GNN-based End-to-end Delay Prediction in Software Defined Networking

by

Zhun Ge

Thesis submitted in partial
fulfillment of the requirements for the
Master of Applied Science
Electrical and Computer Engineering degree

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

Abstract

Nowadays, computer networks have always been complicated and difficult deployment for both the scientific and industry groups as they attempt to comprehend and analyze network performance as well as design efficient procedures for their operation. In software-defined networking (SDN), predicting latency (delay) is essential for enhancing performance, power consumption and resource utilization in meeting its significant latency requirements.

In this thesis, we present a graph-based formulation of Abilene Network and other topologies and apply a Graph Neural Network (GNN)-based model, Spatial-Temporal Graph Convolutional Network (STGCN), to predict end-to-end packet delay on this formulation. It is found that this model outperforms the average baseline predictor in predicting packet delay since the STGCN framework captures both spatial and temporal dimensions of the data.

The evaluation is using STGCN to compare with other machine learning methods: Random Forest (RF) and Neural Network (NN). In the most complex network traffic condition with high traffic intensity, varying capacities and propagation delay, STGCN is 68.5% and 78.7% better than RF and NN, respectively.

More datasets are in used to verify and evaluate the performance of STGCN: 15 nodes network with various distributions and different network traffic distributions.

More Machine Learning (ML) methods with lager network topologies are used for performance evaluation. STGCN outperforms the baseline methodology and other three techniques: Multiple Linear Regression (MLR), Extreme Gradient Boosting (XGBOOST) and RF in 15-node scale-free, 24-node GEANT2, and 50-node networks. Notably, our GNN-based methodology can achieve 97.0%, 95.9%, 96.1%, and 63.1% less root mean square error (RMSE) than the baseline predictor, MLR, XGBOOST and RF, respectively.

All the experiments show that STGCN has good prediction performance with small and stable prediction errors. This thesis illustrates the feasibility and benefits of a GNN approach in predicting end-to-end delay in software-defined networks.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Amiya Nayak, for providing me with guidance, advice, and support throughout my master's study. Also, I am grateful to my senior colleague, Jiacheng Hou, for her valuable advice and support during my thesis work. Finally, I would like to thank my family, who continuously support me both physically and mentally.

Contents

List of Tables

List of Figures

Acronyms

ARIMA	Auto Regressive Integrated Moving Average.
CNN	Convolutional Neural Network.
DCGRU	Diffusion Convolutional Gated Recurrent.
DL	Deep Learning.
DQN	Deep Q-Network.
DT	Decision Tree.
DTPRO	Deep Q-Network and Traffic Prediction based Routing Optimization.
GCN	Graph Convolutional Network.
GGAE	Gated Graph Auto Encoder Network.
GNN	Graph Neural Network.
GPU	Graphics Processing Unit.
GRU	Gated Recurrent Unit.
HA	Historical Average.
KDN	Knowledge-Defined Networking.
KP	Knowledge Plane.
KPI	Key Performance Indicators.
LR	linear regression.
LSTM	Long Short-Term Memory.

MAE	Mean Absolute Error.
ML	Machine Learning.
MLP	Multi-layered Perceptron.
MPNN	Message Passing Neural Networks.
MRE	Mean Relative Error.
MSE	Mean Squared Error.
NB	Naive Bayes.
NC	Nearest Centroid.
NN	Neural Network.
NOS	Networking Operating System.
OD	Origin-Destination.
OVS	open virtual switch.
QoE	Quality of Experience.
QoS	Quality of Service.
RF	Random Fores.
RMSE	Root Mean Square Error.
RNN	Recurrent Neural Network.
SAE	Stacked Auto-Encoder.
SDN	Software Defined Networking.
SFS	Sequential Feature Selection.
SHAP	Shapley additive explanation.
SPF	shortest path first.
STGCN	Spatial-Temporal Graph Convolutional Network.
SVM	Support Vector Machine.

TnC	Traffic and Capacity.
TnCwD	Traffic and Capacity with Delays.
TO	Traffic Only.
TPU	Tensor Processing Unit.
WMAPE	Weighted Mean Absolute Percentage Error.
XGBoost	eXtreme Gradient Boosting.

Chapter 1

Introduction

1.1 Motivation and Objective

A Software Defined Networking (SDN) has the unique ability to manage networks dynamically. Due to its programmability and centralized control, the SDN has the potential to solve many complex design problems. As a result of the capabilities of the SDN (e.g., logically centralized control, global view of the network, software-based traffic analysis, and dynamic updating of forwarding rules), it becomes easier to apply machine learning techniques to SDN.

More specifically, SDN switches are configured by the logically centralized controller and have just the data plane that simplifies the packet forwarding process in the network. In the switch, the control plane is decoupled from the data plane, which makes the control plane logically centralized, with a network-wide view of packet forwarding based on the end-to-end path. An SDN switch, however, is susceptible to packet delay and loss, affecting network performance such as Quality of Experience (QoE) significantly. The SDN has been the subject of a tremendous amount of research in recent years, especially with the development of IoT devices and the resulting possibility of using the flexibility of SDN to manage traffic and improve computer communications. The prediction of traffic conditions is crucial in network operations and management for today's increasingly complex and diverse networks. SDN traffic prediction is crucial to network management and planning, as future traffic can be forecasted in advance, which can lead to improved network performance. SDN network traffic exhibits correlation and self-similarity characteristics, and traffic prediction can be used to increase network performance. Since available network bandwidth is limited, traffic classification allows us to make the most efficient use of it, and Internet service providers can manage the resources by prioritizing packets.

The Graph Neural Network (GNN) [46] is an emerging field of deep learning. The GNNs recognize the dependence of graphs using message passing between nodes through deep learning. They are neural networks that operate within the graph domain. With a strong graph representation learning ability, GNN has been seen to demonstrate impressive results in multiple graph data-related tasks in many applications. While The prediction of QoS parameters can be made quite easily by traditional Machine Learning (ML) techniques, we believe GNNs are quite suitable for this purpose and should be investigated. The representation of graph data here can be either at the node level or at the entire network level, which can make node level or network level prediction quite easy.

This thesis applies GNN to predict end-to-end delays in SDN. The SDN controller has a global view of the network, and if it knows in advance the delay from one node to another, it can improve the routing strategy of the network. Our goal is to use GNN to predict end-to-end delay accurately. In this case, the GNN can be deployed in an SDN controller. The SDN controller has prior knowledge of the network topology and, therefore, the node adjacency matrix information is available. In addition, the SDN controller knows the end-to-end delay from one node to another in each timestamp. During the delay inference phase, the SDN controller can feed both the adjacency matrix and packet delays of previous timestamps to the GNN. Once the GNN predicts delays of the entire network at future timestamps, the SDN controller can allocate traffic more efficiently based on the prediction. For example, the SDN controller can pass forwarding information to the switches via the Openflow protocol to distribute traffic from high-intensity to low-intensity switches in advance.

1.2 Main Contribution

In this thesis, we use the GNN framework to predict end-to-end delays with SDN for the Abilene and other network topologies. The contributions of this thesis are as follows:

- We formulate a line graph of the network and use network links as nodes and routers as edges.
- We apply the GNN-based model with the formulation and evaluate the performance on various datasets.
- We apply the model with the benchmark dataset and compare it with other machine learning (ML) models, Multiple Linear Regression (MLR), Extreme Gradient Boosting (XGBOOST), Random Forest (RF), and Neural Network (NN).

We find that the GNN-based approach outperforms all ML models significantly by as much as 96.1% in the most complex network scenario. After testing more datasets with different typologies and distributions, our results demonstrate the need to include network topology into prediction models explicitly and offer the viable approach of utilizing GNNs in network traffic delay prediction.

1.3 Publication

Zhun Ge, Jiacheng Hou, Amiya Nayak. GNN-based End-to-end Delay Prediction in Software Defined Networking. In Proc. of the 18th Annual International Conference on Distributed Computing in Sensor Systems (DCOSS) Workshop (REFRESH), June 2022.

1.4 Thesis Organization

The remainder of the thesis is organized as follows.

- Chapter 2 reviews concepts of Neural Network, Decision Tree, Random Forest, Recurrent Neural Network, Graph Convolutional Networks, Message Passing Neural Networks, Long Short-Term Memory, and Software-Defined Networking.
- Chapter 3 reviews related work on SDN prediction, including machine learning and deep learning approaches such as Gated Recurrent Unit Framework, Deep Q-Network, LSTM, NARX enabled RNN, NN, RF and other GNN approaches. The chapter also covers the review for STGCN.
- Chapter 4 discusses the SDN delay prediction STGCN model in detail and introduces our modified model training settings.
- Chapter 5 describes the 15-node, Abilene, GEANT2 and 50-node network datasets, the experiment setup, and the experimental results are also presented and analyzed in this chapter.
- Chapter 6 concludes this thesis and proposes several potential future works.

Chapter 2

Background

2.1 Neural Network

Neural Network is a machine learning technique that uses artificial neurons to simulate the way how biological neurons in the human brain work. The basic element of each neural network is the artificial neuron which consists of three components: weights and bias, summation function, and activation function.

As shown in Figure 2.1, each neuron is embedded with one summation function and one activation function; the summation function is usually a linear regression function with weight and bias, while the activation function provides non-linearity transformation for the summation output. Suppose we have n attributes labeled for sample i, where each attribute is labeled as x_i, and the coefficient for each x_i is w_i, and for each neuron we would assign a bias value b. The summation function for each neuron is defined as the following:

$$y_i = \sum_i w_i x_i + bias \tag{2.1}$$

A neural network is made up of layers of nodes that are linked together. Each

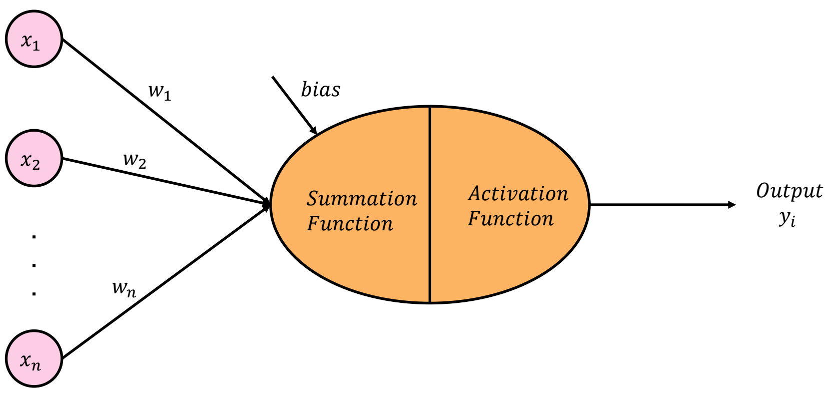

Figure 2.1: Structure of a Basic Neural Network with One Neuron

node is a set of algorithms, which is akin to multiple linear regression. The signal obtained by multiple linear regression is fed into a nonlinear activation function via the training algorithm.

In Figure 2.2, it shows a sample of a simple neural network. In this multi-layered perceptron (MLP), perceptrons are arranged in interconnected layers. The input layer is responsible for collecting input patterns. The output layer contains classifications or output signals that may be mapped to by input patterns. In the middle, there is a hidden layer. Hidden layers adjust the input weightings until the neural network's margin of error is as little as possible by experiments.

2.1.1 Supervised Neural Network Models

To create neural networks, Scikit-Learn[39], a popular Python AI toolkit, can be used. Using a data set to train the supervised learning algorithm, MLP can learn a function $f(X) : R_m- > R_o$, where m represents the number of dimensions for input and o shows the number of dimensions for output.

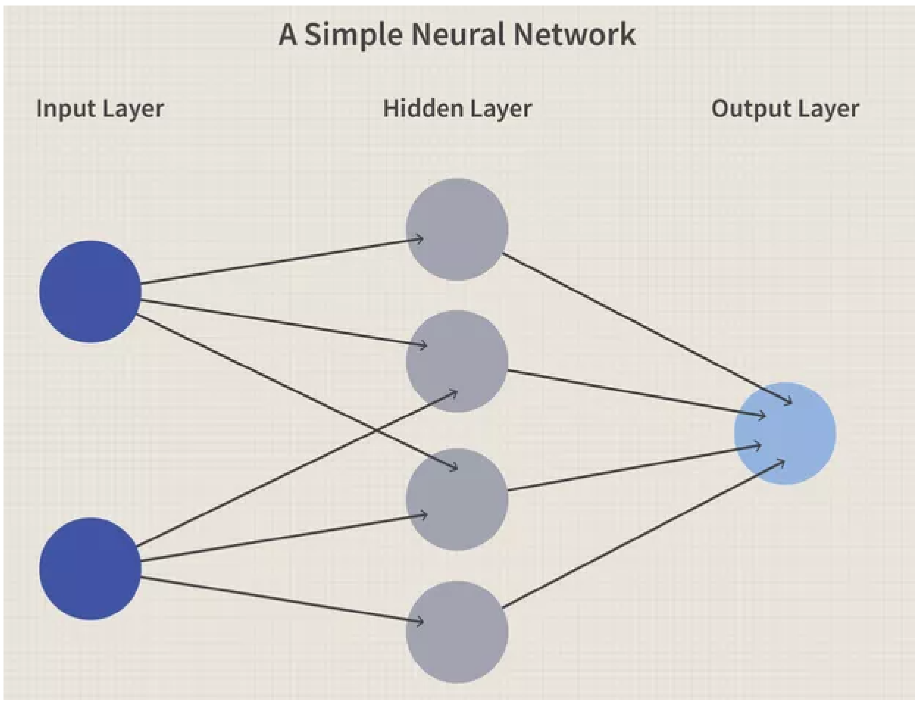

Figure 2.2: A Simple Neural Network with Layers[20]

Assuming a set with various features and one output. The Multiple Linear Regression(MLP) can learn a non-linear function for classification or regression approximately. Compared to logistic regression, it can find out the hidden layers between input features and the output. Here is the picture 2.2: input layer contains the features, and there is one non-linear layer in the middle.

The benefits of MLP are obvious; it is capable of learning non-linear models as well as real-time learning models. However, there are some drawbacks: MLP is sensitive to feature scaling and requires a large number of hyperparameters, such as the number of hidden neurons, layers, and iterations.

2.2 Decision Tree

Decision Tree is one alternative classifier based on tree structures. In general, a decision tree consists of one root node, multiple internal nodes, and multiple leaf nodes. The root node is the original input dataset along the tree path with different internal nodes, each node will split the dataset into smaller datasets based on the chosen splitting condition, and each leaf node is the final class result for a certain set of samples [15].

To achieve better performance, the decision tree will choose among all possible features to be the best splitting feature that can give the best splitting results. The criterion used to choose among these features can define different types of decision trees. The three commonly used criteria include Gini Index, Information Gain, and Information Grain Ratio [32].

2.2.1 Random Forest

A decision tree is not a strong classifier in some cases; many researchers proposed to apply ensemble methods by generating multiple classifiers instead of a single classifier to improve model accuracy. Ensemble methods including boosting, bagging, voting, and stacking. Random forest is one ensemble algorithm using bagging with a decision tree as the base model.

The concept of bagging is to generate multiple data sets from the original data sets by random sampling, and each generated data set is independently used to train a classifier [37]. All the classifiers will act like one voter, and the majority vote will be used as the final result for a certain input data. Random forest algorithm is one classification algorithm that adopts the idea of bagging and extends its concept to not only sample bagging but also features bagging. To apply a random forest algorithm, different random subsets of features are generated for each splitting step. Thus, the effect of some strong features would be eliminated.

There are three main hyper-parameters node size, the number of trees, and the number of features sampled. After setting these three parameters, we can start the training of the decision trees. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned.

2.2.2 eXtreme Gradient Boosting

Extreme Gradient Boosting (XGBoost) [8] is a scalable and distributed gradient-boosted machine learning framework under decision trees. It is the top machine learning package for regression, classification, and ranking tasks, and it supports parallel tree boosting.

To understand XGBoost, you must first understand the machine learning ideas and methods on which it is based: supervised machine learning, decision trees, ensemble learning, and gradient boosting.

Supervised machine learning use algorithms to train a model to detect patterns in a dataset containing labels and features and then employs the trained model to predict the labels on the elements of a new dataset.

Decision trees generate a model that predicts the label by analyzing true/false feature questions and calculating the least number of questions required to assess the probability of reaching a good decision. Decision trees can predict a category or a constant numeric value using classification or regression.

2.3 Recurrent Neural Network

Although the standard feed-forward neural network is a sophisticated deep learning technology that performs well in many domains, it cannot use prior data. Recurrent Neural Network (RNN) is a type of neural network that is derived from the feed-forward neural network [42] has the ability to reuse the saving information at the time of processing input values.

Figure 2.3 shows the typical structure of RNN model, with the input sequence, output sequence, and the hidden layers at time t are represented by x_t, o_t, and h_t

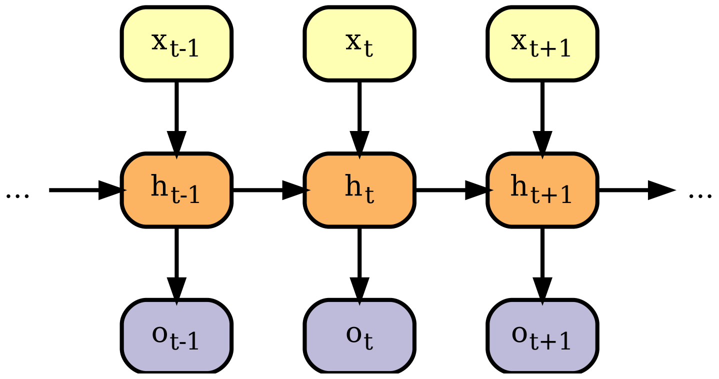

Figure 2.3: Structure of a typical Recurrent Neural Network.

respectively. At time t, the current hidden layers state h_t is calculated based on the current input sequence x_t and the last hidden layers h_{t-1}. After the calculation of h_t is finished, the output at the current time step o_t is generated and the hidden layers state h_t will get involved in the calculation at the next time step $t+1$. Unlike the normal neural network, where the neurons in the same layer of the hidden layers are independent of each other, RNN models usually allow the data to flow within the same layer. In other words, connections between neurons in the same layers or even self-connections are allowed generally allowed in the RNN-based model.

A major advantage of RNN models is that they are able to utilize the information from previous input and apply it at the current time. However, a major drawback of the vanilla RNN is the vanishing and exploding gradients [3]. The vanishing and exploding gradients are usually caused by the multiplication of multiple derivatives in the training process. Although there are several approaches [38] existing to address the vanishing and exploding gradients problem, in practice, it is still difficult for vanilla RNN to memorize and learn the features from long distance, which is described as a long-term dependencies problem. In order to deal with the long-term dependencies problem, many researchers have proposed multiple variants of RNN, such as the

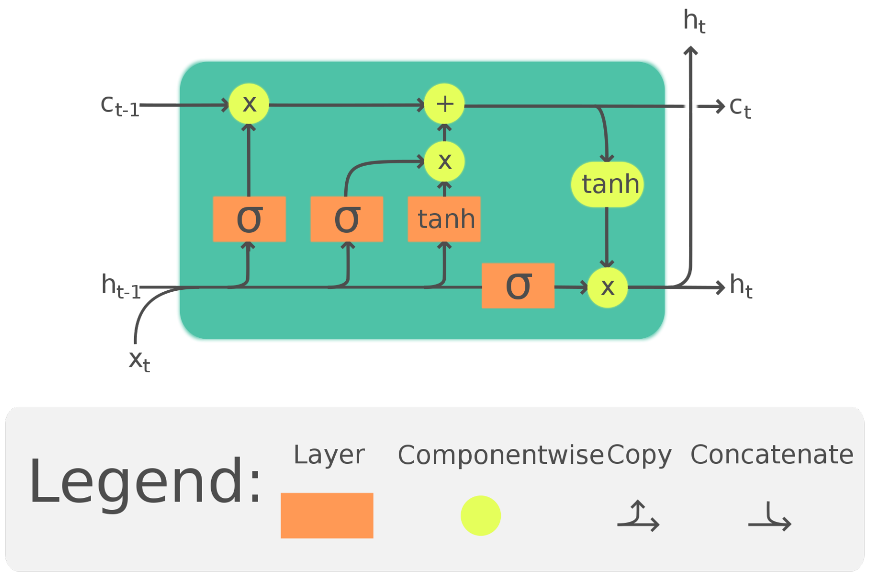

Figure 2.4: Long Short-Term Memory Cell[9]

Long Short-Term Memory (LSTM) [19], the Gated Recurrent Unit (GRU) [11], and the Clockwork RNN (CW-RNN) [23].

2.4 Long Short-Term Memory

The Long Short-Term Memory (LSTM) [19] is a famous variant of RNN. The main idea of the LSTM is the introduction of gate units, which are the structures that can determine whether to keep or discard the current information. A typical LSTM network consists of multiple memory cells, and each memory cell is formed by an input gate, a forget gate, and an output gate.

In LSTM, at time t, the state of a memory cell c_t is calculated based on the input x_t and the last hidden state h_{t-1}. The state of input gate, output gate, and forget gate at time t are represented as i_t, o_t, f_t, respectively. Here is a sample for LSTM cell in Figure 2.4.

Therefore, by using the σ to denote the sigmoid function; the operator $\odot$ to denote the element-wise multiplication, the LSTM transition functions are defined as following[50]:

$$
\begin{aligned}
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
q_t &= tanh(W_q \cdot [h_{t-1}, x_t] + b_q) \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot q_t \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}
\tag{2.2}
$$

When the output of a gate unit is close to 1, the information is more likely to be memorized. On the contrary, a returning value close to 0 from a gate unit means that the information should not be kept. The input gate i_t is the gate unit that controls how much information should be stored at this time. The forget gate f_t is responsible for determining to what extent the memory from the last time c_{t-1} should be kept at time t. The output gate o_t at time t is designed to be used in the computation of the output (hidden state) based on the memory cell state.

2.5 Graph Neural Network

A graph neural network (GNN) is a class of neural networks for processing data best represented by graph data structures. It is a very effective and easy way to do node-level, edge-level, and graph-level prediction tasks.

The most fundamental part of GNN is a Graph. In computer science, a graph is a data structure consisting of two components: nodes (vertices) and edges. A graph G can be defined as G = (V, E), where V is the set of nodes, and E are the edges between them.

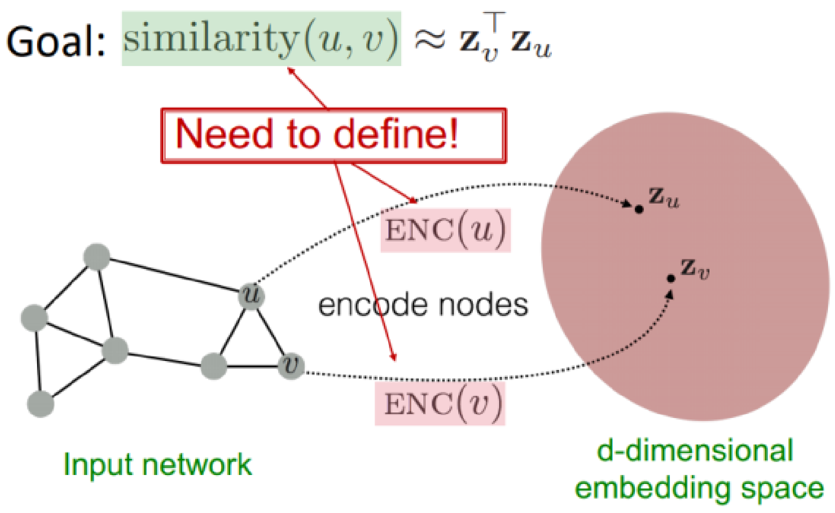

Figure 2.5: Basics of Deep Learning for graphs [27]

To represent graphs, we can use adjacency matrix:

$$W_{i,j} = \begin{cases} 1, & i \ and \ j \ share \ a \ link, \\ 0, & otherwise \end{cases} \tag{2.3}$$

where $W_{i,j}$ is the edge value from node i to j. If a graph has n nodes, adjacency matrix has a dimension of (n $\times$ n).

In GNN, our goal is to map nodes, so that similarity in the embedding space approximates similarity in the network. Let's define u and v as two nodes in a graph; x_u and x_v are two feature vectors. Now we'll define the encoder function $Enc(u)$ and $Enc(v)$, which convert the feature vectors to z_u and z_v. The whole process is shown in Figure 2.5 from [27].

To find the encoder function, we need locality (local network neighborhoods) information, aggregate information, and stacking multiple layers (computation) information. Locality information can be achieved by using a computational graph. After that, we start aggregating by using neural networks.

In Figure 2.6 from [27], Neural Networks are presented in light and dark grey boxes. The required aggregations should be order-invariant, like sum, average, and

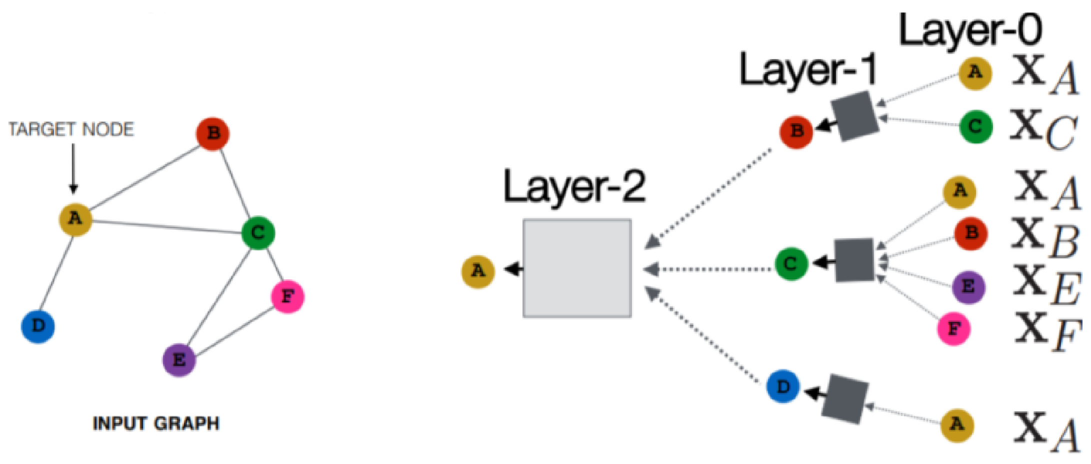

Figure 2.6: Neural Networks with layers [27]

maximum, as the result of permutation-invariant functions. Aggregations can be done prior to this attribute.

Based on the forward propagation rule, the information is transferred from layer to layer. For example, in Figure 2.6, every node has a feature vector. In Layer-0, the inputs are three groups of feature vectors, and the box will take the three groups of feature vectors, aggregate them, and then pass them on to the next layer.

After performing forward propagation in the computational graph layer by layer, we can train the model by defining a loss function on the embeddingS. There are two types of training:

- Unsupervised Training:

  Use simply the graph structure; nodes with comparable embeddings have similar embeddings. An unsupervised loss function can be a loss based on graph node proximity or random walks.

- Supervised Training:

  Train model for a supervised task such as node classification, normal or anomalous node.

The loss function measures the difference between the predicted output and the outcome generated by the machine learning model. We can obtain the gradients that

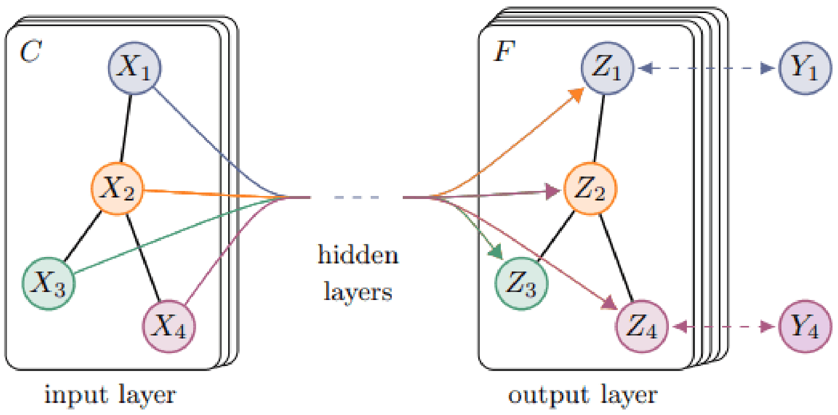

Figure 2.7: Graph Convolutional Network [22]

are utilized to update the weights and parameters from the loss function to get better training performance.

2.5.1 Graph Convolutional Networks

Graph Convolutional Networks (GCN)[22] is one of the most cited papers in the GNN literature and the most commonly used architecture in real-life applications. Because it can work directly on graphs and take advantage of their structural information, it is able to solve the problem of classifying nodes in a graph, where labels are only available for a small subset of nodes (semi-supervised learning). Figure 2.7 from [22] shows the semi-supervised classification with GCN.

There are some important definitions and equations in the GCN. First of all, it enforces self-connections by adding the identity matrix I to the adjacency matrix A.

$$\tilde{A} = A + I \tag{2.4}$$

Then, using the symmetric normalization of the Laplacian, we solve exploding/vanishing gradient with re-normalization. If H is the feature matrix, D is the degree matrix

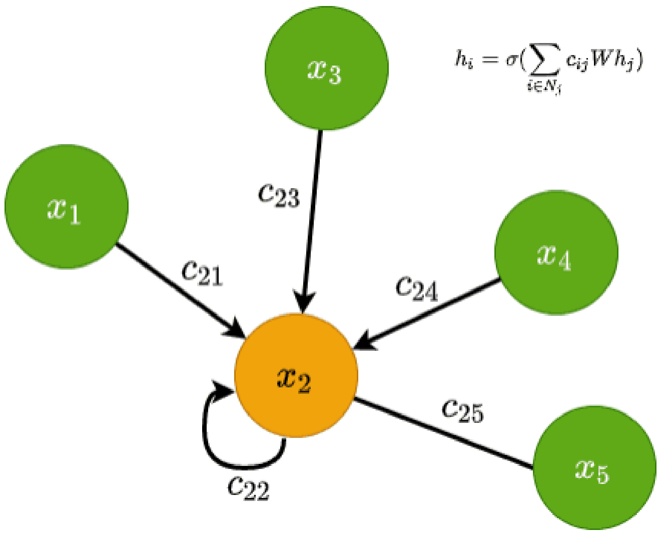

Figure 2.8: Graph Convolutional Network Node Rule [21]

of the graph and W the trainable weight matrix, the update rule for the GCN layer becomes the following:

$$H^{(l+1)} = \sigma(D^{-1/2}\tilde{A}D^{-1/2}H^{(l)}W^{(l)}) \tag{2.5}$$

From a node-wise perspective, the update rule can be written as:

$$h_i^{(l)} = \sigma(\sum_{i \in N_j} c_{ij} W h_j) \tag{2.6}$$

where $c_{ij} = \frac{1}{\sqrt{|N_i||N_j|}}$, and N_i and N_j are the sizes of the nodes' neighbourhoods. The visualization of nodes relationship is shown in Figure 2.8 from [21].

The loss function used in GCN is simply calculated by the cross-entropy error over all labeled examples, where Y_l is the set of node indices that have labels.

$$L = -\sum_{l \in Y_j} \sum_{i=1}^{h_i} Y_{li} ln h_{li} \tag{2.7}$$

2.5.2 Message Passing Neural Networks

Message Passing Neural Networks [16] utilize the notion of messages in GNNs. A message m_{ij} can be sent across edges i and j and is computed using a message function f_e. f_e is generally a small MLP and takes into consideration both the nodes' and edge's features. Mathematically, for two nodes i and j with edge features e_{ij}, we have the following:

$$m_{ij} = f_e(h_i, h_j, e_{ij}) \tag{2.8}$$

After that, all messages arriving at each node are aggregated using a permutation-invariant function, such as summation. The existing node features via f_v (another MLP) is then used to integrate the aggregated representation with the current node characteristics. These yielding an updated node feature vector by the following equation:

$$h_i = f_v(h_i, \sum_{j \in N_i} m_{ji}) \tag{2.9}$$

MPNN is a strong framework and one of the most general-purpose GNN designs. However, it occasionally experiences scaling challenges. As a result, MPNN must store and handle edge messages in addition to node characteristics. Because of that, it is only useful for small-sized graphs in reality.

2.6 Software-Defined Networking

SDN technology is a network management strategy that enables dynamic, programmatically efficient network design in order to increase network performance and mon-

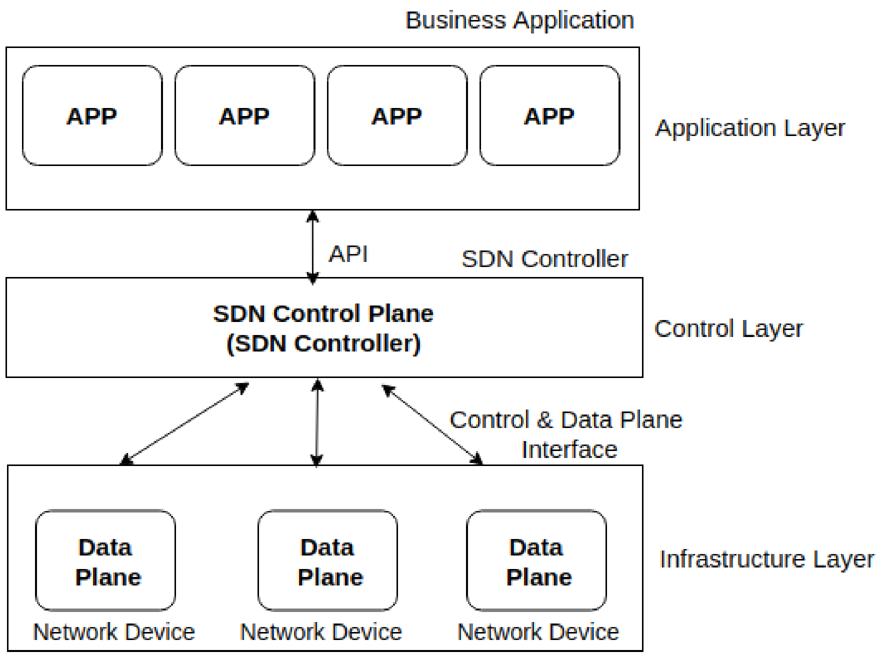

Figure 2.9: SDN Architecture

itoring, making it more similar to cloud computing than traditional network administration [4]. SDN is intended to address traditional networks' static design. SDN tries to consolidate network intelligence in a single network component by decoupling network packet forwarding (data plane) from routing (control plane). The control plane is made up of one or more controllers, which are considered the brain of the SDN network and contain all of the intelligence. Figure 2.9 shows a basic SDN architecture.

SDN separates the control and data planes. In SDN, network resources are governed by a logically centralized controller that also serves as the Networking Operating System (NOS). The network can be dynamically programmed by the SDN controller. Furthermore, by monitoring and collecting real-time network condition and configuration data, as well as packet and flow-granularity information, the centralized controller gets a global picture of the network. For the following reasons, using machine learning techniques to SDN is appropriate and efficient[47]:

- Recent advancements in computer technology like the Graphics Processing Unit (GPU) and Tensor Processing Unit (TPU) present an excellent chance to use promising machine learning approaches (e.g., deep neural networks) in the network area.
- The key to data-driven machine learning algorithms is datasets. The centralized SDN controller provides a global network perspective and can gather diverse network data, allowing machine learning methods to be applied more easily.
- Machine learning approaches may provide intelligence to the SDN controller by conducting data analysis, network optimization, and automated provision of network services based on real-time and historical network data.
- Because SDN is programmable, optimum network solutions (e.g., configuration and resource allocation) generated by machine learning algorithms may be performed in real-time on the network.

SDN also has certain challenges. One major challenge is the single point of failure. Since SDN uses the concept of centralization, the moment the controller, which is the brain of the network, goes down, the whole network goes down. The issue of a single point of failure is also a major security risk. Attackers can mainly focus on taking down the controller in order to bring the whole network down. Another challenge is scalability. As the network size increases, one controller will not be able to effectively handle the network responsibilities. More controllers are added as the network expands. Adding more controllers also raises the challenge of where to place the controllers. This is popularly termed the controller placement problem. Several approaches to dealing with the controller placement problem have been presented in [25] and [14]. Security in SDN networks has also garnered a lot of research contributions in the past years.

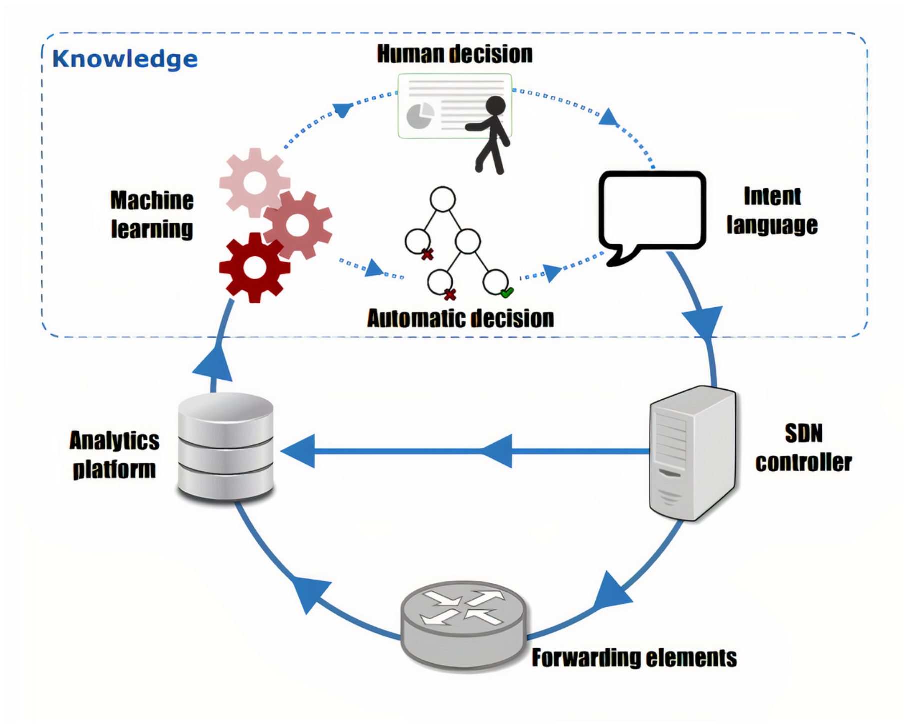

Figure 2.10: KDN operational loop [29]

2.6.1 Knowledge-Defined Networking

The authors in [29] combine the Knowledge Plane with SDN and refer to this new paradigm as Knowledge-Defined Networking (KDN). The Knowledge Plane (KP), as originally proposed by Clark [12], is redefined in this paper under the terms of SDN as follows: *the heart of the knowledge plane is its ability to integrate behavioral models and reasoning processes oriented to decision making into an SDN network.*

The KP in the KDN paradigm uses the control and management planes to gain a comprehensive view and control over the network. It is in charge of learning the network's behavior and, in certain situations, autonomously operating the network accordingly. Fundamentally, the KP analyses the network analytics gathered by the management plane, converts them into knowledge through ML and then uses that information to make choices (either automatically or through human intervention). The overall loop is shown in Figure 2.10.

From the paradigm of the KDN, we can find that SDN, network analytics and

Machine Learning can ultimately provide automated network control to improve and meet the network requirement. It also provide the deployment sample for SDN with machine learning and deep learning.

2.7 Summary

In this chapter, we introduced some background knowledge for our research, including concepts from deep learning, machine Learning, and software-defined networking where we predict the end-to-end delay.

In the following Chapter 3, we will introduce related SDN performance and efficiency improvements with machine learning methods in details.

Chapter 3

Related Works

For delay-sensitive applications, the measurement of Quality of Service (QoS) parameters such as delay, jitter, and throughput is fundamental to meeting QoS requirements. These network-centric metrics are often used to explain network performance by network operators. Therefore, accurate delay estimation is critical to meeting the QoS requirements of delay-sensitive applications. With SDN, statistics can be collected from switches on both a flow-by-flow and port-by-port basis. The prediction of QoS parameters can be made quite easily by traditional Machine Learning (ML) techniques. From the perspectives of traffic classification and QoS/QoE prediction, the authors in [47] discuss how machine learning algorithms have been applied to SDNs. They mentioned several capabilities of SDN that make it easier to apply machine learning techniques, including logically centralized control, a global view of the network, software-based traffic analysis, and dynamic updating of forwarding rules.

Also, in [30], the authors have proposed a Neural Networks(NNs) based delay estimator. They have experimented with various network parameters, such as topology, size, traffic intensity, and routing policies. They have proved that NNs could accurately model the average end-to-end delay of SDN.

The chapter is organized as follows. Section 3.1 reviews several traffic predictions in SDN based on typical machine learning algorithms and GNN methods. Section 3.2 introduces two cases using STGCN in traffic prediction.

3.1 Traffic Prediction in SDN with Machine Learning

The authors in [31] have reviewed some existing Machine Learning (ML), and Deep Learning (DL) approaches for traffic classification and traffic prediction in the SDN context. From their article, we find there are lots of traditional machine learning algorithms like Nearest Centroid (NC), Naive Bayes (NB), Decision Tree (DT), Support Vector Machine (SVM), etc. are in use for SDN prediction problems.

3.1.1 Gated Recurrent Unit Framework for 5G network

The authors in [44] have proposed a traffic prediction and forecasting model for a 5G network in an SDN environment using Gated Recurrent Unit (GRU). GRU is a recurrent neural network gating method introduced by [10]. The GRU functions similarly to a long short-term memory (LSTM) with a forget gate but with fewer parameters because it lacks an output gate.

Fusion learning is used between the data plane and control plane of the SDN environment is able to provide the prediction model with the exchange of model parameters of the SDN client models and its data distribution with a single communication. In the paper, they use fusion learning for long-term traffic prediction and build their forecasting model with diffusion convolution operations in GRU for capturing spatial and temporal dependencies of the features of the network traffic.

As time-series GRU captures only the temporal dependency, to handle dynamics of the network traffic and efficient capture of the traffic pattern, the spatial dependency must be captured. The diffusion convolution process contained in the GRU captures both spatial and temporal dependency of encoder-decoder architectural properties. The stochastic gradient-based planned sampling increases the prediction model's performance with the optimal decay rate.

Their proposal for the framework is tested with the simulated data on Abilene network topology with the RYU SDN controller. The experimental results exhibit improved accuracy in both local and global models of 87%—94%. Besides that, the authors compare LSTM predicted results in the same scenarios and conclude the Diffusion Convolutional Gated Recurrent Unit (DCGRU) has the best performance of all three methods.

3.1.2 Traffic Classification with Deep Learning Models

In [7], the authors have proposed an application-based offline and online traffic classification, with deep learning models, such as Convolutional Neural Network (CNN), Multilayer Perceptron (MLP), and Stacked Auto-Encoder (SAE), over a proposed SDN network testbed, shown in Figure 3.1. They use flow statistics data combined with Packet-in messages as the learning features of the deep learning models designed in the SDN controller. The SDN controller conducts the application-based traffic classification for each flow by establishing the match fields of the flow.

They design the deep learning models resigned in the SDN controller. The SDN controller establishes the match fields of the flow entry and sends them to the open virtual switch (OVS). It also extracts traffic statistics data from the OVS switch. The server IP addresses and transport port numbers of each flow plus the statistics data are designed to be the input features for the deep learning models.

There are three deep learning models in use. The first MLP deep learning model employs back-propagating supervised learning techniques in artificial neural networks. It consists of an input layer, three hidden layers followed by a dropout layer, and an output layer. The input layer consists of server IP addresses and transport port

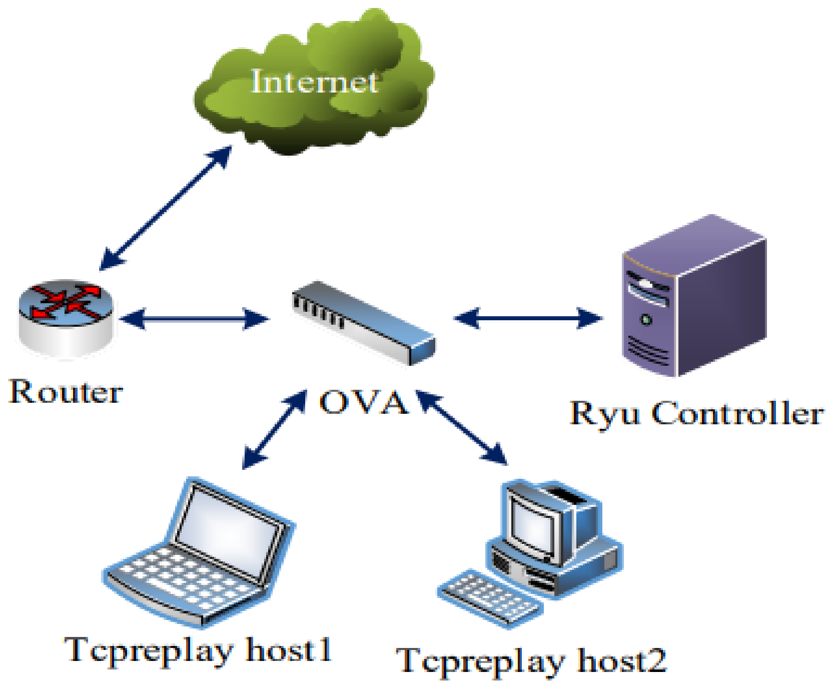

Figure 3.1: Network topology of the SDN testbed [7]

numbers; the dropout layer is used to prevent the overfitting issue; the output layer consists of neurons with a classifier for the MLP deep learning model.

The second model is SAE, containing one single encoding and decoding. It is used for dimension reduction or feature extraction.

The convolutional neural network (CNN) model, which is a common deep learning model used for classification, is the third deep learning model in deep learning training. The CNN model is composed of three convolution layers, three max-pooling layers, and one fully connected layer with the Relu function functioning as an activation function.

They check the accuracy, precision, recall and F1-score for each learning method in real-world network traffic classification, including multimedia, audio and video streaming.

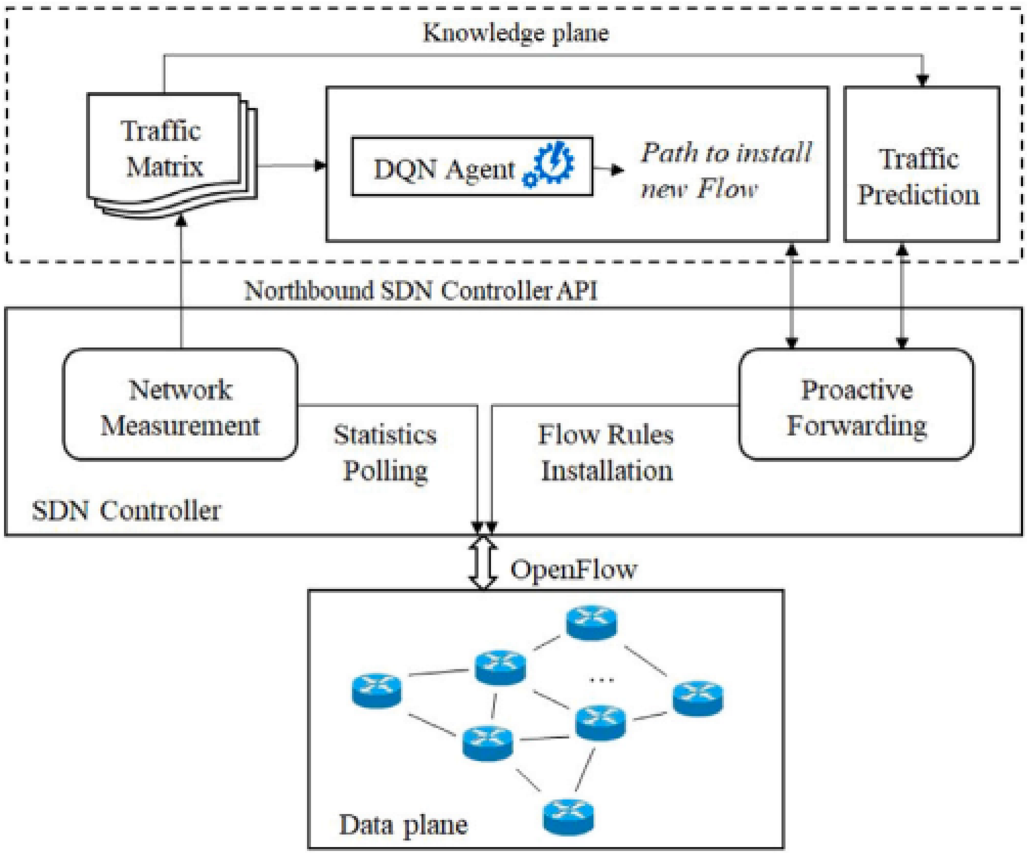

Figure 3.2: DTPRO Architecture [6]

3.1.3 Deep Q-Network and Traffic Prediction based Routing Optimization

In [6], the authors propose a dynamic traffic engineering scheme in SDNs, Deep Q-Network (DQN), and Traffic Prediction based Routing Optimization (DTPRO) for traffic and congestion prediction in providing better routing configurations for traditional routing algorithms in Figure 3.2. The architecture consists of four planes: Data Plane, Control Plane, Management Plane, and Knowledge Plane.

The Data Plane is made up of programmable forwarding devices that process and forward data packets. These devices lack inherent intelligence and rely on the control plane to build their forwarding tables and change their settings in accordance with the OpenFlow protocol.

The Control Plane is regarded as the brain of the SDN network, including the whole intelligence by centralizing network administration and global perspective in a specialized central controller. It consists of two major modules: Network Measurement and Proactive Forwarding. The Network Measurement module is divided into two sub-modules: Statistics, which gathers metrics like the number of packets and bytes per flow to calculate throughput, and Latency Measurement. The Proactive Forwarding module is in charge of selecting the best routing plan and network traffic forecast.

The Management Plane guarantees the network's proper operation and performance by gathering network measurements from the Control Plane, especially from the Network Measurement module, in order to give network analysis. The statistics gathered will be examined and forwarded to the Knowledge Plane.

Finally, the Knowledge Plane makes use of the Control and Management Planes by feeding data from the management plane into ML algorithms, which transform it into the form of knowledge. These planes precisely learn the network's behavior by processing the collected statistics, then extract the optimal paths, representing the knowledge, to route flows by deploying a DQN agent, and finally predict network congestion in the Traffic Prediction module using prediction methods (i.e., LSTM, ARIMA, Linear Regression (LR)).

Then they put those prediction methods in the DQN model, which dynamically determines the optimal policy mapping the set of states (traffic matrices) to the set of actions (changing the vector of link weights) to generate a good estimation accuracy. Finally, they combine DQN with Traffic Prediction and show that network latency, packet loss, and link utilization can be decreased. Moreover, they find that LSTM achieves a high estimation accuracy, which outperforms other traditional prediction methods, and decreases both end-to-end delay and packet loss.

3.1.4 Traffic Classification in SDN-IoT with Machine Learning methods

In [36], the paper proposes a Machine learning model that classifies traffic in SDN-IoT networks for traffic engineering. The classification process is compared with RF algorithm, decision tree algorithm, and the K-nearest neighbors' algorithm.

A deep learning route optimization model based on traffic classification is proposed. The model chooses the route that meets the QoS demands like latency of the identified traffic. They also compare the impact of two feature selection methods, Sequential Feature Selection (SFS) and Shapley additive explanations (SHAP), on the accuracies of the classifiers to reduce the number of features needed for classification.

At last, the authors find the Random forest classifier with SFS feature selection produces the best performance. After comparing with other research works, it is proved to attain accuracy and higher F1 score.

3.1.5 Traffic Prediction in SDNs with LSTM

In [26], the authors have proposed a time-series prediction framework using Long Short-Term Memory (LSTM) for accurately predicting future link throughputs in both SDN and legacy networks. Compared to the traditional long-time-scales prediction, authors analyze that LSTM has abilities to model aggregate network traffic at various traffic aggregation levels and short time scales, which can enable a short-term decision. The LSTM framework models the backbone network traffic at the link level and for various time-epochs and is found to be a good candidate for link-level network traffic modeling. A prediction framework is also presented that can be readily deployed to production SDN topologies without requiring any switch modifications since it uses the OpenFlow API port information to train the LSTM models.

Several variations of LSTM are also tested, including vanilla LSTM, delta LSTM (models the consecutive link throughput deltas), and multi-variate LSTM (models all the link throughput time series at once thus taking into account potential correlations). It can be concluded that LSTM can effectively model backbone network traffic at the link level and for various time-epochs after comparing with several ARIMA baseline models.

3.1.6 Delay Forecasting in SDN-IoT with NARX enabled RNN

The authors in [1] have focused on forecasting delays in Internet of Things (IoT) and tactile internet networks using a k-step prediction approach with nonlinear autoregressive with external input (NARX)-enabled recurrent neural network (RNN).

The RNN essentially has a memory that can predict time-dependent targets, and the NARX network, which can represent dynamic systems, has been used for predicting future information. The widely used NARX model yields promising results for time series issues with lagged input and output variables as well as prediction errors. Unlike traditional RNNs, the NARX network achieves excellent prediction performance for practically every nonlinear function while incurring minimal, or no computational costs [5].

The k-step-ahead modeling is a time series forecasting technique that predicts the subsequent output values using the previous and current input values. It uses different training algorithms to investigate which is the optimal performance generated based on these algorithms. For each step, the suitable ML model is determined using test information, matching several models to the dataset based on historical values.

At last, they use the MSE loss function, RMSE, and MAPE to measure prediction accuracy and conclude the advantages and disadvantages of different ML models in various k values.

3.1.7 End-to-end Delay Prediction in SDNs with NN and RF

In [24], the authors have provided an open simulation platform for generating new datasets with traffic generation and measurement models and have used two machine learning models, Random Forest (RF) and Neural Network (NN), for the prediction of end-to-end delay.

By considering three scenarios of increasing complexities (Traffic Only, Traffic and Capacity, and Traffic and Capacity with Delays), they show that end-to-end delay can be predicted based on traffic matrix samples. In this thesis, we use the work of [24] for traffic generation and comparison purposes.

3.1.8 Traffic Predictors in SDNs with GNN

Unlike CNNs and recurrent neural networks (RNNs), GNNs support inputs of non-Euclidean space data, such as network topologies, knowledge graphs, and molecular structures. Many researchers used GNN in graph structure input data.

The authors in [43] have leveraged a GNN model for network modeling and optimization named RouteNet. Based on GNN, RouteNet can learn and model graph-structured information, and as a result, the model is able to generalize complex relationships between arbitrary topologies, routing schemes, and traffic intensity. By comparing to the original scenario and queuing theory baseline, it is proved that RouteNet can produce accurate estimates produce predictions of Key Performance Indicators (KPI) such as per-source/destination mean delay and jitter in networks with low Mean Relative Error(MRE).

In [49], the authors also have proposed a GNN-based self-encoder model, called Gated Graph Auto Encoder Network (GGAE), for predicting network delay in SDN networks accurately. The model combines the advantages of self-encoders and gated cyclic neural units in neural network technology, which is based on the message passing

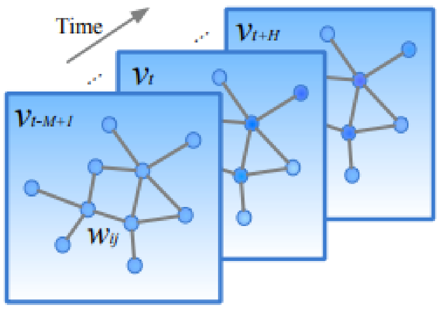

Figure 3.3: Graph-structured traffic data [48]
(Each v_t indicates a frame of current traffic status at time step t, which is recorded in a graph-structured data matrix.)

neural network (MPNN). Their paper makes improvements on RouteNet with lower absolute mean distance (MAD) values.

3.2 Spatio-Temporal Graph Convolutional Networks

Paper [48] has proposed a model STGCN to make traffic predictions in roadways. The STGCN model can learn both temporality and dependence of the input, and it achieves fewer error values on two real-world datasets compared to other state-of-the-art models, including Historical Average (HA), Linear Support Victor Regression (LSVR), Auto-Regressive Integrated Moving Average (ARIMA), and some machine learning methods. The sample graph-structured traffic data is shown in Figure 3.3. STGCN inside structure is shown in Figure 3.4.

Our thesis uses this model to extract spatial and temporal features of the end-to-end delay problem in computer networks.

The graph Fourier transform is applied on the eigenvalues of normalized graph Laplacian matrix as follows:

$$L = I - D^{-1/2} W D^{-1/2} \tag{3.1}$$

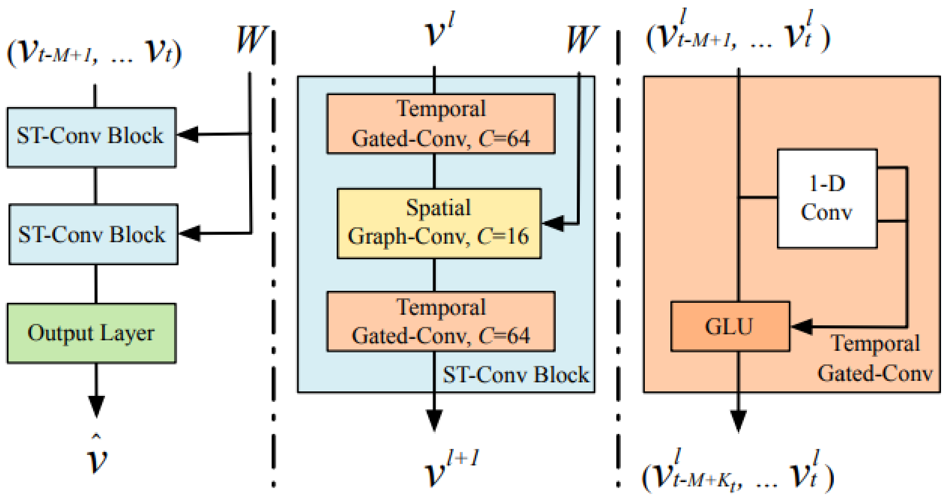

Figure 3.4: STGCN for Traffic Prediction [48]

where I is an N-dimensional identity matrix, D is the diagonal degree matrix with $D_{ii} = \sum_j W_{ij}$ and W is the graph adjacency matrix. Bing Yu[48] et al. introduced a graph convolution operator "$*g$" based on the conception of spectral graph convolution, as the multiplication of a signal x with a kernel Θ,

$$\Theta * gx = \Theta(L)x = \Theta(U\Lambda U^T)x = U\Theta(\Lambda)U^T x \tag{3.2}$$

In this equation, graph Fourier basis U is the matrix eigenvectors of the normalized graph Laplacian L ($L = U\Lambda U^T$). Λ is the diagonal matrix of eigenvalues of L, and $\Theta(\Lambda)$ is also a diagonal matrix.

In this model, two ST-Conv blocks are constructed jointly to process graph-structured in time series. Inside each block, there are two temporal layers and one spatial layer in the middle. Rectified Linear Units (ReLU) is also used to activate the rectified linear unit. For l block t^l, the output t^{l+1} is given by:

$$t^{l+1} = \Gamma_1^l * \tau ReLU(\Theta^l * g(\Gamma_0^l * \tau t^l)), \tag{3.3}$$

where Γ_0^l and Γ_1^l are the upper and lower temporal kernel within block l; Θ^l is the spectral kernel of graph convolution.

The authors originally used L2 Loss Function for model training in Equation 3.4. L2 Loss Function is usually used to minimize the error, which is the sum of all the squared differences between the true value and the predicted value.

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2 \tag{3.4}$$

We are using another loss function to train the model, which is Mean Squared Error (MSE). This will be introduced in the further in the Experiments and Results chapter.

Another paper [18] also applied STGCN to predict cascading delays throughout the railway network. The railway network is represented as a line graph, with nodes representing train lines and edges representing connecting stations. The STGCN model is then applied to this formulation, and its performance is compared to linear regression (LR) and multi-layer perceptron (MLP) models. In terms of MAE and RMSE for forecast intervals, they discover that the STGCN model beats the alternatives. Our thesis uses this model to extract spatial and temporal features of the end-to-end delay problem in computer networks.

3.3 Summary

We review related works on SDN prediction in this chapter. It includes traffic prediction, traffic classification, and delay forecasting with various ML and DL methods. STGCN model is also reviewed, and related research based on STGCN is introduced as well. This variety of SDN prediction proves that it is a concerned and useful area. Also, the STGCN model is verified better than some other delay prediction models in roadway and railway traffic areas.

Chapter 4

The Revised Model for SDN Delay Prediction

Machine Learning methods have shown promising results in predicting delays in transmission systems. Convolution techniques are commonly used in ML to capture spatial correlations in data. GNNs expand existing approaches to work with graph-structured data by utilizing graph convolutions to transmit information between nearby nodes and embed graphs into low-dimensional vectors.

We use the STGCN model for the end-to-end network delay prediction, because it showed great performance on leveraging node-wise features for the graph prediction [48]. The model architecture is summarized in Figure 4.1. It contains two stacked spatio-temporal convolutional blocks (ST-Conv blocks) followed by an output block. Inside each ST-Conv block, there are two temporal gated convolutions and one spatial graph convolution. There is an output layer to generate the prediction delay on the right.

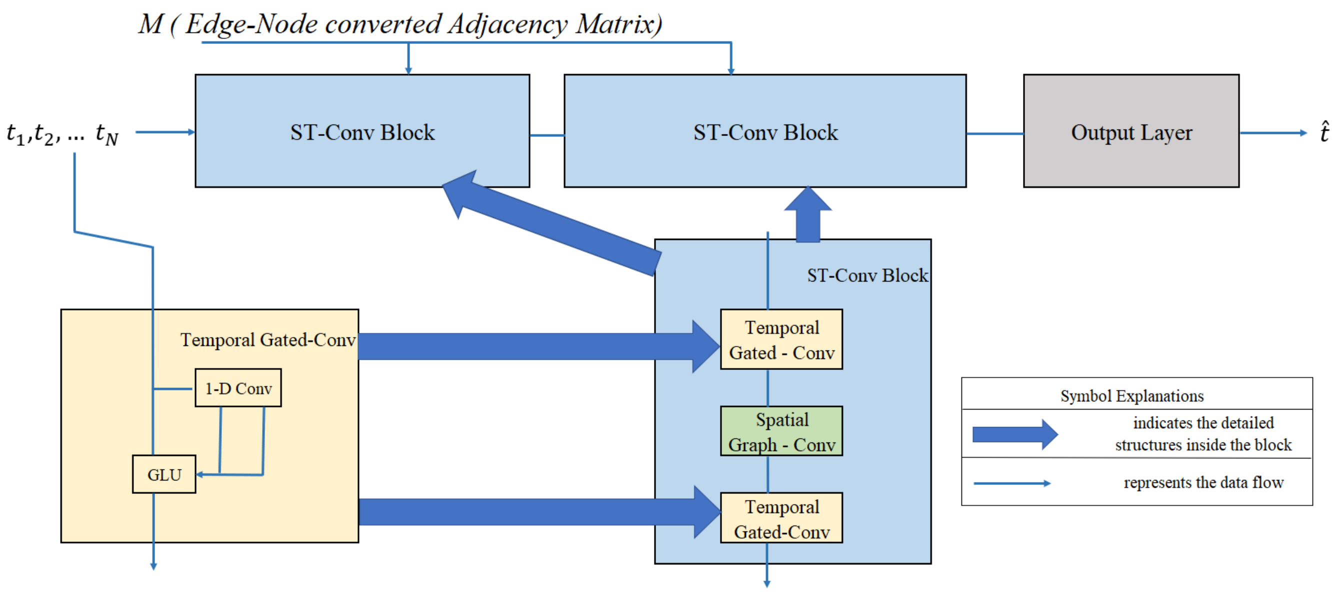

Figure 4.1: STGCN Architecture
(The framework includes two ST-Conv blocks and an output layer. Each ST-Conv block contains two temporal gated convolution layers and one spatial graph convolution layer. The input of delay is uniformly processed by ST-Conv blocks to explore spatial and temporal dependencies coherently. The spatial layer bridges two temporal layers used for fast spatial-state propagation from graph convolution through temporal convolutions. The output is generated prediction delay on the right.)

4.1 Delay Prediction Graphs

We formulate the prediction of delays on the network as a time series regression problem. After observing delays on links with connection relations between nodes, the previous time steps are used to predict the most likely delay at time step ($t_{n+N_{Future}}$). Based on this, the network delay problem can be stated as:

$$\hat{t} = \underset{t_{n+N_{Future}}}{argmax}\ logP(t_{n+N_{Future}}|(t_0, ..., t_n)) \tag{4.1}$$

where t_n is an observation vector of n link segments at time t and $\hat{t}$ is the model prediction at time t.

Therefore, the network can be defined on a graph $G(V, E, X)$, defined by the set of nodes V, edges E, and time-varying node features X. This graph can be represented by its adjacency matrix of M, defined as follows:

$$M_{i,j} = \begin{cases} 1, & \text{if link } i \text{ and } j \text{ share a router,} \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

4.2 Temporal Convolution with Gated Liner Unit

In Figure 4.1, two ST-Conv blocks are constructed jointly to process graph-structured in time series. Inside each block, there are two temporal layers and one spatial graph convolution layer in the middle.

Each temporal convolutional layer contains a 1-D causal convolution with a kernel followed by gated linear units (GLU) for non-linearity. Paper [34] has shown that a 1D convolution along the temporal dimension of data can be more effective than an RNN on shorter sequences while at the same time being quicker to train. Similar to the gating present in RNN models, the nonlinear activation provides a gating that determines the importance of past inputs on future predictions. The resulting temporal convolution is defined as:

$$\Gamma *_T Y = P \odot \sigma(Q) \tag{4.3}$$

where P and Q result from splitting the input of the temporal block along the channel dimension. The Equation 4.3 is then updated to Equation 3.3.

For each node in the graph, the temporal convolution explores k_t neighbors of input elements without padding, which leads to shortening the length of the sequences by $k_t - 1$ each time.

For the sake of clarity, Algorithm 1 presents the pseudo-code for the temporal convolution layer with GLU, where the inputs are: (i) $4DInputTensor$: a four-dimensional tensor where the first dimension is $batchSize$ 100, the second dimension is $inputChannel$ which is 1 in our case, the third dimension is $timeStep$ which is the number of previous timestamps used to predict the OD traffic matrix, and the fourth dimension is the $numVertex$, which is square of node numbers. (ii) $timeEmbeddingSize$: the kernel size used to extract features from the input data (iii) $channelOut$: the number of intervals for prediction. In this thesis, the $timeEmbeddingSize$ is set to 3. In addition, we aim to predict the OD traffic matrix in the next timestamp, and thus $channelOut$ is 1. Three convolutional layers are applied in this module, and a sigmoid layer is utilized to bring the non-linearity. The element-wise addition is then utilized between the transformed data X_p and the input X_{in}. Following that, the element-wise multiplication is used between the computed data from the previous step X_p and the transformed data X_q.

4.3 Convolution in the Spatial Dimension

The ST-Conv blocks of the STGCN architecture leverage graph convolutions to capture spatial relationships in the data. Spectral graph theory provides one method

Algorithm 1: Temporal Convolution with Gated Liner Unit

```
Input: $4DInputTensor$, $timeEmbeddingSize$, $channelOut$
Output: $4DOutputTensor$
 1: $batchSize, inputChannel, timeStep, numVertex = 4DInputTensor.Shape$ // initialize training shape
 2: $X \leftarrow$ copy the data from Data Preprocessing steps as ($4DInputTensor$)
 3: $X_{in} \leftarrow$ reserve the last $timeStep - channelOut$ delay data
 4: Let 1DConv() be a 1D conv-layer with kernal size = $timeEmbeddingSize$ and convolute on the time domain
 5: $X_c = \text{1DConv}(X)$
 6: $pChannel = channelOut$;
 7: $X_p \leftarrow$ pop the first $channelOut$ from $X_c$
 8: $X_p = \text{1DConv}(X_p)$ // update value after 1D conv-layer
 9: $X_q \leftarrow$ pop the last $channelOut$ from $X_c$
10: $X_q = \text{1DConv}(X_q)$ // update value after 1D conv-layer
11: $X_q \leftarrow \text{sigmoid}(X_q)$ // bring the non-linearity
12: $X_p = \text{elementWiseAdd}(X_p, X_{in})$
13: $4DOutputTensor = \text{elementWiseMultiply}(X_p, X_q)$
14: return $4DOutputTensor$
```

(i.e., the graph Fourier transform) for generalizing the convolution operation for graph-structured data. The analysis focuses on the eigenvalues of the normalized graph Laplacian matrix in Equation 3.1, and the graph convolution is introduced in Equation 3.2.

4.4 Loss function

We use mean squared error (MSE) loss to measure the performance of our model. Thus, the loss function of STGCN for delay prediction can be written as:

$$L(\hat{t}; M_\theta) = \frac{1}{n}\sum_{i=1}^{n} ||\hat{t}(t_0, ..., t_n, M_\theta) - t_{n+N_{Future}}||^2 \tag{4.4}$$

where M_θ are trainable model parameters, $t_{(n+1)}$ is the ground truth, and $\hat{t}$ denotes the model's prediction. In our experimentation, we use the OD average traffic matrix in the next timestamp as ground truth.

4.5 STGCN Approximation and Model Setting

To reduce the computation load of kernel Θ with graph Fourier transform, this step utilizes the Chebyshev polynomials approximation method [13]. Accordingly, the graph convolution in Equation 3.2 can be adjusted as Equation 4.5, where the computation cost can be reduced [13].

$$\Theta * gx = \Theta(L)x \approx \sum_{k=0}^{K-1} \Theta_k T_k(\hat{L})x \tag{4.5}$$

where $\hat{L} = 2L/\lambda_{max} - I_N$ was the scaled Laplacian matrix of L; $T_k(\hat{L}) \in R^{M\times M}$ referred to the Chebyshev polynomial of order k evaluated at $\hat{L}$; K denotes the kernel size of graph convolution; $\Theta \in R^K$ refers to the polynomial coefficient vector. With the approximation of kernel Θ, we generate the graph convolutions to capture the spatial and elevation features. In Figure 4.1, there are two ST-Conv Blocks in use. Assuming there are n nodes in the graph. The end-to-end pair will be n^2. The output shapes for both blocks are $[64, 4, n^2]$; after the output block, the data has the size of $[n^2, 1]$. ReLU is activated for the first two layers, and a sigmoid is used on the model output. For the STGCN model, we use a spatial kernel of size $k_s = 3$ and a temporal kernel of size $k_t = 3$. We train each test with 500 epochs and MSE loss.

4.6 Summary

We introduce the SDN delay prediction STGCN architecture in detail. The framework includes two ST-Conv blocks and an output layer. Each ST-Conv block contains two temporal gated convolution layers and one spatial graph convolution layer. For each element, we provide the algorithm and equations with explanations inside.

We also introduce our chosen STGCN approximation method, training and output data shapes, and modified model setting.

Chapter 5

Experiments and Results

5.1 Datasets

This section will introduce why we chose the following datasets and the simulated data generation settings. The first sets are 15-node networks group, which is used in [30]. The second sets are Abilene Network datasets used in [24]. We use three intensities to match and compare the results in their paper. We also generate more datasets with different parameter settings and make it a total of ten different network traffic intensities.

For a 15-node network with different topologies, the authors in [30] use the Omnet++ simulator (version 4.6) [45]; in each simulation, the average end-to-end delay during 16,000 units of time for all pairs of nodes is measured. Several network and traffic factors are changed to see how they impact modeling skills while learning network latency in distinct networks operating under different saturation and packet length regimes.

Specifically, the datasets in use are considered the following parameters:

- Topology:

  Three different network topologies in use: unidirectional ring in Section 5.1.1, star in Section 5.1.2, and scale-free in Section 5.1.3 networks. These three topologies present different connectivities which may affect the learning capabilities.

- Network size:

  The network size is 15 nodes, where all nodes are active transmitters and receivers.

- Traffic Distributions:

  Four different packet length distributions (in Section 5.1.4) are evaluated: deterministic (constant), uniform, binomial and Poisson using a fixed average packet length. In all the cases, the inter-arrival time is exponential.

Each simulation has 1000 Origin-Destination (OD) traffic matrices with a size of 15×15, which contains 225 link delay values. Each scenario we mentioned following contains 225k values. With 3 network topologies and 4 traffic distributions, we will have 12 combinations in total.

We also use another two datasets generated from RouteNet[43] using OMNeT++: one is 24-node GEANT2 [2] network in Fig 5.3, and another is 50-node in Fig 5.4.

5.1.1 Ring Network Dataset

A ring network is a network architecture in which each node links to precisely two other nodes, producing a single continuous signal channel through each node - a ring. Data is sent from node to node, with each node handling one packet at a time.

The advantages of a ring network include low incidence of collision, low cost, and is suitable for small businesses. However, disadvantages are obvious: one faulty node will bring the entire network down; it requires extensive preventative maintenance and monitoring; performance declines rapidly with each additional node; reorganizing the network requires a full system shutdown.

5.1.2 Star Network Dataset

In computer networks, a star network is an application of the spoke–hub distribution model. Every host in a star network is linked to a central hub [41]. In its most basic form, a single central hub serves as a conduit for message transmission. One of the most frequent computer network topologies is the star network. The hub and hosts, as well as the transmission lines connecting them, create a star-shaped graph. A star network's data flows via the hub before continuing on to its destination. The hub oversees and controls all network functions. It also serves as a data flow repeater.

Star topologies are the most often utilized since they let you administer the whole network from a single point: the central switch. As a result, if a node other than the central node fails, the network will remain operational. This adds a layer of security to star topologies against failures that aren't necessarily present in other topology configurations. Similarly, you may add additional machines without taking the network down, as you would with a ring topology. Star topologies require fewer connections than other topologies in terms of physical network construction. As a result, they are straightforward to set up and administer in the long run. The inherent simplicity of the network design makes troubleshooting considerably easier for administrators when dealing with network performance issues.

On the other side, though star topologies may be relatively safe from failure, if the central switch goes down, then the entire network will go down. The performance of the network is also tied to the central node's configurations and performance.

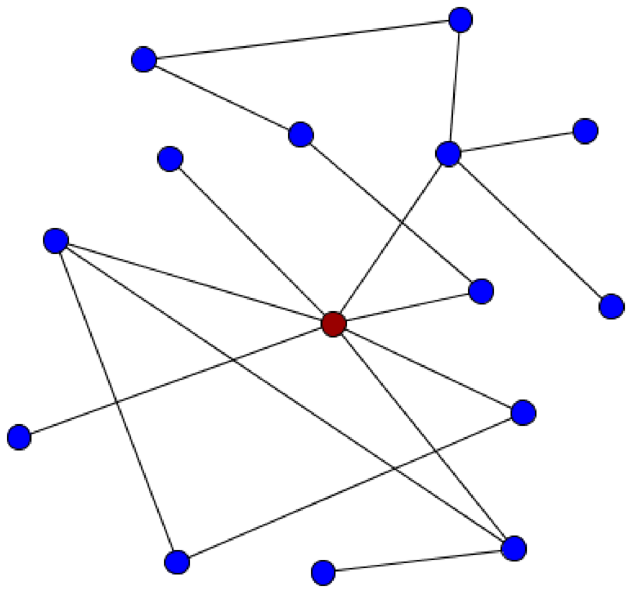

Figure 5.1: Scale-Free Network Topology Sample

5.1.3 Scale-Free Network Dataset

A scale-free network, in Figure 5.1, is a network whose degree distribution follows a power law, at least asymptotically. That is, the fraction $P(k)$ of nodes in the network having k connections to other nodes goes for large values of k as

$$P(k) \sim k^{-\gamma} \tag{5.1}$$

where γ is a parameter whose value is typically in the range $2 < \gamma < 3$ (where in the second moment (scale parameter) of $k^{-\gamma}$ is infinite but the first moment is finite) [33].

The topology of real networks is mostly unknown, and scale-free networks are quite important. Advantages of scale-free networks include robustness against accidental failures, and understanding the characteristics of the scale-free networks can prevent disasters (computer viruses, epidemics of diseases). On the other hand, the disadvantages of scale-free networks are that they are vulnerable to coordinated attacks and do not easily eradicate the viruses or diseases already in the system.

5.1.4 Dataset with Different Traffic Distributions

For each network topology from Section 5.1.1, Section 5.1.2 and Section 5.1.3, we evaluate four different packet length distributions: deterministic (constant), uniform, binomial and Poisson using a fixed average packet length. In all the cases, the inter-arrival time is exponential. Different probability distributions give the probabilities of occurrence of different possible outcomes for traffic datasets; that is why we choose four classical distributions[28].

5.1.5 Abilene Network Dataset

We use the dataset which was simulated by Krasniqi *et al.* [24]. The authors used the NS-3 simulator [40] to generate network traffic. They simulated a high-performance backbone network topology, Abilene [35], which has 12 nodes and 15 links, illustrated by Figure 5.2.

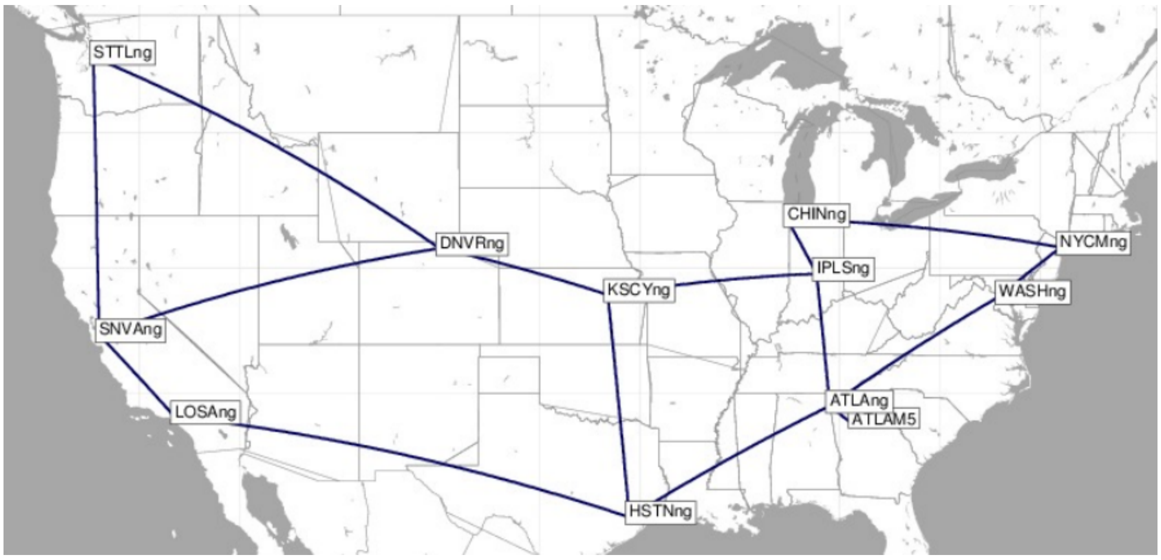

Figure 5.2: Abilene topology [35]

Dijkstra's routing algorithm is used in the NS-3 network simulator to define the shortest path between nodes. This simulator enables users to generate and monitor network traffic with customized characteristics and topology. Network topology and routing policy are fixed during the simulation, but there are three hyper-parameters which has a strong impact on the delay: traffic intensity, link capacity and link propagation delays. So we used three different datasets from their paper [24] with the following scenarios:

- Traffic Only (TO): fixed capacity and propagation delay;
- Traffic and Capacity (TnC): fixed propagation delay and variable capacity;
- Traffic and Capacity with Delays (TnCwD): traffic intensity, link capacity and link propagation delay parameters are variable.

In the simulation, nodes in the network are instructed to generate traffic toward all other nodes. Link capacities are generated randomly according to a truncated Gaussian distribution in the range [10Mbps, 200Mbps]; Link propagation delays are generated following a uniform distribution in the range [10ms, 100ms]. To control the traffic intensity, the authors in [24] specified a value r_k based on link capacities with the following equation:

$$r_k = \phi + (k \times 7 - 20) \times 50\text{Kbps} \tag{5.2}$$

In this equation, ϕ represents the fully utilized link capacity normalized by the maximum number of flows passing through it. Krasniqi *et al.*[24] defined the value k for simulating different levels of the network traffic intensity, where $k = 1$ refers to the low intensity, $k = 7$ to the medium intensity and $k = 10$ to the high intensity. We run the tests with k values from $[1, 10]$ in the TnCwD scenario and compare them with three specific intensities in all scenarios. Each simulation has 500 OD traffic matrices with a size of 12×12, which contains 144 link delay values. Each scenario we mentioned before contains $720k$ values.

5.1.6 GEANT2 and 50-node Networks Dataset

In the dataset, each simulation contains 1000 OD traffic matrices, GEANT2 and 50-node networks have sizes of 24×24 and 50×50, which contain 576 and 2500 link delay values for each matrix accordingly. On top of that, each scenario we mentioned before contains 10 simulations using for average performance evaluation, and there are $5,760,000$ and $25,000,000$ values for two topologies.

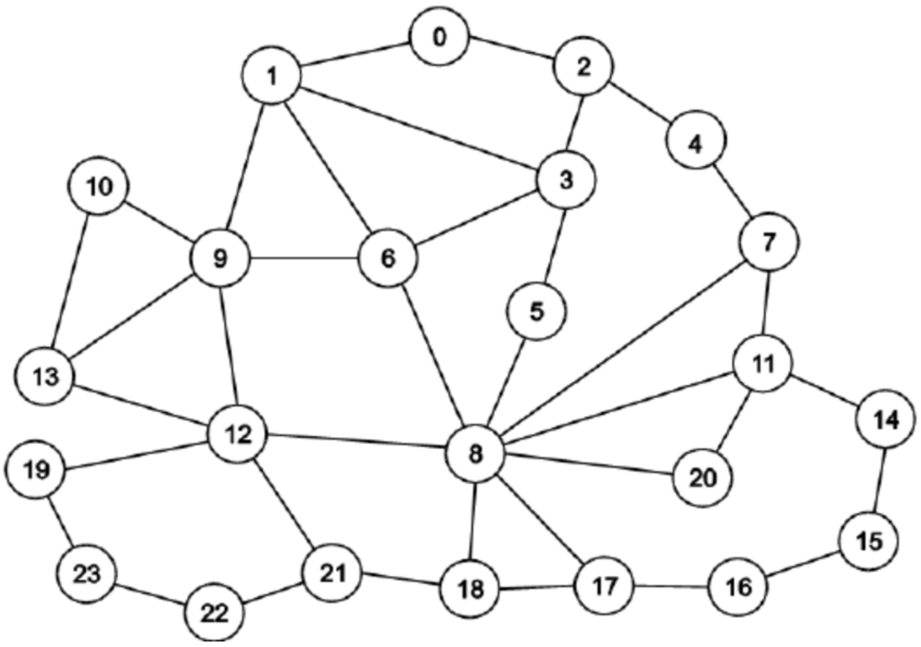

Figure 5.3: GEANT2 Topology

5.2 Data Preprocessing

We use datasets exploring three different network topologies: unidirectional ring, star, and scale-free networks with 15 nodes. The datasets setting will be introduced in Section 5.1. These datasets are the most complicated situation in [30]. In their paper, they prove NN can be used in SDN delay prediction. We will evaluate the performance of STGCN on the same datasets.

Another dataset used in this thesis has been simulated for the Abilene network us-

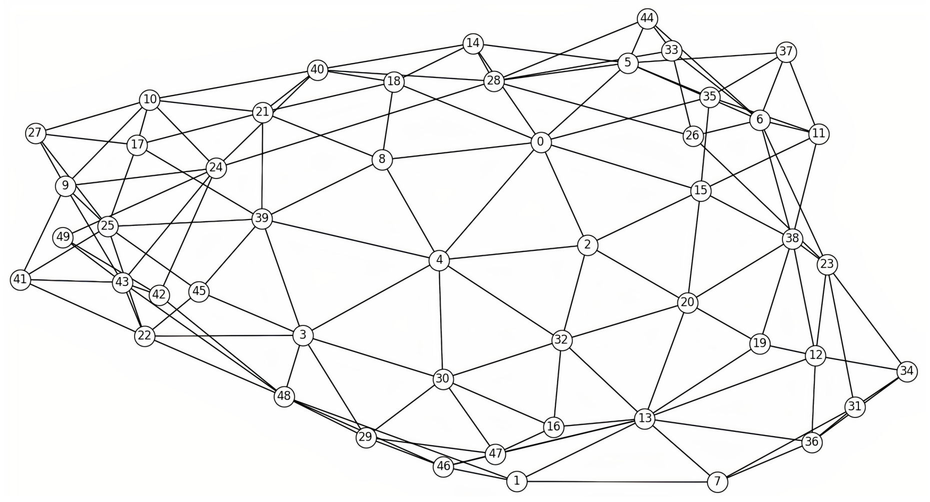

Figure 5.4: 50-node Topology

ing the simulation platform used by Krasniqi *et al.* [24]. Abilene is a high-performance backbone that provides network services to hundreds of member institutions in the United States and abroad. Fiber optic linkages are used to establish connectivity at the physical level for Abilene, with fifteen backbone links connecting routers (i.e., core nodes) in eleven major U.S. cities [17].

First of all, we need to find the adjacency matrix to use in STGCN with different network topologies. Taking the Abilene network as an example, we construct a line graph of the network of all 12 stations. For end-to-end delay prediction, we build the graph as fully connected with the shortest path first (SPF) algorithm. To generate the adjacency matrix for the end-to-end situation, we convert station links as nodes and put 12 backbone routers in between as edges. This node-edge conversion has performed well in delay prediction in railway network used in [18]. And we define the adjacency matrix, M in Equation 4.2.

Then, we use the NetworkX library to create a node-link transfer and also use the library to compute the 144×144 adjacency matrix from the new graph (It will be 225×225 sized adjacency matrix if using a 15-node network). The graph preprocessing process is illustrated in Algorithm 2, where $NetworkTopology$ is the network topology G and the *edgeGraph* is the line graph G'. The *getEdgeGraph* is the NetworkX library used to convert G to G'.

Following graph preprocessing, we preprocess data in order to feed them into the STGCN model. The data preprocessing is described in Algorithm 3, where the inputs are: (i) *end2endDelayTable*: Origin-Destination (OD) traffic matrixes collected from the SDN in each timestamp. (ii) *windowLength*: the number of samples we used to average the OD traffic matrixes. (iii) *batchSize*: the number of samples processed before updating the STGCN model parameters. (iv) *stride*: the number of timestamps of movement over the dataset. In this thesis, each OD traffic matrix is sampled every 0.1s. As in the paper [24], we average OD traffic matrices over a window size of 50, and thus *windowLength* is 50. The *batchSize* is set to 100. In addition, the window is moved to the next OD traffic matrix each time, and thus *stride* is 1. We use the previous ten timestamps of OD traffic matrixes to make predictions, and thus *inputChannel* is 10. Also, *numVertex* is the number of nodes in our graph $G\prime$, which is 144. We transform the OD traffic matrixes into four-dimensional tensors and feed a batch of them into the STGCN model each time.

Algorithm 2: Graph Preprocessing

```
Input: $NetworkTopology$
Output: $edgeGraph$
1: $denseG$ = getCompleteGraph($NetworkTopology$)
   // collect graph nodes information
2: $edgeGraph$ = getEdgeGraph($denseG$)
   // transfer nodes to edges
3: return $edgeGraph$
```

For better understanding, data preprocessing can be explained in another way. Given a network topology, we consider it as a directed graph $G = (N, E)$ where graph G contains K nodes and H edges, denoted as $N = \{n_1, n_2, ..., n_K\}$ and $E = \{e_1, e_2, ..., e_H\}$, respectively. We then convert the graph into a complete directed graph $G\prime = (N, E\prime)$, where $G\prime$ contains the same set of nodes as G and K^2 edges, i.e. $E\prime = \{e_{1,1}, e_{1,2}, ..., e_{K,K-1}, e_{K,K}\}$. We can observe that $e_{i,j} \in E\prime, \forall i \in N, j \in N$ and $i \neq j$.

Algorithm 3: Data Preprocessing

```
Input: end2endDelayTable, windowLength, batchSize, stride
Output: 4DInputTensor
 1: $X$ = 2DTensor,
    $tableLen = end2endDelayTable$.length
    // initialize
 2: for $i = 0;\ i < tableLen - windowLength;\ i = i + stride$ do
 3:     $windowX \leftarrow$ pop the first $windowLength$ row from $end2endDelayTable$
 4:     $windowX \leftarrow$ compute average delay among each column of $windowX$
 5:     extend $X$ with $windowX$
 6: end for
 7: $4DInputTensor$ = Tensor([$batchSize$, $inputChannel$, $timeStep$, $numVertex$])
 8: $batchNum = (tableLen - windowLength) \div stride + 1$
 9: for $i = 0;\ i < batchNum - 1;\ i = i + 1$ do
10:     $batchX \leftarrow$ pop the first $batchSize$ row from $X$
11:     $batchX \leftarrow$ expandDimension($batchX$)
12:     append $batchX$ to $4DInputTensor$
13: end for
14: return $4DInputTensor$
```

Following that, we convert $G\prime$ into a line graph $\hat{G\prime} = (\hat{N}, \hat{E\prime})$ containing K^2 nodes and L edges, where $\hat{N} = \{\hat{n_1}, \hat{n_2}..., \hat{n_{K^2}}\}$. In order to convert $G\prime$ to $\hat{G\prime}$, each edge in $E\prime$ is converted to a node in $\hat{N}$ and $\hat{e_{i,j}} \in \hat{E\prime} \iff e_i$ and e_j share a node n_v, where $e_i \in E\prime, e_j \in E\prime, n_v \in N$.

With $\hat{G\prime}$, each node $\hat{n_i}$ is corresponding to a directed edge in the graph $G\prime$. Instead of appending end-to-end link delay to graph $G\prime$, we attach a delay into each node in $\hat{G\prime}$. This way, we convert the link-level prediction problem into a node-level prediction problem. It allows us to apply GNN in node feature prediction. It is worth noting that the line graph conversion technique achieved great performance in railway delay prediction, as suggested in paper [18].

With graph $\hat{G\prime}$, we compute its adjacency matrix, M with a size of $K^2 \times K^2$, which is defined as follows:

$$M_{i,j} = \begin{cases} 1, & \text{if } \hat{n_i} \text{ and } \hat{n_j} \text{ is connected,} \\ 0, & \text{otherwise} \end{cases} \tag{5.3}$$

After the graph pre-processing step, we pre-process data before feeding them into the STGCN model [48]. Our Origin-Destination (OD) traffic matrix has a uniform sampling rate of ten times per second. We pre-process data according to paper [24], where the first step is to average every 50 OD traffic matrices, i.e., all OD traffic matrices within 5 seconds. Every time we calculate the mean traffic matrices, there are 49 records overlapping with the previous step. We consider the OD traffic matrices prediction a time-series problem, and thus we use ten recent OD traffic matrices in previous time steps to predict the OD traffic matrix in the fifth future time step. Finally, we transform a batch of averaged OD traffic matrices into tensors and feed them into the STGCN model.

5.3 Results and Comparison

5.3.1 Simulation Setup

This thesis develops and implements the STGCN SDN delay prediction model based on PyTorch deep learning framework. All training sets are using NVIDIA GeForce RTX 3070 GPU with memory up to 8GB. and Intel(R) Core(TM) i7-9700K @ 3.60GHz for CPU.

The maximum training epoch is 500. The initial learning rate R is 0.001. All delays are normalized and the z-score method is used to train the model in the STGCN. For each test case, we run ten datasets and get the average for the error evaluation parameters.

Finally, we implement a uniformly sampled 70%/15%/15% split of the data for training, validation and testing purposes, respectively.

5.3.2 Performance Metrics

In this section, we introduce three metrics for evaluating training performances. The three metrics include Root Mean Square Error (RMSE), Mean Absolute Error (MAE) and Weighted Mean Absolute Percentage Error (WMAPE). Accuracy is leveraged as our prediction recognition standard.

RMSE is the criteria used to evaluate all the models with different intensities.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(t_n - \hat{t}_n)^2}{n}} \tag{5.4}$$

where t_n is the true value, $\hat{t}_n$ is the forecast value, and n is total number of values in the test set. The RMSE is a popular method used by statisticians to understand how good is forecast.

We also consider MAE and WMAPE as the evaluation metrics, which are shown in the following equations:

$$MAE = \frac{\sum_{i=1}^{n}|t_n - \hat{t}_n|}{n} \tag{5.5}$$

$$WMAPE = \frac{\sum_{i=1}^{n}|t_n - \hat{t}_n|}{\sum_{i=1}^{n}|t_n|} \tag{5.6}$$

The lower the MAE value, the better the model is; a value of zero means there is no error in the forecast. In other words, when comparing multiple models, the model with the lowest MAE is considered better. RMSE can also be compared to MAE to determine whether the forecast contains large but infrequent errors. The larger the difference between RMSE and MAE, the more inconsistent the error size. The WMAPE metric is helpful for low volume data, where each observation has a different priority when evaluating forecasting models. The lower the value of WMAPE, the better the performance of the model. Observations with higher priority have a higher weight value. The larger the error in high priority forecast values, the larger the WMAPE value is.

In Section 5.3.4, we also compare with the average predictor baseline, which uses the average as a prediction for the future period. After comparing with our STGCN prediction, we can evaluate the performance of different machine learning models.

5.3.3 Results for 15-Node Networks

In Tables 5.1, 5.2 and Figure 5.5, we show that, in all cases, the error is pretty small. Also, the training is very effective. Compared with the work of [30], their ML method takes a maximum training epoch of $7,500,000$, while our setting is only 500. This will reduce the training time.

Table 5.1: MAE, RMSE and WMAPE for 15-Node Ring Network

	Binomial Distr.	Deterministic Distr.	Poisson Distr.	Uniform Distr.
MAE	0.000853	0.000749	0.000874	0.000807
RMSE	0.001137	0.000986	0.001149	0.001084
WMAPE	0.00615327	0.0054008	0.00630066	0.00583009

Table 5.2: MAE, RMSE and WMAPE for 15-Node Scale-Free Network

	Binomial Distr.	Deterministic Distr.	Poisson Distr.	Uniform Distr.
MAE	0.004679	0.004496	0.004552	0.004441
RMSE	0.006195	0.005789	0.006103	0.005885
WMAPE	0.01417278	0.01357348	0.01377185	0.01350639

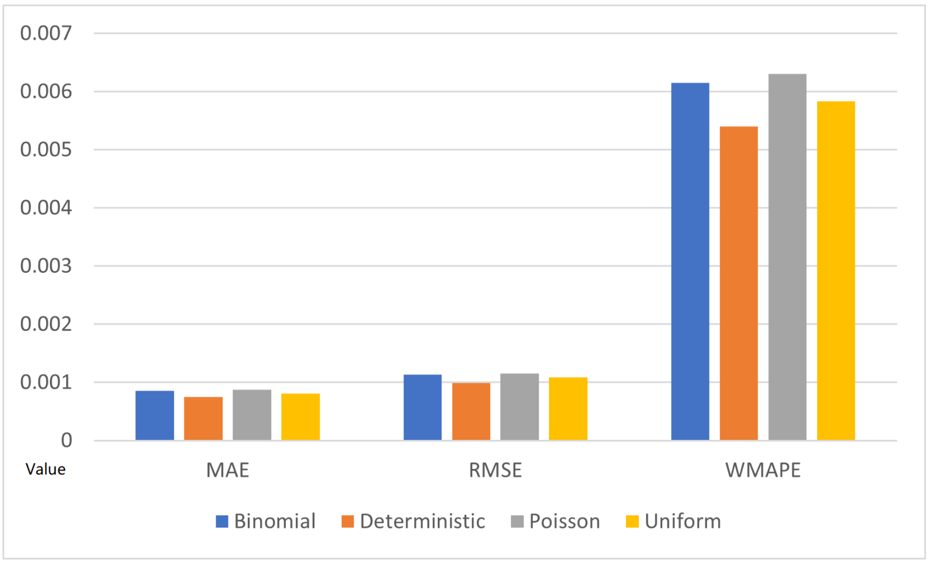

Figure 5.5: MAE, RMSE and WMAPE for 15-Node Star Network

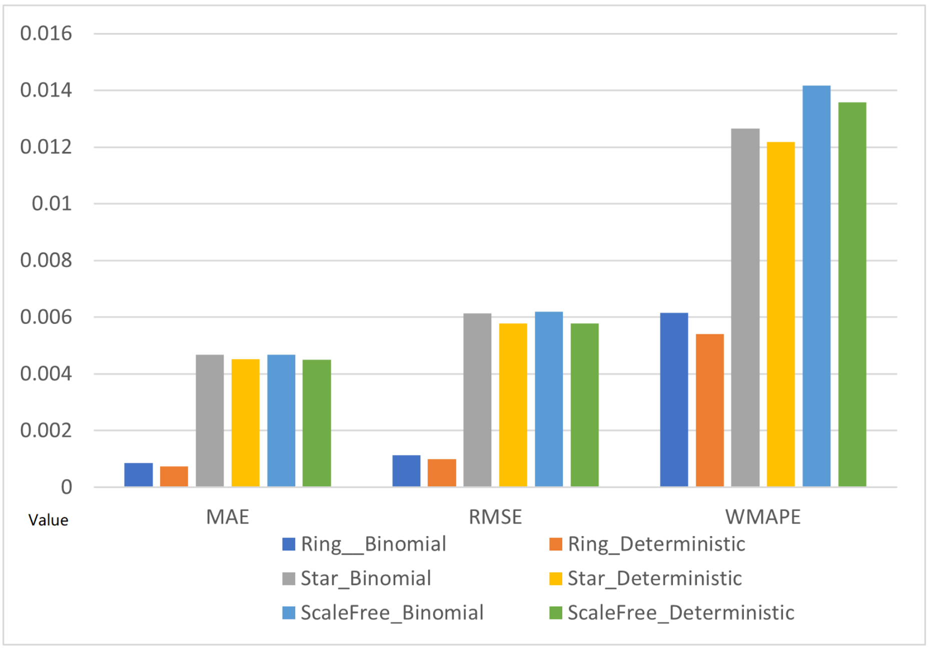

Figure 5.6: MAE, RMSE and WMAPE for 15-Node Networks with Distributions

(Binomial and Deterministic Distributions)

For each network topology, different distributions result in various training errors. For example, in Star Network, shown in Figure 5.5, deterministic distribution outperforms other distributions. Uniform distributed network datasets also perform well in the second smallest RMSE values in all three topologies. Based on RMSE, deterministic and uniform distributions are 4.7% to 14.2% smaller than binomial and Poisson distributions, respectively.

MAE and WMAPE have similar results to RMSE, where deterministic and uniform distributions are smaller than binomial and Poisson distributions in Ring and Scale-Free Network. However, in Star Network, MAE and WMAPE of Poisson distribution are the smallest of all the distributions. In this topology, MAE and WMAPE of deterministic, Poisson and uniform distributions are almost the same, with a difference of about 1%. Binomial distribution has the largest error (about 5% more than others).

In Figure 5.6, we compare three network topologies based on binomial and deterministic distributions. It can be concluded that the ring network has the lowest error values compared with other networks. With respect to RMSE, Ring is only 18.6% whereas it is 18.4% for Star and Scale-Free networks. This shows that STGCN model has better prediction in Ring topology than Star and Scale-Free topologies.

5.3.4 Results and Comparison for Abilene Network

We use the dataset introduced in Section 5.1.5. As mentioned, k is defined in Equation 5.2 from [24], which represents different levels of the network traffic intensity. We want to change the parameter values to find the relation between traffic intensities and STGCN predicting error. The scenario in used is TnCwD, where traffic intensity k is changed from 1 to 10, link capacity follows truncated Gaussian distribution and link propagation delay parameters follow uniform distribution.

Table 5.3: MAE, RMSE and WMAPE for Abilene Network

k Value	MAE	RMSE	WMAPE
1	0.00095525	0.0037815	0.007147533
2	0.0004275	0.001753	0.00332281
3	0.001215	0.003745	0.00819496
4	0.001556	0.005711	0.00912998
5	0.001186	0.004337	0.009106685
6	0.0012125	0.0043175	0.009278315
7	0.00111275	0.00404125	0.008062463
8	0.00197	0.006732	0.01164667
9	0.002072	0.007195	0.01677804
10	0.00230275	0.008101	0.01748078

The result is shown in Table 5.3. We can conclude that, as traffic intensities increase, STGCN learning error increases. This is because the complexity of the network and difficulties of the prediction are increasing.

More specifically, for MAE, it has a general slope of 0.00013. When k is in the range $[4, 7]$, it performs a stable error value. After $k = 7$, it increases rapidly. For RMSE, it is obvious there are two decreases in $k = 2$ and $k = 5$, but the relationship between traffic intensities and RMSE values is a highly positive correlation with the overall slope of 0.0061.

The visualization is shown in Figure 5.7. From the figure, WMAPE values have a significant overall slope of around 0.01. And similar to the other two error values, the error in medium intensity with $k = 3$ to $k = 7$ is almost the same.

In the next section, we will compare STGCN model in three different intensities with other ML and baseline methods.

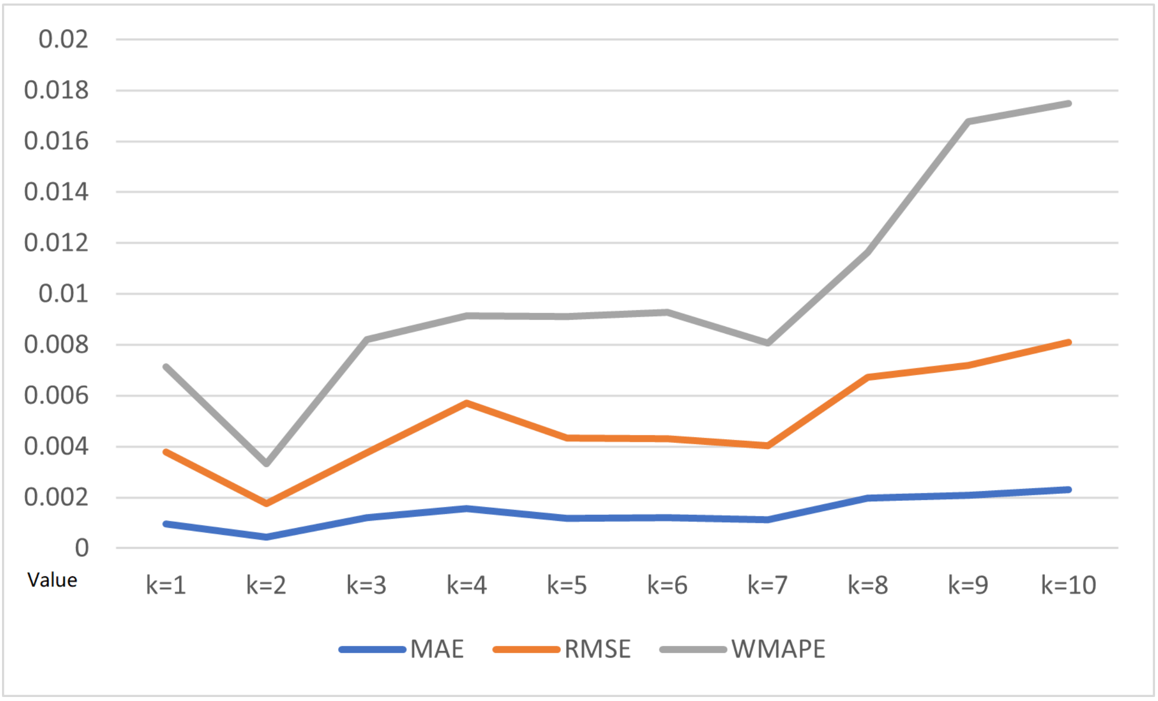

Figure 5.7: MAE, RMSE and WMAPE for Abilene Network in TnCwD scenario

We also use two other ML training methods: RF and NN from Krasniqi *et al.*[24] for comparison.

Three traffic intensities are low, medium and high, corresponding to $k = 1$, $k = 7$ and $k = 10$, respectively. Three scenarios are Traffic Only (TO), Traffic and Capacity (TnC), and Traffic and Capacity with Delays (TnCwD). For stable prediction and comparison purposes, we run our experiment 10 times for each scenario and each intensity. We consider the average end-to-end delays of all OD traffic matrices as the baseline prediction, which is named the baseline predictor. We perform evaluations among the baseline predictor, various ML-based strategies and our proposed GNN-based delay prediction methodology.

Table 5.4 shows the result of the TO scenario. We can observe that almost all the ML-based methods achieve $20\% - 80\%$ less RMSE compared to the baseline predictor. At best, our GNN-based delay prediction method performs 34% better than the baseline predictor. In this case, RF and NN models give the best prediction performance among all three traffic intensities. It is worth noting that the TO scenario is too ideal where it only considers fixed link capacity and propagation delay, and therefore, it is far from a realistic network.

Table 5.4: RMSE of baseline, RF, NN and GNN-based approaches in three traffic intensities of the TO scenario

Method	Low Intensity	Medium Intensity	High Intensity
Baseline Predictor	0.004	0.004	0.004
RF AVE [24]	0.000257	0.00053	0.0011
NN AVE [24]	0.00026	0.00062	0.0007
GNN-based AVE	0.001395	0.003159	0.005442

Table 5.5 shows that the three ML-based methods have similar results in the TnC scenario. Most of these models improve baseline predictor by $87\% - 95\%$ less RMSE. Among all network intensities, NN achieves the best delay prediction. However, in the TnC scenario, the link capacity is variable, but the propagation delay is fixed. It is unrealistic for a network to have a fixed propagation delay because weather,

temperature, and other factors can easily affect the propagation delay.

Table 5.5: RMSE of baseline, RF, NN, and GNN-based approaches in three traffic intensities of the TnC scenario

Method	Low Intensity	Medium Intensity	High Intensity
Baseline Predictor	0.0171	0.0171	0.0171
RF AVE [24]	0.00106	0.0016	0.0025
NN AVE [24]	0.000943	0.0014	0.0022
GNN-based AVE	0.002761	0.005333	0.013776

Figure 5.8 shows the baseline, RF, NN and GNN-based approach performances in the tree traffic intensities of the TnCwD scenario. TnCwD is the most complicated scenario, and it is similar to the real-world SDN environment. In this case, the network link capacities are sampled from a Gaussian distribution, and the link propagation delays are sampled from a uniform distribution. In this closest realistic scenario, our proposed GNN-based approach achieves the best prediction than the other three methods. Remarkably, our GNN-based approach achieves $85 - 90\%$ less RMSE than RF and NN in low and medium intensities. Furthermore, our GNN-based method performs 68.5% and 78.7% better than RF and NN in high intensities, respectively. Compared with the baseline predictor, RF and NN only improve the delay prediction by around $40 - 60\%$, while our GNN-based approach performs $80 - 90\%$ better than the baseline predictor, which is quite significant.

Table 5.6 shows the MAE and WMAPE of baseline predictor and GNN-based approaches in all three scenarios, TO, TnC and TnCwD, with three different intensities. In the best case, our GNN-based approach achieves 96%, 99% and 99% less MAE than the baseline predictor in scenarios TO, TnC, TnCwD, respectively. In addition, our GNN-based approach performs 84% and 97% than the baseline predictor in scenarios TnC and TnCwD, respectively. We can observe that the GNN-based approach performs much better than the baseline predictor in the most complex network scenario, TnCwD.

In short, the RF and NN performances become worse when the network traffic

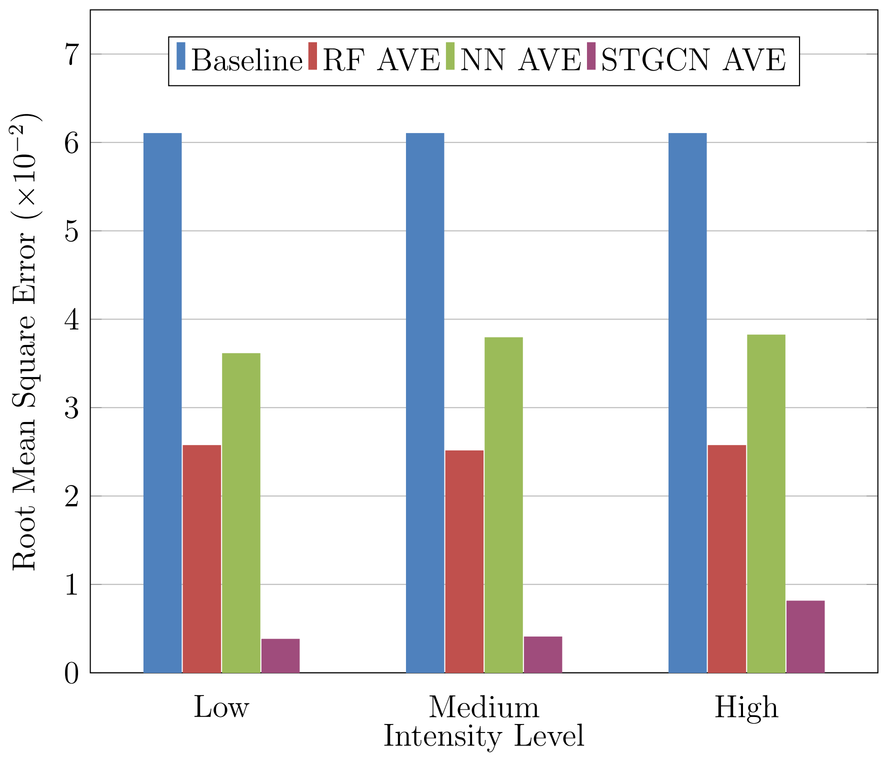

Figure 5.8: TnCwD RMSE with three intensities

(Blue bars indicate the results of the baseline predictor. Red bars represent RF RMSE values, and green bars show NN RMSE values and purple bars present GNN-based RMSE results. As the intensities increase, the RMSE increases. In all intensity levels, GNN-based approach has the smallest RMSE value.)

Table 5.6: MAE and WMAPE of baseline and GNN-based approaches in three traffic intensities of TO, TnC and TnCwD scenarios

	Baseline/Intensity	MAE AVE	WMAPE AVE
TO	*Baseline Predictor*	*0.0282*	*0.0146*
	Low	0.00122	0.0306
	Medium	0.00599	0.32845
	High	0.01394	0.81298
TnC	*Baseline Predictor*	*0.177*	*0.081*
	Low	0.00159	0.01322
	Medium	0.00338	0.02787
	High	0.00885	0.0710
TnCwD	*Baseline Predictor*	*0.646*	*0.274*
	Low	0.00096	0.00715
	Medium	0.00111	0.00806
	High	0.00230	0.01748

intensity increases and network complexity increases. However, our GNN-based delay prediction approach can achieve a satisfying performance regardless of the network complexities. Notably, our GNN-based approach performs much better when the network scenario is realistic and similar to the real-world SDN environment. We can conclude that our GNN-based approach is definitely able to predict end-to-end delays in the SDN environment more accurately.

5.3.5 Results and Comparison for GEANT2 and 50-node Networks

In this section, we will show and compare the result between 15-node, GEANT2 (24 nodes) and 50-node networks. Their corresponding adjacency matrices are of sizes $[255, 255]$, $[576, 576]$ and $[2500, 2500]$, respectively. We execute our experiment ten times for each situation and intensity to ensure consistent prediction and comparison. The average end-to-end delays of all OD traffic matrices are used as the baseline prediction, which is referred to as the baseline predictor. We compare the baseline predictor, several ML-based techniques, and our new GNN-based delay prediction methodology.

Table 5.7 shows the result of the baseline, MLR, XGBOOST, RF and GNN-based approach performances in three traffic topologies. We can conclude that all the ML-based approaches achieve $6\% - 97\%$ less RMSE compared to the baseline predictor. Particularly, XGBOOST has the worst performance in all three methods, which performs $5.9\% - 55.5\%$ better than baseline. MLR has a stable performance, which is $18.6\% - 25.2\%$ smaller than baseline. RF and STGCN give the best prediction performance among all three topologies. RF is $82.7\% - 91.9\%$ and STGCN is $94.7\% - 97\%$ better than the baseline.

Table 5.7: RMSE of baseline, RF, MLR, XGBOOST and GNN-based approaches in three network topologies

Method	15-Node Scale-Free	GEANT2	50-Node
Baseline Predictor	0.15937	0.27449	0.51142
MLR AVE	0.12973	0.20743	0.37759
XGBOOST AVE	0.07124	0.25820	0.396816
RF AVE	0.02738	0.03353	0.04164
GNN-based AVE	0.006103	0.0144635	0.01537

Figure 5.9 shows the comparison between four ML methods. MLR has the biggest RMSE value in 15-Node Scale-Free topology. XGBOOST performs the worst in GEANT2 and 50-Node topology. RF has the third lowest RMSE in all topologies. Our GNN-based approach has the smallest RMSE, which performs $91.4\% - 96.1\%$ better than MLR and XGBOOST; it is also $56.9\% - 77.7\%$ better than RF, which is quite significant.

Table 5.8 shows the MAE and WMAPE of baseline predictor and GNN-based approaches in all three network topologies. We can observe that as the network complexity increases, the error of the baseline prediction increases a lot. However, for our GNN-based model, the error values are stable and small. Numerically, our GNN-based approach achieves 96.7%, 94.8% and 97.0% less MAE than the baseline predictor in 15-Node, GEANT2 and 50-Node scenarios, respectively. In addition,

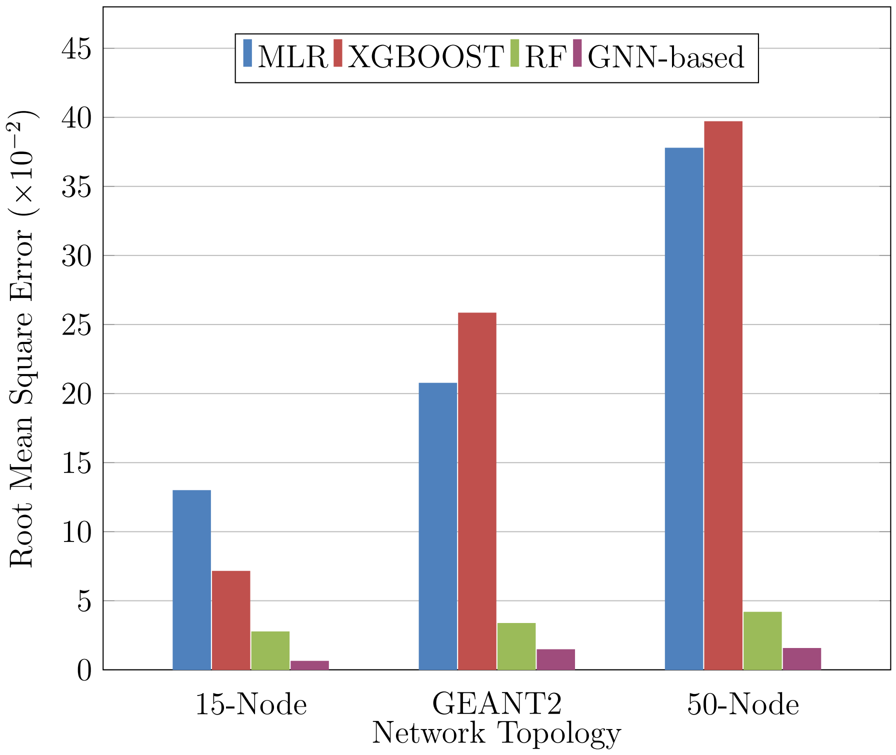

Figure 5.9: Various ML methods RMSE with three networks
(Blue bars indicate the results of MLR prediction RMSE average. Red bars represent XGBOOST RMSE average values, and green bars show RF RMSE average values and purple bars present GNN-based RMSE average results. As the network complexity increases, the RMSE increases. In all topologies, GNN-based approach has the smallest RMSE value.)

Table 5.8: MAE and WMAPE of baseline and GNN-based approaches in three traffic topologies

	Method	MAE AVE	WMAPE AVE
15-Node	*Baseline Predictor*	*0.1371*	*0.41554*
	STGCN	0.004552	0.01377
GEANT2	*Baseline Predictor*	*0.10996*	*0.25046*
	STGCN	0.00568	0.011867
50-Node	*Baseline Predictor*	*0.19663*	*0.29337*
	STGCN	0.005921	0.00579

the GNN-based method performs 96.7%, 95.3% and 98.0% better than the baseline predictor according to WMAPE for three scenarios, respectively. We can find out that the GNN-based approach performs much better than the baseline predictor, and as nodes increase, the performance becomes better compared to the baseline.

In short, when network nodes rise, the baseline and other ML methods (RF, MLR, XGBOOST) performance errors increase. However, regardless of network complexity, our GNN-based delay prediction technique may deliver high accuracy. Notably, in the most complex 50-Node network environment, our GNN-based technique performs significantly better. We can infer that our GNN-based technique is capable of more correctly predicting end-to-end delays in the SDN context.

5.4 Summary

In this chapter, we first introduced 15-node datasets with different topologies and distributions and Abilene Network datasets. Then, data preprocessing and performance metrics were provided. We calculated MAE, RMSE and WMAPE for each dataset and compared it with other ML models.

In 15-node datasets, deterministic and uniform distributions outperformed binomial and Poisson distributions. Ring Network had the least error values of all the topologies. From various intensities in Abilene Network, we can conclude that when traffic intensity and network complexity increase, training error will also increase.

We also used a GNN-based model to forecast the end-to-end latency of SDN in GEANT2 and 50-node networks. We evaluated the performance of our GNN-based delay prediction with MLR, XGBOOST, RF and the baseline predictor.

After comparing our models, it can be concluded that STGCN outperforms other models similar to the real-world SDN environment. This result is likely due to the model's ability to capture dependencies between neighboring nodes in the graph via the graph convolution operation, as well as its ability to capture temporal dependencies via the temporal convolution operation.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we applied a GNN-based model to predict the end-to-end delay in SDN networks. We found that this model outperforms the average baseline predictor in predicting packet delay since our model captures both spatial and temporal dimensions of the data.

First, we used 15-node datasets from the Omnet++ simulator with star, ring and scale-free topologies. We found that the ring network has the lowest error level, which is only 18% of other topologies in RMSE with the binomial distribution. Second, we compared the difference between different distributions. Deterministic and uniform distributions outperformed binomial and Poisson distributions in all 15-node datasets.

We also used Abilene network datasets simulated using the NS-3 simulator. From various intensities in Abilene network, we can conclude that when traffic intensity and network complexity increase, training error will tend to increase in general.

Then, we compared the performance of our GNN-based delay prediction with RF, NN and the baseline predictor. The results showed that the ML-based approach can predict the end-to-end delay more accurately than the baseline predictor. In the best case, our GNN-based delay prediction achieved 94% less RMSE, 94% less MAE and

95% less WMAPE than the baseline predictor. In particular, our GNN-based delay prediction method performed best in the most realistic network scenario, with 68.5% and 78.7% less RMSE than RF and NN, respectively.

Finally, we evaluated the performance of our GNN-based delay prediction with MLR, XGBOOST, RF and the baseline predictor using 15-node scale-free network, GEANT2 with 24 nodes, and a 50-node network topology simulated from RouteNet using OMNeT++. The results showed that the GNN-based technique outperformed the baseline predictor in predicting the end-to-end delay. In the best situation, our GNN-based delay prediction obtained 97.0% less RMSE, 97.0% less MAE and 98.0% less WMAPE than the baseline predictor. When compared to other ML methods, our GNN-based delay prediction technique performed the best in all network scenarios considered. In particular, our GNN-based model outperformed MLR, XGBOOST and RF with 95.9%, 96.1% and 63.1% reduced RMSE in the most complex 50-node network scenario, respectively.

Unlike RF, NN and other prediction methods, our GNN-based model captures not only the temporal but also the spatial dependence of the data. Our GNN-based delay prediction method can certainly be used in SDNs to help controllers understand network traffic conditions more accurately and allocate traffic efficiently.

6.2 Future Work

We outline below some suggestions for future work using our GNN-based delay prediction strategy.

- Test on More Network Traffic Scenarios

  Performance is expected to change for different traffic scenarios. It would be useful to test how accurately the same ML algorithms with the chosen features will perform in different traffic scenarios.

- Test of Different Network Topologies

  The optimized route model can be tested on other network topologies to confirm its performance. Further investigations can be carried out to see how the model will perform in instances where there are link failures.

- Test with Additional Algorithms

  Testing with other ML algorithms or performing parameter tuning on the tested algorithms could produce better results for either the measured accuracy or runtime in the delay prediction problem. This is an area of investigation.

References

[1] Ali R. Abdellah, Omar Abdulkareem Mahmood, Ruslan Kirichek, Alexander Paramonov, and Andrey Koucheryavy. Machine learning algorithm for delay prediction in iot and tactile internet. *Future Internet*, 13(12), 2021. ISSN 1999-5903. URL https://www.mdpi.com/1999-5903/13/12/304.

[2] Fernando Barreto, Emilio Wille, and Luiz Nacamura. Fast emergency paths schema to overcome transient link failures in ospf routing. *International Journal of Computer Networks & Communications*, 4, 04 2012. doi: 10.5121/ijcnc.2012.4202.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[4] Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. Software-defined networking (sdn): a survey. *Security and communication networks*, 9(18):5803–5833, 2016.

[5] Stephen A Billings. *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons, 2013.

[6] EL Hocine Bouzidi, Abdelkader Outtagarts, Rami Langar, and Raouf Boutaba. Deep q-network and traffic prediction based routing optimization in software defined networks. *Journal of Network and Computer Applications*, 192:103181,

2021. ISSN 1084-8045. URL https://www.sciencedirect.com/science/article/pii/S1084804521001909.

[7] Lin-Huang Chang, Tsung-Han Lee, Hung-Chi Chu, and Cheng-Wei Su. Application based online traffic classification with deep learning models on sdn networks. *Advances in Technology Innovation*, 5(4):216–219, 2020.

[8] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL http://doi.acm.org/10.1145/2939672.2939785.

[9] Guillaume Chevalier. Larnn: linear attention recurrent neural network. *arXiv preprint arXiv:1808.05578*, 2018.

[10] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[11] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 1724–1734, 2014.

[12] David D Clark, Craig Partridge, J Christopher Ramming, and John T Wroclawski. A knowledge plane for the internet. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–10, 2003.

[13] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

[14] Yuqi Fan and Tao Ouyang. Reliability-aware controller placements in software defined networks. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 2133–2140. IEEE, 2019.

[15] Yoav Freund and Mason Llew. The alternating decision tree learning algorithm. *International Conference on Machine Learning*, 99:124–133, 1999.

[16] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[17] Tony H Grubesic, Timothy C Matisziw, Alan T Murray, and Diane Snediker. Comparative approaches for assessing network vulnerability. *International Regional Science Review*, 31(1):88–112, 2008.

[18] Jacob SW Heglund, Panukorn Taleongpong, Simon Hu, and Huy T Tran. Railway delay prediction with spatial-temporal graph convolutional networks. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE, 2020.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] Sabrina Jiang. A simple neural network, 2021. URL https://www.investopedia.com/terms/n/neuralnetwork.asp.

[21] Sergios Karagiannakos. Best graph neural network architectures, Sep 2021. URL https://theaisummer.com/gnn-architectures/.

[22] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[23] Jan Koutník, Klaus Greff, Faustino Gomez, and Jürgen Schmidhuber. A Clockwork RNN. *31st International Conference on Machine Learning, ICML 2014*, 5: 3881–3889, 2014.

[24] Filip Krasniqi, Jocelyne Elias, Jérémie Leguay, and Alessandro EC Redondi. End-to-end delay prediction based on traffic matrix sampling. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 774–779. IEEE, 2020.

[25] Stanislav Lange, Steffen Gebert, Thomas Zinner, Phuoc Tran-Gia, David Hock, Michael Jarschel, and Marco Hoffmann. Heuristic approaches to the controller placement problem in large scale sdn networks. *IEEE Transactions on Network and Service Management*, 12(1):4–17, 2015.

[26] Aggelos Lazaris and Viktor K. Prasanna. Deep learning models for aggregated network traffic prediction. In *2019 15th International Conference on Network and Service Management (CNSM)*, pages 1–5, 2019. doi: 10.23919/CNSM46954.2019.9012669.

[27] Jurij Leskovec. Machine learning with graphs, 2021. URL http://web.stanford.edu/class/cs224w/.

[28] Michel Loeve. *Probability theory*. Courier Dover Publications, 2017.

[29] Albert Mestres, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcón, Marc Solé, Victor Muntés-Mulero, David Meyer, Sharon Barkai,

Mike J Hibbett, et al. Knowledge-defined networking. *ACM SIGCOMM Computer Communication Review*, 47(3):2–10, 2017.

[30] Albert Mestres, Eduard Alarcón, Yusheng Ji, and Albert Cabellos-Aparicio. Understanding the modeling of computer network delays using neural networks. In *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, pages 46–52, 2018.

[31] AysŞe Rumeysa Mohammed, Shady A. Mohammed, and Shervin Shirmohammadi. Machine learning and deep learning based traffic classification and prediction in software defined networking. In *2019 IEEE International Symposium on Measurements Networking*, pages 1–6, 2019. doi: 10.1109/IWMN.2019.8805044.

[32] Anthony J Myles, Robert N Feudale, Yang Liu, Nathaniel A Woody, and Steven D Brown. An introduction to decision tree modeling. *Journal of Chemometrics*, 18(6):275–285, 2004.

[33] J-P Onnela, Jari Saramäki, Jorkki Hyvönen, György Szabó, David Lazer, Kimmo Kaski, János Kertész, and A-L Barabási. Structure and tie strengths in mobile communication networks. *Proceedings of the national academy of sciences*, 104 (18):7332–7336, 2007.

[34] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[35] Sebastian Orlowski, Roland Wessäly, Michal Pióro, and Artur Tomaszewski. Sndlib 1.0—survivable network design library. *Networks: An International Journal*, 55(3):276–286, 2010.

[36] Ampratwum Isaac Owusu and Amiya Nayak. An intelligent traffic classification in sdn-iot: A machine learning approach. In *2020 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pages 1–6. IEEE, 2020.

[37] M. Pal. Random forest classifier for remote sensing classification. *International Journal of Remote Sensing*, 26(1):217–222, 2005.

[38] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. PMLR, 2013.

[39] F Pedregosa, G Varoquaux, A Gramfort, V Michel, B Thirion, O Grisel, M Blondel, P Prettenhofer, R Weiss, V Dubourg, J Vanderplas, A Passos, D Cournapeau, M Brucher, M Perrot, and E Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[40] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[41] Lawrence G Roberts and Barry D Wessler. Computer network development to achieve resource sharing. In *Proceedings of the May 5-7, 1970, spring joint computer conference*, pages 543–549, 1970.

[42] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[43] Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Routenet: Leveraging graph neural networks for network modeling and optimization in sdn. *IEEE Journal on Selected Areas in Communications*, 38(10):2260–2270, 2020.

[44] K. Tamil Selvi and R Thamilselvan. An intelligent traffic prediction framework for 5g network using sdn and fusion learning. *Peer-to-Peer Networking and Applications*, 15:751–767, 2022. doi: 10.1007/s12083-021-01280-6.

[45] András Varga. Using the omnet++ discrete event simulation system in education. *IEEE Transactions on Education*, 42(4):11–pp, 1999.

[46] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. on Neural Networks and Learning Systems*, 32(1):4–24, 2021. doi: 10.1109/TNNLS.2020.2978386.

[47] Junfeng Xie, F Richard Yu, Tao Huang, Renchao Xie, Jiang Liu, Chenmeng Wang, and Yunjie Liu. A survey of machine learning techniques applied to software defined networking (sdn): Research issues and challenges. *IEEE Communications Surveys & Tutorials*, 21(1):393–430, 2018.

[48] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*, 2017.

[49] Chen Zebin, Wei Yichi, Hao Tang, and Li Chuanhuang. Research on intelligent perception model of sdn network delay. In *2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS)*, pages 452–457, 2021. doi: 10.1109/ICCCS52626.2021.9449259.

[50] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis Lau. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*, 2015.