As mentioned earlier, the IP-to-features are extracted via KeyCDN API. This process may introduce a network overhead comparable to performing explicit delay measurements. However, the KeyCDN method is considered to be temporary mothed and it should be used till each platform builds its own IP-to-feature database.

Deme is as other AI-based systems are trained on offline datasets, with specific data characteristics. Moving such systems to production usually fails due to the difference between the data in which these systems were trained and the incoming data, i.e., data drift. The reason for this drift is due to the fast-changing nature of the computer network and internet. As a result, the AI systems do not keep their peak performance and degrades over time. The potential

solution for this issue is to surround the AI system with a control loop to continuously measure its performance and execute a proper action to maintain these systems always in top performance. This solution is known as Machine Learning Operations (MLOps) and is explained in the following Section.

Chapter 5. Machine Learning Operation for Deme

5.1 Deme-MLOps Architecture Overview

As mentioned earlier, Deme as well as any machine learning algorithms are trained on offline large datasets to achieve acceptable accuracy. However, once the offline-trained machine learning models are shipped to production, they start getting outdated easily due to the varying nature of real-world problems. In other words, depending on the nature of the application, the dataset used for training the machine learning model may be totally or partially different from the incoming one in terms of distribution and characteristics. To tackle such a problem, the machine learning model is not only treated as a data science development problem but also a software program and is subjected to software development rules, i.e., DevOps. This integration leads to Machine learning Operation (MLOps). The objective of MLOps is to provide all DevOps advantages such as Continuous Integration (CI) and Continuous Delivery (CD). In addition to the ability to fine-tune, transfer learning, or retrain the machine learning model on the incoming data, in case of any performance degradation. This is known as Continuous Training (CT) [37]. Figure 5-1 shows the MLOps architecture. Once the machine learning model is trained and passed the evaluation phase, i.e., reached performance satisfaction, the machine learning model is set to be shipped to production. The model is treated as regular software and undergoes serval tests such as unit testing and integration testing. Then, the machine learning model is ready to be deployed and serve inference. To guarantee CD, the model performance is continuously monitored and compared to a baseline reference, i.e., threshold, and triggers fine-tuning or transfer-learning wherein unsatisfiable performance is detected.

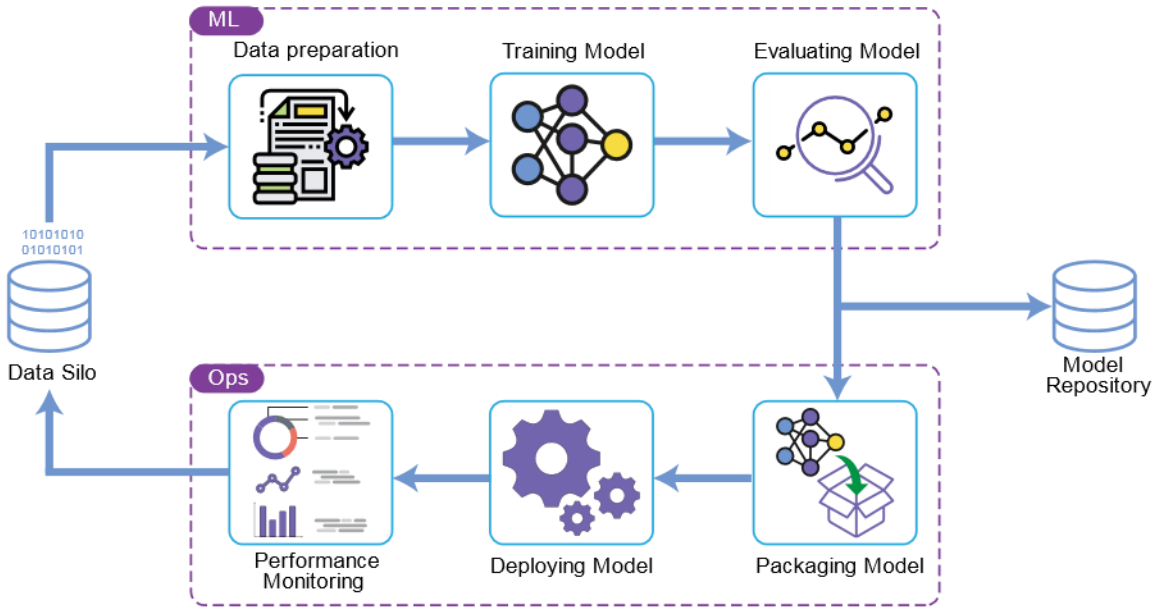


Figure 5-1 MLOps Architecture.

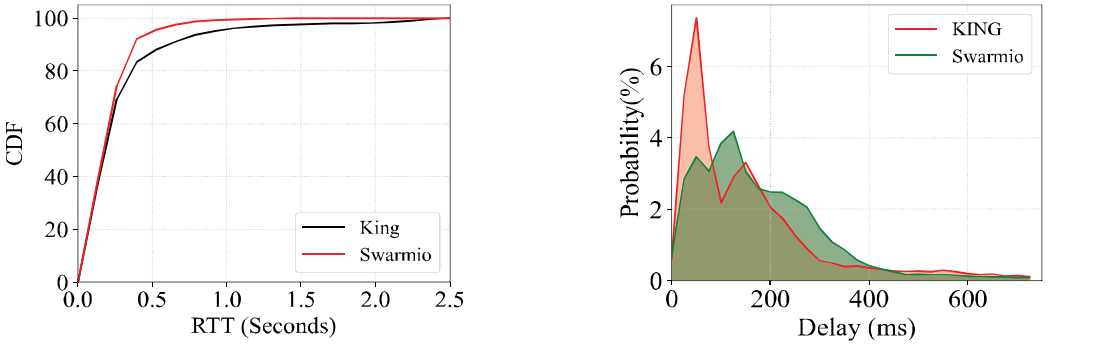
5.2 Testing Deme on Swarmio's Dataset

We had the opportunity to access Swarmio Media's infrastructure and collect live data to test Deme's performance after several months of its training. The collected dataset is around 94, 000 data rows in terms of end-to-end RTT, source, and destination IP addresses. The collected data is fed to the preprocessing pipeline for extra-features extraction and normalization. We noticed an immense performance degradation (–22%) while testing with the newly collected dataset. After analyzing and troubleshooting the system, we found that the newly collected data slightly differs from the dataset used in training the Deme.

5.2.1 Datasets: KING VS Swarmio

In this section, we compare the statistical characteristics for both KING and Swarmio's lively-collected datasets. As mentioned earlier, the KING dataset is used for training the Deme and consists of 100 million pairwise RTTs in total. Swarmio's dataset is collected for

model validation via Swarmio infrastructure and datacenters. Swarmio's dataset is collected from centric ping commands between Swarmio's datacenters and Swarmio's clients. The total size of Swarmio's live dataset is around 94 thousand pairwise RTTs in total. Figure 5-2a shows the CDF of latencies measured for both datasets, where KING measurements are slightly greater than those measured by Swarmio. Moreover, the average measured latencies are 0.28 and 0.21 for the KING and Swarmio datasets, respectively. Figure 5-2b shows the probability mass function of the measured latencies in both datasets. The two datasets do not show any single concentration. Furthermore, the peak measurements of Swarmio's dataset is slightly shifted by 75ms compared to KING. This shift is considered to be normal due to the varying nature of the Internet. However, this variation may lead to a gradual performance deterioration for any offline supervised model.



(a) RTT distribution for each pair nodes (b) Probability mass function of measured latencies.

Figure 5-2 KING and Swarmio's Dataset Analysis.

5.2.2 Continuous Development and Training

Comparing Deme's current performance against the baseline threshold (90%), It is obvious that Deme requires fine-tuning to restore its high performance. To do so, we studied freezing Deme's all layers except for the last four. Also, instead of randomly initializing the last four layers, we load the outdated weights and

start training. In other words, loading the outdated weights enable the model to get updated, and the new data is treated as batch training. Moreover, we studied the effect of discarding Deme's past training and train a model from scratch. The results are presented in Section 7.1.2.

Chapter 6. Network Automation System

ISPs pay lots of attention to failure management and traffic to balance their network utilization overhead, providing high QoS to meet their customers' Service Level Agreements (SLA), and minimizing ISPs' operational costs (OPEX). ISPs hire network experts to quickly intervene and fix any network failure, especially for today's diversified networks in terms of services and advanced networking. In this section, we intend to introduce a system that automates ISPs' network operations center (NOC), as shown in Figure 6-1. Raw Data (e.g. SNMP, ping, traceroute, etc.) is fed into network Analytics software to give an insight for NOC team about the states of network's devices. SolarWinds monitoring tools [81], [82], Network Analytics modules include Ciena's Unified Assurance and Analytics (UAA) [22], and Route Optimization and Analysis (ROA) [23] are well-known examples of such Network Analytics tools used in monitoring today's network. If any abnormalities are detected, the NOC team opens a ticket with the issue, studies the potential actions, and executes the optimum one. The last two operations of exploring the possible solutions and the best action's execution usually are not easy tasks and may take a long-time. As a result, end-users' QoE degrades.

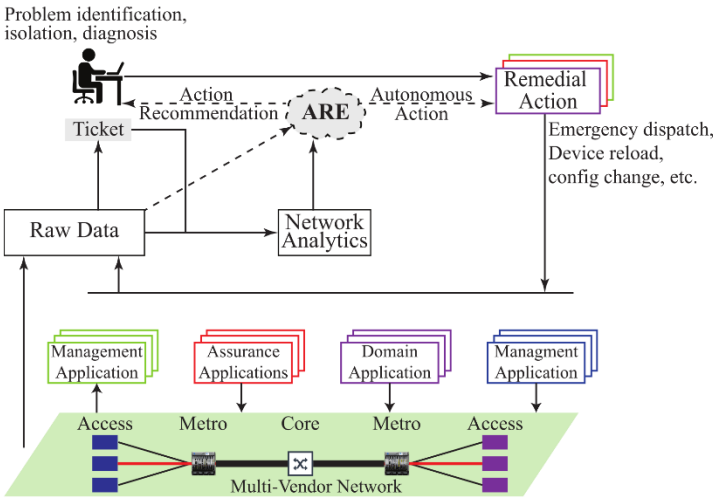


Figure 6-1 NOC typical operation.

We present a constantly running closed-loop ML-based network status prediction and action recommendation system, shown in Figure 6-2. The system has the ability to differentiate between three network states. The first state is the "Normal state," a problem-free state, where all end-users are satisfied, i.e., maximum QoE is maintained. While the second state is the "Congestion state," where one or more of the network paths are congested, and its end-users suffer from QoE degradation. The last network state is the "Network Physical Issue" state "NPI". This state resembles any physical issue that may occur due to a network problem such as router failures, link disconnections, or BGP anomalies, which lead to a drastic decrease in QoE.

The proposed system first predicts network status from QoS and QoE measurements. Based on the predicted network state, the system recommends a suitable action to maximize clients' QoE and minimize ISP's OPEX. However, some states may have more than one valid action. For example, a disconnected link can either be fixed or end-users utilizing this link can be rerouted to other paths. Looking at ISP's philosophy of minimizing their OPEX and maintaining a high customer satisfaction level, the cheapest action that guarantees end-

users satisfaction will be chosen. Therefore, we defined the cost of each possible action in terms of dollars.

We searched for any public network dataset with details like SLA measurements, QoE levels, and NOC actions to train our ML models, but we did not find any. Consequently, we built our testbed to collect the required data needed to train our model. Our testbed is chosen to be simple enough to extract good rules from common sense, yet complex enough to not determine optimal rules trivially. In the following subsections, we will discuss our testbed setup, the hypothetical scenario utilized to collect data for training and testing the ML models as well as the methods we used in data collection and analysis applied to the collected data.

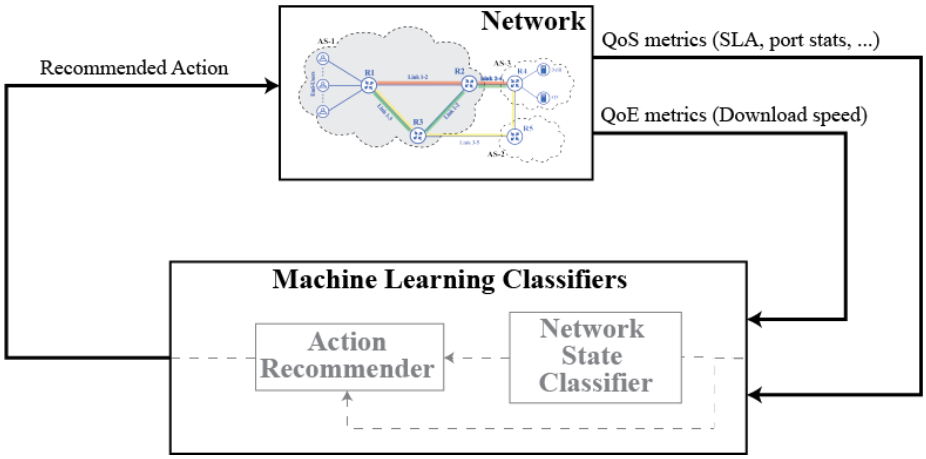


Figure 6-2 A High-level overview of the proposed architecture.

6.1 Testbed Setup

We employ the GNS-3 to emulate the network shown in Figure 6-3. GNS-3 is an open-source software used by telecommunication giants like AT&T, Google, and NASA to emulate their systems [83]. This network is chosen to be similar to the Internet's backbone network, where it has multi-paths between end-users and servers. The chosen topology consists of three Autonomous systems (AS), AS-1 is the network we are interested in predicting its state and recommend

actions to keep it in the best conditions. AS-2 resembles a neighbour ISP, where forwarding packets are considered more expensive compared to routing the packets internally, i.e., inside AS-1. AS-3 contains services such as FTP servers.

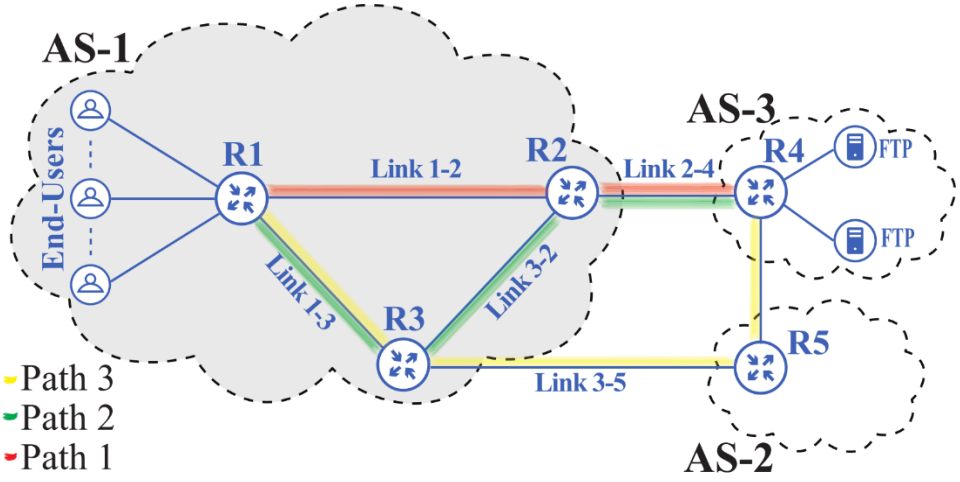


Figure 6-3 Network topology consisting of 3 ASes.

To guarantee network connectivity, we used the OSPF routing protocol among the routers. In addition, we employed the MPLS routing technique and Cisco's Access-Lists (ACLs) to create three main paths between end-users and streaming servers. ACLs are considered the primary key for traffic engineering since it allows ISPs to force their end-users' traffic to adopt specific routes.

We model end-users as containers running a headless Ubuntu 18.04 OS image since containers are lightweight. The content servers are built on the same machine that operates the testbed to eradicate any unknown and external network conditions. Moreover, we use Cisco's IOSv images to model routers and switches. However, the IOSv images have bandwidth limitations since their primary usage is to test new network algorithms and network configurations. None of these applications require high bandwidth to operate. As a result, the maximum bandwidth of IOSv images can support up to 2Mb/sec. In

our case, bandwidth is essential since end-users are either streaming or file transferring relatively big data.

To workaround IOSv's bandwidth limitation, we limit the download speed for each of the nine end-users to 20 KB/sec. Also, we defined the Normal state as each path having a maximum of three end-users streaming simultaneously, and any additional users above three are considered as path congestion. To do so, we applied layer-3 policies to limit MPLS tunnels' bandwidth. All policies are in Table 6-1. We use Wireshark to ensure the effectiveness of the applied policies versus the number of end-users per path [84], as shown in Figure 6-4. In the figure, there are two types of links: "unique" and "shared" links. Unique links are a member of only one path. Therefore, the unique link's maximum bandwidth is 60 KB/sec, i.e., three end-users' bandwidth. In contrast, shared links aggregate two or more paths, and their maximum bandwidth is double the unique one.

Layer-3 ACLs help in defining current network states. The normal state is when any link has a maximum of three end-users. A congestion state is when there are more than three end-users connected to a path. Additionally, we defined network issues such as any network device failure or path disconnectivity by a composition of delay and jitter ranging between 80-350 ms and 5-60 ms, respectively.

Table 6-1 A summary of Layer-3 policies.

Link	Type	Max BW	Member	Policy
Link 1-2	Unique	60	Path 1	R2: G0/0 peak 60
Link 2-4	Shared	120	Path 1&2	R4: G0/1 peak 120
Link 1-3	Shared	120	Path 2&3	R3: G0/0 peak 120
Link 2-3	Unique	60	Path 2	R2: G0/1 peak 60
Link 3-5	Unique	60	Path 3	R5:G0/0 peak 60

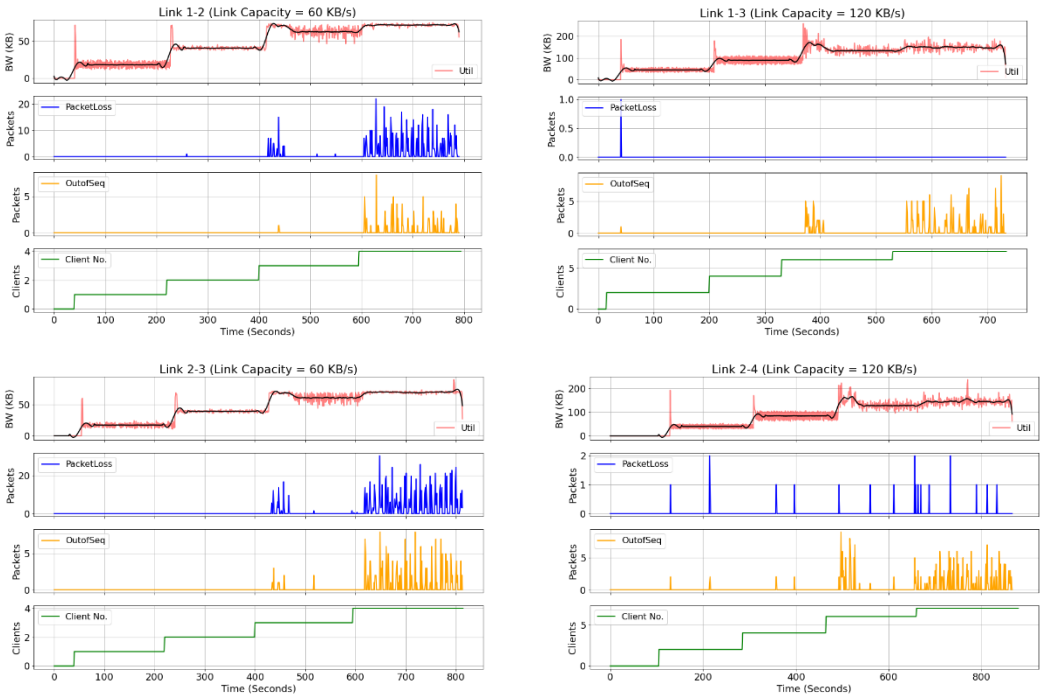


Figure 6-4 MPLS links' utilization.

6.2 Scenario

The scenario adopted matches exactly ISPs' philosophy described above. In the network, AS-1 is the network of interest in which we try to predict its state and recommend actions in case of any abnormalities. AS-1 consists of three routers and contains nine end-users, file transferring big files from servers available on AS-3. AS-2 represents an available infrastructure that can be used by AS-1 but with an additional charge. We assume that AS-1 is the only AS that generates traffic in the network, and there is no incoming traffic to AS-1.

Nine end-users are file transferring big data via FTP protocol through locally built servers. Each end-user has three paths to the content servers. PATH_1 and PATH_2 are internal routes within AS-1, and their costs depend on the number of hops the traffic passes through. In contrast, PATH_3 is an external path that forwards the

traffic to the AS-2 and then to the content services. To match the real situation of ISP and in order to distinguish between possible paths and action, we implement a billing system that sets the cost of each action and path. The billing system favors internal routes and charges them less compared to external ones. For that reason, PATH_1 is considered to be the cheapest available path in terms of cost given that it is an internal route and consists of only two hops; i.e., OPEX=2(IGP hops) × -1 = -2. On the contrary, PATH_3 is the most expensive path whereas AS-2 charges for allowing AS-1 traffic to use its infrastructure; i.e., OPEX = 2(IGP hops) × -1 + -5 {External AS} = -7.

PATH 2 is more expensive than PATH 1 though cheaper than PATH 3 given that it is an internal route but with more hops than PATH 1; i.e., OPEX = 3(IGP hops) × -1 = -3. Moreover, the billing system takes into consideration that fixing network issues needs money and time, so OPEX = -5. Also, the system either awards or charges the AS based on the end-users' QoE per path. All costs and benefits are summarized in Table 6-2.

Table 6-2 Cost and Benefits

Action	Cost ($)	QoE	Cost ($)
AS-1 Each hop	- 1	Bad	-5
Fix Network device	-5	Medium	0
Rerouting	-2	High	5
External AS	-5		

To summarize our possible state spaces, i.e., network state classifier labels: Network issue states due to a physical issue *NPI:1→3*, *NPI:3→2*, *NPI:2→1*, *NPI:2→4*, *NPI:3→5*, congestion states *Cong:P1, Cong:P2*, and *Normal* state.

As for action space, i.e., action recommender estimator labels: reroute due to congestion *TE-P1→P2, TE-P1→P3, TE-P2→P3*, reroute to optimize cost *TE-P2→P1, TE-P3→P1, TE-P3→P2*, and

network issue *Fix LR-1-3, Fix LR-2-4, Fix LR-2-1, Fix LR-3-2, Fix LR-3-5*. Table 6-3 summarizes all the ARE's labels.

Table 6-3 Summary for labels utilized by ARE ML-models.

State Classifier Labels	
Label	**Description**
NPI:1→3	The physical issue is present between routers 1 and 3.
NPI:2→1	The physical issue is present between routers 2 and 1.
NPI:2→4	The physical issue is present between routers 2 and 4.
NPI:3→2	The physical issue is present between routers 3 and 2.
NPI:3→5	The physical issue is present between routers 3 and 5.
Cong-P1	Congestion is detected at Path-1.
Cong-P2	Congestion is detected at Path-2.
Action Recommender Estimator Labels	
Label	**Description**
TE-P1→P2	Traffic Engineer (TE): Move traffic from Path-1 to Path-2 due to congestion.
TE-P1→P3	Traffic Engineer (TE): Move traffic from Path-1 to Path-3 due to congestion.
TE-P2→P3	Traffic Engineer (TE): Move traffic from Path-2 to Path-3 due to congestion.
Fix LR-1-3	Fix the physical issue between router 1 and router 3.
Fix LR-2-4	Fix the physical issue between router 2 and router 4.
Fix LR-2-1	Fix the physical issue between router 2 and router 1.
Fix LR-3-2	Fix the physical issue between router 3 and router 2.
Fix LR-3-5	Fix the physical issue between router 3 and router 5.

6.3 Data Collection

A python-based script is developed to interact with GNS3's API. The API offers the options to activate or deactivate any of the network devices. Also, the API can apply a combination of delay and jitter on any of the network links. The python script describes various network scenarios utilized in data collections, presented in Algorithm 6-1. The scenarios are written carefully to mimic the NOC operations of detecting and recommend possible actions to restore the network to its normal state. Because our testbed network is not complex, it was relatively easy for us to define these rules. But, in a more realistic network, these rules become more complicated to define. The NOC Data collection parameters and facts are presented in Table 6-4.

Algorithm 6-1 Operational rules mimicking NOC actions for ARE.

```
Input:  Sleep Units (U), client_number, physical_devices_space
Output: Network_state, action

While True:
        choice = random(add_client, physical_prob);

        # Network Status Definition
        if choice == physical_prob:
           Network_state = NPI_x; # "x" resembles the link/router
        else:
           if client_len(PATH1>3):
              Network_state = Cong:P1;
           elsif client_len(PATH2>3):
              Network_state = Cong:P2;
           else:
              Network_state = Normal;

        sleep(5U); # sleep 5 units of time.

        # Action Definition
        if Network_state == NPI_x:
           action = Fix physical_issue;
        elsif Network_state == congestion:
           if internal_path_avail:
              action = reroute_intnally;
           else:
              action = reroute_externally;
        else:
           action = do_nothing;

        # Collect and align QoS & QoE metrics, then dump to CSV
        collectmetrics();
        append_csv();
```

Table 6-4 A summary of data collection parameters and facts.

Data Collection Frequency	30 Seconds
Number of features	66
Number of Datapoints	56,000

The collection started by randomly assigning each end-user to a path and we introduced a problem by randomly choosing between disconnecting a link or intentionally having congestion on a specific link. We collect two clusters of data every 30 seconds: QoS and QoE metrics. QoS metrics include (i) SLA measurements of the three paths, which are end-to-end delay, jitter, and packet loss and (ii)

dropped packets, ingress rates, and egress rates values of the router ports within AS-1. File-transferring speed is the only QoE metric we collect. The collected QoS and QoE metrics are accurately aligned with their corresponding timestamps. Table 6-5 summarizes the features collected. In the end, our dataset consisted of 56,000 data points, which are used to train the ML algorithms for both network state classification and action recommendation tasks.

It is worth mentioning that any transient problem that can appear and disappear between the sampling time period, 30 seconds, are ignored and ARE cannot sense it. As a result, network maintainers need to tune the sampling time according to their network nature.

Table 6-5 Collected features summary.

QoS Features (57 features):
▪ Per-Path-QoS (30 features; 3 Paths × 10 features):
  ▪ Delay: End-to-end Path delay (min/avg/max).
  ▪ Jitter: End-to-end Path jitter (min/avg/max).
  ▪ Packet loss: End-to-end Path packet loss (min/avg/max).
  ▪ Out of Sequence: In-ordered packets.
  ▪ Port statistics loss: average packet loss of AS-1's routers ports.
▪ Port statistics (27 features; 3 routers × 3 Ports × 3 Features):
  ▪ Drops: Packet Drop count for ports 0, 1, and 2.
  ▪ Input Rate: Ingress packet rate for ports 0, 1, and 2.
  ▪ Output Rate: Egress packet rate for ports 0, 1, and 2.

QoE Features (9 features; 3 Paths × 3 features):   ▪ Download speed: FTP transfer speed (min/avg/max).

6.4 Data Preparation and ML Model Selection

The collected dataset includes 66 unique features. The pie charts in Figure 6-5 show the class distribution of network status in the form of normal and abnormal classes and the corresponding action distribution. The breakdown of the abnormal classes and their recommended actions are shown in Figure 6-6. As can be seen, the Normal state class dominates the dataset. This is due to the data collection strategy, which mandates the network to be restored to the

Normal state before transitioning to other states. As a result, we face imbalanced classes; one class has more instances than others. This is an intended outcome since, in real-life situations, the network state of Normal is actually the condition of most of the time.

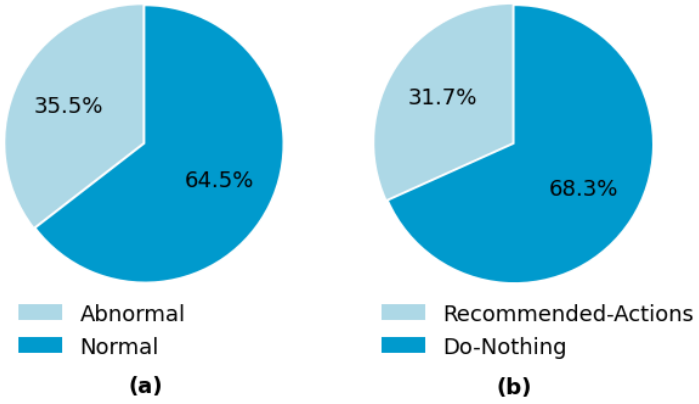


Figure 6-5 Space distribution for (a) state (b) action.

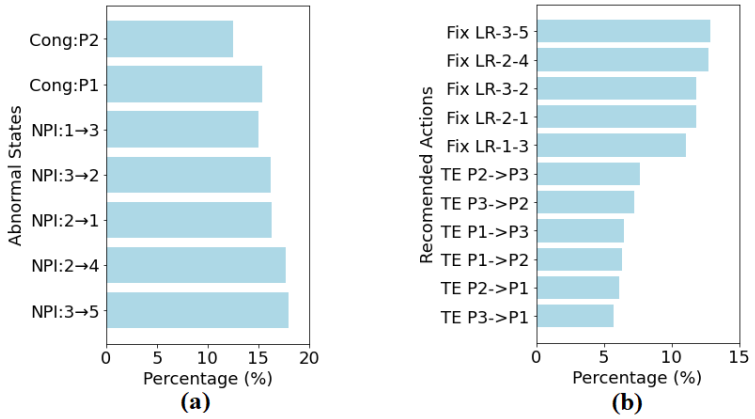


Figure 6-6 Minority class breakdown for (a) state (b) action.

However, imbalanced datasets are problematic by their nature. When dealing with such datasets, the ML models might end up facing two different situations. The first one is that the ML model might focus on classifying the majority class correctly at the expense of misclassifying the minority classes in an attempt to minimize false positives. The second one is that the minority class could not have enough data points to be representative enough. These consequences

are tricky to deal with since oftentimes minority class is the center of attention and our research is not an exception to this phenomenon. Fortunately, ensemble learning methods are powerful supervised learning techniques that improve classifiers' performance, enhance their generalization ability, and reduce the bias in classification tasks by combining several classifiers into a stronger one. When a new data point is introduced, each learner votes for the classes. Then the point is assigned to the class with the most votes [85]. These characteristics make ensemble methods an invaluable tool for challenging tasks such as the classification of imbalanced datasets [86], [87].

Hence, to mitigate the challenge of imbalanced class distributions, we take advantage of ensemble methods and implement two algorithms: Gradient Boosting and XGBoost algorithm. In addition, we also try the Decision Tree. Moreover, we intentionally biased the three chosen ML models by introducing class weights towards the rare classes, to slightly balance them against the majority class.

6.5 Action Recommendation Engine

Action Recommender Engine (ARE) is divided into two main processes; network state classification and action recommendation estimator, as shown in Figure 6-2.

6.5.1 Network State Classifier

The state classifier is fed by the 66 features at every time step to generate a prediction for the current network status along with the probabilities of each state. However, during training the state classifier, we found out that the waiting period of 30 seconds was not enough for some states, especially congestion states, to be clearly seen.

To mitigate this problem, we aggregated the dataset by a factor of two. The aggregation is done with regard to the nature of the feature. For instance, for RTT, we collect RTT_{min}, RTT_{avg}, and RTT_{max}. For $RTT_{min,}$ we select the minimum of the previous measurement:

$$RTT_{min_{agg}}[T] = min(RTT_{min}[T-1], RTT_{min}[T]$$

Similarly, for RTT_{max} the aggregation becomes:

$$RTT_{max_{agg}}[T] = max(RTT_{max}[T-1], RTT_{max}[T]$$

Likewise, for RTT_{avg} the aggregation becomes:

$$RTT_{avg_{agg}}[T] = avg(RTT_{avg}[T-1], RTT_{avg}[T]$$

For action, the dominant action is chosen to be the label for the aggregation vector. In other words, if the two actions are Do Nothing and any other action; e.g., reroute, or fix, we chose the other action to be our label.

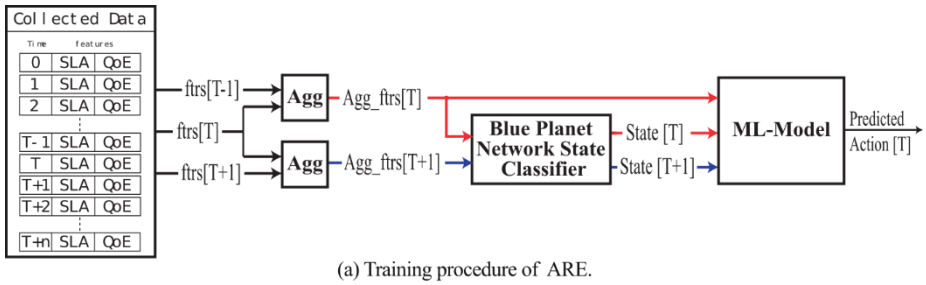
6.5.2 Action Recommender Estimator

The objective of the action recommender engine is to recommend an action to restore network status from abnormal to normal. Although our interest lies only in the actions that restore the network to its normal state. We generally trained the estimator to diagnose the root cause action that leads to a change in network status from $state_A$ to $State_{B.}$ This way, the action recommender estimator can also be used as an action simulator to help the NOC team understands the impact of every action on the network.

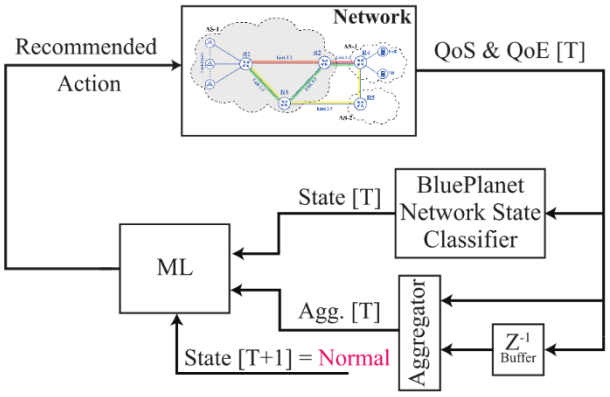
During training, we fed the ML models with the aggregated features at the time [T] together with the network state probabilities at the time [T] and [T+1] slot resulting from the network state classifier. The label is the action at the time [T-1]. Network State probabilities is a list of probabilities for all available states; it is very similar to something like this: {*normal*: 90%, *congestion* 3%,

network_physical_problem: 7%}. The training procedure is illustrated in Figure 6-7a.

After training, we integrate the ARE with GNS-3 to form a live closed-loop system and configured it to autonomously execute its predicted actions on the network, shown in Figure 6-7b. Every time [T], SLA and QoE measurements are collected and fed to the network state classifier to generate a state probabilities vector. Furthermore, SLA and QoE measurements are aggregated with the measurements generated at the time [T-1], represented by Z^{-1} in the figure. After that, the state probabilities vector and the aggregated measurements are fed to the action recommender ML model at the time [T]. Also, the state probabilities vector of [T+1] is always set to the Normal state. The predicted action is applied automatically to the network via the python script mentioned above.



(a) Training procedure of ARE.



(b) ARE's Closed-loop diagram.

Figure 6-7 Training and Operation modes of Action Recommendation Engine

Chapter 7. Testing and Evaluation

In this chapter, we present all the results for Delay Measurement Estimator and Action Recommendation Estimator.

7.1 Distributed Delay Measurements Estimation

7.1.1 Standalone Deme

We assess our proposed system, DEME, on the KING's testing sub-dataset. First, we compared DEME's performance with state-of-the-art algorithms such as Vivaldi, DMF, Phoenix, and IDES in terms of accuracy. Table 7-1 shows the accuracy of DEME versus the other algorithms. DEME scored around 96.1% of accuracy and Phoenix has the highest accuracy among the other algorithms at 93.5%.

Table 7-1 Average accuracy for AI approaches and state-of-the-art algorithms in terms of percentage.

DEME	96.1%
Vivaldi	90.6%
IDES	92.3%
DMF	92.8%
Phoenix	93.5%

Figure 7-1 presents the cumulative distribution function of relative errors for the proposed system and the other algorithms. We note that DEME outperforms all other algorithms in all scenarios. In addition, we measured the NPRE of DEME to be 0.25, while Phoenix had the nearest performance to MDN with 0.43. Furthermore, the mean error of prediction for DEME is 50ms.

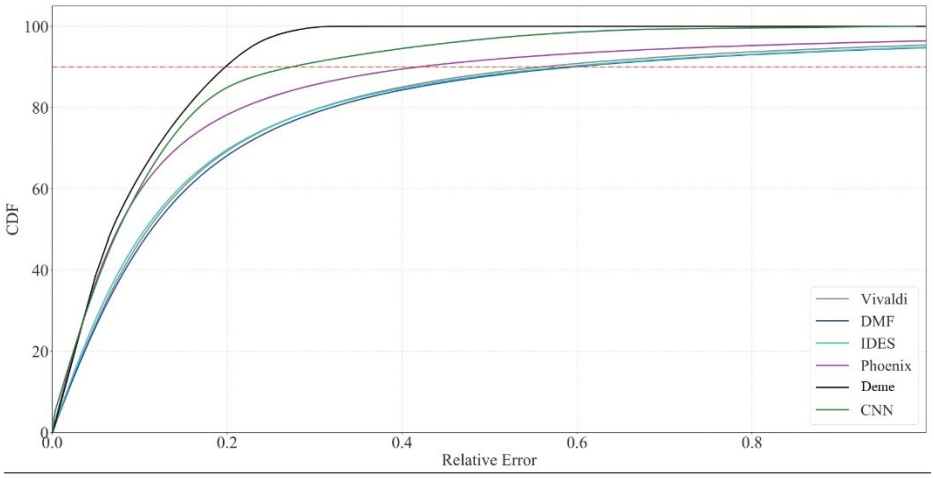


Figure 7-1 Cumulative distribution function (CDF) of relative errors (RE).

Finally, Figure 7-2 shows a pie chart representing shows the delay distribution of the dataset based on the ratio between predicted and measured latencies (i.e., predicted/measured). The ratio of less than one means underestimated latency, On the other hand, the ratio of more than one means overestimated latencies. In our results, 76.4% of the ratios lie between 0.8 and 1.2, the purple region. In other words, approximately three-quarters of the estimated latencies in the KING dataset are within an estimation error of 20%.

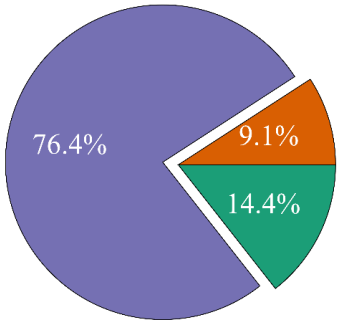


Figure 7-2 Latency distribution ratio of KING dataset

7.1.2 MLOps-Enabled Deme

In this section, we evaluate our proposed MLOps-Enabled Deme and compare its performance with: (i) standalone Deme, i.e., testing Deme As-Is (ii) a trained Deme from scratch. We split Swarmio's

lively-collected dataset into three parts and used it for training (60%), validation (10%), and testing (30%). The evaluation is done on the same machine used to train Deme before. The workstation runs Ubuntu 18.04 with specifications of intel core i9-9900k, 64 GB of Ram, and Nvidia GeForce RTX 2080 as GPU. Table 7-2 shows the average accuracy of the proposed MLOps-Enabled Deme against the other two approaches. The accuracy of the MLOps-Enabled Deme is 93%. while the standalone Deme system scores 74%. Similarly, the measured NPRE for MLOps-Enabled Deme is 0.33. While standalone Deme had the closest performance with 0.73. Figure 7-3 presents the CDF of relative errors for MLOps-Enabled Deme and the other approaches. We observe that the MLOps-Enabled Deme outperforms all other approaches in all scenarios.

Table 7-2 Performance summary for MLOps-Enabled Deme and other approaches.

Model	Accuracy	NPRE
DL-Pipeline	93%	0.33
As-Is Deme	74%	0.73
Deme Trained From-Scratch	63%	0.81

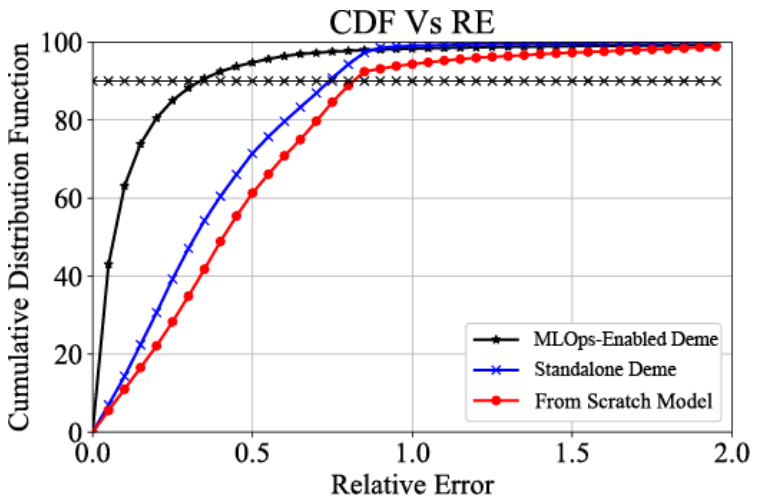


Figure 7-3 CDF of Relative Errors.

To study the effect of data amount on the proposed model's performance, we utilized batches of 1000 data points during online learning. Figure 7-4 and Figure 7-5 show the effect of data size on

the proposed model's NPRE and accuracy, respectively. It can be seen that the number of data points required to reach our threshold, 90%, is around 50 thousand. However, we preferred to continue training to get the best possible performance. Finally, we employed a Monte-Carlo Dropout methodology to test the model's confidence. The worst confidence is reported at an average delay of 100 ms with a standard deviation of ± 0.0007%.

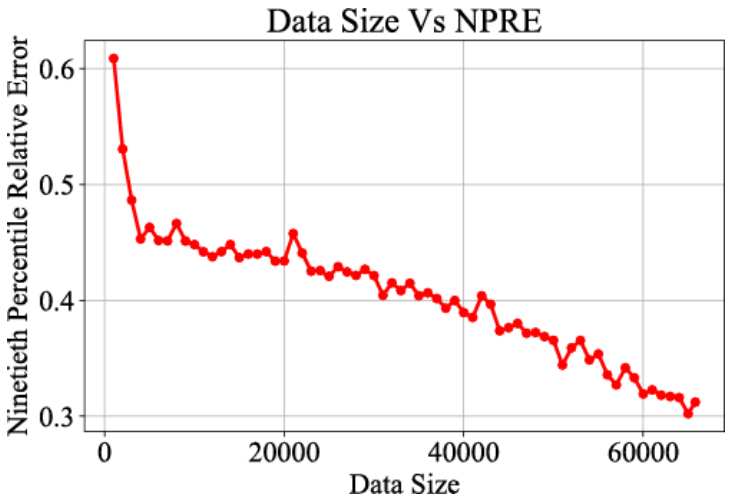


Figure 7-4 Effect of data size on model's NPRE.

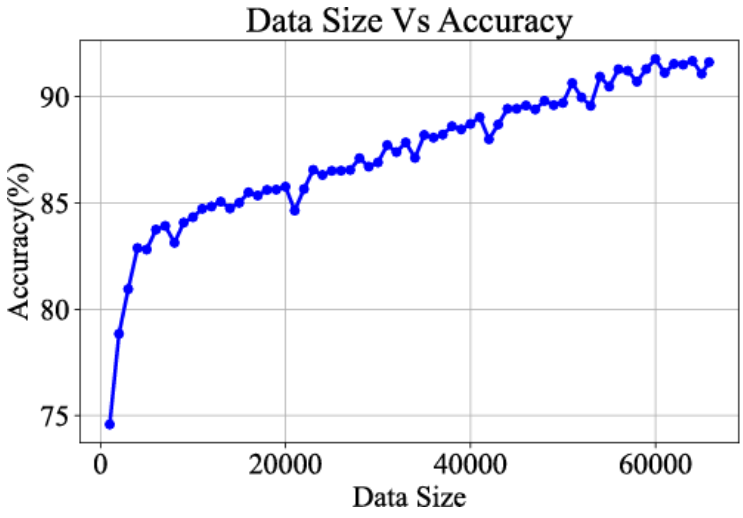


Figure 7-5 Effect of data size on model's accuracy.

7.2 Network Automation – Action Recommendation Engine (ARE)

As mentioned in chapter 5, ARE is divided into two main processes; network state classification and action recommendation estimator. In the following subsection, we present the results regarding accuracy and confusion matrices for the processes.

7.2.1 Network State Classifier

40% of the dataset is reserved for testing and the rest is used in training the GB, DT, and XGBoost models. Table 7-3 lists all model parameters and accuracies. Table 7-4 shows the F1 scores of the three models. XGBoost algorithm slightly outperforms the other algorithms in terms of accuracy and F1-score. As shown in the F1 scores, the congestion states suffer from performance degradation for both GB and DT. On the other hand, this relatively poor performance did not affect both models' overall accuracy due to the dataset imbalance (i.e., The number of Normal datapoints are comparable with the other classes combined).

Table 7-3 State – Model's Parameters and Accuracies

Model	Parameters	Accuracy
DT	Split Criterion: Gini, Splitter: best, Depth: 19, Leaves: 149	97.119%
GB	Learning rate: 0.1, Max depth: 3, No. of estimator: 100	98.138%
XGB	Learning rate: 0.3, Max depth: 6, booster: gbtree, No. of estimator: 100	99.02%

Table 7-4 State – F1 Scores.

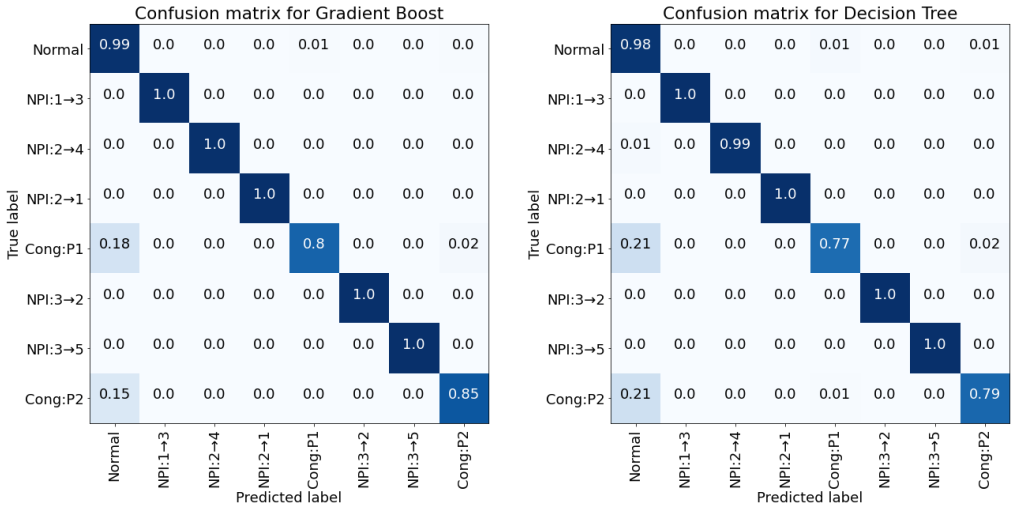
	Normal	NPI:3→5	NPI:2→4	NPI:2→1	NPI:3→2	NPI:1→3	Cong:P1	Cong:P2
DT	0.98	1	0.99	1	1	1	0.75	0.78
GB	0.99	1	1	1	1	1	0.84	0.86
XGB	0.98	1	0.99	1	0.97	1	0.87	0.90

To further assess the classification models, we decide on utilizing confusion matrices. Figure 7-6 shows the normalized confusion matrices for the three classifiers. It can be seen that all models can correctly classify NPI states. However, there is confusion between congestion and normal states. It can be said that this is the only

considerable confusion causing the models to degrade. DT fails to distinguish Congestion in PATH_1 and PATH_2 from the Normal state, while GB is slightly better at separating the two Congestion classes from the Normal state. Unlike its counterparts, XGB's performance is not weakened by this task. Table 7-5 shows a comparison of this work with state-of-the-art.

Table 7-5 State – Performance and Summary Comparison.

Paper	Platform	ML-Type	Model	Pro/Re-Active	Contribution	Reported Results
[56]	SDN	Unsupervised	K-nearst Neighbour	Reactive	Classify network status into congested or not-congested	Acc: 98%
[57]	Core	Supervised	Naïve Bayes	Reactive	Categorize the network status as faulty or normal and localize faults	Acc: 95%
[58]	NFV	Supervised	SVM	Both	Classify network status and predict if the problem happened or not	Acc: 95%
Our Work	Core	Supervised	XGB	Reactive	Classify network status into 3 classes and localize faults	Acc: 99%



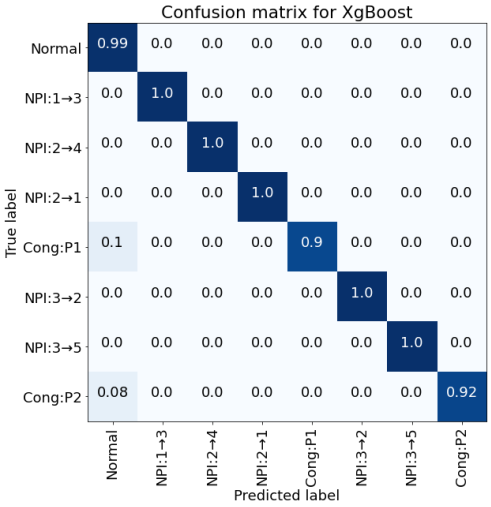


Figure 7-6 State Classifier - Confusion Matrices for GB, DT and XGB.

7.2.2 Action Recommender Estimator

The collected data were randomly split into 75% for training and 25% for testing. Furthermore, we opt to utilize 10-fold cross-validation on the training datasets. Table 7-6 lists all models' parameters and accuracies. Table 7-7 shows the same trend as the network classifier. XGB slightly outperforms the other two models. It is obvious that all the models can clearly detect the majority class "Do Nothing" with an F1 score of almost 100%. Also, the three models can identify any link issues with F1 scores ranging from 0.97 to 1.00. Nevertheless, it is obvious that performance degrades in detecting rerouting due to congestion.

Table 7-6 Action – Model's Parameters and Accuracies.

Model	Parameters	Accuracy
DT	Split Criterion: Gini, Splitter: best, Depth: 19, Leaves: 149	99.62%
GB	Learning rate: 0.1, Max depth: 3, No. of estimator: 100	99.72%
XGB	Learning rate: 0.3, Max depth: 6, booster: gbtree, No. of estimator: 100	99.87%

Table 7-7 Action – F1 Scores.

Model	TE-Action					
	P1-P2	P3-P2	P1-P3	P3-P2	P2-P1	P3-1
DT	0.88	0.94	0.9	0.93	1	0.81
GB	0.91	0.98	0.93	0.92	0.84	.088
XGB	0.92	1	0.96	0.97	1	0.96
Model	**Fix-Action**					
	DoNoth	LR-1-3	LR-2-4	LR-2-1	LR-3-2	LR-3-5
DT	1	0.99	0.99	0.97	1	1

GB	1	1	0.99	1	1	1
XGB	1	1	1	1	1	1

Figure 7-7 shows the confusion matrices for the three algorithms. As can be seen, DT and GB are the least accurate models, especially in rerouting actions. XGBoost algorithm shows are able to perfectly classify actions due to network issues and with higher accuracy on rerouting actions compared to the other algorithms.

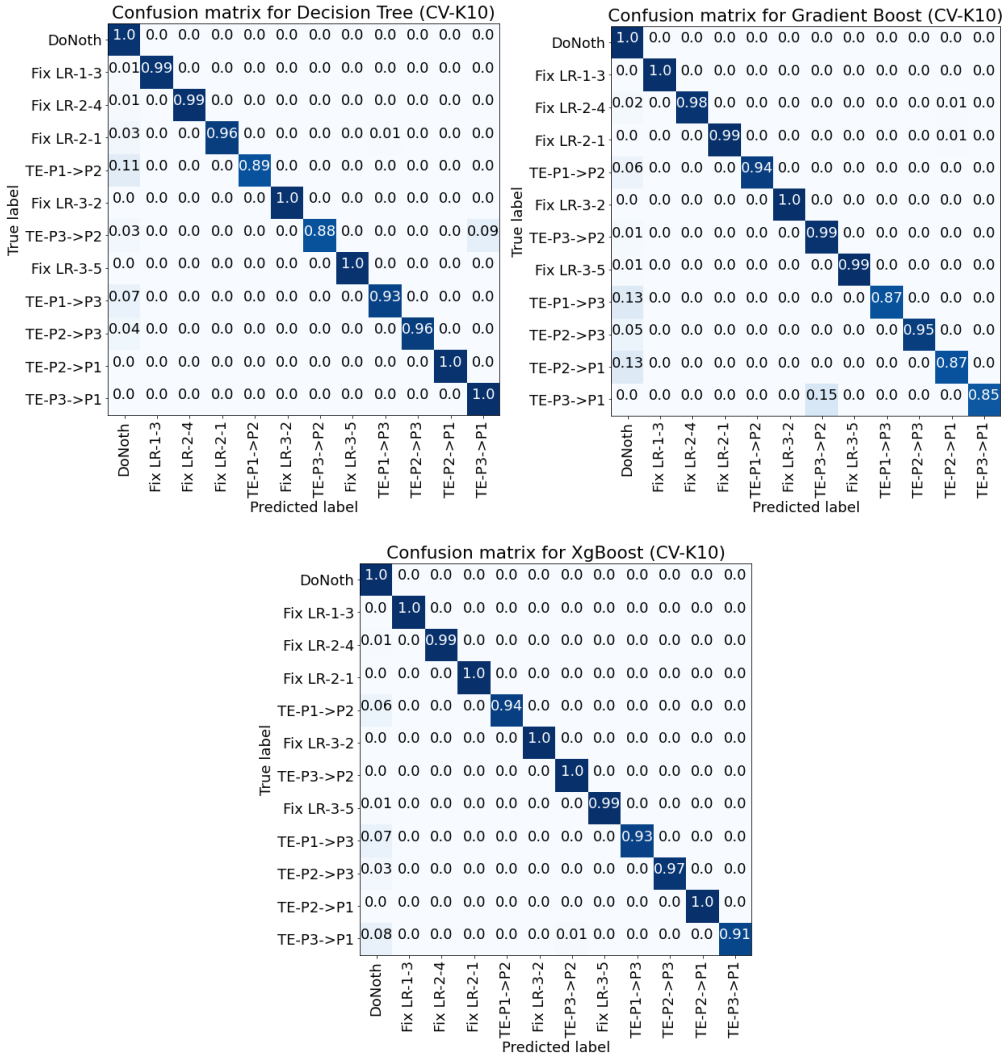


Figure 7-7 Action - Confusion Matrices for GB, DT and XGB.

To further assess both classifiers, we employ a live closed-loop system emulation that is quite similar to a real-life situation faced by NOCs. We compare the system's performance in terms of

incremental QoE-OPEX as the main metric of comparison. We compare ARE with *NOC "expert rules"*, *"Static"* which represents an unmanaged network, and an *"Anchor"* method which represents existing works in that it is a congestion-aware traffic engineering algorithm without taking into consideration OPEX. We used the same expert rules in Anchor implementation but we considered the cost of all paths to be equal, in essence not taking into account OPEX. We ran this experiment for more than 1200 steps (i.e., 10 hours). Figure 7-8 shows a comparison between the three cases. It is evident that all methods outperform the static network one. The second-worst performing method is the Anchor method, which is expected since it does not consider OPEX. Furthermore, XGB-model closely follows the expert rules, proving that the automated recommendations by ARE are valid.

The only time NOC rules lead to noticeably better results than ARE is at approximately iteration 770. When we checked that instance, we found that the network state classifier has incorrectly classified the network state as "Normal". It is worth noting that in the real world, NOC technicians might also be misled by the data and/or make mistakes. Also, NOC technicians can always override the ARE recommendations, at least at the beginning of the deployment till the ML model mature. But generally, what the Figure shows is that letting ARE take autonomous actions will lead to results that are quite comparable to the manual decisions taken by NOC technicians, despite ARE's mistakes, and this is a positive step towards NOC automation.

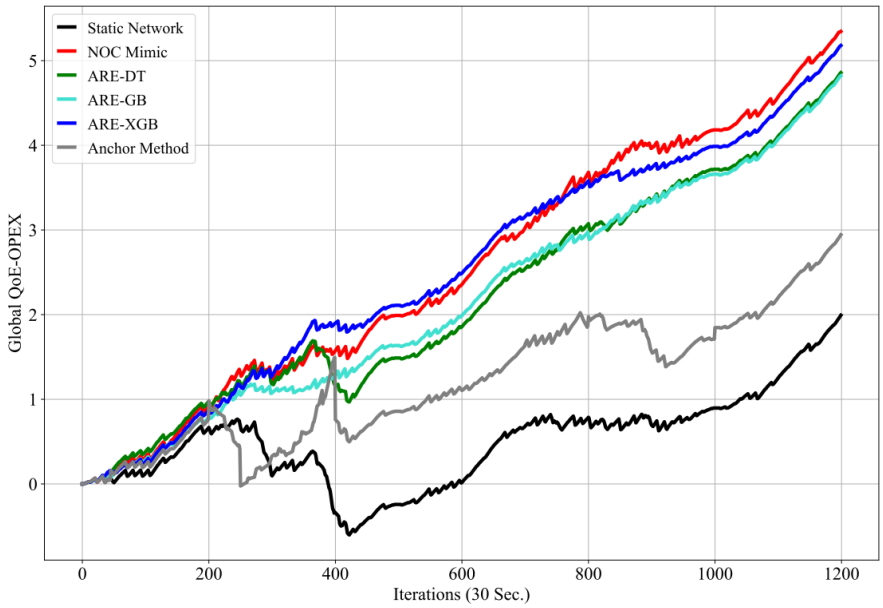


Figure 7-8 Cumulative QoE-OPEX comparison between ARE, Static, Anchor, and NOC rules methods.

Chapter 8. Conclusion and Future Work

In this chapter, we conclude the work presented in this thesis and propose future research directions.

8.1 Conclusion

In recent years, computer networks have remarkably become more diverse and complex. Network maintainers perform a numerous number of network measurements. Although these measurements provide network maintainers with information regarding instant network behaviour and enable them to interact. It adds significant overhead to the network's infrastructure. To this end, the overall focus of this thesis is on applying artificial intelligence to tackle the measurement overhead and help in realizing autonomous network automation and management. We chose two main challenges: (1) predicting delay distributed measurements to save the network from the overhead of the explicit ones, and (2) network automation, in terms of predicting network status, localize faulty devices and links, identify the root cause problem, and suggest suitable action set to keep the network in its best state, i.e., normal state.

Chapter 2 provided the required background concepts for both machine learning and the core network concepts. Chapter 3 documented the related works for both challenges. This chapter helped us obtaining insights into the shortcomings of the existing state-of-art solutions. For instance, the current stream delay measurements are not mathematically proven to settle. Taking into our consideration these shortcomings, we investigated a novel distributed delay estimator system (Deme) in chapter 4. Deme is considered the first end-to-end delay estimator to utilize artificial intelligence. Afterwards, we investigated the real-world deployment

issues and solutions for Deme in chapter 5. We proposed MLOps feedback pipelines that continuously monitor Deme's performance and ensure continuous integration and development. In chapter 6, we switched gears to our second challenge, *Network Automation Systems*. In this chapter, we proposed an AI-assisted Action Recommendation Engine (ARE). First, we presented the testbench we built to collect data. Then, we demonstrated ARE's architecture which is divided into two main components; network status predictor and action estimator. The ARE is not only able to predict the network status and localize the faulty device/link but also recommend a suitable set of actions to keep the network in its normal state.

8.2 Topics for Future Work

8.2.1 Distributed delay measurements – Deme

As shown in this thesis, Deme is the first distributed end-to-end delay estimator to utilize artificial intelligence as a core predictor. On that account, there is still a research opportunity to refine the proposed architecture.

The dataset used in designing Deme lacks timing information. Therfore, we were only able to predict the delay between any two network nodes once, ignoring the time as well as the current state of the network. New data can be collected in a form of time-based series. A time-based dataset opens the door for applying time-aware neural networks such as recurrent neural networks and their derivatives.

8.2.2 Network Automation System - ARE

The main improvement for ARE is to solve its scalability issue. The ML model in ARE needs to be adjusted every time the ISP's network grows. We can examine applying Graph Neural Networks (GNN), which is proven to work well with graphs and networks.

Similar to Deme, ARE is an AI-based system that is prone to working in a real environment. The incoming inference data will be different from the data ARE trained with. Therefore, we can investigate the feasibility of applying MLOps workflow to continuously monitor ARE's performance and automatically trigger retraining in case of noticeable performance degradation.

In this thesis, we proved the feasibility of training a machine-learning algorithm to mimic all the actions taken by the NOC manually. However, the automation problem can be tackled by a different approach, *Reinforcement Learning (RL)*. RL can even take actions that may be better than the actions taken by NOC. This could potentially revolutionize the field of network automation and management.

Reference

[1] I. Farris, T. Taleb, Y. Khettab, and J. Song, "A survey on emerging SDN and NFV security mechanisms for IoT systems," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 1, pp. 812–837, 2019, doi: 10.1109/COMST.2018.2862350.

[2] M. Ojo, D. Adami, and S. Giordano, "A SDN-IoT architecture with NFV implementation," *2016 IEEE Globecom Workshops, GC Wkshps 2016 - Proceedings*, 2016, doi: 10.1109/GLOCOMW.2016.7848825.

[3] N. Omnes, M. Bouillon, G. Fromentoux, and O. le Grand, "A programmable and virtualized network & IT infrastructure for the internet of things: How can NFV & SDN help for facing the upcoming challenges," *2015 18th International Conference on Intelligence in Next Generation Networks, ICIN 2015*, pp. 64–69, 2015, doi: 10.1109/ICIN.2015.7073808.

[4] R. Zaheer and S. Khan, "Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges," pp. 257–260, 2012, doi: 10.1109/FIT.2012.53.

[5] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 4, pp. 2347–2376, Oct. 2015, doi: 10.1109/COMST.2015.2444095.

[6] Y. Francillette, L. Abrouk, and A. Gouaich, "A players clustering method to enhance the players' experience in multi-player games," in *Proceedings of CGAMES 2013 USA - 18th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational and Serious Games*, 2013, pp. 229–234. doi: 10.1109/CGames.2013.6632639.

[7] S. Agarwal and J. R. Lorch, *Matchmaking for Online Games and Other Latency-Sensitive P2P Systems*. 2009. Accessed: Oct. 17, 2020. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/1592568.1592605?casa_token=n6rFHWhIYzUAAAAA:phRnYV2Di5HK10Vwkqv98pKjsUjmDZioV1j1nV9lyBH1e16Rn0fpJe3YE8y9z5h6f_2-cO3DvlE

[8] R. Krishnan *et al.*, "Moving beyond end-to-end path information to optimize CDN performance," in *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, 2009, pp. 190–201. doi: 10.1145/1644893.1644917.

[9] M. Steiner and E. W. Biersack, "Where is my peer? evaluation of the vivaldi network coordinate system in azureus," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5550 LNCS, pp. 145–156. doi: 10.1007/978-3-642-01399-7_12.

[10] L. Velasco and M. Ruiz, *Provisioning, Recovery, and In-Operation Planning in Elastic Optical Networks*. 2017. Accessed: Oct. 17, 2020.

[Online]. Available: https://books.google.com/books?hl=en&lr=&id=9EkzDwAAQBAJ&oi=fnd&pg=PA13&dq=Provisioning,+Recovery,+and+In+Operation+Planning+in+Elastic+Optical+Networks&ots=avbpDDUlQI&sig=9y36OH3J9C8ICW6JfJDG06LNy4g

[11] A. Malik, B. Aziz, M. Adda, and C. H. Ke, "Smart routing: Towards proactive fault handling of software-defined networks," *Computer Networks*, 2020, doi: 10.1016/j.comnet.2020.107104.

[12] A. Kvalbein, T. Čičiv, and S. Gjessing, "Post-failure routing performance with multiple routing configurations," 2007. doi: 10.1109/INFCOM.2007.20.

[13] T. Čičić *et al.*, "Relaxed multiple routing configurations: IP fast reroute for single and correlated failures," *IEEE Transactions on Network and Service Management*, 2009, doi: 10.1109/TNSM.2009.090301.

[14] D. Imahama, Y. Fukushima, and T. Yokohira, "A reroute method using multiple routing configurations for fast IP network recovery," in *2013 19th Asia-Pacific Conference on Communications, APCC 2013*, 2013, pp. 433–438. doi: 10.1109/APCC.2013.6765984.

[15] A. Capone, C. Cascone, A. Q. T. Nguyen, and B. Sansò, "Detour planning for fast and reliable failure recovery in SDN with OpenState," 2015. doi: 10.1109/DRCN.2015.7148981.

[16] M. R. Parsaei, S. Habib Khalilian, and R. Javidan, "A Comparative Study on Fault Tolerance Methods in IP Networks versus Software Defined Networks Introduction," *International Academic Journal of Science and Engineering*, vol. 1, no. 1, pp. 29–37, 2014, Accessed: Oct. 17, 2020. [Online]. Available: www.iaiest.com

[17] D. Mirkovic, G. Armitage, and P. Branch, "A survey of round trip time prediction systems," *IEEE Communications Surveys and Tutorials*, 2018, doi: 10.1109/COMST.2018.2816917.

[18] M. Meeker, "Internet Trends 201."

[19] "US20190280942A1 - Machine learning systems and methods to predict abnormal behavior in networks and network data labeling - Google Patents." https://patents.google.com/patent/US20190280942A1/en (accessed Oct. 17, 2020).

[20] D. W. Boertjes, M. Reimer, and D. Côté, "Practical considerations for near-zero margin network design and deployment [Invited]," *Journal of Optical Communications and Networking*, vol. 11, no. 9, pp. C25–C34, Sep. 2019, doi: 10.1364/JOCN.11.000C25.

[21] "Cisco Visual Networking Index: Global Mobile Data," 2019.

[22] "Unified Assurance and Analytics." https://www.blueplanet.com/products/uaa.html (accessed Jan. 04, 2021).

[23] "Products | Route Optimization and Analysis." https://www.blueplanet.com/products/route-optimization.html (accessed Jan. 04, 2021).

[24] "Cisco DNA Analytics ." https://www.cisco.com/c/en/us/solutions/collateral/enterprise-

networks/nb-06-ai-nw-analytics-wp-cte-en.html?oid=wpren017300 (accessed Oct. 17, 2020).
[25] K. P. Murphy, *Machine learning: a probabilistic perspective (adaptive computation and machine learning series)*. 2012.
[26] A. Burkov, "Hundred Page Machine Learning Book," *Expert Systems*, 2018.
[27] T. G. Dietterich, "Experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization," *Machine Learning*, 2000, doi: 10.1023/A:1007607513941.
[28] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, 2001, doi: 10.1214/aos/1013203451.
[29] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," 2016. doi: 10.1145/2939672.2939785.
[30] H. Schulz and S. Behnke, "Deep Learning: Layer-Wise Learning of Feature Hierarchies," *KI - Kunstliche Intelligenz*, 2012, doi: 10.1007/s13218-012-0198-z.
[31] J. F. Trevor Hastie, Robert Tibshirani, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. 2000.
[32] D. Svozil, V. Kvasnička, and J. Pospíchal, "Introduction to multi-layer feed-forward neural networks," 1997. doi: 10.1016/S0169-7439(97)00061-0.
[33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," 2012. doi: 10.1061/(ASCE)GT.1943-5606.0001284.
[34] I. Goodfellow, "Tutorial on Neural Network Optimization Problems," *Deep Learning Summer School*, 2015.
[35] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," 2011.
[36] "Gartner Says Nearly Half of CIOs Are Planning to Deploy Artificial Intelligence." https://www.gartner.com/en/newsroom/press-releases/2018-02-13-gartner-says-nearly-half-of-cios-are-planning-to-deploy-artificial-intelligence (accessed Aug. 27, 2021).
[37] "MLOps: Continuous delivery and automation pipelines in machine learning." https://cloud.google.com/solutions/machine-learning/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning#devops_versus_mlops (accessed Jan. 04, 2021).
[38] S. v. Stehman, "Selecting and interpreting measures of thematic classification accuracy," *Remote Sensing of Environment*, 1997, doi: 10.1016/S0034-4257(97)00083-7.
[39] S. Shirmohammadi and H. al Osman, "Machine Learning in Measurement Part 1: Error Contribution and Terminology Confusion," *IEEE Instrumentation and Measurement Magazine*, vol. 24, no. 2, pp. 84–92, Apr. 2021, doi: 10.1109/MIM.2021.9400955.

[40] H. al Osman and S. Shirmohammadi, “Machine Learning in Measurement Part 2: Uncertainty Quantification,” *IEEE Instrumentation and Measurement Magazine*, vol. 24, no. 3, pp. 23–27, May 2021, doi: 10.1109/MIM.2021.9436102.

[41] P. Francis *et al.*, “IDMaps: A global Internet host distance estimation service,” *IEEE/ACM Transactions on Networking*, 2001, doi: 10.1109/90.958323.

[42] T. S. Eugene Ng and H. Zhang, “Towards Global Network Positioning,” 2001. doi: 10.1145/505203.505206.

[43] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti, “Lighthouses for scalable distributed location,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2735, pp. 278–291, 2003, doi: 10.1007/978-3-540-45172-3_26.

[44] T. Ng, H. Z.-U. A. T. Conference, undefined General, and undefined 2004, “A network positioning system for the internet.,” *usenix.org*, Accessed: Oct. 18, 2020. [Online]. Available: https://www.usenix.org/event/usenix04/tech/general/full_papers/ng/ng_html/

[45] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, “Vivaldi: A decentralized network coordinate system,” in *Computer Communication Review*, 2004, vol. 34, no. 4, pp. 15–26. doi: 10.1145/1030194.1015471.

[46] F. Zhao, J. Chen, D. Ye, and X. Luo, “A Network Coordinate System Constructing Algorithm Based on Optimal Neighbor Nodes,” in *Proceedings - 2017 IEEE 2nd International Conference on Data Science in Cyberspace, DSC 2017*, Aug. 2017, pp. 140–144. doi: 10.1109/DSC.2017.94.

[47] Y. Chen, Y. Xiong, X. Shi, B. Deng, and X. Li, “Pharos: A decentralized and hierarchical network coordinate system for internet distance prediction,” in *GLOBECOM - IEEE Global Telecommunications Conference*, 2007, pp. 421–426. doi: 10.1109/GLOCOM.2007.85.

[48] N. Hariri, B. Hariri, and S. Shirmohammadi, “A distributed measurement scheme for internet latency estimation,” in *IEEE Transactions on Instrumentation and Measurement*, May 2011, vol. 60, no. 5, pp. 1594–1603. doi: 10.1109/TIM.2010.2092871.

[49] L. Tang and M. Crovella, “Virtual landmarks for the internet,” in *Proceedings of the 2003 ACM SIGCOMM conference on Internet measurement - IMC ’03*, 2003, p. 143. doi: 10.1145/948205.948223.

[50] R. Zhu, B. Liu, D. Niu, Z. Li, and H. V. Zhao, “Network Latency Estimation for Personal Devices: A Matrix Completion Approach,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 724–737, Apr. 2017, doi: 10.1109/TNET.2016.2612695.

[51] J. Cheng, Y. Liu, Q. Ye, H. Du, and A. v. Vasilakos, “DISCS: A Distributed Coordinate System Based on Robust Nonnegative Matrix Completion,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 934–947, Apr. 2017, doi: 10.1109/TNET.2016.2615889.

[52] Y. Mao, L. K. Saul, and J. M. Smith, “IDES: An internet distance estimation service for large networks,” *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 12, pp. 2273–2283, Dec. 2006, doi: 10.1109/JSAC.2006.884026.

[53] Y. Liao, P. Geurts, and G. Leduc, “Network distance prediction based on decentralized matrix factorization,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010, vol. 6091 LNCS, pp. 15–26. doi: 10.1007/978-3-642-12963-6_2.

[54] Y. Chen *et al.*, “Phoenix: A weight-based network coordinate system using matrix factorization,” *IEEE Transactions on Network and Service Management*, vol. 8, no. 4, pp. 334–347, Dec. 2011, doi: 10.1109/TNSM.2011.110911.100079.

[55] Y. Chen, P. Sun, X. Fu, and T. Xu, “Improving prediction accuracy of matrix factorization based Network Coordinate systems,” 2010. doi: 10.1109/ICCCN.2010.5560092.

[56] R. Kumar, U. Venkanna, and V. Tiwari, “A Binary Classification Approach for Time Granular Traffic in SDWMN based IoT Networks,” in *2020 International Conference on COMmunication Systems and NETworkS, COMSNETS 2020*, Jan. 2020, pp. 531–534. doi: 10.1109/COMSNETS48256.2020.9027336.

[57] A. Saple and L. Yilmaz, “Agent-based simulation study of behavioral anticipation: Anticipatory fault management in computer networks,” in *Proceedings of the Annual Southeast Conference*, 2006, vol. 2006, pp. 383–388. doi: 10.1145/1185448.1185533.

[58] L. Gupta, T. Salman, M. Zolanvari, A. Erbad, and R. Jain, “Fault and performance management in multi-cloud virtual network services using AI: A tutorial and a case study,” *Computer Networks*, vol. 165, p. 106950, Dec. 2019, doi: 10.1016/j.comnet.2019.106950.

[59] J. Park and J. S. Kim, “A classification of network traffic status for various scale networks,” in *International Conference on Information Networking*, 2013, pp. 595–599. doi: 10.1109/ICOIN.2013.6496693.

[60] Z. Shu *et al.*, “Traffic Engineering in Software-Defined Networking: Measurement and Management,” *IEEE Access*, vol. 4, pp. 3246–3256, 2016, doi: 10.1109/ACCESS.2016.2582748.

[61] Y. Zuo, Y. Wu, G. Min, and L. Cui, “Learning-based network path planning for traffic engineering,” *Future Generation Computer Systems*, vol. 92, pp. 59–67, 2019, doi: 10.1016/j.future.2018.09.043.

[62] Y. M. Lee, “Classification of node degree based on deep learning and routing method applied for virtual route assignment,” *Ad Hoc Networks*, vol. 58, pp. 70–85, Apr. 2017, doi: 10.1016/j.adhoc.2016.11.007.

[63] M. Arifuzzaman, O. A. Dobre, M. H. Ahmed, and T. M. N. Ngatched, “Joint routing and MAC layer QoS-aware protocol for wireless sensor networks,” 2016. doi: 10.1109/GLOCOM.2016.7841933.

[64] Y. Niu, Y. Liu, Y. Li, X. Chen, Z. Zhong, and Z. Han, “Device-to-device communications enabled energy efficient multicast scheduling in

mmWave small cells," *IEEE Transactions on Communications*, vol. 66, no. 3, pp. 1093–1109, Mar. 2018, doi: 10.1109/TCOMM.2017.2773529.
[65] E. Oh and B. Krishnamachari, "Energy savings through dynamic base station switching in cellular wireless access networks," 2010. doi: 10.1109/GLOCOM.2010.5683654.
[66] F. Tang *et al.*, "On Removing Routing Protocol from Future Wireless Networks: A Real-time Deep Learning Approach for Intelligent Traffic Control," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 154–160, Feb. 2018, doi: 10.1109/MWC.2017.1700244.
[67] A. Azzouni, R. Boutaba, and G. Pujolle, "NeuRoute: Predictive dynamic routing for software-defined networks," in *2017 13th International Conference on Network and Service Management, CNSM 2017*, Jul. 2017, vol. 2018-January, pp. 1–6. doi: 10.23919/CNSM.2017.8256059.
[68] A. Hari, U. Niesen, and G. Wilfong, "On the Problem of Optimal Path Encoding for Software-Defined Networks," in *IEEE/ACM Transactions on Networking*, Feb. 2017, vol. 25, no. 1, pp. 189–198. doi: 10.1109/TNET.2016.2571300.
[69] A. Lazaris and V. K. Prasanna, "DeepFlow: A deep learning framework for software-defined measurement," in *CAN 2017 - Proceedings of the 2017 Cloud-Assisted Networking Workshop, Part of CoNext 2017*, Dec. 2017, pp. 43–48. doi: 10.1145/3155921.3155922.
[70] Z. Xu *et al.*, "Experience-driven Networking: A Deep Reinforcement Learning based Approach," in *Proceedings - IEEE INFOCOM*, Oct. 2018, vol. 2018-April, pp. 1871–1879. doi: 10.1109/INFOCOM.2018.8485853.
[71] M. Huang, W. Liang, Z. Xu, W. Xu, S. Guo, and Y. Xu, "Dynamic routing for network throughput maximization in software-defined networks," in *Proceedings - IEEE INFOCOM*, Jul. 2016, vol. 2016-July. doi: 10.1109/INFOCOM.2016.7524613.
[72] F. Geyer and G. Carle, "Learning and generating distributed routing protocols using graph-based deep learning," in *Big-DAMA 2018 - Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks, Part of SIGCOMM 2018*, Aug. 2018, pp. 40–45. doi: 10.1145/3229607.3229610.
[73] Y. Zhang, Y. Zhang, S. Sun, S. Qin, and Z. He, "Multihop packet delay bound violation modeling for resource allocation in video streaming over mesh networks," *IEEE Transactions on Multimedia*, vol. 12, no. 8, pp. 886–900, Dec. 2010, doi: 10.1109/TMM.2010.2065217.
[74] "lisa-lab/pings." https://github.com/lisa-lab/pings (accessed Oct. 30, 2021).
[75] K. P. Gummadi, S. Saroiu, and S. D. Gfibble, "King: Estimating Latency between Arbitrary Internet End Hosts," 2002.
[76] "KeyCdn- content delivery made easy." https://www.keycdn.com/
[77] "Google Code Archive - Long-term storage for Google Code Project Hosting." https://code.google.com/archive/p/word2vec/ (accessed Aug. 28, 2021).

[78] A. Gupta, A. Vedaldi, and A. Zisserman, “Synthetic Data for Text Localisation in Natural Images,” 2016. Accessed: Oct. 18, 2020. [Online]. Available: http://www.robots.ox.ac.uk/~vgg/data/scenetext

[79] “Feature crosses-machine learning crash course by google.” https://developers.google.com/machine-learning/crash-course/featurecrosses/%0Avideo-lecture

[80] E. Winarno, W. Hadikurniawati, and R. N. Rosso, “Location based service for presence system using haversine method,” in *Proceedings - 2017 International Conference on Innovative and Creative Information Technology: Computational Intelligence and IoT, ICITech 2017*, Mar. 2018, vol. 2018-January, pp. 1–4. doi: 10.1109/INNOCIT.2017.8319153.

[81] B. Nojavanasghari, D. Gopinath, J. Koushik, T. Baltrušaitis, and L. P. Morency, “Deep multimodal fusion for persuasiveness prediction,” in *ICMI 2016 - Proceedings of the 18th ACM International Conference on Multimodal Interaction*, Oct. 2016, pp. 284–288. doi: 10.1145/2993148.2993176.

[82] L. Adato, “Monitoring 101: A primer to the philosophy, theory, and fundamental concepts involved in systems monitoring.”

[83] “Network Performance Monitor | SolarWinds.” https://try.solarwinds.com/network-performance-monitor?&CMP=KNC-TAD-GGL-SW_NA_X_PP_CPC_LD_EN_PBRDE_TIN-X-945268740~46582888306_g_c_solarwinds%20monitoring%20tools-e~411230658944~~p53914499890&s_kwcid=AL!11508!3!411230658944!e!!g!!solarwinds%20monitoring%20tools&ds_cid=71700000067663068&ds_agid=58700005885037323&network=g&device=c&keyword=Solarwinds%20monitoring%20tools&matchtype=e&creative=411230658944&feeditemid=&gclid=Cj0KCQjw8rT8BRCbARIsALWiOvTE1vbgU8dbQKWE9wMTM6rosjOxuY9G26eI7dMhhm8-3DWh81OpV1YaAs6dEALw_wcB&gclsrc=aw.ds (accessed Oct. 18, 2020).

[84] “GNS3 | The software that empowers network professionals.” https://www.gns3.com/ (accessed Oct. 18, 2020).

[85] “Wireshark · Go Deep.” https://www.wireshark.org/ (accessed Oct. 18, 2020).

[86] “Multiple Classifier Systems - First International Workshop, MCS 2000 Cagliari, Italy, June 21-23, 2000 Proceedings | Josef Kittler | Springer.” https://www.springer.com/gp/book/9783540677048 (accessed Oct. 18, 2020).

[87] T. M. Khoshgoftaar, A. Fazelpour, D. J. Dittman, and A. Napolitano, “Ensemble vs. data sampling: Which option is best suited to improve classification performance of imbalanced bioinformatics data?,” in *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, Jan. 2016, vol. 2016-January, pp. 705–712. doi: 10.1109/ICTAI.2015.106.

[88] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, “A review on ensembles for the class imbalance problem: Bagging-, boosting-, and hybrid-based approaches,” *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 42, no. 4. pp. 463–484, Jul. 2012. doi: 10.1109/TSMCC.2011.2161285.