PREDICTION AND PREEMPTIVE CONTROL OF NETWORK CONGESTION IN DISTRIBUTED REAL-TIME ENVIRONMENT

by

Ramandeep Dhanoa

B.Tech., Rajasthan Technical University, Jaipur, India, 2012

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER SCIENCE

UNIVERSITY OF NORTHERN BRITISH COLUMBIA

January 2016

©Ramandeep Dhanoa, 2016

Abstract

Due to ever increasing demand for network capacity, the congestion problem is inflating. Congestion results in queuing within the network, packet loss and increased delays. It should be controlled to increase the system throughput and quality of service. The existing congestion control approaches such as source throttling and re-routing focus on controlling congestion after it has already happened. However, it is much more desirable to predict future congestion based on the current state and historical data, so that efficient controlling techniques can be applied to prevent congestion from happening in future.

We have proposed a Neural Network Prediction-based routing (NNPR) protocol to predict as well as control the network traffic in distributed real time environment. A distributed real time transaction processing simulator (DRTTPS) has been used as the test-bed. For predictions, multi-step neural network model is developed in SPSS Modeler, which predicts congestion in future. ADAPA (Adaptive Decision and Predictive Analytics) scoring engine has been used for real-time scoring. An ADAPA wrapper calls the prediction model through web services and predicts the congestion in real-time.

Once predicted results are obtained, messages are re-routed to prevent congestion. To compare our proposed work with existing techniques, two routing protocols are also implemented - Dijkstra's Shortest Path (DSP) and Routing Information Protocol (RIP). The main metric used to analyze the performance of our protocol is the percentage of transactions which complete before their deadline. The NNPR protocol is analyzed with various simulation runs having parameters both inside and outside the neural network input training range. Various parameters which can cause congestion were studied. These include bandwidth, worksize, latency, max active transactions,

mean arrival time and update percentage. Through experimentation, it is observed that NNPR consistently outperforms DSP and RIP for all congestion loads.

Dedicated to my family

Acknowledgement

Firstly, I would like to thank Dr. Waqar Haque for his guidance, motivation, enthusiasm, and immense knowledge in this field. His guidance and support helped me overcome many difficult situations in this journey. Throughout this research, his professionalism, attention to detail, and high quality of standards tremendously helped me to find myself in a better academic standing. I could not have achieved my goal without his constant support and generosity.

I also would like to express appreciation to my supervisory committee Dr. Alex Aravind and Dr. Iliya Bluskov for their support and direction in my research. I would like to thank my colleagues for their ongoing support for discussing new ideas and problems.

Last but not the least, I would like to thank my parents, my siblings, and my husband for their patience and encouragement, and for the sacrifice that they have made while I completed this research.

Raman Dhanoa

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Database system is a collection of information shared by many users [1]. In today's world, databases and database systems have become a crucial part of life. When a database system is accessed, read and write operations are executed. When these operations are executed on the database in a particular order, it is called a transaction [1], for example depositing money in a bank. To maintain integrity and consistency of the database, a transaction must follow ACID properties: atomicity, consistency, isolation and durability [8]. Atomicity makes sure that a transaction will fail, if any part of the transaction dies. Consistency ensures that a transaction will always leave the database in a consistent state, whether it is completed or not. When many transactions are executing concurrently, isolation guarantees that all the transactions are running independent of each other. Durability implies that a committed transaction cannot be undone, even if the system crashes.

Database systems can be classified as centralized and distributed. In a centralized database system, data is stored and maintained on a single node; whereas in a distributed database system many nodes run independently at different locations, and they interact and share data with each other via a communication network [9]. In distributed database systems, availability of data can be augmented by replication of data. In a fully replicated system, each node stores the entire database, thus allowing

easy access to all the database entities. In this system, there is no risk of loss of data if any node fails. However, it also raises issues related to concurrency control. In a partially replicated system, the database is replicated on a subset of the network nodes.

In a real-time system, every transaction has a specified time allocated to complete. If a transaction does not complete in the assigned time, it is called a *tardy* transaction [8]. System's throughput is determined by the transactions completed in the assigned time. Distributed real-time database system (DRTDBS) [10] is a collection of databases scattered over different data sites connected via a communication network, where transactions have consistency and time constraints. The primary objective of DRTDBS [5] is to increase the number of completed transactions before their deadlines, while maintaining the serializability of transactions. If the outcome of the transactions running concurrently is same as of running them serially, then transactions are said to be *serializable*.

In DRTDBS, nodes communicate through messages by finding an efficient link, as specified by the network. A link (pipe) has parameters, such as latency, bandwidth, source and destination node. Latency is the amount of time taken to send a message from source to destination node. Bandwidth represents the total amount of data which can be passed through the link at any given time. The number of messages being sent from source to destination node depends on the source-destination link's bandwidth. If the messages coming on the link are more than link's bandwidth, they wait in a queue. Because of resource and time constraints, every computer tries to complete its operations in the assigned time. However, when resource requirement surpasses the capacity of network, congestion occurs [11]. In a distributed network, congestion occurs if a network link has queued messages waiting to get processed thereby decreasing the quality of service. Some effects of congestion are increased delay and dropping of packets.

Congestion should be controlled to improve the system throughput and quality of

service. There are many techniques to control congestion [11], but these techniques control the congestion after it has happened. With an increased demand for highly efficient networks, it is important to analyze the potential network traffic beforehand and predict the traffic [12], so that efficient control techniques can be applied.

Our hypothesis is that network congestion can be reduced by predicting traffic and dynamically re-routing the messages, thus increasing the percentage of completed transactions before their deadlines.

1.1 Transaction Execution

Transaction is a list of operations executed as a basic unit of data processing in database systems. The transaction's operations to read and write a database object can be denoted by $R_T(O)$ and $W_T(O)$, respectively. A transaction can be either local or global depending on the way it needs to access data [1]. Local transaction demands data access at its local node only, whereas global transaction requires data access at multiple nodes and may spawn many sub-transactions. In DRTDBS, execution of a transaction is managed by processes executing on different nodes to collaborate between a transaction and its sub-transactions. The process executing on the node where transaction generates is called *coordinator*, and processes executing at other nodes on behalf of coordinator process to support global execution of transaction by synchronizing transaction with its sub-transactions are called *cohorts* [1]. When cohorts complete their tasks, they send acknowledgments to their coordinator, resulting in the successful execution of a transaction.

When a transaction executes, it goes through various stages as shown in Figure 1.1.

- Active State - In this stage, transaction starts its execution.
- Partially Committed - In this state, when cohorts complete their assigned tasks, they send an acknowledgment to the coordinator process.

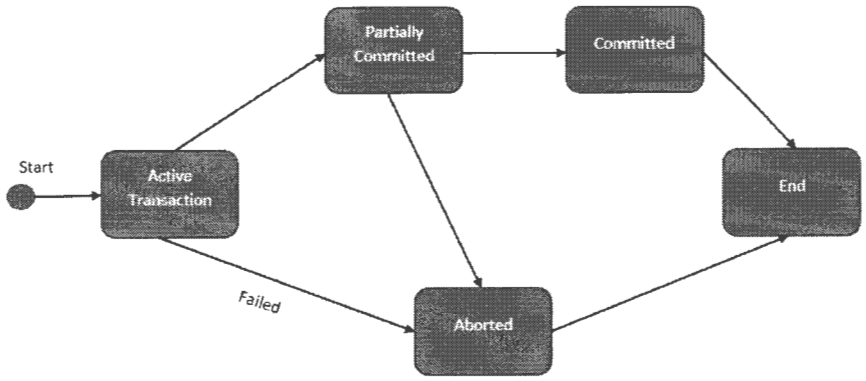

Figure 1.1: Life Cycle of a Transaction

- Aborted - If transaction's execution gets interrupted by terminating the current ongoing operation, it gets aborted. Thereby, the transaction is rolled back i.e. database is returned to its starting state. Once the transaction is aborted, it can be restarted or terminated.
- Committed - When coordinator receives an acknowledgment from all the cohorts, the transaction successfully completes or commits. All the changes resulting from this transaction are permanently updated to the database.

In real-time systems, availability of data at the required time is very significant. When the entire database is stored on each node, data is available to a transaction on its local node. However, when the database is partitioned across the nodes, the data may not be directly accessible. Hence, the type of data distribution decides data's accessibility.

1.2 Distribution of Data

In DRTDBS, database is distributed across the system and the system's performance significantly depends on the way data is distributed. There are two ways in which data can be distributed: partitioned and replicated.

- In partitioned distribution, there is only one copy of the database, which is distributed across some or all nodes as shown in Figure 1.2. An advantage of such distribution is that it enhances the performance of the system because of easy management, back-up, and restore of data. Since a small volume of data is accessed, data retrieval operations are also efficient. On the contrary, a drawback is that failure of one node may lead to the breakdown of the entire system because the failed node may have high valued data that is accessed by different nodes.

- A partially replicated system stores copies of database fragments at each node. In a fully replicated system, there are multiple copies of the database and each node stores one copy of the entire database. It increases the availability of data facilitating transaction's execution. Even if one node fails, the system will not collapse because other nodes will have the required data. However, maintaining consistency is the main problem here. To prevent inconsistencies, if one copy of the database gets updated, that change should get reflected in other copies as well. This can be achieved by concurrency control protocols [13].

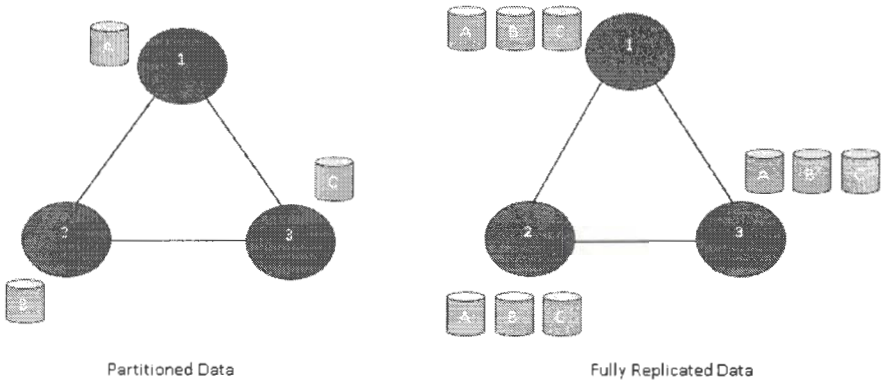

Figure 1.2: Partitioned vs. Replicated Data Distribution

The primary factors affecting the decision to use data replication are database size and usage frequency. When the usage frequency is excessive, data replication decreases the cost of data retrieval operations [14].

1.3 Transaction Deadlines and Priority Scheduling

In DRTDBS, the performance of the system depends on the number of transactions completing before their deadlines. In general, transaction's deadline can be classified into three types namely hard, soft and firm deadline [1]. For each type of deadline, there is a value associated that declines as the function of completion time if deadline is missed.

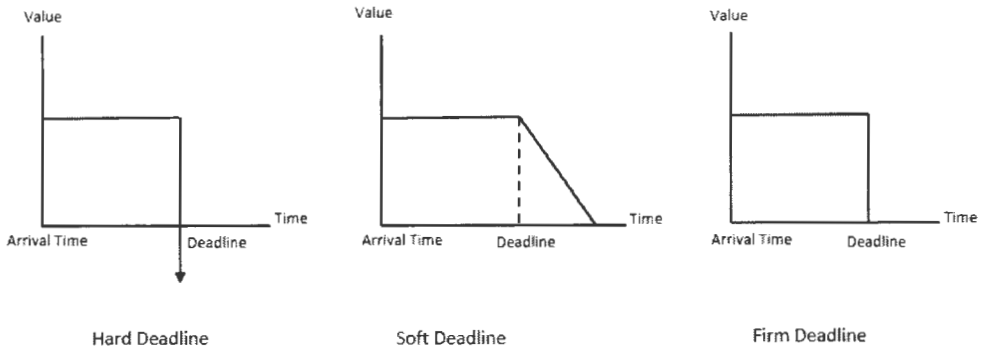

Figure 1.3: Transaction Deadlines [1]

- In hard deadline, transaction is assigned a rigid deadline to complete its operations. If deadline is missed, the value gets reduced to negative amount, indicating disastrous consequences on the system. The examples of hard deadline applications include radar tracking system and military applications. So, to handle hard deadline application effectively there is a need to design a fault-tolerant system. Redundancy is a standard approach to attain fault tolerance [15]. There are three types of redundancies: hardware redundancy, software redundancy, and time redundancy [15]. Hardware redundancy means critical hard-

ware components of the system are duplicated, when failure occurs duplicated components replace the faulty ones. Software redundancy refers to the software components duplication, which are functionally identical. It comes into effect when the working software crashes/fails. Time redundancy is giving extra time to execute failed operations (Failed operations can be software failure or any other kind of failure).

- A soft deadline allows a transaction to miss its deadline and allocates extra time (slack time) to complete its operations. Once deadline is missed, the value starts decreasing; but the transaction is still allowed to complete its tasks. When the transaction has still not completed within the additional allocated time, its value becomes zero. The examples can be stock trading and payment of taxes.

- A firm deadline is same as the soft deadline, but it does not provide extra time to transaction to complete its operations. Once deadline is missed, the value becomes zero. For example - incorrect forecasting of a sales company can lead to a great loss in terms of revenue.

Priorities are allocated to transactions in order to define their significance and order of executions. Priority assignment algorithms can be predominantly categorized into three types: static, dynamic and hybrid [1]. In static priority scheduling algorithm, transaction's priorities are fixed and do not change once they are assigned. When priorities are assigned at run time, it is called dynamic scheduling of transactions. Dynamic scheduling algorithm assigns priority to transaction considering the factors such as deadline, slack time (extra time given to a transaction after deadline to complete its operations), execution time, and others. A hybrid algorithm is a mixture of static and dynamic priority scheduling i.e. some transactions have fixed priorities, while others get assigned at run time. Some of the examples of priority scheduling algorithms are described below [8]:

- **Earliest Deadline First (EDF):** In EDF, transaction with the closest deadline is allocated the highest priority. A limitation of this algorithm is that it

may assign higher priority to a transaction which may have almost missed its deadline, and thus restricting other transactions to execute which may complete before the deadline.

- **First Come First Serve (FCFS):** According to FCFS, transaction with earlier arrival time will be designated higher priority. It does not consider deadline information of the transactions, and may assign higher priority to a transaction having longer deadline and vice-versa.

- **Shortest Job First (SJF):** SJF assigns the highest priority to the transaction having minimum execution time. However, SJF does not perform well in applications where execution time of transactions cannot be computed beforehand.

- **Minimum Slack First (MSF)**: According to MSF, transaction having minimum slack time will be assigned the highest priority.

These priority scheduling algorithms decide the order in which transactions execute in case of congested scenarios, hence increasing the overall performance of the system.

1.4 Network Congestion and Control Techniques

Network congestion is defined as a state when the quality of service deteriorates because of an increase in network load [11]. It occurs when data arriving on node's link exceeds its bandwidth capacity. During congestion, data gets queued on the node's buffer, which results in increased packet delays and hence decreases the number of transactions completed before the deadline. Congestion cannot be just determined from queue's length, it also depends on link's bandwidth and latency. For example, suppose Node A needs to send 40 message units to Node B. If link A-B has 20 message units bandwidth, with 5 seconds latency, it will take 10 seconds to clear Node A's queue. But if bandwidth is 5 and latency 10 seconds, it will take 80 seconds to clear the queue. So, even if queue length is same in both the scenarios, bandwidth and

latency play a major role in determining the congestion of link. Congestion can be classified as low, medium or high depending on the threshold time limit(time taken to clear a node's queue). We define congestion of a link as follows:

$$Congestion_l = \left\lceil \frac{\sum_{i=0}^{n} size_i}{BW_l} \right\rceil \times L_l \tag{1.1}$$

where $size_i$ is the size of queued message i, BW is the bandwidth of the link l, and L is the latency of the link.

In DRTDBS, if queues on individual nodes are not cleared/processed for a prolonged time, transactions will start missing their deadlines at a very high pace, consequently engendering low system throughput. To be efficient, DRTDBS should have a congestion control algorithm that can manage traffic efficiently. The aim of congestion control algorithm is to increase the network efficiency i.e. throughput of the network by decreasing the delay and packet drops. Congestion control schemes are classified as [11, 16]:

- **Open loop control**: Open loop control does not use any feedback to control the system, for example - light switch automatically gets turned off after certain duration [11]. It has no knowledge of the output state i.e. it does not inspect system dynamically. It is further divided into source control and destination control.
  - Source control [17] tries to control traffic at the source level by controlling arrival rate of traffic, for example - Input buffer limit algorithm [18] controls traffic by applying a limit on input buffer, which eventually prevents the input traffic from entering the system.
  - Destination control tries to control the traffic at intermediate or destination nodes, for example - Selective packet discarding policy [19] simply discards the packet when there are no empty buffers.
- **Closed loop control**: Closed loop system uses feedback from the output to control the system [20], for example, when the room temperature is high, the

thermostat sends a signal to air conditioner to switch on the cooling unit. Closed loop system has complete knowledge of the output state i.e. it analyzes network performance dynamically. It is further divided into global and local feedback control.

- In global feedback control, feedback is attained from complete path i.e. destination to source, for example - Rate based control [21] examines the incoming traffic on each node and compares it with threshold capacity. If incoming traffic is greater than threshold value, it tunes the rate of incoming traffic and broadcasts the messages to all nodes in the network.
- According to local feedback control, feedback is attained just from the neighbour node, for example - Hop by hop control [16]. In this algorithm, each node will monitor its outgoing link traffic to the neighbour node. It then realizes the congestion status using feedback information from neighbour and adjusts traffic rate dynamically.

There are many congestion control techniques (such as controlling the arrival rate of packets in the network, discarding packets and routing techniques) proposed by the researchers over many years [16, 18, 19, 22–24]. However, these algorithms come into effect once congestion has been discovered by the system. An ideal approach would be to analyze the present data, predict network congestion and take corrective actions to prevent the congestion.

1.5 Data Mining and Predictive Analytics

Data mining is a mechanism to extract meaningful information from data through different mathematical algorithms [25]. This information can be discovering patterns or trends, finding correlation or clustering of data. Data mining is a crucial step in the process of predictive modelling. For example - to predict which customers are most likely to purchase the new product, data mining provides hidden and relevant information from the present data and identifies meaningful trends or patterns in

data. By analyzing present scenarios and predicting events using statistical and mathematical techniques, predictive analytics determines uncertainty and risk within data, as a result deciding optimal outcomes for the system. Based on the business requirement, predictive analytics life cycle has the following phases [25]:

- **Data Understanding** - This phase includes understanding business/system needs, determining system goals and planning accordingly.
- **Data Preparation** - Data preparation collects, selects and verifies data according to system objectives. In this phase, data cleaning, integration and transformation are the vital steps. Data cleaning handles missing or irrelevant information present in the data. Disparate data sources from different locations are integrated into one data source through data integration. Data transformation converts the format of data from one type to another.
- **Data Modelling** - In data modelling, a model is developed trying to achieve the business goals. Modelling techniques can be classified into the following categories [25]:
  - **Clustering** is grouping of data objects based on the similarity in one way or other (distance, nature and characteristics). Each group, called cluster, is dissimilar to other clusters. Clustering helps to identify the natural groupings present in the data. It is being used in many applications such as pattern recognition, image processing, and others.
  - **Association** modelling discovers relationships between one or more variables in a dataset. The relationships defined between attributes are called association rules. For example, customers who buy cheese in grocery market are most likely to buy bread as well. So, bread and cheese relationship can be expressed as an association rule - *cheese implies bread, with 90 percent probability.*
  - **Classification** predicts and categorizes the data based on their attributes. For example, the classification model can be used to identify student as a

good or bad scorer based on student's attributes such as attendance details and exam scores.

- **Result Interpretation** - The model is analyzed and evaluated in this phase. One way to evaluate a model is to randomly divide the large dataset into three sub-sets, which are then used for training, validation and testing. The training set is used to train the model; validation set assesses model's performance by analyzing the trained model; and testing set is used to estimate the error rate of the model. If results are not satisfactory, this phase goes back to data preparation phase (Figure 1.4) to remove model's deficiencies; else next phase is followed.

- **Decision Optimization** - In this end phase, an optimal decision is taken on the basis of result evaluation phase. One best solution is chosen among many feasible solutions considering factors like cost, profit, and others.

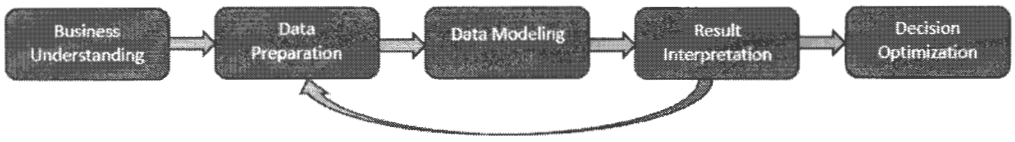

Figure 1.4: Data Mining Process

A desirable feature of analytics is to develop a predictive model, which is readily available to the users present at different locations. The data generated by predictive models is extremely critical to the businesses, therefore the model should be accurate and accessible to the decision makers on-demand. The deployment of predictive models on the cloud allows users to deploy and score the predictive models in real-time (explained in Chapter 3).

1.6 Contribution

Due to increase in network demand, congestion problem is increasing. Congestion results in queuing within the network, packet loss and increased delays. With an increased demand for high-performance networks, there is a need to increase the network's throughput and quality of service. Various congestion control techniques are discussed in [11,16]. However, these techniques attempt to control congestion after it has already happened. It is important to proactively analyze network traffic and predict potential congestion before it occurs, so that efficient controlling techniques can be applied as a preemptive measure.

We have proposed a protocol to predict congestion as well as control the network traffic in distributed real-time environment using distributed real-time transaction processing simulator (DRTTPS) as a test-bed. For predictions, multi-step neural network technique is used, which predicts congestion (1 step ahead, where 1 step is 100 ticks). After predicting the network traffic, the predicted congested link's messages are re-routed to other links. To compare the proposed work with other techniques, two routing protocols are implemented - Dijkstra's Shortest Path (DSP) algorithm [26] and Routing Information Protocol (RIP) [27]. The primary metric used to analyze the performance is the percentage of transactions completed before their deadline. Some of the key considerations of our work are listed below:

- To meet real-time system requirements, the proposed technique ensures that prediction results are attained in a reasonable time.
- Instead of reducing the traffic in system or increasing the inter-arrival time of packets [17, 20, 21, 28, 29], network traffic is controlled by routing techniques.
- Invocation of prediction model is a controllable parameter that is configured by the user according to the requirement (by default, it is set to 100 ticks).

1.7 Thesis Organization

The rest of the thesis is organized as follows. In addition to literature survey, Chapter 2 also contains some information about different data mining methods. Chapter 3 provides a brief discussion of our test-bed i.e. DRTTPS, neural networks and cloud computing. Chapter 4 explains the prediction model developed to predict congestion, and gives the implementation details. Chapter 5 presents experimentation and analysis of results. Chapter 6 provides the conclusion and directions for future research.

Chapter 2

Related Work

With an increased growth in network applications, network congestion is growing rapidly. Congestion should be controlled not only to increase system throughput, but also to improve quality of service and data transmission reliability. Many congestion control schemes have been proposed by researchers to control and prevent congestion. This chapter is divided into two sections. The first section outlines general congestion control mechanisms proposed in the literature, and second section describes congestion control using data mining and predictive algorithms.

2.1 Congestion Control Mechanisms

In literature, many approaches have been proposed to handle congestion, either by preventing or controlling or avoiding mechanisms. Some of the congestion control mechanisms are discussed in the following subsections:

2.1.1 Slow Start Congestion Control

Slow start [22] is a type of congestion control mechanism, in which the sending rate of packets is proportional to the transmitting rate of the network. Initially, sender determines network's capacity by transmitting one packet to the receiver. When a node receives the packet, it sends an acknowledgement (ACK) to the sender. The

sender then increases the congestion window (cwnd) size to 2, and sends 2 packets as shown in Figure 2.1. After receiving each ACK from the receiver, it elevates congestion window size by 1. For example, when ACK 1 and ACK 2 are received, congestion window size becomes 3 and 4 respectively (as shown in Figure 2.1). This results in exponential growth of congestion window size over round trips. Once cwnd is more than the threshold value, slow start algorithm increases cwnd linearly, rather than exponentially (i.e. window size is increased by 1 for each round trip time).

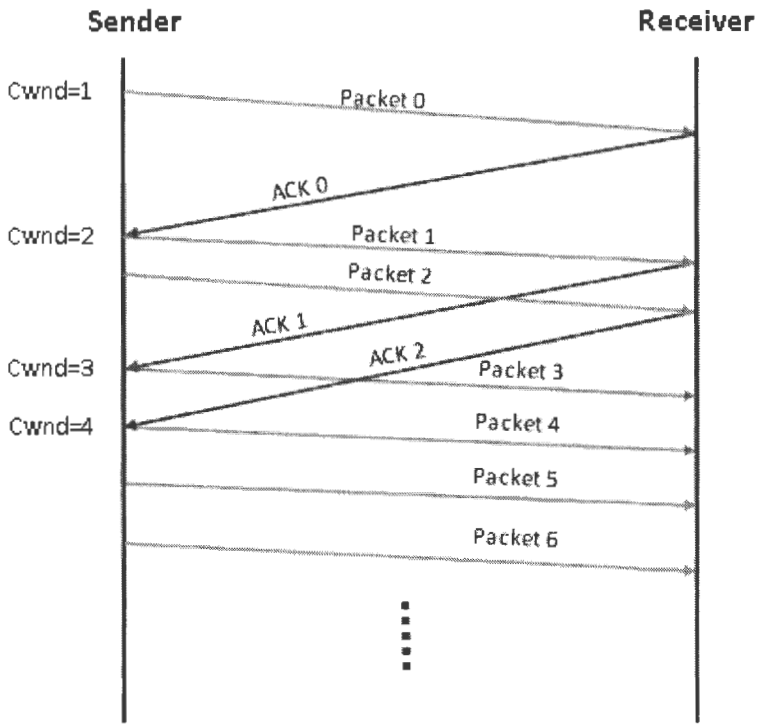

Figure 2.1: Slow Start Algorithm [2]

Slow start is crucial in avoiding congestion collapse [30]. Congestion collapse occurs when the sending rate of packets exceeds the network capacity. Thus, when a sender sends a large file, network can be overloaded and packets may start dropping. Slow start algorithm overcomes this problem but can cause delays in transmission,

which can be a significant limitation for real-time applications.

2.1.2 Fast Retransmit and Fast Recovery

Fast retransmit and fast recovery [22] is used to promptly regain the lost packets. Unlike transmission control protocol (TCP[1]), it does not use a transmission time-out to resend the packets. If a node receives a data segment which is not in sequence, it will send a duplicate acknowledgement. Duplicate acknowledgement (DUP ACK) is an acknowledgement sent by a node to sender while receiving an out of order data packet. For example, when sender transmits packet 1 to receiver, upon receiving the packet receiver sends an ACK 1 to sender and expects packet 2 from sender. Now if packet 2 gets lost and receiver receives packet 3, it will send again ACK 1 to sender (as shown in Figure 2.2), which is called duplicate acknowledgement.

If sender receives three duplicate acknowledgements from the receiver, it will consider that data packet (packet number greater than duplicate ACK number) is lost and retransmits the packet. This algorithm works efficiently when data packet losses are not frequent.

2.1.3 Random Early Detection

Random Early Detection (RED) is a congestion avoidance technique proposed by Floyd and Jacobson [23], which controls congestion by dropping packets when queue size exceeds a threshold value. In this technique, once average queue length surpasses maximum threshold value, new arriving packets are discarded or marked by setting a bit in packet's header with a certain probability, where probability is a function of average queue size. If average queue size is less than the minimum threshold value, packets are allowed to enter into the queue.

Contrarily, May et al. [31] argued and experimentally proved that if the average queue size exceeds the maximum threshold value, then dropping good packets do not

[1]TCP is a connection-oriented protocol responsible for setting-up the connections and handle data-communication over network layer.

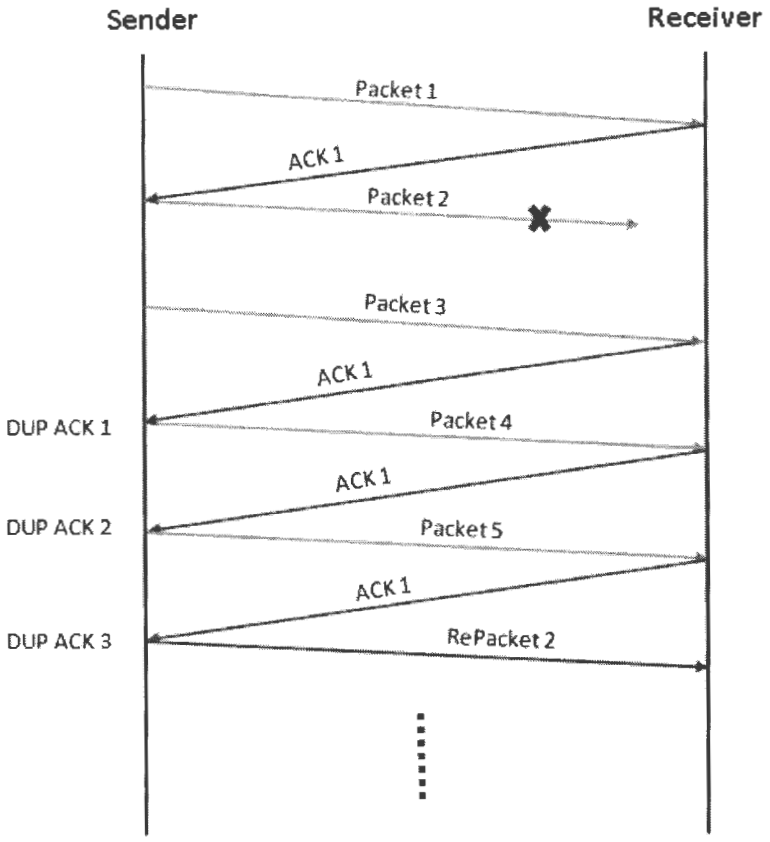

Figure 2.2: Fast Retransmit and Fast Recovery Algorithm [2]

increase the performance of the system. They demonstrated that RED implemented with small buffers does not alleviate system throughput significantly, whereas large buffers increase system throughput but parameter setting is challenging.

2.1.4 Back Pressure Technique

In back pressure technique [32], once a node becomes congested, it ceases receiving packets from its immediate upstream node. It may result in the congestion of immediate upstream nodes, which consequently stop receiving packets from their preceding nodes as shown in Figure 2.3.

This technique starts with congested node and disseminates in the opposite di-

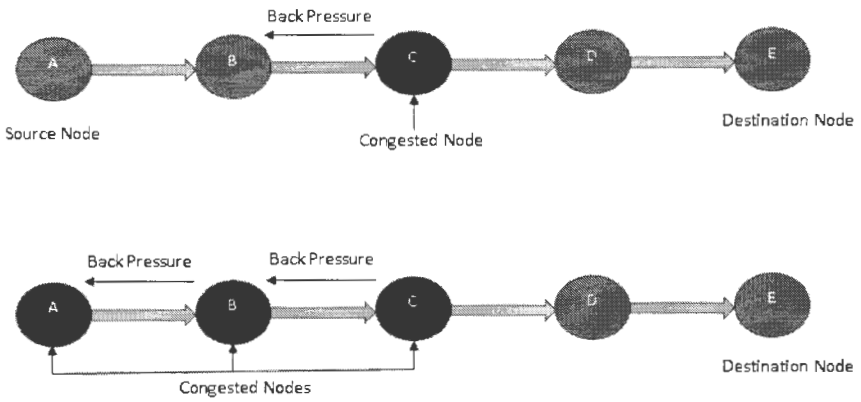

Figure 2.3: Back Pressure Technique [3]

rection of packet flow, to the source node. However, this technique can be of limited value, since it can only be used for connection-oriented networks.

2.1.5 Choke Packet Technique

Choke packet [32] is a packet generated by the congested node and transmitted back to the source node for congestion notification as shown in Figure 2.4. Inevitably, source has to reduce its sending rate until it stops receiving these packets. Unlike back pressure technique, choke packet technique gives congestion warning directly to the source node, skipping intermediate nodes. Therefore, it is a fast technique to notify sender to decrease its sending rate. This technique is good when there are fewer source nodes causing congestion at a particular time. But, when the congested node has queued data from different sources, it is difficult to determine where to transmit choke packets.

2.1.6 Implicit and Explicit Congestion Notification

In implicit signalling, source node waits for a hint/signal to take congestion control steps in the system [24]. From those signals, sender believes that there is congestion in the system. For example, when sender does not receive confirmation of sent packets,

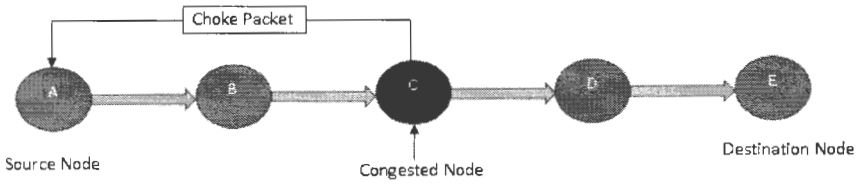

Figure 2.4: Choke Packet Technique [3]

it assumes there is a congestion. As a result, sender reduces its transmission rate.

Explicit Congestion Notification [24] is a congestion avoidance technique, which notifies congestion by setting a congestion notification bit, rather than dropping the packets. A Congestion Experienced (CE) bit is included in each packet's header, which is set by routers to signal congestion. So, when the network starts approaching congested state, receiver sends an acknowledgement to sender. In response to this acknowledgement, sender reduces the sending rate of packets and it further informs the receiver about the reduction in sending rate, so that it stops resending acknowledgements.

2.2 Congestion Control using Predictive Analytics

In the developing world of networking, more emphasis is placed on network's reliability and efficiency. To ensure this, network congestion should be handled in a way so that it minimizes network breakdowns. There is a necessity to detect network problems as soon as possible, so that preventive measures can be taken accordingly. Hence, historical network traffic needs to be analyzed to predict network behaviour so that efficient congestion control techniques can be applied.

There are two approaches to build a prediction model: empirical and analytical. Empirical approach is based on the observation and experimentation. Model is experimented on a real system or a simulator representing real system. Simulator-based models are the most common models to predict network congestion because simula-

tor captures the details of the underlying system and provides a variety of real-time scenarios to test the model. On the other hand, analytical models are less expensive, easy to evaluate but complex to build. Some of the techniques proposed in literature are discussed in the following subsections:

2.2.1 Time Series Prediction

To forecast, past data is analyzed to identify trends and seasonality. Using this information, data is projected into the future using statistic modeling techniques. Time series is an ordering of data noted at regular intervals of time, for example, monthly sales of a store. It is very important in the time series to analyze each point's correlation (measure of degree of association) with the previous point in the series. Two functions to analyze this correlation are discussed in [33] and presented below:

- **Autocorrelation function (ACF)**: It is defined as the correlation of an observed value with its past values, for example, autocorrelation of X time series at lag 2 is the coefficient of correlation between X(t) and X(t-2), and similarly X(t-2) with X(t-4). So, it is also expected that there is a correlation between X(t) and X(t-4).

- **Partial Autocorrelation function (PACF)**: It is defined as the correlation of the observed data with its lag after removing the observed data correlation with lower order lags.

In order to predict through time series, the stationary assumption should be satisfied, which is defined by ACF and PACF functions [34]. If the stationary assumption is not satisfied, Mean Square Error (MSE) will be high, leading to inaccurate predicted results. MSE indicates the difference between predicted and estimated values. Some models for time series prediction are described in [12] and summarized below:

- **Auto Regressive (AR) model**: AR model is based on the belief that each observed value in the time series is correlated to its previous lag values, for example, if the order of AR model in time series X is 3, it means X(t-3) is required to predict X(t). This model is used when the present data is related to the previous data, for example - student's previous scores are required to predict present scores.

- **Moving Average (MA) model** : In this model, a moving average is measured to analyze the seasonality or trend of data, for example, moving average order of 2 indicates that deviation of previous two values from the mean should be inspected to predict the current value. This model is generally used to forecast financial data and stock prices.

- **Auto Regressive Moving Average (ARMA) model** : This model consists of AR and MA models, called ARMA(p, q) model, where p is the order of AR model and q is the order of MA model.

- **Auto Regressive Integrated Moving Average (ARIMA) model** : When the data is not stationary, some differencing or transformation is applied on the data to satisfy the time series stationary assumption. If the time series data is differenced by the order of d, the model is called ARIMA(p, d, q).

Different time series models have been used to predict network traffic. It is, however, assumed that the data is stationary [34]. Stationary data has constant mean, constant variance, and the covariance is independent of time. The stationary assumption can be measured by ACF and PACF. To predict the network packets, traffic monitoring was done for one year by connecting to intra-network [12]. The total number of packets was measured every hour. ACF and PACF functions were not satisfied when predictions were done with the collected data or even after transforming the data. The data was then classified first by monthly, then by weekly periods; still the stationary assumption was not satisfied. Finally, when the data was classified

on a daily basis, the stationary assumption was satisfied and network packets were predicted using AR model.

Jung et al. [12] used AR model to predict network congestion, followed by another method [33] which focuses on controlling the traffic using routing techniques. Each node has its own routing table and routing decisions are made using Dijkstra's algorithm. The proposed algorithm checks if the predicted packets are greater than the given bandwidth and updates the routing table accordingly. Network Simulator-2 (NS-2) [35] has been used as a test-bed to compare the proposed algorithm with OSPF routing protocol and prove algorithm's efficiency. This work is similar to our work with following exceptions:

- Unlike Jung et al.'s research, neural network is being used in this research to predict network congestion because DRTTPS has many parameters affecting the congestion value (queue length), and neural network can understand the complicated relationships between the parameters and predict non-linear complex functions. If one parameter is tweaked, congestion value changes, and neural model analyzes the congestion successfully (shown in Chapter 5).
- Our prediction model is robust in a sense that it can analyze and predict different types of congestion loads (low, medium or high) in the system.
- Routing algorithm is dynamic i.e. if congestion is not predicted, then routing tables are not updated.

Long-range dependence and self-similarity in larger time span are the characteristics which should be captured by a good traffic model [36]. Zhou et al [36] proposed a model which is a combination of linear time series ARIMA and non-linear time series GARCH[2] model. Three separate time scales have been used to predict the network traffic from one-step-ahead to k-step-ahead prediction. It captures long as well as

[2]GARCH [36] is a non-linear time series model used to capture the varying variance over time.

short-range dependence. The model was compared with FARIMA[3] model to prove its efficiency.

2.2.2 Neural Networks

Artificial neural network is influenced by the biological nervous system, such as brain. The basic units are neurons and these units are organized in layers [4]. There are three layers in neural network (Figure 2.5) explained below:

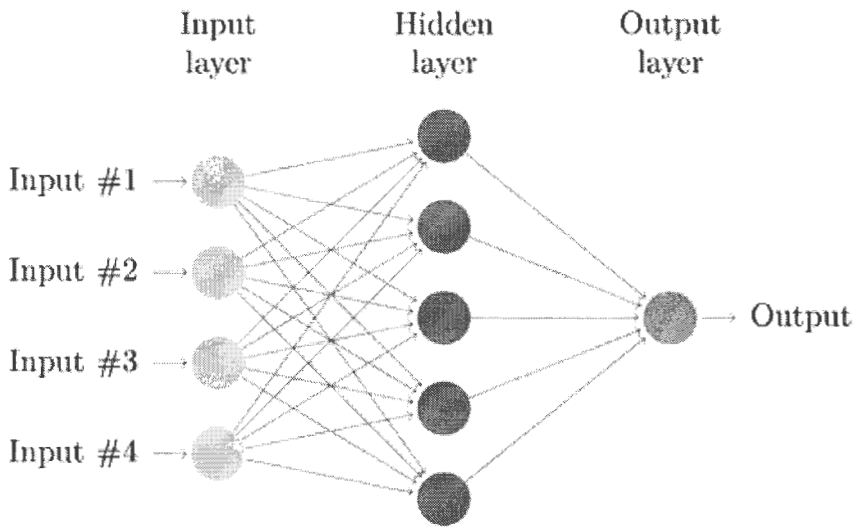

Figure 2.5: Neural Network Structure [4]

- **Input Layer**: Units in the input layer are the inputs of neural network.
- **Hidden Layer**: Depending on the network structure, there can be one or more hidden layers in the neural network. Hidden layer transforms the input data and capture non-linear dependencies in the data through activation functions (explained in Chapter 3).

[3]FARIMA [37] model is capable to capture the property of real traffic with long-range and short-range dependent behavior.

- **Output Layer**: Output layer represents the output of the system.

The edges through which these units are connected in the layers have some weights, representing a numeric value controlling the input. Neural network learns through training i.e model is fed with many datasets of known outputs. As training continues, the model keeps on adjusting its weights according to the input, and gradually becomes more accurate. Because of an efficient learning mechanism of neural network, it can predict network congestion with good accuracy [20,28]. After prediction, congestion can be controlled by many ways. One way is to throttle the input arrival rate [20,28], or apply different routing mechanisms [38–40]. The following subsections explain the congestion prediction done through neural network model and congestion control by throttling the source or applying different routing techniques.

2.2.2.1 Congestion control by throttling the source

Bivens et al. [28] have used simple feed forward neural network to predict the source of congestion. Once the congestion is predicted, flow rate[4] restriction is applied to the source node which is responsible for congestion. This technique is able to detect and correct congestion in almost 90 percent of the cases (out of 31 cases, congestion is correctly predicted 27 times) [28]. Network Simulator (NS), a discrete event simulator, is used to model the traffic patterns. In this simulator, a topology is chosen where all the network nodes are trying to send messages to one node at a random bit rate to depict the real network traffic. For each node, statistics are recorded during the simulation run, such as the packets received and queue size. The bandwidth allocation to links is random and the latency is set to a constant value. A control agent has been implemented that executes at a polling interval and contains two programs - C wrapper and MetaNeural Network Software. The communication between NS simulator and these two programs is done through files. Initially, C wrapper is called from the simulator, which reads the files updated by the simulator and then performs calculations like the average number of packets, the variance of

[4]Flow rate is the number of messages moving in a link/pipe in a given time frame

packets, and the third moment[5]. These values are normalized and then sent to neural network to make a decision. Neural network program generates the prediction result in another file which is converted to a readable format by C wrapper so that the simulator can process the file. Once the congested node is predicted, bit rate is reduced by adding a small amount of time to the sending interval.

When the incoming traffic packets exceed the outgoing packets, congestion is reduced by controlling the traffic rate [17, 20, 21, 28, 29]. A feedback control algorithm is proposed in [21] to predict the buffer occupancy L step-ahead through multi-step neural network. It also estimates the resources required through Back-Propagation (BP) neural network which is then used by the source node to adjust the sent-out rate accordingly. With the help of simulation model, it is shown that high prediction accuracy can be achieved by using fewer predictive steps. Thottethodi et al. [17] proposed a self-tuned mechanism which throttles the source upon congestion detection. Congestion is estimated by comparing the global information of network with the threshold value. If the global estimate is larger than the threshold value, packet injection is controlled. Threshold value is not static; it gets changed through self-tuned mechanism.

Liu et al. [29] considered queue length as a measure to estimate the performance of Asynchronous Transfer Mode (ATM) network. The congestion control of Available Bit Rate (ABR) service in ATM networks is achieved by implementing predictor and controller in the system using BP neural networks. The future arrival rate of traffic is predicted with metrics, such as past arrival rates and bandwidth. The controller predicts the queue length by taking inputs like predictive available bandwidth, queue length, control law[6] and their historical values. By using fairness algorithm at different connections, dynamic fair rate is allocated to each virtual circuit. At last, the performance of predictor and controller is compared to FARIMA model to prove its efficiency.

[5]Third moment is used to define the skewness of numbers.

[6]Control law computes source arrival rate

Fan and Mars [20] predicted the video traffic by finite impulse response (FIR) neural network and controlled congestion by throttling the input arrival rate. FIR neural network is a modification of conventional neural network, where network's weights are replaced by FIR linear filter. "FIR means that for an input excitation of finite duration, the output of the filters will also be of finite duration" [20]. Ogras and Marculescu [41] predicted the congestion on Network-on-Chip (NOC) and proposed a flow control algorithm which controls the total number of packets in the network. The primary drawback of both the aforementioned work is that they are reducing the input arrival rate, rather than controlling the existing congestion.

2.2.2.2 Congestion control by Re-routing

With the increasing complexity of network structure, it is difficult to find the best path [30]. In [38], two approaches are implemented using neural networks to resolve the routing and congestion control problem. The first approach uses feed-forward neural networks. This neural network indicates whether the link is congested or not by receiving input, such as the average number of packets, the variance of packets and the polling flag of sending packets. The second approach uses a recurrent neural network to decide the complete path from source node to destination node. Neural network inputs are source node, destination node, link time costs and congestion status (output obtained from first neural network). The best path is obtained as an output through several iterations. Both approaches are applied to two different network topologies and demonstrated promising results.

Prediction models are much more efficient as compared to the mathematical models [39]. Mohan et al. [39] has implemented two approaches to predict the congestion free path. In the first approach, association rule mining and traditional artificial neural network are used. Association rule mining defines the constraints, rules and statements derived from the data. Neural network takes the input like packet drop, response time and node degree, and yields output as the congestion weight that is used to determine the best path. The second approach is an improved version of

the feed-forward neural network, called self-motivated functional link feed-forward neural network. With an improvement in the architecture, the neural network was trained with additional inputs to give the best reliable path. Finally, the two approaches have been compared to prove that the prediction error of the proposed work is comparatively less as compared to the traditional feed-forward neural network.

Barabas et al. [40] incorporated neural network with multipath routing algorithm (Situation Aware Multipath algorithm) to improve the performance of the congested network. A comparison of proposed work has been made with OSPF and EMP routing protocols. Through experimental results, it is proved that the percentage of lost packets is reduced significantly and hence the performance of the network is improved.

2.2.3 Fuzzy Logic

The fuzzy logic technique has been used by many researchers to predict the network congestion [42–44]. In this context, fuzzy logic scales the degree of congestion, rather than defining complete congestion or not. Xiang et al. [42] developed a fuzzy neural network[7] to predict the arrival rate of traffic in future. After predicting arrival rate, the queue length is calculated by Lindley equation [45]. If the queue length is estimated to overflow, encoding rate of the source is reduced by 25 percent of the current sending rate. The fuzzy logic approach was examined with BP network and no-feedback control method, and concluded that its packet discarding rate is much smaller as compared to these methods. Swathiga and Chandrasekar [44] developed a fuzzy logic system to predict congestion level in wireless sensor networks. The congestion level is scaled as low (A1), medium (A2) and high (A3) (based on threshold values). After a certain interval, each node in wireless sensor networks measures node degree, data arrival rate and queue length. These three values are accepted as input by the fuzzy logic system, and the congestion level is yielded as an outcome. If the congestion level is A1, no control algorithm is applied. However, if the congestion

[7]Fuzzy neural network is a combination of fuzzy logic and neural network.

level is between A2 and A3, adaptive rate adjustment technique is triggered. In this technique, node sends a rate regulation message to upstream nodes and a new data sending rate is generated using current rate, node degree and queue length. NS-2 has been used to implement the proposed technique and compare with Hybrid congestion control [46] in wireless sensor networks. It has been shown that the proposed technique is superior by analysing a number of performance metrics such as average packet delivery ratio and packet drops.

2.2.4 Other Techniques and Applications

A Kalman-filter based prediction technique is proposed in [47], [48]. Haught et al. [47] extended the work done by Stuckey et al. [48] on Kalman-filter based prediction. Stuckey et al. worked with very small network whereas Haught et al. are dealing with complex network structure. Kalman filters are placed on network links to record data periodically. The sample rate of the filter is one second. Periodically, it records the queue size and current arrival rate. It predicts the queue size at next interval based on the current and past queue sizes. So, the arrival rate of the packet is controlled by analyzing the predicted queue size with the help of control algorithm implemented in the network. NS-2 is used to implement the proposed algorithm.

A new system called Network Bandwidth Predictor (NBP) to forecast the network bandwidth is proposed in [49]. NBP uses Network Weather Service (NWS)[8] to gather traffic statistics. The raw data is further processed by Network Traffic Pre-Processor and neural network is trained with the input such as timestamp, minimum and maximum number of bits in one second in that bin size (the period at which user wants to make prediction), average number of bits in one second, and the predicted value. By testing many real-time datasets, NBP prediction mechanism is shown to be superior to NWS. A Graphical User Interface (GUI) is also provided, which gives a report for analysis and accuracy comparison with NWS.

Neural networks have also been used to predict road traffic with high accuracy

[8]NWS is a methodology which measures the hop-by-hop available bandwidth on all links.

[50, 51]. Yu et al. [50] successfully captured bursty nature of traffic by developing a back-propagation feedforward neural network model. Hussein Dia [51] has developed dynamic neural network models (time-lag recurrent network and hybrid networks) to predict short-term traffic. The models are trained with the speed measurements from the historical time intervals. Through experimentation, it has been proved that high degree of accuracy is obtained when speed data was predicted up to 5 (or 15 minutes) into the future.

2.3 Summary

Congestion Control techniques using prediction algorithms are far more superior than general congestion control techniques in terms of network's performance. For congestion prediction, it has been demonstrated that neural network is a very feasible method because of its highly sophisticated learning mechanism and complex computational capability. To predict congestion, the input parameters for a predictive model are generally based on the packet arrival rate and queue length. Many solutions are proposed to control congestion by reducing/controlling the sending rate. But decreasing traffic rate is like avoiding congestion, rather than controlling. More efficient techniques are needed to control the congestion without reducing the existing traffic rate.

Chapter 3

Simulator Architecture, Neural Networks and Cloud-Computing

This chapter contains three sections. The first section explains the design and architecture of the distributed transaction processing simulator (DRTTPS), which is used as a test-bed in this research to predict network congestion. The second section gives an overview of neural networks, and the training process. The last section discusses the importance of deployment of predictive models on the cloud. It further describes ADAPA on the Amazon cloud and the model deployment process.

3.1 The Simulator - DRTTPS

Studying and conducting experiments directly on a real system is not feasible due to cost, time, complexity and error-prone nature. Therefore, simulation is used to understand, model and analyze the system. A simulation model describes the real-system workflow and relationship between different entities. Simulation of a system can be either continuous or discrete [52]. When the state of the system continuously changes over time, it is called continuous event simulation. For example, an air-plane has state variables - velocity and position, which change continuously with respect to time. In discrete event simulation, the state of the system is based on the occurrence

of events, which occur at discrete points in time. The banking system is an example of discrete event simulation, where the total number of customers is one of the state variables. This variable changes only by the occurrence of events such as the arrival of new customer and departure of the customer after being served.

To predict and control network congestion in distributed real-time database system, distributed real-time transaction processing simulator (DRTTPS) is used as a test-bed. DRTTPS [5] is a discrete event simulator developed to simulate distributed real-time transaction-based database system. It enables the user to configure various parameters through its highly interactive graphical user interface. Different protocols such as routing protocols, concurrency protocols, pre-emption protocols and priority protocols can be added to DRTTPS to test and analyze their performance. Its discrete event simulation engine consists of tick (simulation clock), entities, events, event queue, event scheduler and event processor. A tick is a unit used in the simulator to measure discrete amount of time. Events are created by entities and inserted in the event queue based on their execution time [5]. Examples of events in DRTTPS include: sending a message from one node to another, transaction arriving at a node and transaction committing. Event scheduler extracts an event from the top of the event queue and calls the event processor. Once the event is processed completely, the state of the system gets updated. Different simulator modules and their vital features are explained in detail in the following subsections.

3.1.1 Network Architecture

Network is the core component of DRTTPS, having one or more sites connected to each other through wide area network. Each site can have many nodes, and each node can have one or more real-time databases as shown in Figure 3.1. Nodes are connected to each other through local area network and communicate through messages using network connections. These network connections are called links (pipes), which are bi-directional in nature. Each node has its own routing table, where the routing table stores paths to all the receivable destination nodes. If a link gets congested due to

heavy workload, the routing protocol updates the routing table to deliver the message in a timely manner.

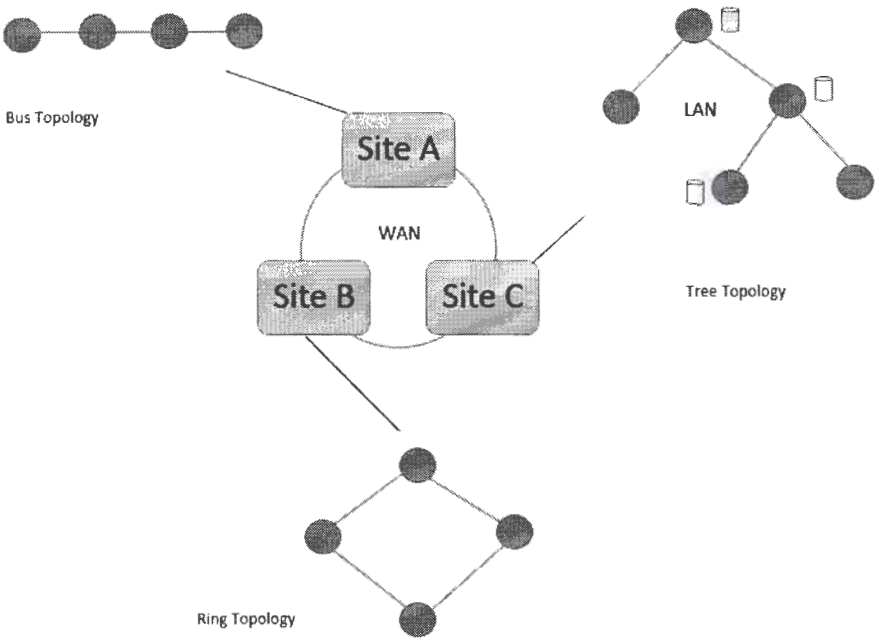

Figure 3.1: Network Architecture [5]

Each link or network connection has the following components:

- Source Node: node initiating network connection.
- Destination Node: final node receiving the messages. A message originating from source node can travel through one or more intermediate nodes to reach its final destination node.
- Bandwidth: the maximum capacity (number of messages) which can be transferred from one node to another in a particular time period. Bandwidth is one of the major factors that affects network performance.

- Latency: time taken by a message to travel from one node to another. High latency network connection suffers long delays.

- Queue Length: number of messages waiting on link's queue to get processed. When bandwidth is fully utilized, messages start queuing. Once queue length starts increasing, network delay is caused.

Nodes' connections to each other and data flowing within a network are determined by the network topology. Some of the network topologies implemented in DRTTPS are ring, star, tree, fully connected and hypercube.

3.1.2 Node Architecture

A node represents a single computer, which can perform computations and communicate with other nodes via messages. Messages can be sent directly to the neighbouring nodes, whereas the messages have to pass through some intermediate nodes to reach non-neighbouring nodes. A node consists of many hardware components such as processor manager, disk manager, buffer and swap disk. These components are explained as follows:

- **Processor manager** manages all the processors in a node. A node can have more than one processor, and each processor can process one page at a time. The number of ticks taken by a processor to process one page is called process time. More than one page can be processed at a particular instance, if hyper-threading feature is enabled.

- **Disk manager** manages the disks of a node. Each disk stores specified set of pages. A disk can read or write one page at a time. The ticks taken by a disk to read or write a page is called access time. The property that enables a disk to distribute data as partitioned or replicated is called data distribution (explained in section 1.2).

- **Buffer** represents the memory storage of a node which temporarily stores data (pages). The page limit of the buffer depends on the buffer **size** attribute. If a page is available in the buffer, transaction accesses the page directly from the buffer instead of reading from the disk.
- **Swap disk** represents the virtual memory of a node. It stores pages when buffer is full.

A node may also have a workload generator which controls the workload of the system. The system's workload can be controlled by varying its various attributes, such as inter-arrival time, slack time, work-size, update percentage and workload size. Any distribution can be selected for these attributes as shown in Figure 3.2.

Arrival time represents the inter-arrival time, which is the time difference between the two transactions arriving at a node. Low inter-arrival time results in high system load because many transactions enter the system within a short duration. The workload generator creates transactions with associated deadlines to complete the operations. Each transaction processes the pages to perform read or write operations. The number of pages accessed by a transaction depends on an attribute called **worksize**. After processing, pages are updated on the disk; determined by an attribute called **update percent** (percentage of transaction's operations required to be updated on the disk). An extra time called **slack** time is allocated to the transaction for completing its operations. The total number of transactions generated by the workload generator depends on the **workload size** attribute.

When a transaction gets delayed (even after passing the slack time) because of inevitable problems like deadlock and congestion, it may get aborted. There is a **transaction time-out** attribute, which decides transaction's maximum waiting time limit during its execution. Another node attribute called **maximum active transactions** represents the maximum number of transactions which can run simultaneously on a node. When multiple transactions are generated by different nodes' workload generators, numerous problems (such as concurrent access, resource sharing, and congestion)

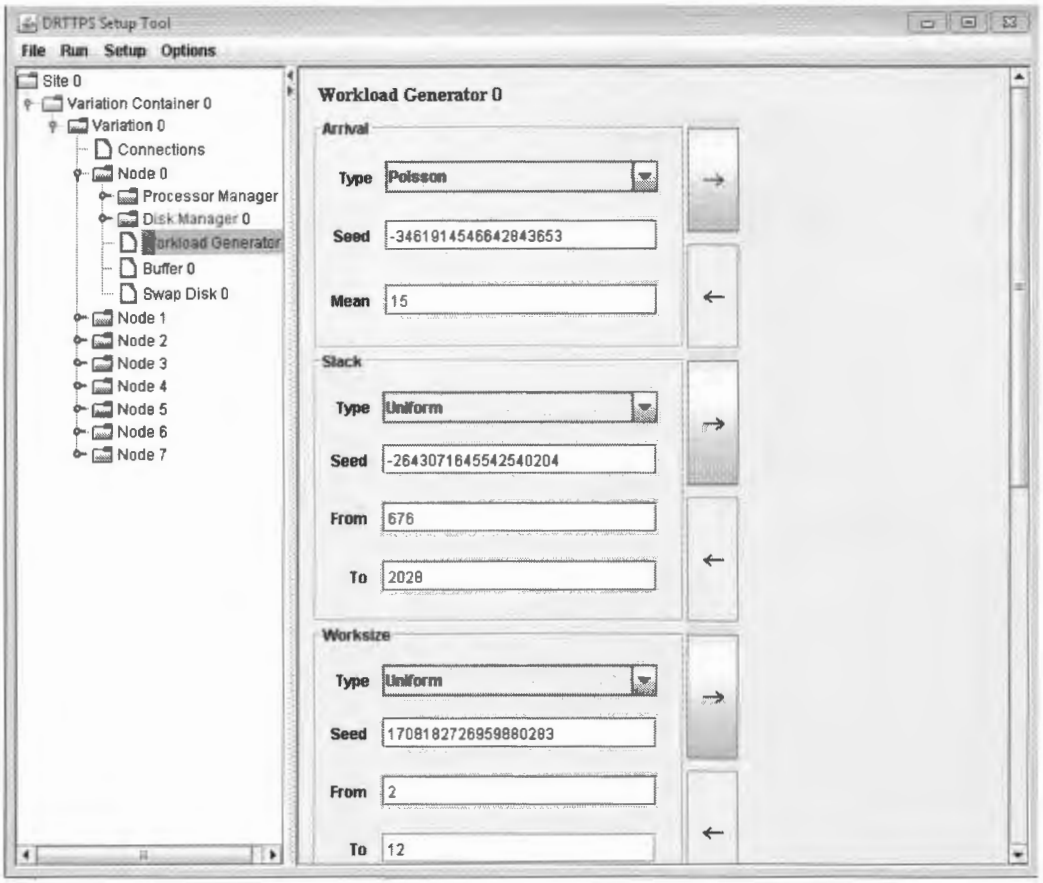

Figure 3.2: Workload Generator Attributes

can arise. To handle these problems various protocols are implemented in DRTTPS at the node level. Each node can run their own protocols based on their simulation environment. Some of these protocols are discussed hereunder:

3.1.2.1 Concurrency Control Protocol

Concurrency control protocols are used to handle simultaneous access of database by different users [13]. The aim is to coordinate concurrent access to the database system. These protocols control the transaction's requests for locks on pages. Transactions need a shared lock (or exclusive lock) to read/access the pages, and an exclusive lock is required to perform the write operation on the pages. There are various concurrency control protocols implemented in DRTTPS such as greedy locking, greedy

locking all copies, speculative locking and adaptive speculative locking [53, 54].

Greedy locking all copies protocol has been used in this research to handle concurrency. In this protocol, when a transaction starts its execution, it requests locks on all the pages which will be required during its lifetime. The transaction does not start its operation until all the locks have been granted to it. Locks are released when the transaction is completed. Greedy locking and greedy locking all copies are similar, except for the fact that the later protocol supports replication across nodes.

3.1.2.2 Preemption Protocol

When two or more transactions try to access the same page, priority inversion may occur [1]. Priority inversion happens when a high priority transaction waits for low priority transaction to commit. Preemption protocols handle these scenarios by controlling the preemption of transactions. The preemption protocols implemented in DRTTPS are listed below:

- **High Priority Preempts**: A transaction holding a lock on page can be preempted only if a transaction trying to preempt has higher priority.
- **Never Preempts**: No preemption will happen.
- **Priority Inheritance**: Priority inheritance is a method for avoiding priority inversion. When a transaction with high priority waits for the transaction having low priority, low priority transaction inherits the priority of high priority transaction in order to complete itself instead of being aborted.

3.1.2.3 Routing Protocols

In a network, nodes communicate with each other via messages. Routing algorithms determine the route of a message from one node to another, according to the network topology. In a fully connected network, routing is simple because there is a direct path from one node to the other. But, topologies like hypercube, ring and tree

have intricate routing due to indirect paths between nodes. Messages from the source node (node sending messages) are routed to intermediate nodes before reaching the final destination node (node receiving messages), and these paths are determined by the routing algorithms. We have selected DSP and RIP for comparison with our proposed NNPR protocol. DSP is a classical routing protocol, whereas RIP is widely used in today's distributed networks. These protocols are described below:

- **Dijkstra's Shortest Path (DSP)** [26] - Dijkstra's Shortest Path Algorithm computes the shortest path between two nodes. The cost of the route is the sum of the latencies of all the intermediate routes from source to destination node. For example, in Figure 3.3 there are many paths from Node 1 to Node 8, but the shortest path between these nodes is [1-2-5-8] with a total cost of 4. The routing table format for Node 1 is shown in Table 3.1.

  In DSP algorithm, once the routing table is set-up it never changes during the simulation run. The cost of the route is not affected by the amount of congestion.

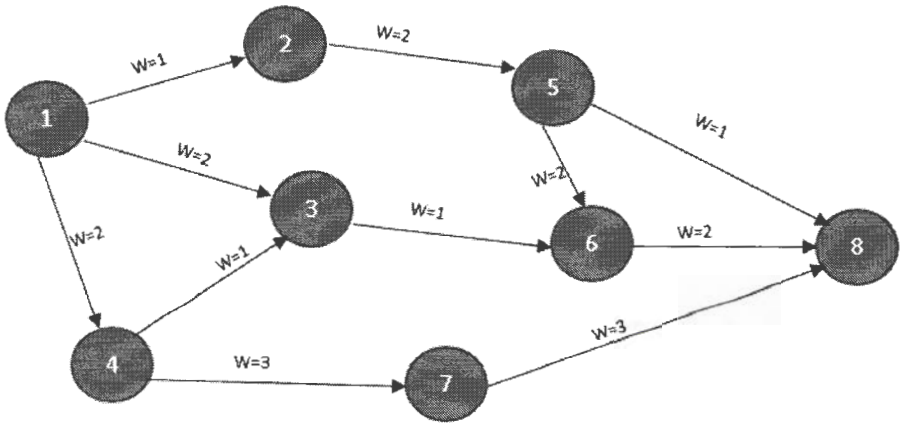

Figure 3.3: Shortest path between Node 1 and 8 using DSP Algorithm

- **Routing Information Protocol (RIP)** - RIP is a distance vector routing protocol [27] in which each node shares routing information with its neighbour-

Table 3.1: Routing table for Node 1

Source Node	Destination Node	Shortest path	Cost
1	2	[1-2]	1
1	3	[1-3]	2
1	4	[1-4]	2
1	5	[1-2-5]	3
1	6	[1-3-6]	3
1	7	[1-4-7]	4
1	8	[1-2-5-8]	4

ing nodes at set time intervals. In turn, the neighbours exchange the routing information with their nearest neighbours, and so on, until all the nodes have the same routing information of the network (this state is known as convergence). While generating a simulation with RIP, a parameter called **updateTime** defines how often each node sends its entire routing information to the neighbouring nodes. Unlike DSP, RIP takes into account both congestion and latency metrics while updating the routing tables.

In RIP, when a path becomes congested, the network does not discover it immediately due to the slow convergence. Another disadvantage of RIP is the **count to infinity** problem. It happens when a network link breaks, and nodes mislead each other by calculating the shortest path to infinity (a node sends the wrong and outdated shortest path calculation to another node, which propagates through the entire network and reaches infinity).

- **Neural Network Prediction-based Routing protocol (NNPR)** - This is the main contribution of this thesis and has been discussed in detail in Chapter 4.

3.1.3 Simulation Set-up

DRTTPS's set-up tool provides the user with a graphical user interface to configure and simulate a real-time distributed database system. It involves setting up site components, network architecture, node structure and other simulation settings. The user can save the configured settings and run the simulation from the set-up interface. It has another important component called variation container, which allows the user to run and compare many simulations simultaneously. After setting up the required parameters, variation tab can be opened to vary the desired parameters. For example, in Figure 3.4, a variation is created by varying the bandwidth parameter. Many variations can be generated through the variation tab.

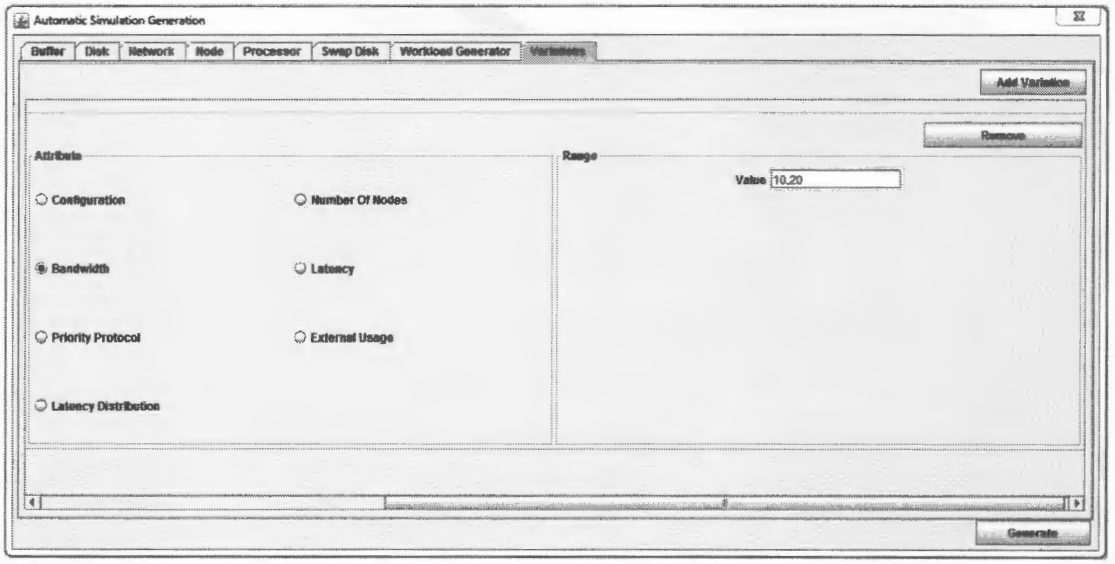

Figure 3.4: Variation Container

3.1.4 Performance Analysis

When a simulation runs, run-time performance analysis can be done through report-tool graphical user interface (as shown in Figure 3.5). There are many perfor-

mance metrics implemented which can be analyzed through this window. PTCT is the main performance metric of DRTTPS, which indicates the percentage of transactions completed on time. Report tool has a feature called loading and opening report, which facilitates the user with a functionality of saving and opening the graphs later for analysis purpose.

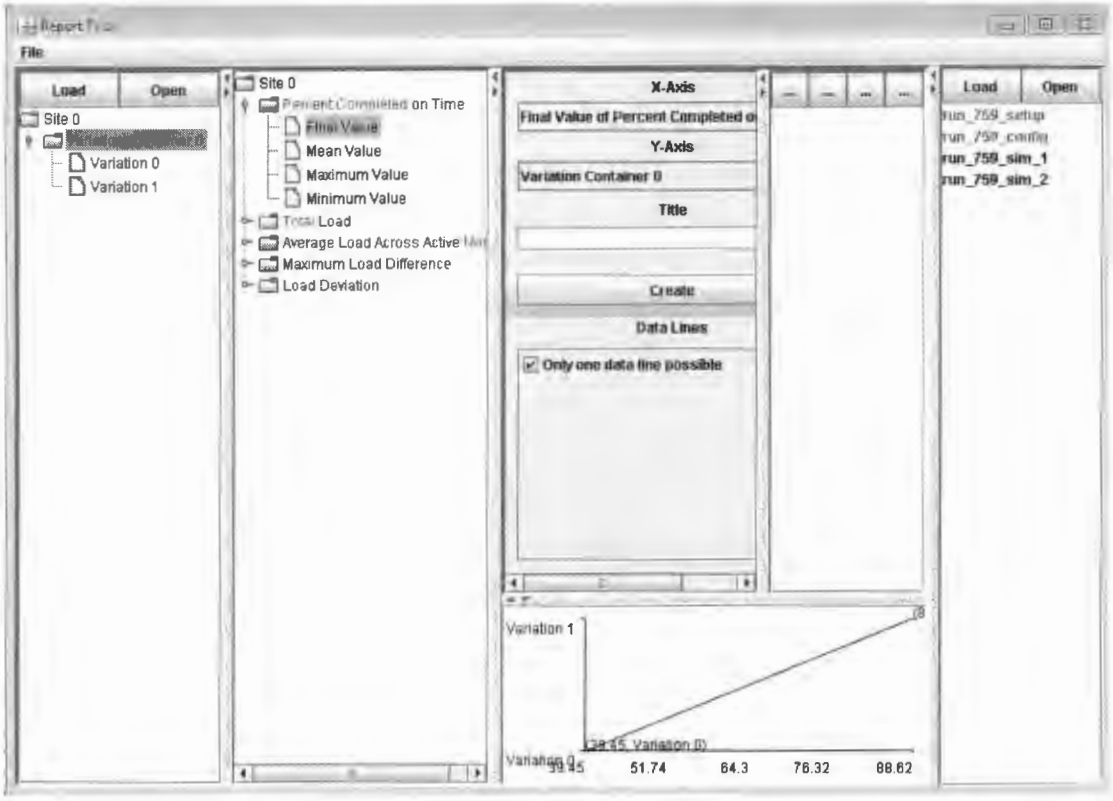

Figure 3.5: Report Tool

3.2 Neural Networks

"Human brain is the most powerful processor and the most efficient" [55]. It has millions of neurons connected to each other in a complex manner. There are multiple layers of neurons interacting with each other. Each neuron in the layer

receives inputs from the previous layer and forwards it to the next layer and vice-versa. Based on the learning and memorization, neurons keep sending signals and finally adjust themselves for the best decision making. Hence, researchers mimicked the human brain's functionality to solve complex problems and achieve the same extent of accuracy. This is called Artificial Neural Network (ANN).

ANN is a mathematical model, which consists of three parts: input layer, one or more hidden layers and output layer [6]. These layers have many neurons connected to each other through edges as shown in Figure 2.5 (This figure is an example of a simple feed-forward neural network). Each neuron consists of numerical information called activation value, which is a function of its input value (input fields and weights). A sample of neuron structure and its working design is shown in Figure 3.6.

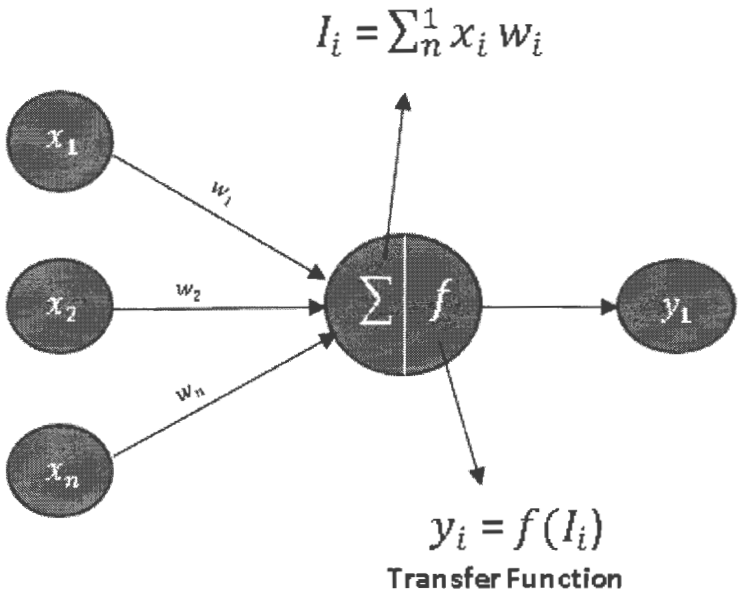

Figure 3.6: Neuron Design [6]

The activation value is high when the input value is large. The activation values are passed from one layer to another, where each neuron aggregates the received

activation values and changes the aggregated value by its activation function. The activation function (also known as output, or transfer function) converts input values to output values. Some of the activation functions, as explained in [6], are shown below:

- Identity Function - Linear (Identity) function does not apply any transformation, which means it is identical to the input.

$$y_i = I_i \tag{3.1}$$

  where I_i is the input, and y_i is the output signal.

- Binary Step - In binary step, output has two states (0 and 1) and these states change depending on the threshold value. If the activation value exceeds the threshold value (T), the output is 1, otherwise 0 (shown in equation 3.2).

$$y_i = f(I_i) = \begin{cases} 0 & \text{if } I_i < T \\ 1 & \text{if } I_i \geq T \end{cases} \tag{3.2}$$

- Sigmoid - The most frequently used sigmoid functions are logistic and hyperbolic tangent [6]. Logistic function is defined by the equation (3.3).

$$y_i = \frac{1}{1 + e^{-\beta I_i}} \tag{3.3}$$

  where β is the slope parameter. The range of logistic and tangential functions are 0 to 1 and -1 to 1 respectively. It is continuous and easily differentiable and is therefore widely used in the models.

- Gaussian - Gaussian function is also called radial basis function. It can be defined by the following equation:

$$y_i = e^{-\frac{I_i^2}{2\sigma^2}} \tag{3.4}$$

  The function value declines when absolute input value increases.

3.2.1 Types of Neural Networks

Neural networks are classified into two types: feed-forward and feed-back [6]. **Feed-forward** is a one-directional neural network, that is, information flows from input layer, through hidden layers (if present), to output layers and not vice-versa. A simple feed-forward neural network has already been explained in section 2.2.2. The most common kind of feed-forward networks are **perceptron** and **radial basis**. **Feed-back** is a recurrent neural network where information can flow forward and backward, which leads to the formation of cycles or loops. The state of the network is changing continuously until an equilibrium point is reached, and therefore it exhibits dynamic behavior. Once equilibrium is attained, the state of the network will not change until the input is changed. This kind of networks can become very complicated and are generally used in systems like motion detection and speech recognition.

Perceptron network has been used in this research to predict network congestion, therefore its detailed design structure is discussed here.

3.2.1.1 Single Layer Perceptron

Single layer perceptron (SLP) [7] is a simple artificial neural network. The input layer sends directly inputs to output layer and there is no hidden layer. The output value is binary and it depends on the threshold value. If the sum of the product of input value and weight is greater than the threshold value, output is 1, otherwise it is 0. When the predicted results do not match with the expected results, it means network performance is not satisfactory. To decrease the network error, weights are adjusted. Since SLP is based on the linear function, it does not understand simple non-linear cases, such as the XOR problem. However, multi-layer perceptron can easily analyze XOR problem.

3.2.1.2 Multi-layer Perceptron

Multi-layer perceptron's structure is similar to single layer perceptron with one or more hidden layers. It is based on the **back-propagation** [7] learning technique.

Back-propagation consists of two phases:

- Forward Phase - In this phase input values are fed to the output layer, and output is computed using sigmoid transfer function.
- Backward Phase - When the performance of the model is not satisfactory, the error is fedback to the network. As a result, network modifies link weights to reduce the error rate. The error of the model usually becomes very small after repeating the same process for several training cycles.

3.2.2 Neural Network Training Process

Training is the most important part of a neural network model. Model's performance is directly related to the training process. It is an iterative process that starts with the preparation of input data. After preparing input data, neural network type (multilayer perceptron or radial basis function network) and structure (hidden layers and neurons) are determined. Network training is started after configuring the parameters such as network type, number of hidden layers, number of neurons, training time, and the training cycles. Once the training is completed, the analysis of results can guide the user to identify issues with the input data or other neural network settings. The complete process is repeated until satisfactory results are obtained (as shown in Figure 3.7).

There are few steps that should be performed before training as shown in Figure 3.7. The steps involved in the training process are elaborated in detail in the next subsections [7].

3.2.2.1 Preparing Input Data

Neural network's performance depends highly on the input data. Ideally, the training data should span the complete input range of the network model. If the input of the model is not within the range of training set, it may not perform well. After deciding the input data and its ranges, data is divided into three sets: training,

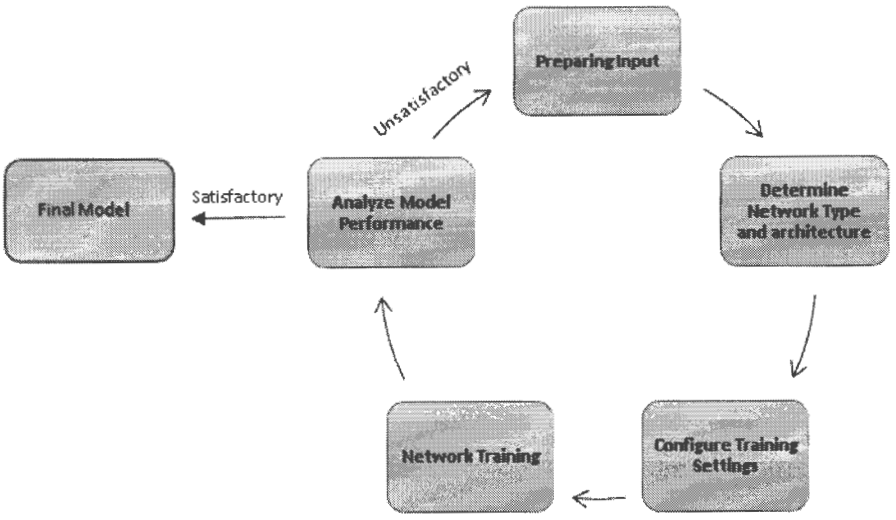

Figure 3.7: Neural Network Training Process [7]

testing, and validation. The training set generally consists of 60-70 percent of the complete dataset; and testing and validation approximately 15-20 percent each. A simple way to create these sets is to randomly divide the complete dataset into three sets. The amount of data needed to train the model can never be estimated ahead of the training because it depends on the accuracy of the model (if accuracy is low, model needs to be trained with more data, otherwise no further training is required with additional data).

Data cleansing or transformation needs to be done before feeding the input to the neural network. By doing so, the neural network can easily extract the meaningful information from data. Data cleansing/transformation may include normalization, handling missing values in data and other transformations. In normalization, data is normalized to fall within a specific range, generally -1 to 1 or 0 to 1. This is done because activation functions in the hidden layers may get saturated (reach a constant value) with the large input numbers. The reason for avoiding saturation is because the derivative of the activation function is almost 0 in the saturation region, which

substantially slows down the network learning. For example, in multilayer perceptron network, sigmoid transfer function gets saturated when the input is large. To handle the missing input data, a possible solution is to replace the missing values with the average value of that particular variable.

3.2.2.2 Neural Network Type and Architecture

The type of neural network (Section 3.2.1) depends on the nature of the problem. When the type is determined, network architecture (number of hidden layers and neurons) needs to be decided. The complexity of the neural network model depends on the number of inputs and the relationships between them. The more complex model requires more neurons and hidden layers.

To determine the number of layers in the network, a conventional way is to begin the training with one hidden layer. If the performance is not satisfactory, then two hidden layers can be used. The network becomes more complicated and unstable when more than two layers are used.

The number of neurons is determined by the complexity of the problem which cannot be defined without training the network. The general way is to commence the training with a large number of neurons (randomly selected) and enable early **stopping** (explained in the following section) feature in the model to avoid over-fitting.

3.2.2.3 Neural Network Training and Configuration

Network training can be started when the input data is prepared and the network architecture is chosen. Firstly, weights are initialized for the complete network. Then, stopping rules are used to prevent the over-fitting problem (modelling errors rather than defining relationships in the data). One way of stopping is to limit the maximum amount of time the model needs to be trained. Another possible way is to specify the maximum number of training cycles required.

3.2.2.4 Analyze Model Performance

In the training phase, the model is trained with the actual outcome. As training continues, the model keeps on adjusting its weight to minimize error. The training and actual outcomes are then compared to find the error of the model. The error can be computed by analyzing statistical metrics such as R-square, root mean square error, and mean absolute percentage error. To analyze the network congestion prediction model results, R-square [56] metric is used in this research. R-square, also known as coefficient of determination, is a statistical measure which defines the fitness of data for a model. The higher the R-square, the better is the prediction model. R-square is defined as follows:

$$R^2 = 1 - \frac{\sum_{i=0}^{n}(y_i - \hat{y})^2}{\sum_{i=0}^{n}(y_i - \bar{y})^2} \tag{3.5}$$

where y_i is the actual value, $\hat{y}$ is the predicted value, and $\bar{y}$ is the mean of actual values.

The aim is to minimize the error rate, which is attained by iterating the complete training process again and again. Once the model results are satisfactory, the network is analyzed against the test-sets. The test-sets may not perform sufficiently due to the following reasons:

- Overfitting - There is a possibility that stopping rule is not configured properly.
- Local Minima - It is possible that network has entered its local minima of the error rate. A widely used solution for this problem is to retrain the network by random weights.
- Extrapolating - Extrapolating means test data does not fall in the training data sets range. In this situation, model needs to be re-trained by combining the test data into trained datasets.

3.3 Cloud Computing

IBM SPSS Modeler 16.0 [57], a data mining and analytic tool, has been used in this research to model neural network. In predictive analytics, creating a data model is the first phase. The second and important phase is the deployment of model, which enables the decision makers to use the model for real applications [58]. Business users need data which is accurate, on-time and easy to understand. Hence, the real-time scoring engine is needed so that timely decisions are made and new business rules are derived.

The deployment of predictive model is a complicated task because it can be very resource and time-consuming. For real-time applications, when results from the predictive model are delayed, it is of no value because it is possible that when a model is deployed business rules are changed.

ADAPA (Adaptive Decision and Predictive Analytics) [58] scoring engine has been used in this research for real-time scoring. It provides cloud computing capabilities and open standards, which facilitate quick deployment of the model by integrating with data mining platform (SPSS Modeler). ADAPA uses Predictive Model Markup Language (PMML) [59] to deploy the models from data mining tools to cloud. PMML offers open standards for the predictive models. It is an XML-based language, which represents data mining models, business rules, input data, and data transformations.

```
</TransformationDictionary><NeuralNetwork activationFunction="tanh" algorithmName="MLP" functionName="regression">
<MiningField importance="0.019100048153744" name="Latency"/>
<MiningField importance="0.010967619999791" name="Max Active Transactions"/>
<MiningField importance="0.488057103834506" name="Number of Messages"/>
<MiningField importance="0.040715584056622" name="Pipe"/>
<MiningField importance="0.343857909410228" name="Queue Length"/>
<MiningField importance="0.011459496919937" name="Source Arrival Time"/>
<MiningField importance="0.015883957603419" name="Source Update Percentage"/>
<MiningField importance="0.014231532042607" name="Source Workrate"/>
<MiningField importance="0.016122403484521" name="Tick"/>
<MiningField importance="0.014222063663529" name="Total Bandwidth"/>
<MiningField importance="0.025403162363643" name="Used Bandwidth"/>
<MiningField name="Future Queue Length" usageType="predicted"/>
```

Figure 3.8: PMML Sample Code

Figure 3.8 shows a small PMML code fragment of a trained neural network model

(the model trained to predict network congestion) developed in SPSS Modeler. Training algorithm, function name and activation function used in the model are represented in the initial line of code. The subsequent lines specify network inputs and their **importance** values denoting predictive contribution, and the last line denotes that 'Future Queue Length' is the predicting field.

3.3.1 Model Deployment Process

The steps involved in the deployment of the model on ADAPA are shown in Figure 3.9 [58]. These steps are explained as follows:

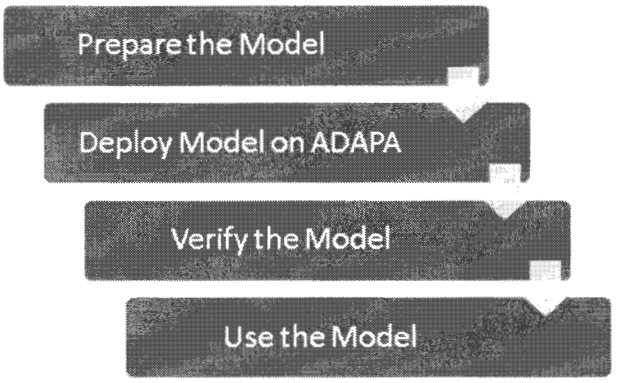

Figure 3.9: Predictive Model Deployment Process

1. In this research, the neural network model is developed in SPSS Modeler. SPSS Modeler can export its model in PMML format, which is interpreted by ADAPA. Once the model is finalized, it is deployed on ADAPA so that real-time scoring can be done.

2. Model is deployed directly by uploading PMML file on the ADAPA console. The ADAPA console is an interface that allows user to upload models and rule-sets.

3. After model deployment, ADAPA results need to be verified. For verification, a CSV data file (containing input data) is uploaded on the ADAPA console.

ADAPA processes the file and provides the predicted results in a new file, which can be downloaded and analyzed. To verify the ADAPA results, there is a need to compare ADAPA results with SPSS Modeler results. If both the results are identical, the model is verified.

4. Once the model deployment is verified, the model can be used for scoring. One way of scoring is through ADAPA console as explained in the above step. Another way is real-time scoring through web service. Web service uses XML to interpret the data. Data can be easily transferred from one application to another through SOAP standard [60]. ADAPA web service uses Java Data Mining (JDM) standard to process the models defined in PMML [58]. Web service properties are configured in Web Service Description Language (WSDL) file. WSDL is defined by JDM standard. The detailed explanation of the implementation of web service is provided in Chapter 4.

3.3.2 ADAPA in the Amazon Cloud

ADAPA is set up and installed on the virtual server in the Amazon Cloud. It is called an **ADAPA instance**. The instance can be launched on demand, and terminated whenever required. Each instance has its own secure server with pre-installed ADAPA. When an instance is initiated on the cloud, it can be accessed from any application through web services.

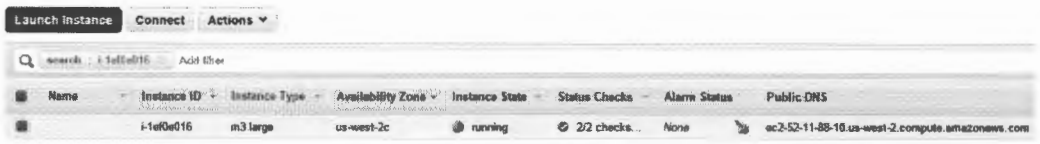

Figure 3.10: ADAPA Control Centre Interface

Amazon Web Service (AWS) account is required to access ADAPA Control Centre (ACC), which handles security and manages ADAPA instances [58]. In terms of security, the instance has its own secure server, which does not allow the user to start

an instance without its access keys. Instance is only accessible via HTTPS protocol, and both ADAPA and web service are password protected. When an instance is terminated, the model information and data are permanently deleted. ACC provides an interface, which shows the status of different instances as shown in Figure 3.10 (running, stopped and terminated). There are different types of instances: small, large and extra-large (in terms of memory and processing power). An instance can be chosen depending on the processing requirement of the model.

Chapter 4

Prediction Model and Implementation

This chapter explains the neural network prediction model developed to predict network congestion. Furthermore, it provides the implementation detail of different classes implemented in DRTTPS. These classes briefly explain the generation of the trace, integration of ADAPA into DRTTPS, and routing mechanism.

4.1 Network Congestion Prediction Model

As defined in Section 1.4, congestion is directly related to the queue length (Equation 1.1) i.e. when the queue length increases, congestion also increases, resulting in a low PTCT (percentage of transactions completed on time). Hence, the predictor attribute in this research is **queue length**. The aim is to predict the queue length of each link and reduce the queue length by re-routing the messages. A neural network model is developed to predict queue length of each link. This queue length is predicted 100 ticks ahead (the reason to chose 100 ticks is to reduce the overhead of prediction module).

4.1.1 Determining Inputs of Network Model

DRTTPS has been used as a test-bed in this research, hence it is important to analyze its primary parameters affecting congestion. To determine the primary parameters affecting congestion, numerous experiments are performed with different parameters. The parameters with significant impact on congestion are selected as the inputs of neural network. Through experimentation, it was determined that the primary parameters affecting congestion are bandwidth, max active transactions, update percentage, latency and work-size (as shown in Figure 4.1, 4.2, 4.3, 4.4, 4.5). The input range for each parameter is chosen in a way that it shows observable results. The aim is to train the neural network with different types of congestion loads i.e. high, medium, low, and no congestion. These loads can be generated by varying different parameters in the simulator.

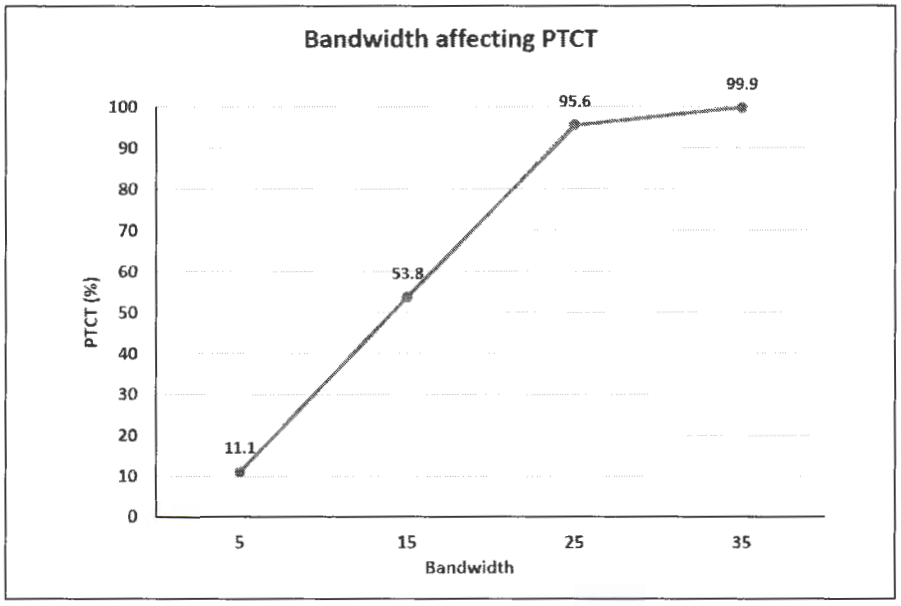

Figure 4.1: Bandwidth vs PTCT

Figure 4.1 illustrates that bandwidth is a major factor impacting congestion (congestion is inversely related to PTCT). A large number of transactions are aborted

when bandwidth is 5 because of excessive queuing of messages on the nodes' links. As the bandwidth increases, the message queues are drastically reduced, and PTCT climbs to almost 100 percent with a bandwidth of 35. Hence, the bandwidth range chosen for neural network training is 5-35.

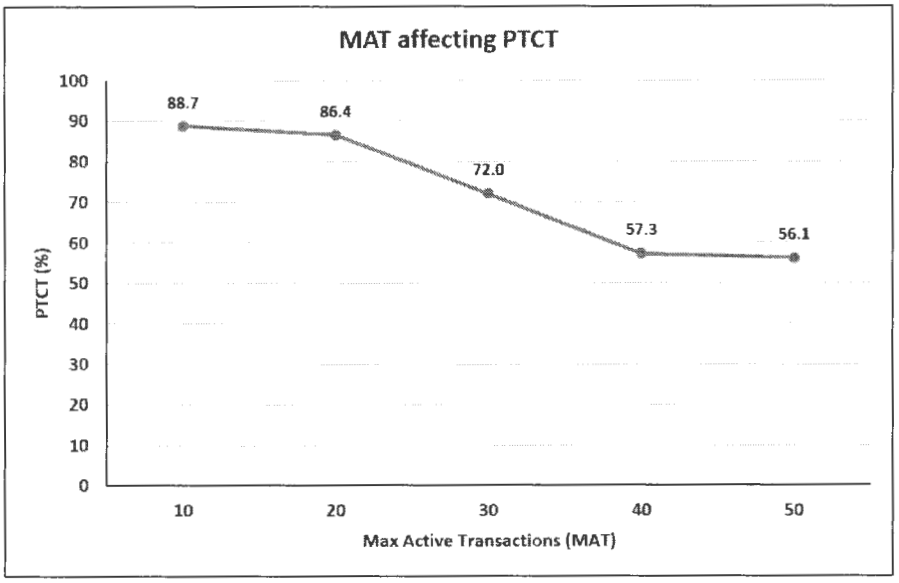

Figure 4.2: MAT vs PTCT

When the max active transaction (MAT) increases, congestion of the system increases (Figure 4.2). This is because, the transactions running concurrently in the system cause a large number of messages to flow within the network, which in turn lead to queuing up of messages. There is no observable difference beyond MAT 40, because it saturates the total number of transaction workload of the system in a short duration (total transactional workload is 150). Therefore, the range of MAT is selected as 10-50.

Effects of update percentage parameter on the system load are shown in Figure 4.3. This parameter indicates the percentage of write operations in transaction processing. Write operations block other transactions because of the transaction's exclusive lock on data. Transactions' execution time increases because of the blocking, and hence

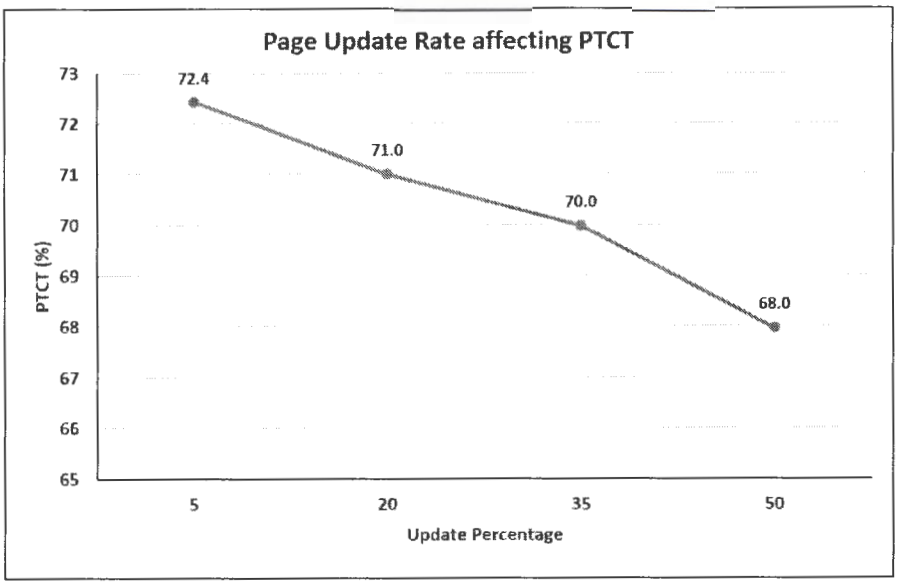

Figure 4.3: Update vs PTCT

transactions do not complete before their deadline. The chosen range of update percentage is 5-50 percent. The input range is not chosen beyond 50 % update rate because the linear decline of PTCT (Figure 4.3) is correctly captured by the neural model (which is validated in Section 5.1.2).

Latency and work-size are the other two parameters which impact PTCT (Figure 4.4 and 4.5). Increased latency results in delaying of the messages to reach the destination node. This causes more transactions to miss their deadlines and reduces PTCT. Work-size exhibits a similar trend. When the number of pages required by a transaction increases, it sends more messages to access the pages, resulting in congestion within the network. The chosen range for work-size is 2-8 because it shows a major variation of PTCT within this range. After work-size 8, PTCT decreases at a slower pace.

Finally, arrival time also affects the congestion of the system. In DRTTPS, each node has a different inter-arrival time to reflect real-time scenario (arrival rate scaling). Inter-arrival time is generated on each node by a random number generator within a

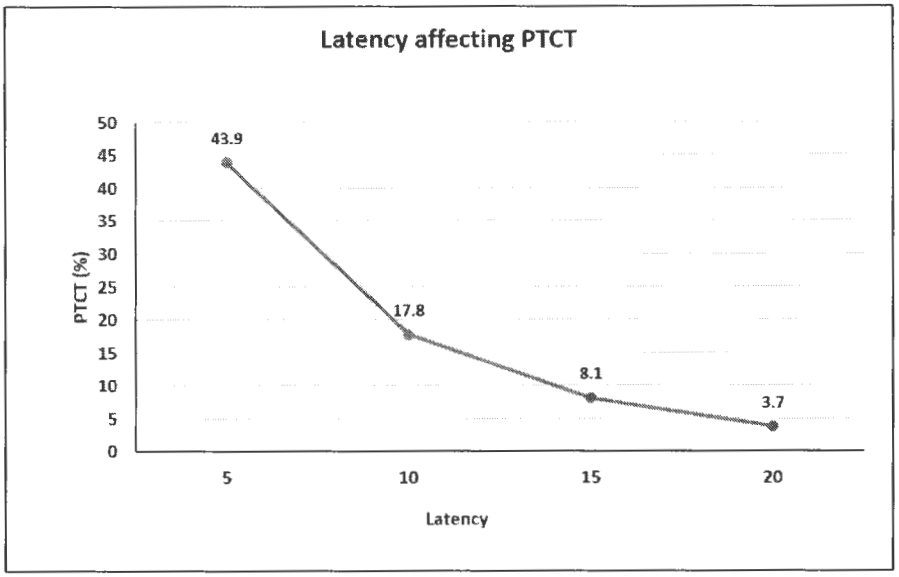

Figure 4.4: Latency vs PTCT

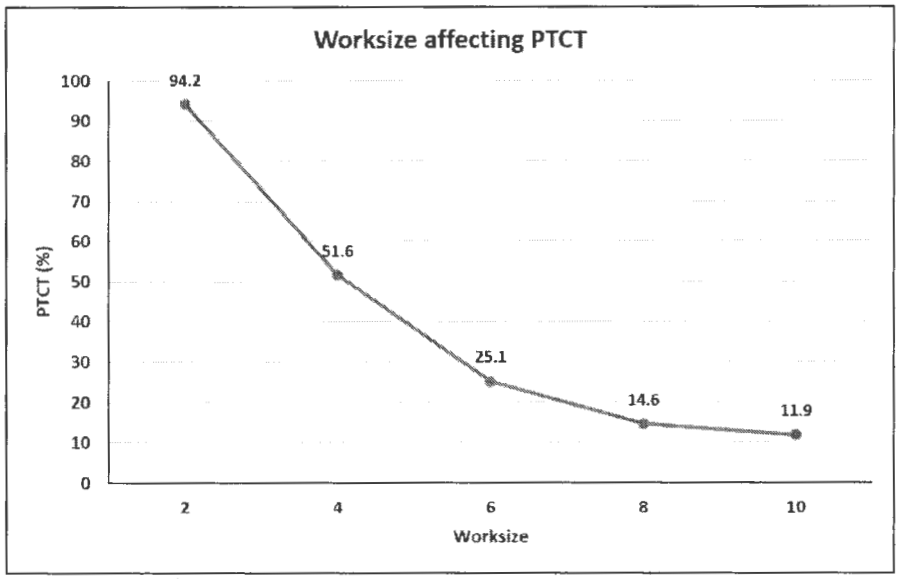

Figure 4.5: Worksize vs PTCT

range of 5-300 ticks using Poisson distribution.

The parameters configured in DRTTPS are shown in Table 4.1. After conducting different experiments (explained above), it is concluded that the inputs of neural net-

Table 4.1: Parameters Settings

Parameter Type	Parameter Name	Value
Network	Topology	Hypercube
	Number of Nodes	8
	Bandwidth	5 - 35
	Latency	5 (Poisson Distribution)
Node	Max Active Transactions	10 - 50
	Disk Count Per Node	20
	Disk Access Time	35 ticks
	Buffer Size	100
	Swap Disk Access Time	35 ticks
	Transaction Process Time	5 ticks
	Replication Protocol	Full Replication
	Routing Protocol	DSP
	Concurrency Control Protocol	Greedy Locking all Copies
	Priority Protocol	Earliest Deadline First
	Preemption Protocol	Higher Priority
Workload Generator	Inter-arrival Time	15 - 300 ticks
	Scale Arrival Rate	Enabled
	Slack Time	676 - 2028
	Work-size	2 - 8 pages
	Update Percentage	5 - 50 %
	Total Transaction Workload	150

work are latency, tick, link, total bandwidth, used bandwidth, queue length, number of messages, update percentage, work-size, arrival time and maximum active transactions. Output/Target is future queue length.

4.1.2 Neural Network Model

A periodic trace file is generated from the simulator, which contains all the neural network inputs. The trace file stores the snapshot of each link at every 100th tick. For example, the snapshot of link 6#2 and 7#3 at tick 200 is displayed in Figure 4.6. Trace file depicts that at tick 200, link 6#2 has received a total of 39 messages i.e. 195 message units (it is assumed that each message is of 5 message units), out of which 180 message units are queued and 15 are in the link (because total bandwidth is 15). Trace file further informs that link 6#2 will have 175 message units queued at tick 300.

Latency	Tick	Pipe	Total Bandwidth	Used Bandwidth	Queue Length	No. of Messages	Update %	Worksize	Arrival Time	MAT	Future Queue Length
5	200	6#2	15	15	180	39	25	4	130	30	175
2	200	7#3	15	5	0	1	25	6	30	30	140

Figure 4.6: Snapshot of links at 200th Tick

Various simulations have been run to create different congestion scenarios by varying parameters like inter-arrival rate, max active transactions, update percentage, work-size, and latency. The neural network model is trained with 30 simulation runs having different congestion loads. The congestion load of a simulation run is the average time taken (in ticks) to clear the queue length. Four ranges of congestion load have been created, i.e. negligible, low, medium, and high congestion load (Table 4.2). These ranges are defined by analyzing the average congestion load of a simulation run. Table 4.3 displays the simulation runs used to train the neural network. Training runs have different kinds of congestion loads i.e. negligible, low, medium and high.

The neural network model is developed in SPSS Modeler 16.0 [57] (Figure 4.7). It

Table 4.2: Congestion Load Ranges

Congestion Load	Avg. Congestion Range (in Ticks)
Negligible	0 - 2.99
Low	3 - 15.99
Medium	16 - 30
High	>30

Table 4.3: Neural Network Training Simulation Runs

Max Active Transaction	Update (%)	Bandwidth	Avg Congestion	PTCT (%)	Congestion Load
10	5	5	25.32	60.7	Medium
		30	1.00	99	Negligible
	25	10	15.48	70.67	Low
		20	3.98	93.43	Low
	40	5	36.50	53.32	High
		25	2.15	96.18	Negligible
20	10	15	15.26	75.77	Low
		25	3.18	98.44	Low
	30	10	28.45	52.33	Medium
		30	1.82	99	Negligible
	45	5	57.14	20.84	High
		20	6.81	90.45	Low
30	15	15	15.82	68	Low
		20	7.33	81.48	Low
	20	20	6.21	77.6	Low
		25	3.06	94.24	Low
	35	5	70.38	9.93	High
		30	1.87	98.69	Negligible
40	10	10	32.79	29.62	High
		15	18.5	46.66	Medium
	25	5	69.05	7	High
		25	3.26	84	Low
	50	20	7.53	64.83	Medium
		30	2.10	95.53	Negligible
50	5	5	72	7.12	High
		30	1.77	99	Negligible
	30	20	7.48	80	Low
		25	3.82	94	Low
	45	10	29.54	16.97	Medium
		15	17.57	32.8	Medium

has different types of nodes, which are responsible for different kinds of functionalities. These nodes are explained as follows:

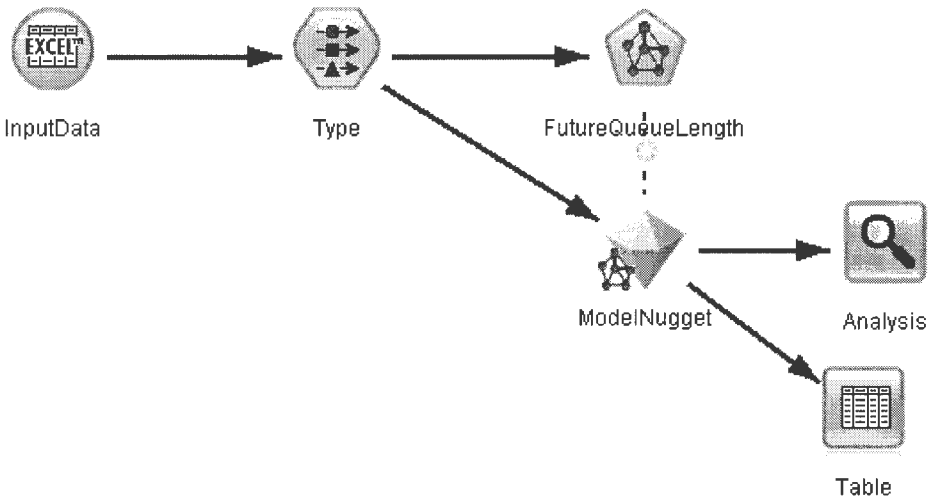

Figure 4.7: Neural Network Model

- **InputData** node is the excel source node, which imports data from the periodic excel trace file.
- **Type** node specifies the role (input, target, or both) and format of input fields (Figure 4.8).
- **FutureQueueLength** is the neural network modelling node, where all the network settings are configured. The model parameters settings configured in the model are specified in Table 4.4. The type of neural network is **Multilayer Perceptron** with 2 hidden layers. The first hidden layer has 10 neurons and the second hidden layer has 5 neurons. Determining the number of hidden layers and neurons is an iterative process (explained in Section 3.2.2.2). The number of training cycles which are used to train multilayer perceptron network is 250.

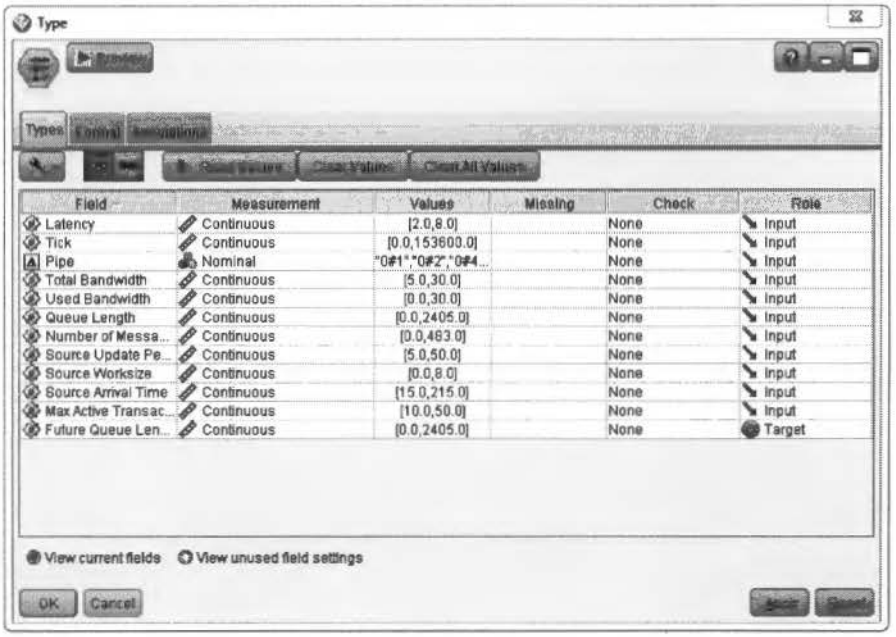

Figure 4.8: Type Node

Overfit prevention set is 50%, which means 50% of the total data is separated from the modelling data so that the network does not model errors in the system.

Table 4.4: Neural Network Settings

Neural Network Parameter	**Value**
Type	Multilayer Perceptron
Hidden Layers & Neurons	Hidden layer 1: 10 Hidden layer 2: 5
Training Cycles	250
Overfit Prevention set (%)	50 %

- When the model completes its execution, a **model nugget** (diamond-shaped) is created (Figure 4.7). This nugget contains the complete model information (consisting of rules and equations), which is used for scoring and analysis of the data. Model summary and details can be browsed by opening this nugget. The

summary view shown in Figure 4.9 displays the model summary table and a network quality chart. The model summary gives the details regarding network architecture and other training parameters. Network quality chart displays the final accuracy of the model, which is defined by R-square (explained in Chapter 3). The accuracy (R-square) of the neural network model is 92.7%. Various other details of the model can be browsed through the nugget, for example - the nugget shows the predictive importance chart, which helps to identify the relative importance of each predictor in the model (Figure 4.10).

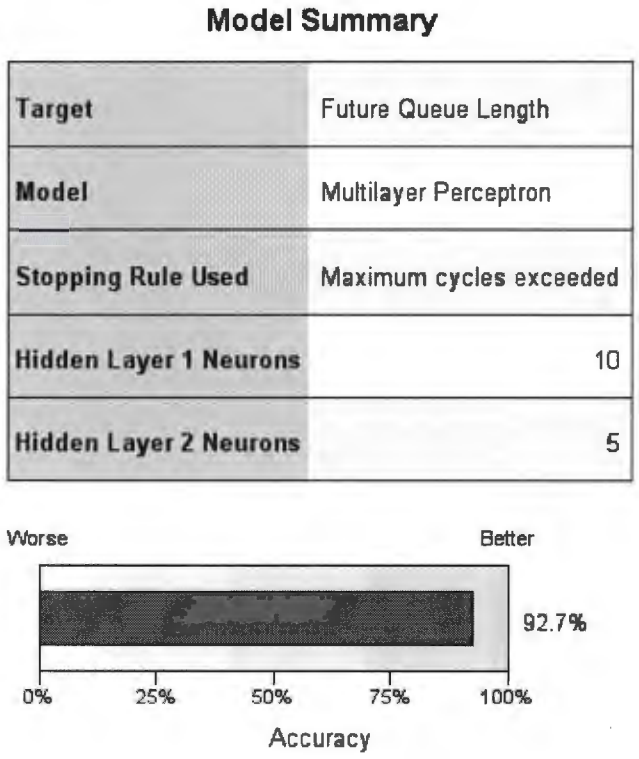

Figure 4.9: Neural Network Model Summary View

- Finally, in the analysis node, training outcome and actual outcome are analyzed

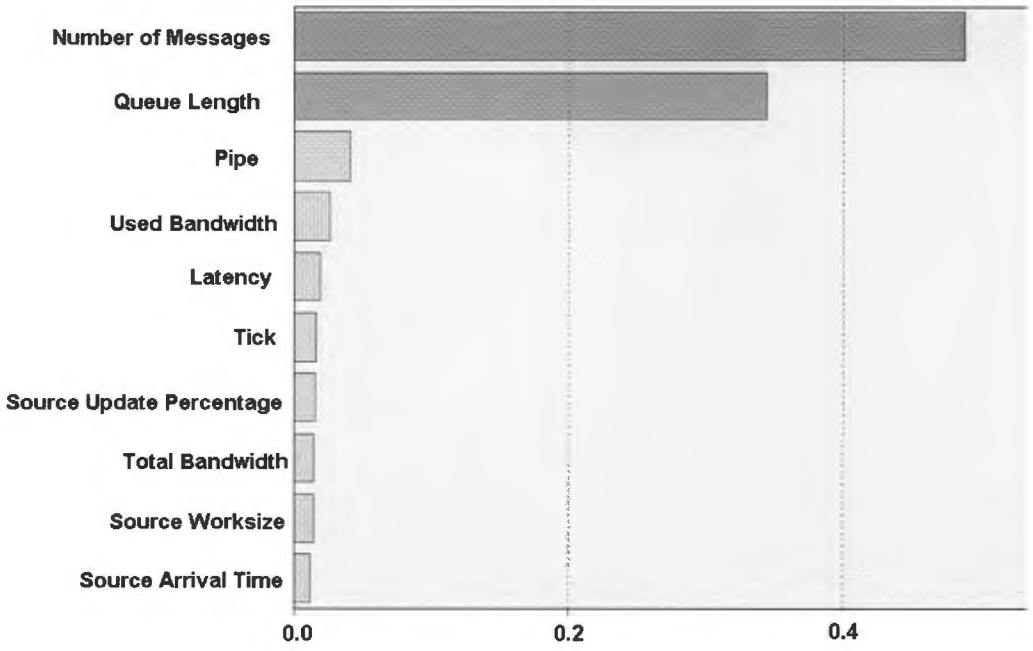

Figure 4.10: Predictive Importance Chart

to find the error of the model. The statistical measure used here is R-square. The higher the R-square, the better is the prediction model.

Determining neural network architecture and other settings is an iterative process. After the first training cycle, the results are analyzed through model nugget and analysis node. This analysis guided us to identify issues in the network settings and helped us to reconfigure the network parameters. This process is repeated again and again until satisfactory results are achieved (R-square is above 90%).

4.2 Implementation

In this section, the implementation details of different modules are explained. Generally, simulation tools have underlying assumptions which simplifies the modelling and design of the system. To design the Neural Network Prediction-based Routing (NNPR) protocol, the following assumptions have been made:

- Hardware failures cannot occur.
- Queue length is infinite.
- All resources are acquired prior to the start of transaction, which means deadlocks cannot occur.
- Once a transaction is committed, it cannot be rolled back.

Figure 4.11 illustrates the overall architecture of our research work. The modules for this framework are discussed in the following sub-sections.

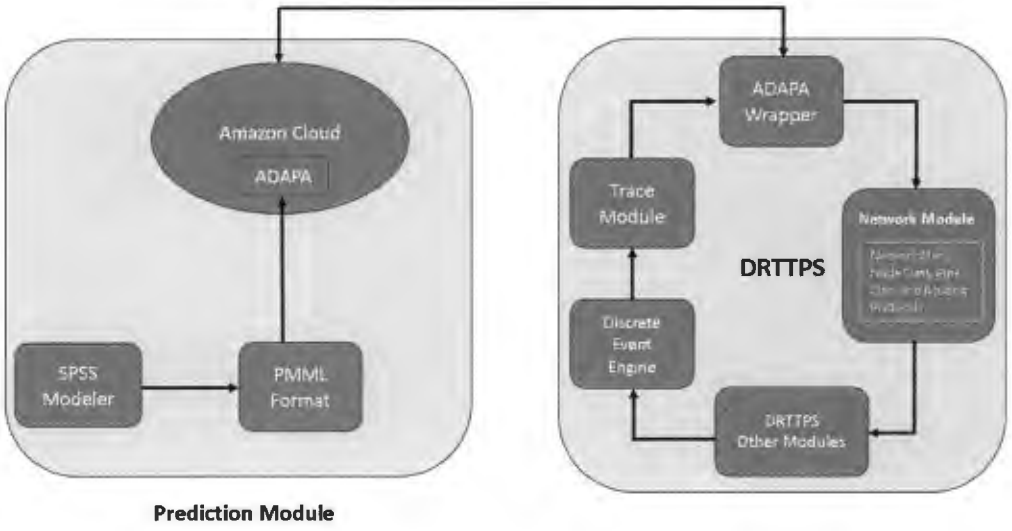

Figure 4.11: Architecture

4.2.1 Prediction Module

Neural network prediction model is developed in SPSS Modeler [57], which is explained in Section 4.1.2. SPSS modeler exports a file in PMML format containing the neural network model, which is then uploaded on the ADAPA instance running on Amazon cloud (explained in Section 3.3). ADAPA wrapper (in DRTTPS) calls the

prediction model through web services and loads the predicted results in the network Class.

4.2.2 Trace module

Simulation is used to model the dynamic systems and collect statistical results. Some simulation tools save the results in a data structure produced at run time, while others save results in a trace file. In our case, simulation execution is saved in trace files due to neural network training purposes as explained in Section 4.1.2. There are two different trace files generated from the execution of a simulation run. The first trace file includes the information of all the simulation ticks, wherever the events are generated. After completion of each simulation, this file is used to calculate the future queue length. Future queue length is the snapshot of link's queue length few ticks ahead, for example - if link A has queue length of 10 messages at tick 5, then this queue length (10 messages) will be considered as a future queue length at tick 0, as long as there are no intermediate events between these two ticks. The second trace file records the data periodically, for example - the first snapshot of the link is saved at 100th simulation tick and next will be saved at 200th simulation tick (if snapshot interval is 100 ticks), and so on. This interval (difference) can be set in the simulation set-up.

Trace module is composed of multiple classes such as *Tracer, XLSWriter, Converter, EntryBuffer and PipeEntry* (Figure 4.12). Each class is responsible for different tasks including preparing trace files, loading trace files and sending the status of each link to ADAPA for prediction. *Converter* class prepares the periodic trace file. *PipeEntry* loads the link's parameters at specific tick, and pushes the data into *EntryBuffer*. *XLSWriter* class fetches the data periodically from the *EntryBuffer* class and records in the trace file. In the next sub-section, the main routines of *Tracer* class are explained.

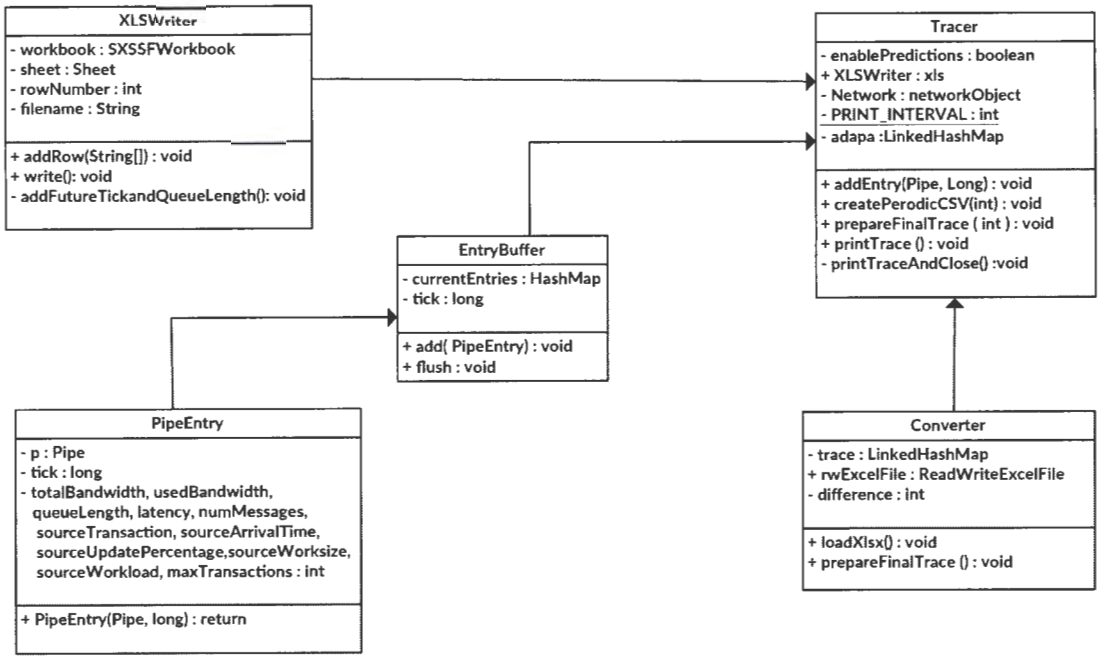

Figure 4.12: Trace Module Class Diagram

4.2.2.1 Tracer Class

This class is responsible for storing snapshot of each link at different simulation ticks. Simulation's core classes store the event details in a HashMap data structure. After 100 simulation ticks, all the events are written to the trace file. Each execution of DRTTPS generates a large number of events which makes it difficult to store all events in the memory and generates an out of heap memory exception. The reason for writing the trace file after 100 ticks and clearing up the events in data-structure is because of the memory exception. The tick interval can be set to a number that will ensure that the memory exception doesn't occur. The main functions of this class are discussed as follows:

- **createTrace** method writes the events into the trace file, and enables the prediction module depending on the selection of a routing protocol. This method checks the routing protocol's name and enables the prediction module when

NNPR is selected. After 100 simulation ticks, this method loads the predicted results in the network class and handles the congestion by updating the routing table. A code snippet describing this functionality is shown below:

```
private boolean enablePredictions = true;
private Simulation simObject;
long tick;
private final int PRINT_INTERVAL = 100;
if (tick - this.lastPrint >= this.PRINT_INTERVAL)
 {
   this.lastPrint = tick;
   RoutingProtocol rp=(RoutingProtocol)networkObject.getNodes().get
       (0).getAttribute("routing_protocol");
    if (enablePredictions && (rp.getNameOfProtocol().equals("NNPR")
        ))
            {
              createPerodicCSVAndPredict((int)lastPrint);
              handleCongestion();
            }
    printTrace();
 }
```

- **createPerodicCSVAndPredict** method prepares the data structure which holds the periodic trace of each simulation run. It calls the ADAPA web services (Section 4.2.3) and loads the predicted results into the network class, so that the results are available to each node. Each call loads the predicted results 100 ticks ahead of the simulation. For example, if a call is made at 100th tick, then it will load the results for each link at 200th tick.

- **handleCongestion** method is responsible for handling the congestion and rerouting the messages along the best path, to improve the performance by reducing congestion. It iterates through all the predicted results and enables the re-routing on those nodes which have congestion greater than the congestion limit (the congestion limit is set to 0). It generates an event that executes

a routine at that specific tick when congestion started to build, this event in turn invokes the re-routing based on the predicted values from ADAPA. The event makes a call to *invokeReroutingPrediction* method with arguments (Pipe CongestedPipe, long SimualtionTick, double CongestionValue). In each simulation, another event is generated to call *resetRoutingTable*, which resets the routing table before prediction starts. For example, if current simulation tick is 240 then it will reset the routing table at 299th tick (congestion will be predicted at 300th tick), and this method resets the current routing table to DSP's routing table. It is reset to DSP because neural network is trained with DSP algorithm. Following code shows the logic (all statements are not included to avoid complexity):

```
//logic to reset the routing tables depending on the congestion. For
    loop iterates over the predicted results using ADAPA and generates
    the events based on congestion at different links (pipes).
int congestionLimit = 0;
 for (Map.Entry<String, LinkedHashMap<String, Double>> entrySet :
    networkObject.getPredictedValues().entrySet())
 {
      String pipe = entrySet.getKey();
      LinkedHashMap<String, Double> value = entrySet.getValue();

      for (Map.Entry<String, Double> entrySet1 : value.entrySet())
      {
          String clock = entrySet1.getKey();
          Double congestion = entrySet1.getValue();
          if (congestion > congestionLimit)
           {
// find the object of congested pipe based on the predicted results from
    ADAPA
           Pipe congestedPipe = networkObject.returnPipeWithString(pipe.
              split("#")[0], pipe.split("#")[1]);
```

```
// find the tick when congestion started building based on the
    congestion formula (Equation 1.1)
            double tickToReroute = Double.parseDouble(clock) - (Math.ceil
                (congestion / networkObject.getPipes().get(0).
                getBandwidth()) * congestedPipe.getLatency());
// This will generate an event which will be invoked at specific
    simulation tick.
            networkObject.getEventQueue().addEvent(new Event(
                networkObject, "invokeReroutingPrediction",new Object[]{
                congestedPipe, clock, congestion}, new Time((int)
                tickToReroute)));
            }
        }
}
// A reoccurring event at every 99th tick of simulation to reset the
    routing table
    EventQueue eq = networkObject.getEventQueue();
    if(!eq.hasOnlyRecurringEvents())
    {
        eq.addEvent(new Event(networkObject, "resetRoutingTable", new
            Object[] {}, new Time(99)));
    }
```

4.2.3 Adapa Wrapper

ADAPA [58] is a predictive scoring tool based on the PMML (Predictive Model Markup Language) [59] standard. The user can create the predictive model in SPSS Modeler, export it as a PMML file, and upload it to the ADAPA user interface. Once the prediction model is uploaded to ADAPA, the model's name is specified in the Java code to make web service calls to ADAPA (score the data in real-time). IBM SPSS modeler doesn't provide any API to score the data in real-time and that lead us to use ADAPA.

In ADAPA, the user needs to initiate an instance and generate a default code package based on the instance. Each instance has its own settings in different files

of SDK. In our experiments, the instance is started and the default source code files have been generated and the instance is never terminated. Termination of the instance will make those source code files unusable. ADAPA provides a development kit with sample programs that allows us to send the data in an acceptable format and obtain the predicted values. The primary classes of ADAPA wrapper and their interaction are shown in Figure 4.13. There are two main files which hold the information such as web service URL, user name and password:

- *adapa.properties* holds critical information such as username, password and web URL (URL is called to get the predicted values).
- *models.wsdl* holds the information about the model parameters. Most of them are added by default, but the requesting URL needs to be changed whenever an instance is started.

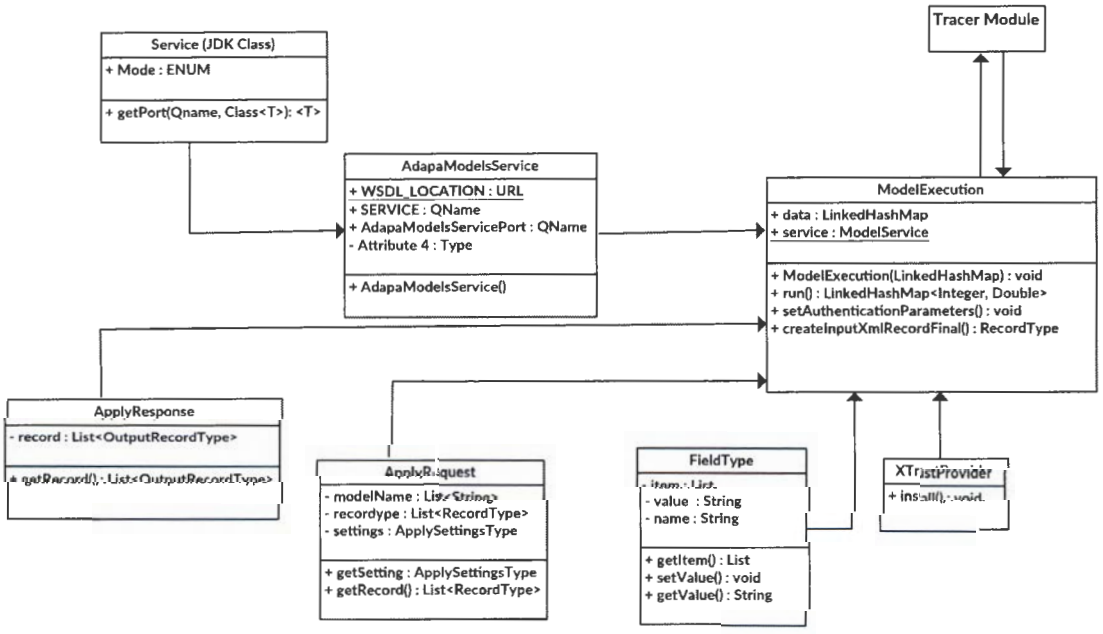

Figure 4.13: ADAPA Wrapper Class Diagram

4.2.3.1 ModelExecution Class

ModelExecution class has four important methods responsible for preparing the XML for scoring, authenticating the web requests, processing the web requests, and storing the predicted values in a data structure. The object of this class is created in the Tracer class which is explained in Section 4.2.2. The tracer class prepares a LinkedHashMap with event entries which are sent to the ADAPA scoring engine. It passes the LinkedHashMap as an argument to the constructor of the ModelExecution class. This way data is available to all the methods in this class. Following are the important methods of this class along with their roles:

- *setAuthenticationParameters()* reads the adapa.properties file and its content. It prepares an object of *BindingProvider* [61] interface which provides the access to the protocol bindings and their associated context objects for request and response message processing in web services calls.

- *createInputXml()* method has many arguments such as tick, latency, pipe (link), total bandwidth, used bandwidth, queue length, update percent, worksize, arrival rate and MAT. This method is responsible for preparing the XML, which holds all the parameters and their values as shown in Figure 4.14. A call to this method prepares an XML record, which holds one row of the trace file data (one row of data per link).

- *run()* method initiates the web service call. However, before making the call, it sets the parameters for user authentication and specifies the model name. Then, it prepares the XML for all the records and sends the records in an XML format. All the parameters are supplied to the object of *ApplyRequest* [61] and that object is passed to *ADAPAModelsService* [61] (both the classes are provided by the ADAPA team). Object of *ADAPAModelsService* provides the access to the response of web service call with its inbuilt function *apply()*, which accepts the object of *ApplyRequest*. The response of web service is a list of values

```
<ns2:applyRequest
    xmlns:ns2="http://www.sementis.com/adapa/ws/models">
    <modelName>Future Queue Length</modelName>
    <record>
        <field name="Latency" value="5"/>
        <field name="Tick" value="100"/>
        <field name="Pipe" value="1#0"/>
        <field name="Total Bandwidth" value="20"/>
        <field name="Used Bandwidth" value="0"/>
        <field name="Queue Length" value="0"/>
        <field name="Number of Messages" value="0"/>
        <field name="Source Update Percentage" value="25"/>
        <field name="Source Worksize" value="0"/>
        <field name="Source Arrival Time" value="15"/>
        <field name="Max Active Transactions" value="30"/>
    </record>
    <record>
        <field name="Latency" value="2"/>
        <field name="Tick" value="100"/>
        <field name="Pipe" value="2#0"/>
        <field name="Total Bandwidth" value="20"/>
        <field name="Used Bandwidth" value="20"/>
        <field name="Queue Length" value="350"/>
        <field name="Number of Messages" value="74"/>
        <field name="Source Update Percentage" value="25"/>
        <field name="Source Worksize" value="0"/>
        <field name="Source Arrival Time" value="30"/>
        <field name="Max Active Transactions" value="30"/>
    </record>
</ns2:applyRequest>
```

Figure 4.14: Input XML Format

stored in LIST data structure. The details of other related classes and its usage is provided in the ADAPA manual [61] and sample codes.

4.2.4 Routing protocols

In DRTTPS, three routing protocols (DSP, RIP and NNPR) have been implemented, and the user can select any protocol in the simulation setup. Standard DSP algorithm supports static routing i.e. the routes are determined and loaded into a node's data structure and they never change during the simulation execution. How-

ever in NNPR, this limitation is removed and the powerful feature of predictions has been added. The routing tables of the congested nodes are updated based on the predicted congestion. The routing protocol (NNPR) has different methods which are responsible for different tasks:

- **initialize** method prepares the routing table. Unlike DSP; it stores more than one path from one node to another node. When the protocol initiates for the first time, it behaves like standard DSP, but when the prediction module generates the predicted results, it starts updating the routing table according to the congestion of the network.
- **updateRouting** method updates the routing table of the node to the best effective paths. This method is triggered (from the **Tracer** Class) when link's congestion is more than the congestion limit. Once routing tables are updated, NNPR sends messages to the shortest path.
- **resetRouting** method is called by the Tracer class to reset the routing table before prediction module is triggered. This method resets the routing table to DSP's routing table.
- **reroutePath** method returns the best path from source to destination node. It accepts two arguments - source node and destination node. It iterates over the routing table and finds the path that will deliver the message in the shortest duration. The shortest path is determined by calculating the number of ticks required to free the queue length.

4.2.5 Execution Flow

The sequence diagram of the execution flow of our work is shown in Figure 4.15. When transactions are generated by node's workload generator, they require locks on different pages existing on the different nodes. Hence, nodes exchange messages amongst themselves to acquire these locks. When the source node needs to send a

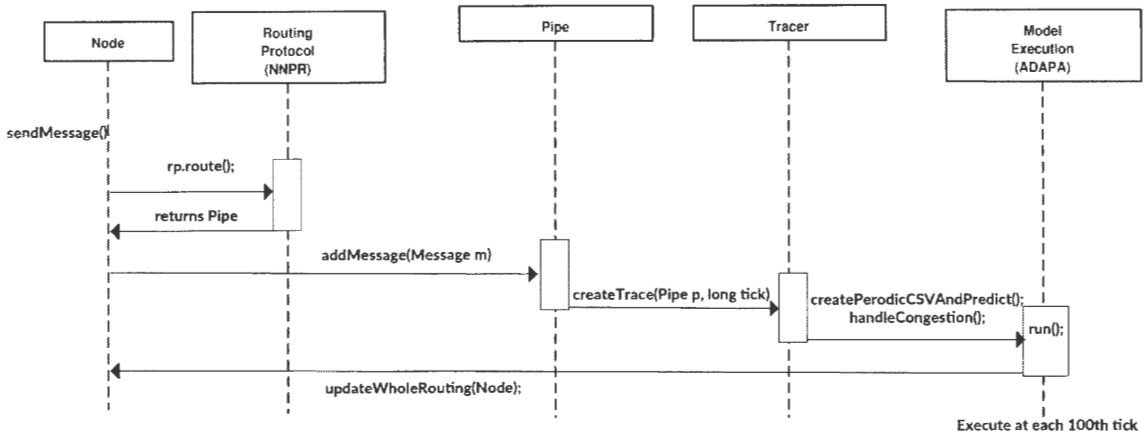

Figure 4.15: Execution Flow

message to the destination node, source node accesses its routing table to get the best path to the destination node. Then, it sends the message to the returned link. Pipe class records all the events in a trace file. In trace class, a method prepares the periodic trace to send an XML record to ADAPA scoring engine and gets the predicted values. Based on the predicted congestion, the routing protocol (NNPR) updates the routing table of all the congested nodes.

4.3 Summary

This chapter explained the neural network prediction model developed in SPSS Modeler 16.0, and discussed the selection of inputs and its ranges through experimentation. Finally, the implementation details of tracer module, ADAPA integration and routing protocol were presented, which included a brief explanation of the Tracer class, XLSWriter class, Converter class, ModelExecution class, routing mechanism, and the execution flow.

Chapter 5

Experiments and Analysis of Results

This chapter presents the experimental results and analysis of the neural network prediction-based routing protocol. The first section demonstrates the accuracy of the prediction model for various congestion loads. The second section provides a comparison between DSP, RIP, and our proposed Neural Network Prediction-based Routing (NNPR) protocol with different parameters, and exhibits that NNPR is superior in performance.

5.1 Prediction Model Testing

In this research, hypercube network topology is chosen because of its superior topological characteristics which includes small diameter (communication delays are less when network's diameter is small), high connectivity, simple routing, and fault tolerance [62]. A 3-dimensional (8 nodes) hypercube (Figure 5.1) is used for the experiments. Since, links are bi-directional in nature, the total number of links in the topology are 24. For example - there are two links between 0 and 1 (bi-directional), link outgoing from Node 0 and Node 1 is named as Link 0-1 and Link 1-0 respectively. Each link may have different congestion load, depending on its attributes (bandwidth and latency) and source node's workload (each node has different inter-arrival rate).

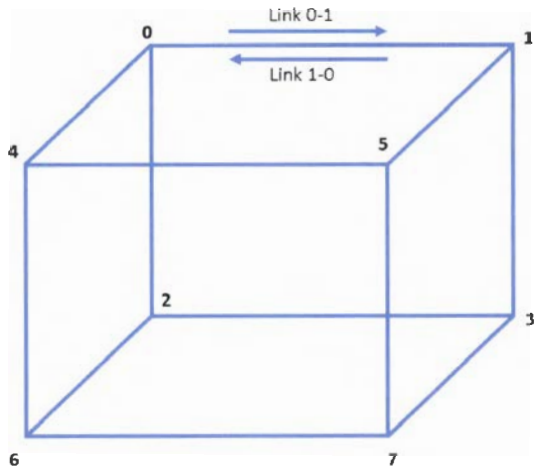

Figure 5.1: Hyper-Cube Network Topology

Each link in hypercube network is analyzed to examine the performance of prediction model, but only those links are presented in this chapter, which show observable results, trends, cyclic patterns, or fluctuations. The predicted results are analyzed by comparing the actual and predicted queue lengths.

5.1.1 Model testing with non-trained parameters, but within the trained input range

In this set of experiments, accuracy of the neural network model is tested with the non-trained parameters, but these parameters lie within the trained input ranges. The simulation runs used to test the model are shown in Table 5.1. These runs represent different types of congestion load, and therefore they are chosen to test the accuracy of neural network model. For better accuracy of the results, each simulation is run multiple times and an average of multiple simulation runs is obtained.

5.1.1.1 Run 1 - High Congestion Load

First, model is tested with a simulation run depicting high congestion. The parameters for this simulation run are shown in Table 5.1. It is a highly congested scenario because links' total bandwidth is very low (5 message units/ticks). The accuracy i.e.

Table 5.1: Neural Network Testing Simulation Runs

Simulation Run	MAT	Update (%)	Bandwidth	Avg. Congestion Load	PTCT (%)	Congestion Load	Accuracy (%) (R square)
1	50	10	5	69.6	7.8	High	95.7
2	50	25	10	27.9	18.1	Medium	95.4
3	30	35	30	1.5	96.9	Negligible	91.0

R square (explained in Chapter 4) of the run is 95.7%. Hypercube network has 8 nodes having different inter-arrival rates, therefore the network links depict different congestion loads. To demonstrate the model's performance, only one representative link is selected for each load.

i) **Links showing negligible congestion load** - In order to exhibit the behavior of prediction model for links with no/negligible congestion loads, the analysis of the least congested link (Link 0-2) is presented, for example, the link displays almost no congestion in the time-span 1400 to 2000 ticks (Figure 5.2).

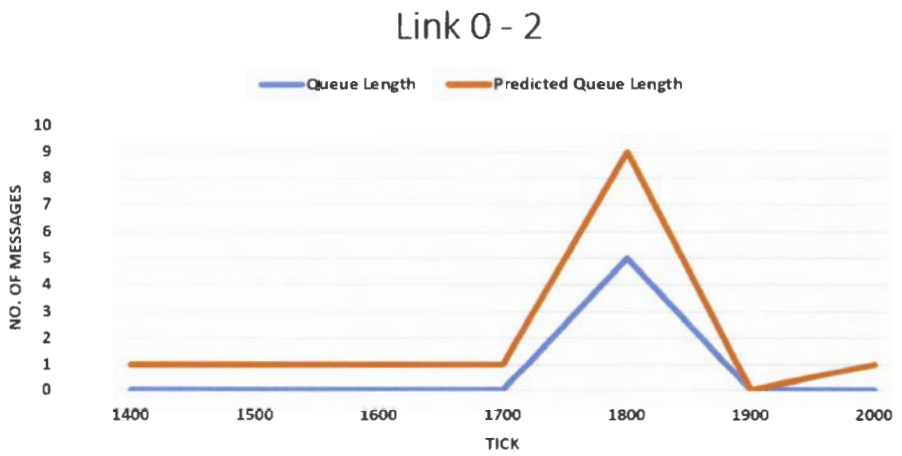

Figure 5.2: Analysis of Link 0-2

In this time-span, the number of queued messages is 0, except at tick 1800 (5 messages queued). As seen in the graph, predicted queue length trending is same as that of actual queue length. It is noted that the model results are promising

(because it can capture outlier) under negligible congestion scenarios, and the model accuracy needs to be observed for links having higher congested scenarios.

ii) **Links exhibiting varying congestion loads** - In this experiment, links are analyzed for longer time-span (around 10,000 ticks) to view the trend and seasonality of the number of queued messages, and it is observed that some links have varying number of queued messages (or congestion load). For instance, Figure 5.3 shows that Link 1-3 has a cyclic pattern between 300 and 4500 ticks. The reason for this pattern is when multiple transactions arrive on a node simultaneously, a large number of messages is exchanged between nodes, resulting in increased queue length. Further, when transactions start completing, queue length eventually reduces. Beyond 4500th tick, queue length reduced drastically (lies between 100 and 200 messages approximately). This decline is caused by the saturation of transactions workload (transaction workload is 150). Similarly in Figure 5.4, the queue length is initially high (340 messages at 1500th tick), and eventually decreases to 0 at 3400th tick, and then further exhibits minor fluctuations.

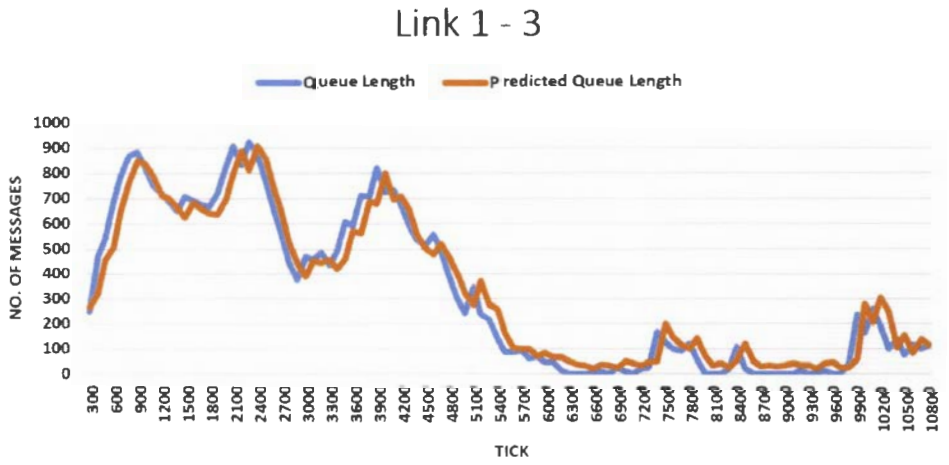

Figure 5.3: Analysis of Link 1-3

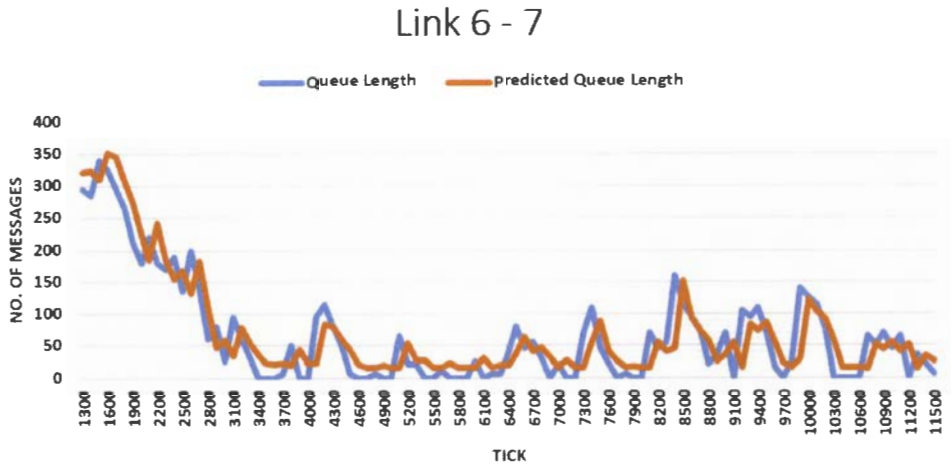

Figure 5.4: Analysis of Link 6-7

Even though both the links have different trends and cycles, it is observed in both the cases that the predicted queue length has followed the same trends as that of actual queue length. The analysis of these links inferred that the neural network model is reliable because it can sense different congestion scenarios in the simulation run and predict accordingly.

iii) **Highly congested links** - To examine the model's performance under highly congested scenarios, the model is analyzed with the links having high congestion load. An example of a link experiencing high congestion is Link 4-6. As shown in Figure 5.5, the number of messages queued on Link 4-6 fall approximately between 1000 and 2000 messages. This denotes that it takes longer (200 to 400 ticks) to clear the queue, resulting in high congestion. As displayed in the graph, the prediction model has shown very promising results in predicting the congestion for highly congested scenarios. For more explicit comparison, the snapshot of predicted results for selected 1000 ticks is shown in Table 5.2. The model's correctness for the selected ticks is always above 90% (out of 10 cases, 6 have above 95% correctness).

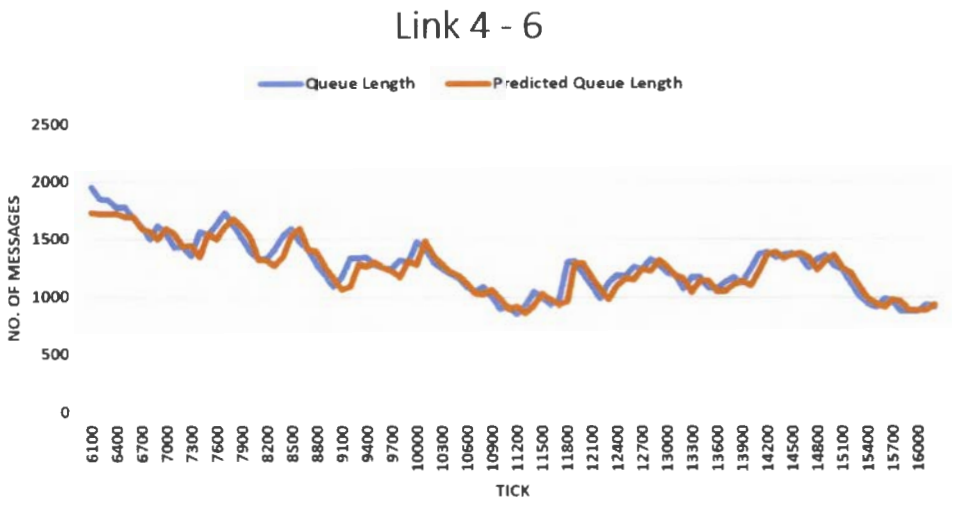

Figure 5.5: Analysis of Link 4-6

Table 5.2: Prediction Result Snapshot of Link 4-6

Tick	Queue Length	Predicted Queue Length	Model Correctness (%)
15300	1005	1094	91.1
15400	940	993	94.4
15500	915	940	97.3
15600	990	919	92.8
15700	970	981	98.9
15800	875	964	89.8
15900	875	885	98.9
16000	880	885	99.4
16100	935	889	95.1
16200	920	936	98.3

5.1.1.2 Run 2 - Medium Congestion Load

To prove model's accuracy under medium congestion scenario, the model is studied with parameters as shown in Table 5.1. Even though update percentage is increased

to 25, simulation run 2 is depicting medium congestion scenario. It is because the bandwidth is doubled i.e. total bandwidth is increased from 5 to 10 message units/ticks. By increasing the bandwidth, fewer messages are queued which results in lower congestion (as compared to simulation run 1). The average congestion load of this simulation run is 27.9. The accuracy (R-square) of simulation run 2 is 95.4%. Similar to simulation run 1, an example scenario of different links experiencing low and high congestion loads is presented (explained below).

i) **Low congested link** - The results for less congested links are presented in Figure 5.6 and 5.7. In simulation run 2, Link 5-7 has the lowest congestion load because node 5 has the highest inter-arrival rate amongst all nodes. Even though there are three links outgoing from node 5 (in hypercube network), link 5-7 has the lowest load because it has the lowest latency as compared to the other two outgoing links. As shown in Figure 5.6, there are fluctuations in queued number of messages, which confirm varying congestion load of the link at different instances. Link 2-3 is another example of low congestion load (Figure 5.7).

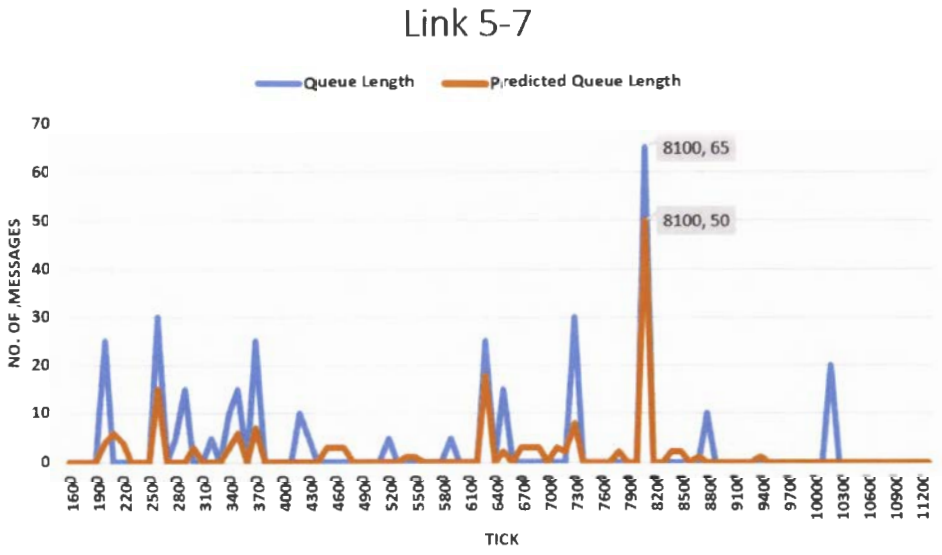

Figure 5.6: Analysis of Link 5-7

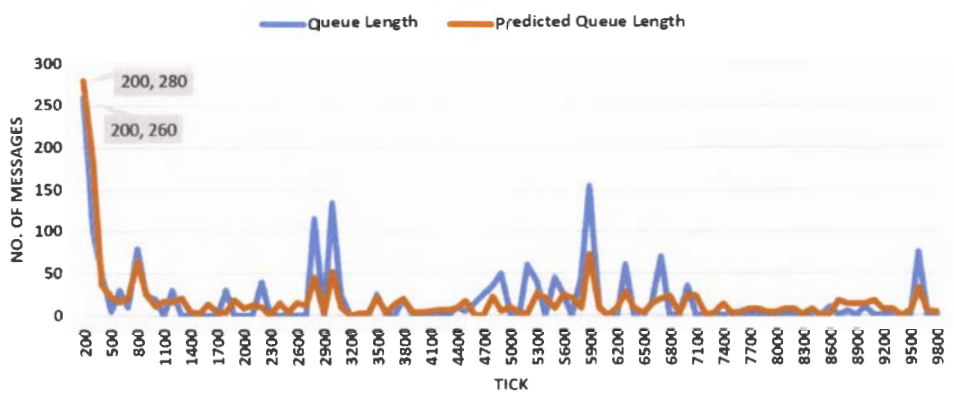

Figure 5.7: Analysis of Link 2-3

As seen in both the links (5-7 and 2-3), results from our prediction model are satisfactory because it can capture the deviations/outliers presented at different instances in the simulation run. One outlier example is displayed in each link (displayed through data labels), and are explained below:

(a) In Figure 5.6, the highest outlier at tick 8100 has 65 queued messages (0 messages queued at 8000th tick). Our prediction model captured this outlier successfully, and predicted 50 messages at 8100th tick (predicted 0 messages queued at 8000th tick).

(b) In Figure 5.7, the highest outlier at tick 200 has 260 queued messages, and our prediction model predicted 280 messages, which is quite promising (92 % accuracy).

ii) **High congested link** - In this experiment, the model is tested with the links having high congestion load. Since, simulation run 2 has medium congestion load, therefore there are very few congested links in the network. An example of a congested link is 3-2, which has varying congestion in the time-span as shown in Figure 5.8. It has a large number of queued messages in the first 5000 ticks of the simulation run (because of transaction workload), and then eventually drops

between 0 and 200 messages. It is observed that the trends of predicted and actual queue lengths are same (Figure 5.8).

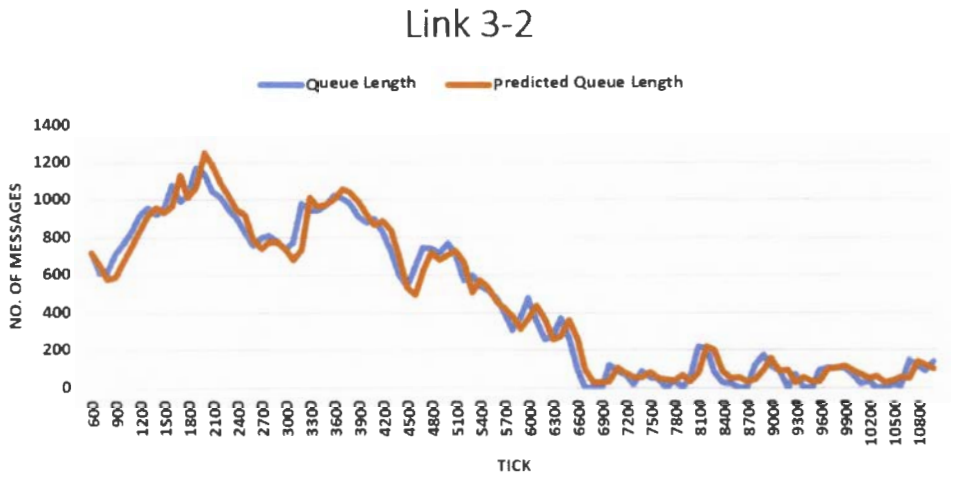

Figure 5.8: Analysis of Link 3-2

5.1.1.3 Run 3 - Negligible Congestion Load

Finally, the prediction model is tested with simulations runs having negligible/no congestion load. The parameters of the simulation run are specified in Table 5.1. The accuracy (R-square) of the run is 91.0%. In this case, the links have negligible or low congestion load, and an example link is presented for each load.

i) **Links showing negligible congestion load** - In order to see the model's behavior under minimal congestion, the model was analyzed with the links having negligible congestion load. Links 1-5 and 7-5 have 0 messages in the queue (no congestion), and insignificant fluctuations. As seen in both the links (Figure 5.9 and 5.10), model is able to successfully recognize the links having negligible congestion load and determine even the minor fluctuations. The highest outliers in Link 1-5 (Figure 5.9) and 7-5 (Figure 5.10) have been successfully recognized by the prediction model. Both the outliers are shown in the figures through

data-labels, and are discussed below:

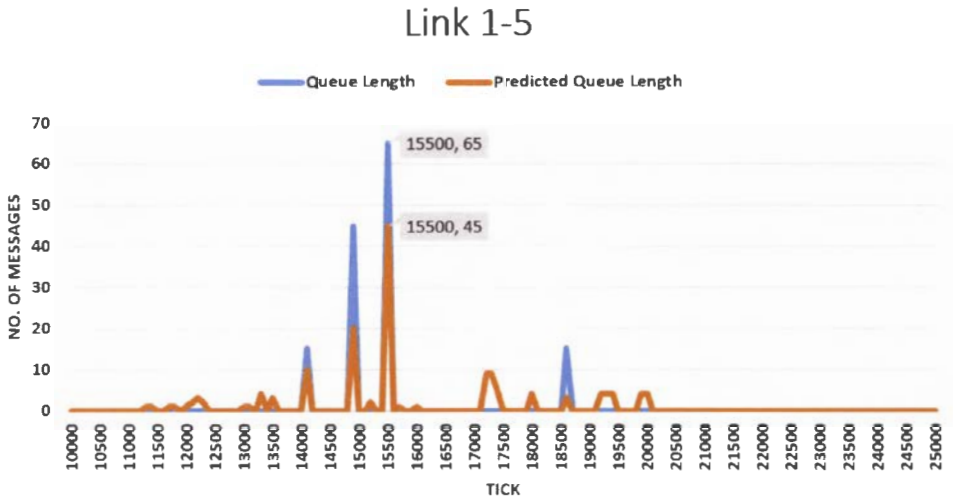

Figure 5.9: Analysis of Link 1-5

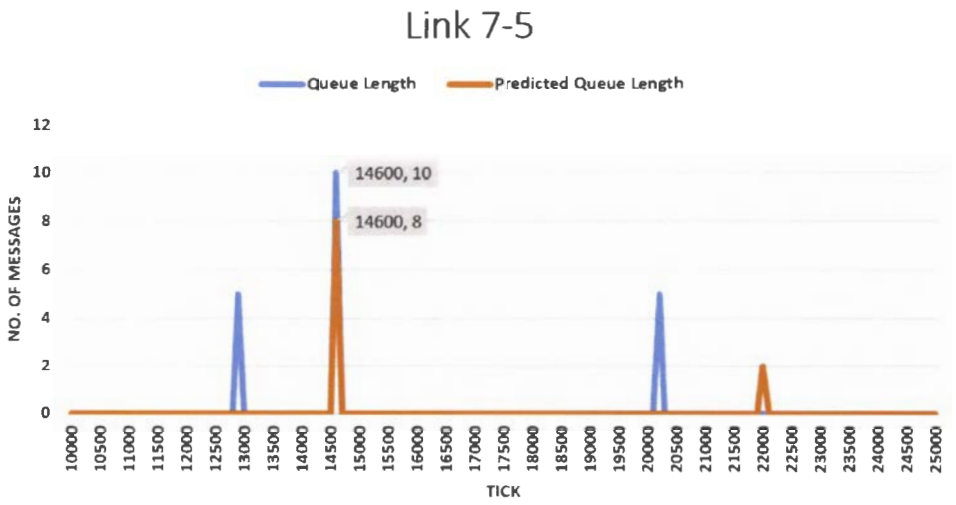

Figure 5.10: Analysis of Link 7-5

(a) At tick 15,500, Link 1-5 has 65 queued messages whereas model predicted 45

messages (Accuracy - 70 %). Even though the number of queued messages (65 messages) is significantly higher than the mean value (approximately 0), the model is still able to predict this spike with reasonable accuracy (70%).

(b) At tick 14,600, Link 7-5 has 10 queued messages, and model predicted 8 messages (Accuracy - 80 %).

ii) **Links exhibiting low congestion load** - There are few links exhibiting low congestion load (the majority of the links have negligible congestion load), for example - Link 6-2 (Figure 5.11).

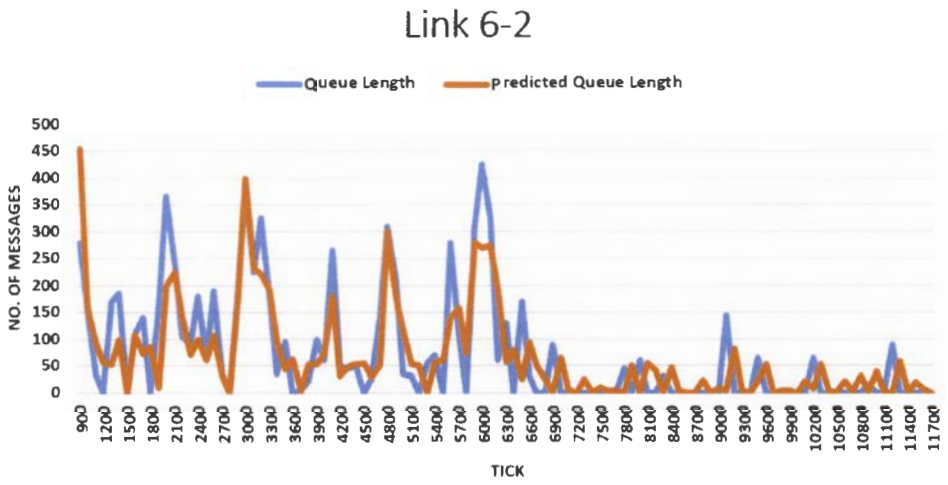

Figure 5.11: Analysis of Link 6-2

Even though, Link 6-2 has a large bandwidth (30 message units/ticks), this link has some congestion load because the inter-arrival rate of transactions on the source node is very low, and link has a larger latency as compared to the other links.

5.1.2 Model testing with parameters outside the input training range

In this section, the prediction model is tested with the simulation runs, where the parameters of the simulation runs fall outside the input training ranges. Recalling the training ranges from Chapter 4, training range of MAT is 10 to 50 transactions, update percentage 5 to 50 %, and bandwidth 5 to 30 message units/ticks. The testing simulation runs are shown in Table 5.3, where the values of the parameters fall outside the input training ranges. Again, the simulation runs with different congestion loads are analyzed; but only one representative link from each simulation run is displayed.

Table 5.3: Neural Network Testing Simulation Runs (outside input training ranges)

Simulation Run	MAT	Update (%)	Bandwidth	Avg. Congestion Load	PTCT (%)	Congestion Load	Accuracy (%) (R-square)
1	60	55	15	31.1	24.1	High	92.0%
2	70	80	25	5.1	64.7	Low	91.4%
3	70	80	50	0.3	100	Negligible	89.8%

1. **Run 1 - High Congestion Load** - In this experiment, high congestion load is created by setting up MAT to 60, update percentage 55, and bandwidth 15. All the parameters are outside the input training range, except total bandwidth. It is because the goal here was to create high congestion scenario, which would not have been possible if bandwidth value was selected from outside the input training range (when bandwidth value is large, congestion load is very low). The accuracy (R-square) of the simulation run is 92.0%.

   An example of a link with high congestion load is shown in Figure 5.12. The predicted results of approximately 2000 ticks is displayed with the data error bar for each result. The data error bar represents the deviation of predicted value from the actual value. The dark gray bar indicates that the predicted value is smaller as compared to the actual value; whereas the white bar means that the predicted value is larger than the actual value. The more the prediction error,

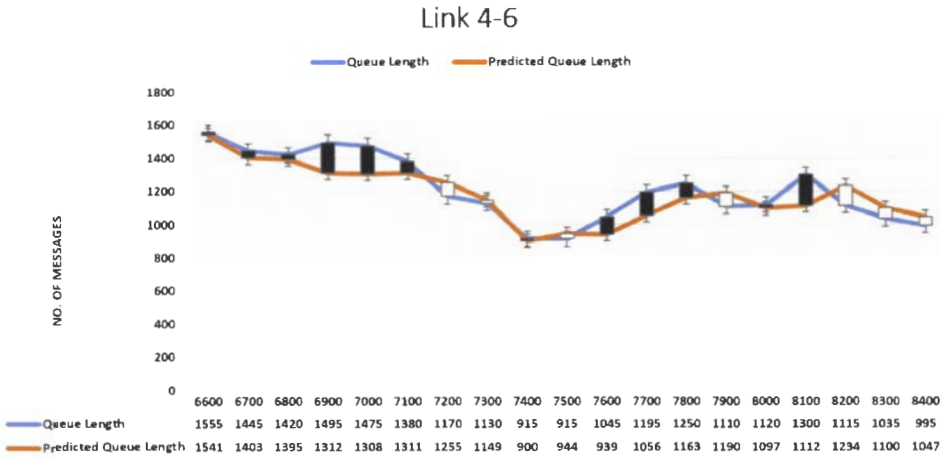

Figure 5.12: Analysis of Link 4-6

the wider (or larger) is the error bar. For example, in the time range shown in Figure 5.12, the maximum error is at 8100th tick (hence, the largest error-bar).

2. **Run 2 - Low Congestion Load** - In this run, the prediction model is tested with low congestion load (parameters are specified in Table 5.3), and the accuracy (R-square) of the run is 91.4%. This simulation run has fluctuating queue length as shown in Figure 5.13, for example - at tick 1200, the number of queued messages is 250 (high congestion load), then drops to 10 messages (low congestion load), and climbs to 95 messages (medium congestion load). The predicted result for the aforementioned example is 180 messages queued at tick 1200 (high congestion load), declines to 28 messages (low congestion load), and then climbs to 87 messages (medium congestion load). This indicates that our prediction model can capture the fluctuating congestion load scenarios in the hypercube network.

3. **Run 3 - Negligible Congestion Load** - When tested with negligible congestion load, the accuracy (R-square) of the run is 89.8%. The highest outlier in the range shown in Figure 5.14 falls at 1900th tick, and the remaining ticks show negligible/no congestion and are predicted accurately by the model.

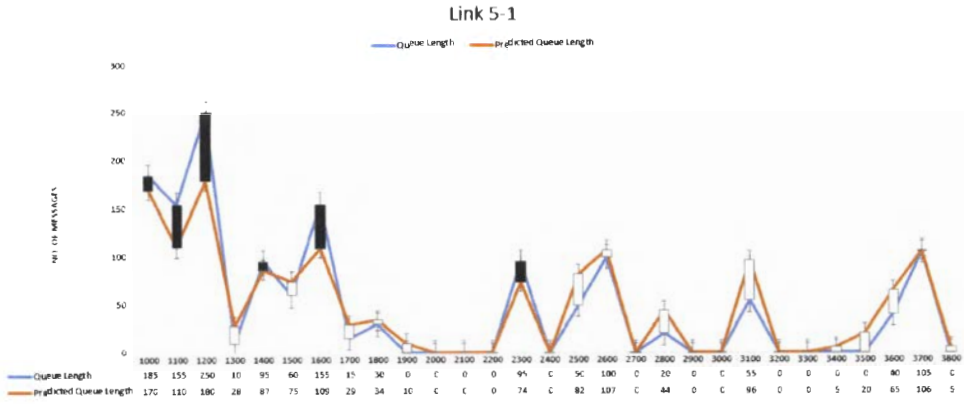

Figure 5.13: Analysis of Link 5-1

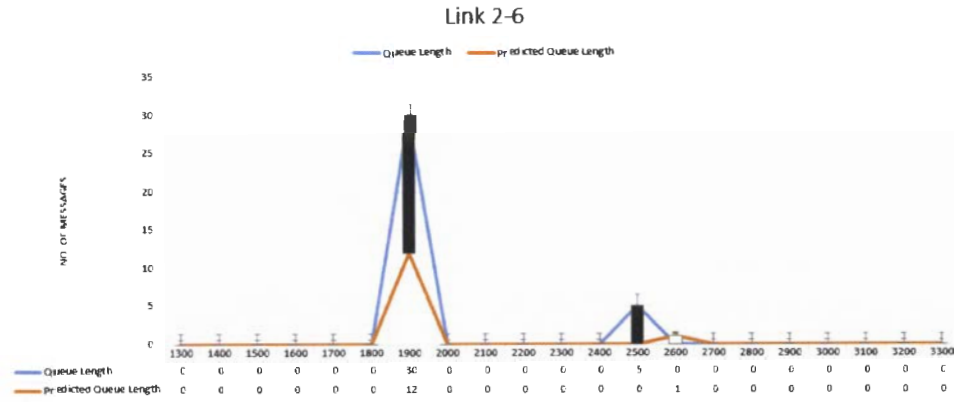

Figure 5.14: Analysis of Link 2-6

5.1.3 Summary

The prediction model is tested with different simulation runs (inside and outside training ranges), and R-square is observed for each simulation run (Figure 5.15). From the testing set (set of simulation runs), it is noted that the maximum accuracy of the testing set inside the input training ranges is 95.7% at high congestion load (minimum is 91.0% at negligible congestion load); whereas the maximum accuracy of the testing set outside the input training ranges is 92.3% at medium congestion load (minimum is 89.8% at negligible congestion load). Since the model is not trained with

the testing set outside the input training range, the accuracy has decreased slightly in these cases.

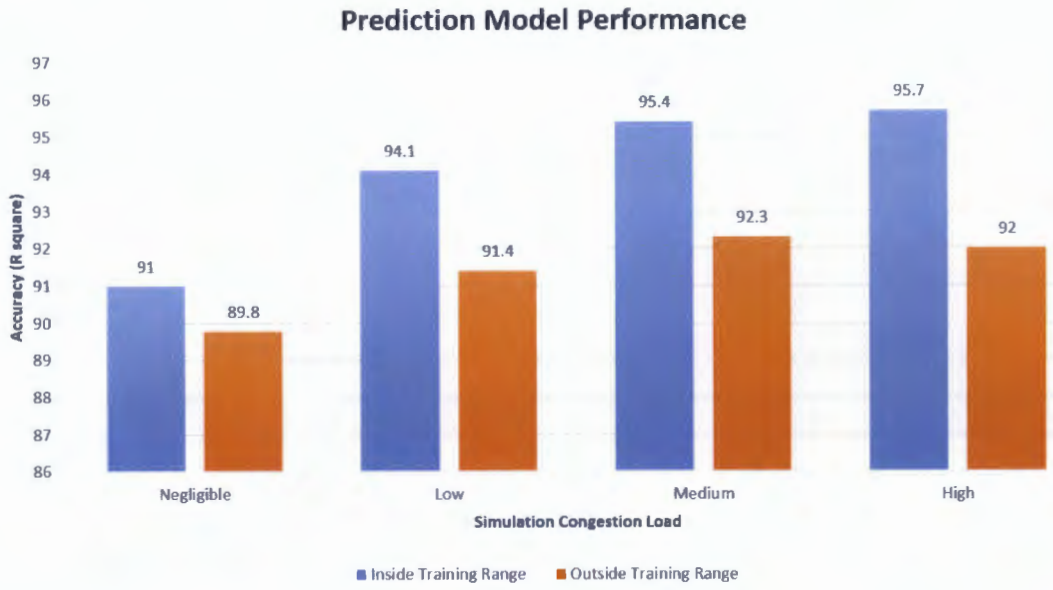

Figure 5.15: Performance of Prediction Model

Apart from the simulation runs shown in the above subsections, more simulation runs were tested with parameters inside and outside training ranges, and satisfying results were achieved. From the analysis of various simulation runs with different congestion loads, it is concluded that our prediction model is robust because it can sense different congestion scenarios in a link; capture fluctuations; handle outliers; and analyze negligible, low, medium, and high congestion loads.

5.2 Comparison between DSP, RIP, and NNPR

In this section, the experiments conducted to compare the performance of NNPR (Neural Network Prediction-based Routing) protocol with DSP and RIP protocol are presented. The key performance metric used in this thesis is the percentage of

transactions completing on time (PTCT).

Table 5.4: Baseline Experiment Parameters Settings

Parameter Type	Parameter Name	Value
Network	Topology	Hypercube
	Number of Nodes	8
	Bandwidth	20
	Latency	5 (Poisson Distribution)
Node	Max Active Transactions	30
	Disk Count Per Node	20
	Disk Access Time	35 Ticks
	Buffer Size	100
	Swap Disk Access Time	35 Ticks
	Transaction Process Time	5 Ticks
Workload Generator	Inter-arrival Time	15 - 300 Ticks (Poisson)
	Scale Arrival Rate	Enabled
	Slack Time	676 - 2028
	Work-size	2 - 8 pages (Uniform Distribution)
	Update Percentage	25 %
	Total Transaction Workload	150

In DRTTPS, the congestion load of the system can be varied by many parameters, including bandwidth, max active transactions, update percentage, latency, work-size, and total transaction workload. In the conducted experiments, the baseline parameters were set as shown in Table 5.4, and few parameters are tweaked to change the congestion load of the system. NNPR is examined with different simulation runs (parameters lying inside and outside the model input training range), and it is observed that it consistently performs better than DSP and RIP protocol for different parameters.

Why NNPR performs better? The superior performance of NNPR can be attributed to model's prediction of congestion with reliable accuracy (shown in Section 5.1). When prediction results are accurate, routing mechanism becomes efficient because it can now determine the specific tick when congestion will start to build, and routing tables are updated at that particular tick. For example, if the model predicts 150 queued messages at 200th tick, routing tables are updated, say at 160th tick (it is the calculated tick when congestion started to build up). If model predicts no congestion, routing tables are not updated and DSP's routing table is used by default, which reduces the routing overhead.

Several experiments were conducted to compare the performance of protocols. Each experiment varied one specified parameter to create congestion scenarios and analyze PTCT. The range of values for the parameter is chosen in such a way that it shows significant results within the selected range. The performance of the protocols is shown as relative to each other.

5.2.1 Impact of Bandwidth

Bandwidth is a key factor which can impact the congestion in a system. If there is not enough bandwidth, messages are queued; resulting in congestion within the network and thus a low PTCT. The experiment shown in Figure 5.16 uses the baseline parameters settings, except bandwidth, which is varied from 5 to 35 message units/ticks (bandwidth range). It can be seen that NNPR performs significantly better than DSP and RIP for the entire range of selected bandwidth.

When bandwidth value is between 5 and 20, NNPR demonstrates very high performance as compared to DSP and RIP (DSP and RIP perform close to each other). With a bandwidth of 5, NNPR demonstrates 40% PTCT gain than DSP (NNPR - 21.5% PTCT and DSP - 15.27% PTCT). It indicates that when the congestion load is high, NNPR's performance is superior, which implies that the protocol is able to predict and control the congested system effectively. Within the bandwidth range of 10 and 20, NNPR performs 20 to 30% better than the other two protocols. As

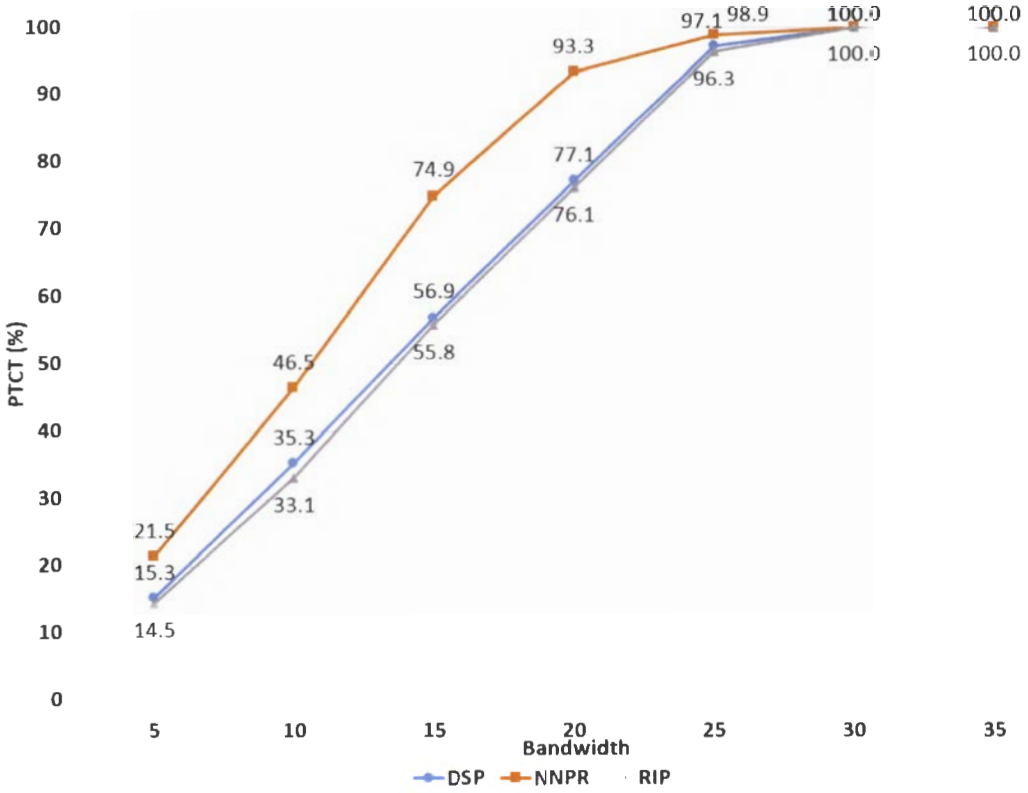

Figure 5.16: Impact of Bandwidth (MAT - 30)

bandwidth increases beyond 25, there is no congestion load, and all protocols climb to 100 % PTCT.

Next, MAT is increased from 30 to 60 transactions (Figure 5.17). Congestion load is high in this case because when more transactions run concurrently in the system, it causes more messages to flow within the network, resulting in congestion. Due to high congestion load, the performance of all the protocols decreases significantly. However, again NNPR outperforms DSP and RIP protocols, and consistently performs better when bandwidth range is between 5 and 30. For example, when bandwidth is 15, NNPR exhibits a 73% PTCT increment as compared to DSP (DSP - 33.8% PTCT and NNPR - 58.5% PTCT). All protocols climb to 100% as the bandwidth increases, however, in this case the 100% PTCT is achieved at a bandwidth of 35.

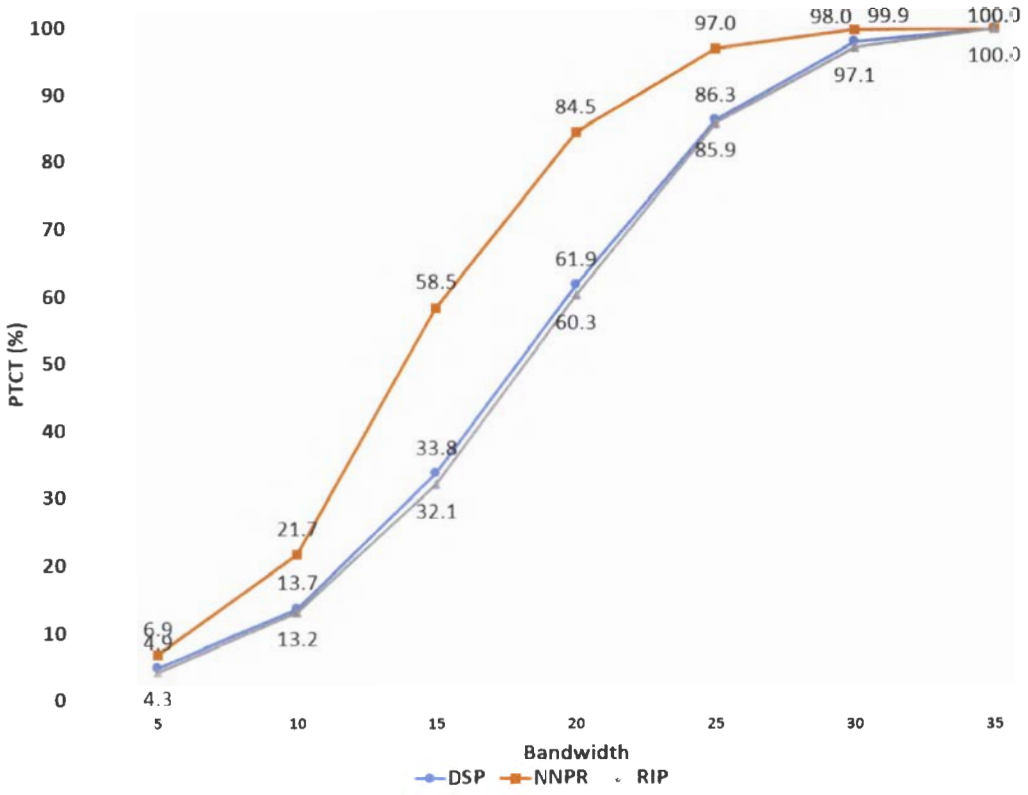

Figure 5.17: Impact of Bandwidth (MAT - 60)

As seen in Figure 5.16 and 5.17, RIP and DSP perform very closely to each other. RIP analyzes the congestion in the system and periodically updates the routing table accordingly, but it is not able to perform better than DSP. Since system's congestion load varies frequently, it is possible that the updated paths of the routing table by RIP are no longer the best paths. For example - at tick 200, RIP updated the routing table indicating that the best path from node 1 to node 3 is 1->2->3; and further at 210th tick, link 2-3 becomes highly congested; which means that path 1->2->3 is no longer the best path. The late realization of the congestion scenario causes RIP to perform poorly. So, there can be some scenarios where RIP is sending the messages to the highly congested links. After extensive experimentation, it is observed that RIP performs almost the same as DSP, and thus RIP's performance is not exhibited

in the later experiments.

5.2.2 Impact of Page Update Rate

Page update rate decides the percentage of write operations in a transaction execution. Write operations block other transactions because of the exclusive lock on data. This blocking causes delay in transaction's execution, resulting in low PTCT. A page update rate of 0% implies that the transaction contains read operations only.

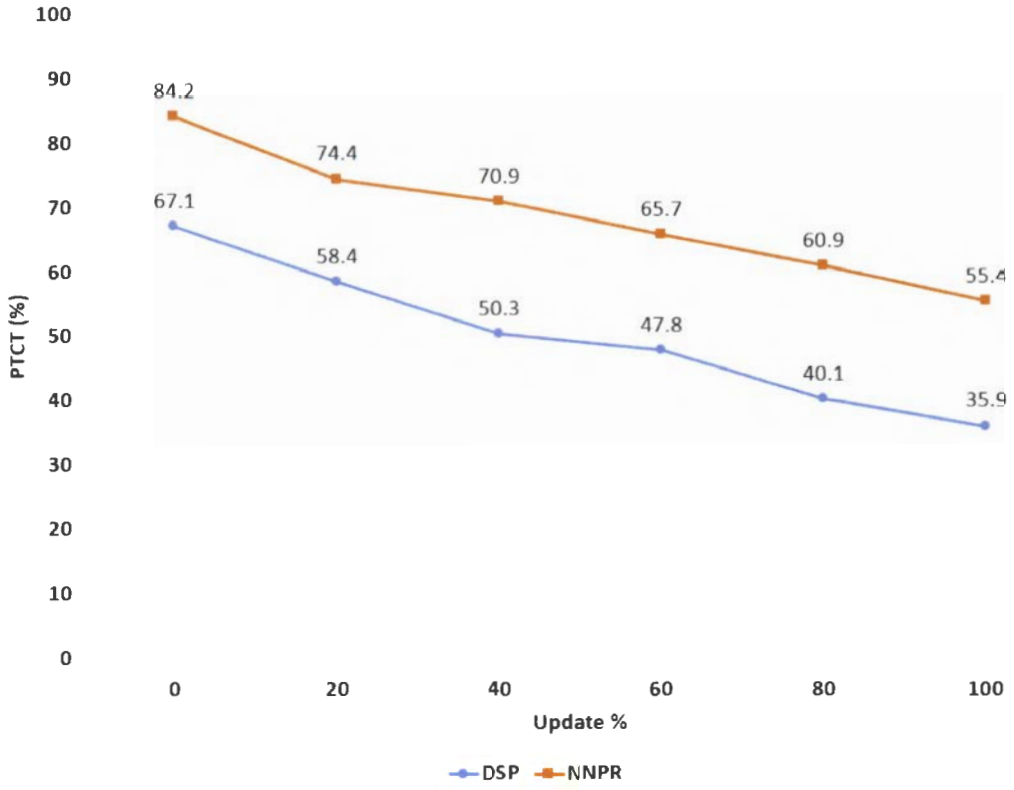

Figure 5.18: Impact of Page Update Rate (Bandwidth - 15)

Two experiments are conducted to see the impact of page update rate on PTCT. The parameters of both the experiments are based on the baseline parameter setting, except bandwidth, which is varied to create high and low congestion scenarios. In

the first experiment, the bandwidth value is chosen as 15 which indicates medium or low congestion (depending on the page update percentage value).

DSP and NNPR have maximum PTCT at 0% update because of no blocking of transactions (Figure 5.18). When page update rate is increased, the congestion load of the system rises and eventually becomes tremendous at 100% update, that is, when all operations in a transaction require a write to the database. NNPR consistently performs better than DSP, with approximately 25%, 38%, and 50% PTCT gain within the range of 0 to 20, 40 to 60, and 80 to 100 update percentage, respectively.

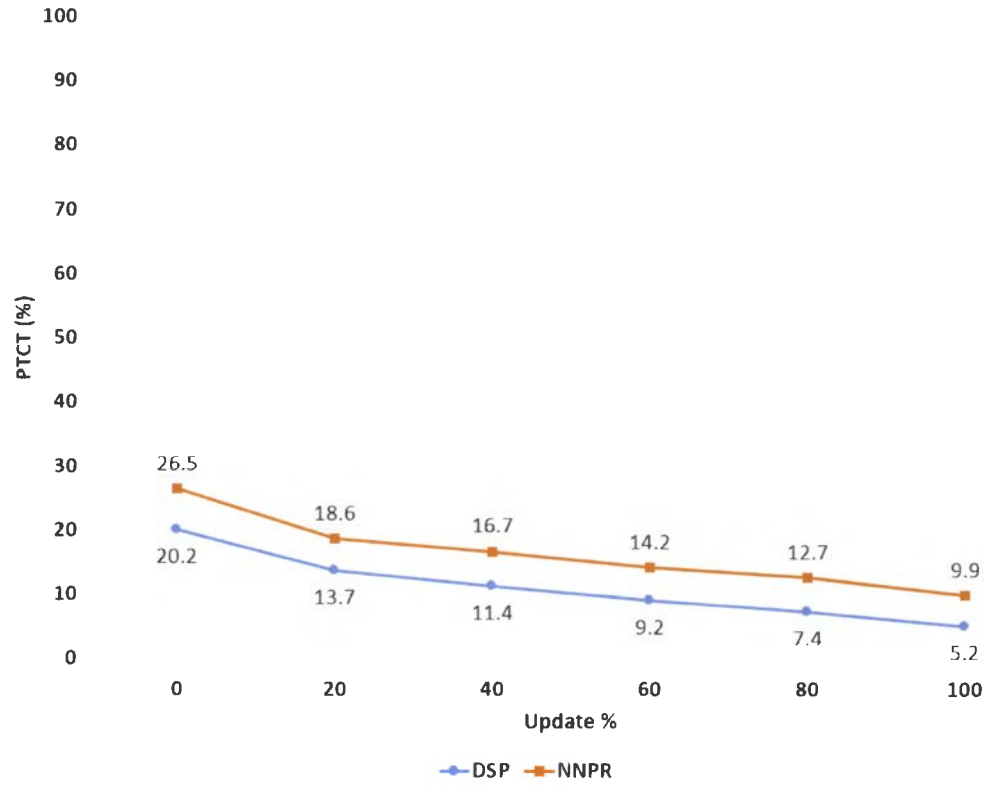

Figure 5.19: Impact of Page Update Rate (Bandwidth - 5)

In the second experiment, the bandwidth value is reduced to 5, which produces high congestion load in the system (Figure 5.19). As expected, the performance of

DSP and NNPR degrade. However, even in the highly congested scenario, NNPR continues to outperform DSP by 31 to 90% (demonstrates approx. 5% more PTCT).

5.2.3 Impact of MAT

Max Active Transactions (MAT) indicates the number of transactions running concurrently on a node. When more transactions run simultaneously on a node, more messages are exchanged within the network, resulting in congestion. In this section, the impact of MAT on PTCT is studied. The range of MAT is varied from 10 to 70, as there are no noticeable results outside of this range.

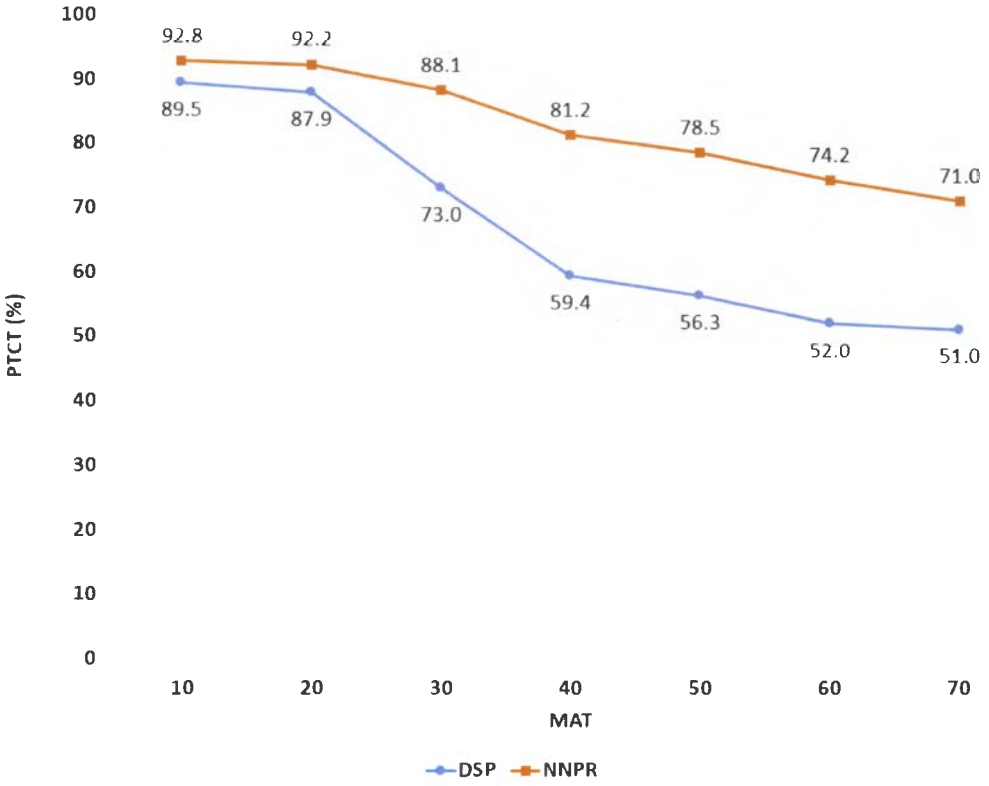

Figure 5.20: Impact of MAT

Figure 5.20 demonstrates that the performance of both DSP and RIP decreases when MAT increases. When MAT is varied from 10 to 20 transactions, the perfor-

mance of both the protocols is not impacted largely, i.e. DSP is reduced from 89.5 to 87.9 % and NNPR degrades from 92.8 to 92.2 %. It is because bandwidth is abundant (20 message units) to handle the queuing of the messages. However, beyond a MAT of 20, there is a transitioning from low to high congestion load, and bandwidth is no longer enough to handle the queued messages, resulting in low PTCT. During this period, DSP degrades rapidly and NNPR consistently outperforms it by 20 to 43% (on an average of 20% more PTCT).

5.2.4 Impact of Latency

Latency is the time taken by a message to travel from source to destination node. High latency network connections suffer long delays, causing more transactions to miss their deadlines, and therefore a low PTCT. In this section, the impact of latency on the congestion load of the system (and hence PTCT) is studied. For neural network training/testing, latency is always chosen as 5, with Poisson distribution across the links. However, to study the impact of latency, the same latency is chosen for all links. The chosen range for latency is 2 to 12 ticks.

As shown in Figure 5.21, both DSP and NNPR demonstrate 100% PTCT at 2 ticks of latency. When latency is increased to 4 ticks, there is a huge decline of PTCT for both the protocols, i.e 30.7% (NNPR) and 42.2% (DSP). It is because, at latency 4, messages are experiencing more delays as compared to latency 2. During this period, NNPR demonstrated 20% PTCT increment (11.5% more PTCT) than DSP. When latency is increased further, PTCT for both the protocols consistently declines as there is a transitioning from low to medium, and then high congested load systems. However, NNPR continues to exhibit better performance than DSP by 30 to 141% (representing on an average of 12% more PTCT).

5.2.5 Impact of Work-size

Work-size depicts the number of pages accessed by a transaction. When more pages are required by a transaction for its execution, more messages are exchanged

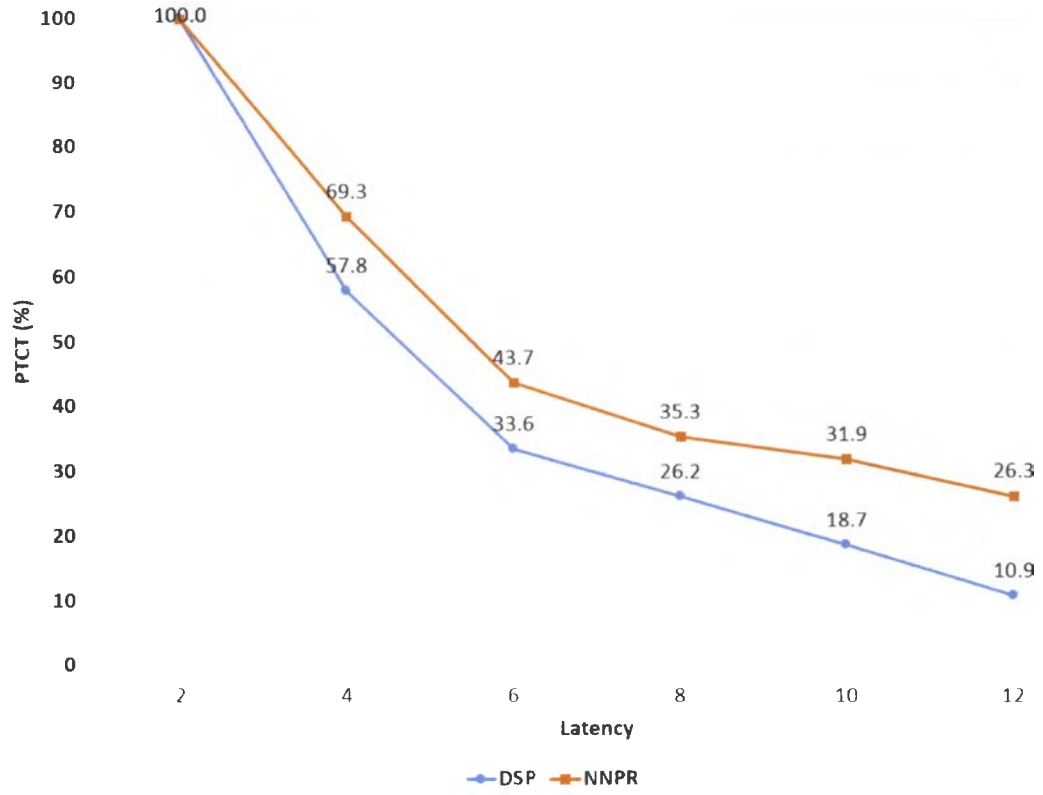

Figure 5.21: Impact of Latency

within the network, which causes congestion. In this experiment, the impact of work-size is studied. The range of work-size is chosen from 2 to 12 pages.

In Figure 5.22, it is observed that there is a steep linear decrease in PTCT within the range of 4 to 10 pages (for both protocols). During this period, NNPR shows 5 to 19% PTCT gain than DSP. Beyond 10 pages, there is a small decline of PTCT for both the protocols, i.e. 3.7% (NNPR) and 4.4% (DSP).

5.2.6 Summary

- In each experiment, NNPR is tested with the simulation runs having parameters outside the model's input training range, for example - Figure 5.17 has MAT 60 (input training range of MAT is 10 to 50 transactions), Figure 5.18 has page

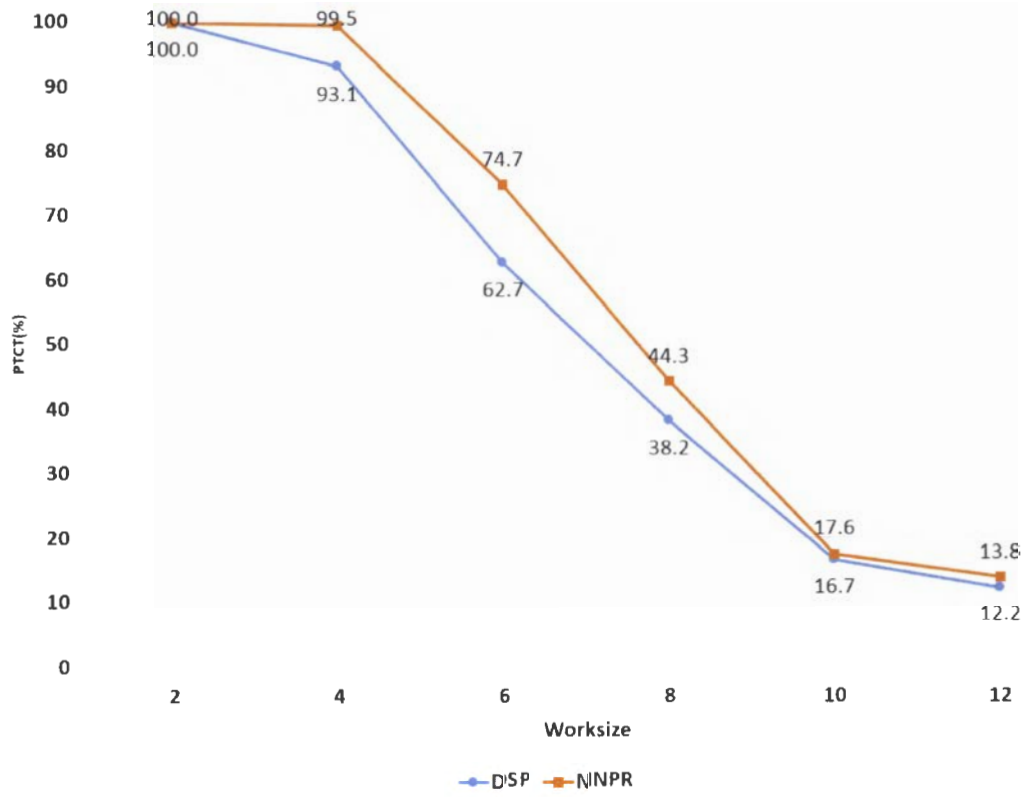

Figure 5.22: Impact of Worksize

update range from 0% to 100% (input training range is 5 to 50%), and Figure 5.22 has work-size range from 2 to 12 pages (input training range is 2 to 8 pages). Despite that, NNPR outperformed DSP and RIP for each congestion scenario. DSP and RIP perform very close to each other.

- Increasing bandwidth from 5 to 35 message units improved the performance of all the routing protocols sharply.

- A change in page update rate from 0% to 100%, MAT 10 to 70 transactions, Latency 2 to 12 ticks, and work-size from 2 to 12 pages decreased the performance of all the routing protocols. However, NNPR consistently outperforms both DSP and RIP.

Chapter 6

Conclusion and Future Work

Congestion problem has increased because of increased network demand. Congestion results in queuing within the network, packet loss and increased delays. It should be controlled to increase the system throughput and quality of service. There are many congestion control approaches such as controlling the arrival rate of packets in the network, throttling the source, routing techniques, etc developed by the researchers over many years. However, these techniques focus on controlling congestion after it has already happened. With an increased demand for high performance networks, there is a need to increase the network's throughput and quality of service. Hence, it is important to analyze the historical data and predict future congestion, so that the efficient controlling techniques can be applied.

We have proposed a protocol (NNPR) to predict as well as control the network traffic in distributed real time environment using distributed real time transaction processing simulator (DRTTPS) as a test-bed. For predictions, multi-step neural network technique is used, which predicts congestion in future (1 step ahead - 100 ticks). Neural network is fed with the inputs such as *latency*; *tick*; *pipe*; *total bandwidth*; *used bandwidth*; *queue length*; *no. of messages*; *worksize*; *page update rate*; *MAT*; *arrival rate*. All these input values are recorded periodically after 100 ticks. Output of neural network is queue length, 1 step (100 ticks) ahead. Queued messages

cause larger delays and dropping of messages, leading to congestion. The network congestion of a link is defined with the following formula:

$$Congestion_l = \left\lceil \frac{\sum_{i=0}^{n} size_i}{BW_l} \right\rceil \times L_l \tag{6.1}$$

where $size_i$ is the size of queued message i, BW is the bandwidth of a link and L is the latency of a link.

Neural network prediction model is developed in SPSS Modeler [57] (explained in Section 4.1.2). Model is trained in an off-line mode with data obtained from simulation runs by varying different parameters in the simulator. For real time predictions, ADAPA is used. SPSS modeler exports a file in PMML format containing the neural network model, and then the file is uploaded on the ADAPA instance running on Amazon cloud (explained in Section 3.3). ADAPA wrapper (in DRTTPS) calls the prediction model through web services and predicts the data in real-time. After predicting the network traffic, the predicted congested link's messages is re-routed to other links. To compare the proposed work with other techniques, two routing protocols are implemented - DSP and Routing Information Protocol (RIP). The main metric used to analyze the performance of our protocol is the percentage of transactions which complete before their deadline.

6.1 Future Work

This research can be extended in several ways as indicated below:

- Our prediction model is trained with three dimensional hypercube topology. However, the model can be trained and tested with different network topologies by varying number of nodes.
- In prediction module, queue length is predicted 100 ticks ahead. Prediction accuracy of the model can be compared and analyzed by varying this prediction step, for example - analyze the behavior of the model when predicted 50 or 150

ticks ahead.

- The test-bed of this research DRTTPS is not designed to handle fault tolerance (for example, failure of link or node). Therefore, it will be interesting to see NNPR's performance in such a environment.

Bibliography

[1] Udai Shanker, Manoj Misra, and AnilK. Sarje. Distributed real time database systems: background and literature review. *Distributed and Parallel Databases*, 23(2):127–149, 2008.

[2] Dr. Wanida Putthividhya University of Southern California. Network simulator - java network visualiser (javis), Accessed - August 2015.

[3] Congestion Control Algorithms. http://ecomputernotes.com/computernetworking notes/communication-networks/what-is-congestion-control-describe-the-congestion-control-algorithm-commonly-used, Accessed - August 2015.

[4] Example Neural Network. http://www.texample.net/tikz/examples/neural-network/, Accessed - August 2015.

[5] Waqar Haque and Paul R. Stokes. Simulation of a complex distributed real-time database system. In *Proceedings of the 2007 Spring Simulation Multiconference - Volume 2*, SpringSim '07, pages 359–366, San Diego, CA, USA, 2007. Society for Computer Simulation International.

[6] Laurene Fausett. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[7] Martin T. Hagan, Howard B. Demuth, and Mark Beale. *Neural Network Design.* PWS Publishing Co., Boston, MA, USA, 1996.

[8] B. Kao and H. Garcia-Molina. An overview of real-time database systems. Technical Report 1993-6, Stanford University, 1993.

[9] Hector Garcia-Molina and Bruce Lindsay. Research directions for distributed databases. *SIGMOD Record*, 19(4):98–103, December 1990.

[10] Yu-Wei Chen and Le Gruenwald. Effects of deadline propagation on scheduling nested transactions in distributed real-time database systems. *Inf. Syst.*, 21(1):103–124, 1996.

[11] Michael Welzl. *Network Congestion Control: Managing Internet Traffic (Wiley Series on Communications Networking & Distributed Systems)*. John Wiley & Sons, 2005.

[12] Sangjoon Jung, Chonggun Kim, and Younky Chung. A prediction method of network traffic using time series models. In *Computational Science and Its Applications - ICCSA 2006*, volume 3982 of *Lecture Notes in Computer Science*, pages 234–243. Springer Berlin Heidelberg, 2006.

[13] Philip A Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

[14] Peter Rob and Carlos Coronel. *Database Systems Design, Implementation and Management*. Course Technology Press, Boston, MA, United States, 5th edition, 2002.

[15] SandraR. Thuel and JayK. Strosnider. Enhancing fault tolerance of real-time systems through time redundancy. In GaryM. Koob and CliffordG. Lau, editors, *Foundations of Dependable Computing*, volume 285 of *The Kluwer International Series in Engineering and Computer Science*, pages 265–318. Springer US, 1994.

[16] C.-Q. Yang and A.V.S. Reddy. A taxonomy for congestion control algorithms in packet switching networks. *Network, IEEE*, 9(4):34–45, Jul 1995.

[17] M. Thottethodi, A.R. Lebeck, and S.S. Mukherjee. Self-tuned congestion control for multiprocessor networks. In *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on*, pages 107–118, 2001.

[18] S.S. Lam and Y.C.L. Lien. Congestion control of packet communication networks by input buffer limits-a simulation study. *IEEE Transactions on Computers*, 30(10):733–742, 1981.

[19] Nanying Yin, San qi Li, and Thomas E. Stern. Congestion control for packet voice by selective packet discarding. *IEEE Transactions on Communications*, 38(5):674–683, 1990.

[20] Z. Fan and P. Mars. Access flow control scheme for atm networks using neural-network-based traffic prediction. *Communications, IEE Proceedings-*, 144(5):295–300, Oct 1997.

[21] Yanxiang He, Naixue Xiong, and Yan Yang. Data transmission rate control in computer networks using neural predictive networks. In *ISPA*, volume 3358 of *Lecture Notes in Computer Science*, pages 875–887. Springer, 2004.

[22] Janey C. Hoe. Improving the start-up behavior of a congestion control scheme for tcp. *SIGCOMM Comput. Commun. Rev.*, 26(4):270–280, August 1996.

[23] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1(4):397–413, August 1993.

[24] Chunlei Liu and Raj Jain. Improving explicit congestion notification with the mark-front strategy. *Computer Networks*, 35:2–3, 2000.

[25] Oded Maimon and Lior Rokach. *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[26] E. W. Dijkstra. A note on two problems in connexion with graphs. *NUMERISCHE MATHEMATIK*, 1(1):269–271, 1959.

[27] C. Hendrick. Routing information protocol. RFC 1058, IETF, June 1988.

[28] J. Alan Bivens, Boleslaw K., Mark J. Embrechts, and Boleslaw K. Szymanski. Network congestion, arbitration, and source problem prediction using neural networks. *Smart Engineering System Design*, 4:243–252, 2002.

[29] Zhixin Liu, Xinping Guan, and Huihua Wu. Bandwidth prediction and congestion control for abr traffic based on neural networks. In *ISNN (2)*, volume 3973 of *Lecture Notes in Computer Science*, pages 202–207. Springer, 2006.

[30] W.Price D.Davies, D.Barber and C.Solomonides. *Computer networks and their protocols.* Wiley and Sons Ltd, New York, 1979.

[31] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons not to deploy red. In *Quality of Service, 1999. IWQoS '99. 1999 Seventh International Workshop on*, pages 260–262, 1999.

[32] Mario Gerla and Leonard Kleinrock. Congestion control in interconnected lans. *IEEE Network*, 2(1):72–76, 1988.

[33] Sangjoon Jung, Mary Wu, Youngsuk Jung, and Chonggun Kim. Traffic-predicting routing algorithm using time series models. In *Computational Science and Its Applications - ICCSA 2006*, volume 3983 of *Lecture Notes in Computer Science*, pages 1022–1031. Springer Berlin Heidelberg, 2006.

[34] T. H. Lai. Time series analysis univariate and multivariate methods : William w.s. wei, (addison-wesley, reading, ma, 1990). *International Journal of Forecasting*, 7:389–390, 1991.

[35] Teerawat Issariyakul and Ekram Hossain. *Introduction to Network Simulator NS2.* Springer Publishing Company, Incorporated, 1 edition, 2008.

[36] Bo Zhou, Dan He, and Zhili Sun. Traffic modeling and prediction using arima/-garch model. In *Modeling and Simulation Tools for Emerging Telecommunication Networks*, pages 101–121. Springer US, 2006.

[37] Yantai Shu, Zhigang Jin, Lianfang Zhang, Lei Wang, and O. W W Yang. Traffic prediction using farima models. In *Communications, 1999. ICC '99. 1999 IEEE International Conference on*, volume 2, pages 891–895, 1999.

[38] Abdulkareem Y. Abdalla, Turki Y. Abdalla, and Khloud A. Nasar. Routing with congestion control in computer network using neural networks. *International Journal of Computer Applications*, 57(2):34–41, November 2012. Published by Foundation of Computer Science, New York, USA.

[39] B.Chandra Mohan, R. Sandeep, and D. Sridharan. A data mining approach for predicting reliable path for congestion free routing using self-motivated neural network. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, volume 149 of *Studies in Computational Intelligence*, pages 237–246. Springer Berlin Heidelberg, 2008.

[40] Georgeta Boanea Melinda Barabas and Virgil Dobrota. Multipath routing management using neural networks based traffic prediction. *EMERGING 2011 , The Third International Conference on Emerging Network Intelligence*, 2011.

[41] U.Y. Ogras and R. Marculescu. Prediction-based flow control for network-on-chip traffic. In *Design Automation Conference, 2006 43rd ACM/IEEE*, pages 839–844, 2006.

[42] Fei Xiang, He Xiaoyan, Junzhou Luo, Jieyi Wu, and Guanqun Gu. Fuzzy neural network based traffic prediction and congestion control in high-speed networks. *J. Comput. Sci. Technol.*, 15(2):144–149, 2000.

[43] U. Urathal alias, Swathiga, and C. Chandrasekar. Article: An efficient fuzzy based congestion control technique for wireless sensor networks. *International Journal of Computer Applications*, 40(14):47–55, February 2012. Published by Foundation of Computer Science, New York, USA.

[44] Dr. C. Chandrasekar U. Urathal alias Swathiga. Congestion prediction and adaptive rate adjustment technique for wireless sensor networks. *International Journal of Research in Computer Science*, 3:1–7, July, 2013.

[45] D. V. Lindley. The theory of queues with a single server. Proc. Camb. Phil. Soc., vol. 48, pp. 277-289, 1952.

[46] Jang-Ping Sheu, Li-Jen Chang, and Wei-Kai Hu. Hybrid congestion control protocol in wireless sensor networks. *J. Inf. Sci. Eng.*, 25(4):1103–1119, 2009.

[47] James Haught, Kenneth Hopkinson, Nathan Stuckey, Michael Dop, and Alexander Stirling. A kalman filter-based prediction system for better network context-awareness. In *Winter Simulation Conference*, pages 2927–2934. WSC, 2010.

[48] N Stuckey. Stochastic estimation and control of queues within a computer network. Master's thesis, Air Force Institute of Technology, 2007.

[49] Alaknantha Eswaradass, Xian-He Sun, and Ming Wu. Network bandwidth predictor (nbp): A system for online network performance forecasting. In *CCGRID*, pages 265–268. IEEE Computer Society, 2006.

[50] E. S. Yu and C. Y. R. Chen. Traffic prediction using neural networks. In *Global Telecommunications Conference, 1993, including a Communications Theory Mini-Conference. Technical Program Conference Record, IEEE in Houston. GLOBECOM '93., IEEE*, pages 991–995 vol.2, Nov 1993.

[51] Hussein Dia. An object-oriented neural network approach to short-term traffic forecasting. *European Journal of Operational Research*, 131(2):253–261, 2001.

[52] Bernard P. Zeigler, Tag Gon Kim, and Herbert Praehofer. *Theory of Modeling and Simulation.* Academic Press, Inc., Orlando, FL, USA, 2nd edition, 2000.

[53] W. Haque and P. R. Stokes. Adaptive speculative locking protocol for distributed real-time database systems. In *19 th IASTED Interntional Conference on Parallel and Distributed Computing and Systems, Cambridge, Massachusetts*, pages 382–390, 2007.

[54] P. Krishna Reddy and M. Kitsuregawa. Speculative locking protocols to improve performance for distributed database systems. *Knowledge and Data Engineering, IEEE Transactions on*, 16(2):154–169, Feb 2004.

[55] Edris Zaman Farsa, Soheila Nazari, and Morteza Gholami. Function approximation by hardware spiking neural network. *Journal of Computational Electronics*, 14(3):707–716, 2015.

[56] Michael J. Crawley. *The R Book*. Wiley Publishing, 2nd edition, 2012.

[57] IBM. *IBM SPSS Modeler 16 User's Guide*, August 2015.

[58] Alex Guazzelli, Kostantinos Stathatos, and Michael Zeller. Efficient deployment of predictive analytics through open standards and cloud computing. *SIGKDD Explor. Newsl.*, 11(1):32–38, November 2009.

[59] Data mining group, pmml version - 4.1. http://dmg.org/pmml/pmml-v4-1.html, 2015.

[60] Robert A. Van Engelen. Pushing the soap envelope with web services for scientific computing. In *In proceedings of the International Conference on Web Services (ICWS), pages 346Ũ352, Las Vegas*, pages 346–352, 2003.

[61] Zementis. *ADAPA Solutions Guide*. Zementis Incorporation, 2015.

[62] Ahmed Louri and Brent Weech. Scalable optical interconnection networks for large-scale parallel computers. In Keqin Li, Yi Pan, and SiQing Zheng, editors, *Parallel Computing Using Optical Interconnections*, volume 468 of *The Springer International Series in Engineering and Computer Science*, pages 47–76. Springer US, 1998.