IMPLEMENTATION OF TCP RL BASED CONGESTION CONTROL SCHEME AND ITS PERFORMANCE EVALUATION UNDER ADVERSARIAL ATTACK

A PROJECT REPORT

Submitted by

HARISH J	**2019103524**
KAUSHIK NARAYAN R	**2019103536**
NANDA KUMAR R	**2019103545**

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING, GUINDY CAMPUS

ANNA UNIVERSITY : CHENNAI 600 025

MAY 2023

BONAFIDE CERTIFICATE

Certified that this project report "**IMPLEMENTATION OF TCP RL BASED CONGESTION CONTROL SCHEME AND ITS PERFORMANCE EVALUATION UNDER ADVERSARIAL ATTACK"** is the bonafide work of "**HARISH J (2019103524), KAUSHIK NARAYAN R (2019103536) AND NANDA KUMAR R (2019103545)**" who carried out the project work under my supervision, for the fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion on these or any other candidates.

Place : Chennai **Dr. Valli S**

Date : Professor,

Department of Computer Science and Engineering,

Anna University, Chennai - 25.

COUNTERSIGNED

Dr. S. Valli

Head of the Department,

Department of Computer Science and Engineering,

Anna University, Chennai.

Chennai – 600025

ACKNOWLEDGEMENT

We express our deep gratitude to our guide, **Dr. S. Valli**, Professor & Head of the Department, Department of Computer Science and Engineering, for guiding us through every phase of the project. We appreciate her thoroughness, tolerance and ability to share her knowledge with us. We would also like to thank her for her kind support and for extending the facilities of the Department towards our project and for her unstinting support.

We express our thanks to the panel of reviewers **Dr. S. Bose**, Professor, Department of Computer Science and Engineering, **Dr. P. Geetha**, Professor, Department of Computer Science and Engineering for their valuable suggestions and critical reviews throughout the course of our project.

We express our thanks to all other teaching and non-teaching staff who helped us in one way or other for the successful completion of the project. We would also like to thank our parents, family and friends for their indirect contribution in the successful completion of this project

HARISH J **KAUSHIK NARAYAN R** **NANDA KUMAR R**

ABSTRACT

In modern computer networking, congestion control plays a crucial role in maintaining the performance, stability, and reliability of networked systems. As network traffic continues to become more complex and dynamic, efficient congestion control mechanisms are becoming more important than ever. This work focuses on TCP congestion control schemes. Machine learning-based congestion control (MLCC) is a relatively new approach to manage network congestion. It uses machine learning algorithms to analyse network conditions and adjust the sending rate of data to prevent congestion. Reinforcement learning-based congestion control (RLCC) is a subset of machine learning-based congestion control that uses reinforcement learning algorithms to adjust the sending rate of data to prevent congestion. This work showcases that reinforcement learning based algorithms can deal with realistic congestion in networks with dynamic and sophisticated state space and have higher online learning capability. Due to RL-based congestion control, systems gain competitive performance and adaptability benefits over traditional approaches by enabling strong learning capabilities. The downside is that RL based systems are prone to adversarial attacks. Adversarial attacks in congestion control refer to deliberate attempts to manipulate or disrupt the performance of congestion control algorithms in computer networks. An adversarial attack is simulated on the RL models to showcase their vulnerability.

திட்டப்பணிச் சுருக்கம்

நவீன கணினி வலையமைப்பில், பிணைய அமைப்புகளின் செயல்திறன், நிலைத்தன்மை மற்றும் நம்பகத்தன்மை ஆகியவற்றை பராமரிப்பதில் நெரிசல் கட்டுப்பாடு முக்கிய பங்கு வகிக்கிறது. நெட்வொர்க் ட்ரா ஃபிக் மிகவும் சிக்கலானதாகவும் மாறும் தன்மையுடனும் தொடர்ந்து வருவதால், திறமையான நெரிசல் கட்டுப்பாட்டு வழிமுறைகள் முக்கியமானதாகி வருகிறது. எங்கள் பணி முக்கியமாக பரப்புகை கட்டுப்பாடு நெறிமுறை திட்டங்களில் கவனம் செலுத்துகிறது. இயந்திர கற்றல் அடிப்படையிலான நெரிசல் கட்டுப்பாடு என்பது நெட்வொர்க் நெரிசலை நிர்வகிப்பதற்கான ஒப்பீட்டளவில் புதிய அணுகுமுறையாகும், இது நெட்வொர்க் நிலைமைகளை பகுப்பாய்வு செய்ய இயந்திர கற்றல் வழிமுறைகளைப் பயன்படுத்துகிறது.வலுவூட்டல்கற்றல் அடிப்படையிலான நெரிசல் கட்டுப்பாடு என்பது இயந்திர கற்றல் அடிப்படையிலான நெரிசல் கட்டுப்பாட்டின் துணைக்குழு ஆகும், இது நெரிசலைத் தடுக்க தரவு அனுப்பும் விகிதத்தை சரிசெய்ய வலுவூட்டல் கற்றல் வழிமுறைகளைப் பயன்படுத்துகிறது. வலுவூட்டல் கற்றல் அடிப்படையிலான வழிமுறைகள், ஆற்றல்மிக்க மற்றும் அதிநவீன நிலை இடவசதியுடன் நெட்வொர்க்குகளில் யதார்த்தமான நெரிசலை சமாளிக்கும் மற்றும் அதிக ஆன்லைன் கற்றல் திறனைக் கொண்டிருக்கும் என்பதை எங்கள் பணி காட்டுகிறது. பாரம்பரிய அணுகுமுறைகளை விட கணினிகள் போட்டி செயல்திறன் மற்றும் தகவமைப்பு நன்மைகளைப் பெறுகின்றன. எதிர்மறையான அம்சம் என்னவென்றால், வலுவூட்டல் கற்றல் அடிப்படையிலான அமைப்புகள் எதிரிகளின் தாக்குதல்களுக்கு ஆளாகின்றன. நெரிசல் கட்டுப்பாட்டில் உள்ள விரோத தாக்குதல்கள் கணினி நெட்வொர்க்குகளில் நெரிசல் கட்டுப்பாட்டு வழிமுறைகளின் செயல்திறனைக் கையாள அல்லது சீர்குலைக்கும் வேண்டுமென்றே முயற்சிகளைக் குறிக்கிறது. இந்த பாதிப்பை வெளிக்காட்ட, அவற்றின் மீது எதிர் தாக்குதலை உருவகப்படுத்துகிறோம்.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS, ABBREVIATIONS AND NOMENCLATURE

AI	Artificial Intelligence
CC	Congestion Control
CPI	Conservative Policy Iteration
DQN	Deep Q-Network
IP	Internet Protocol
LDA	Loss Discrimination Algorithm
ML	Machine Learning
MTU	Maximum Transmission Unit
NS3	Network Simulator 3
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
RTT	Round-trip Time
TCP	Transmission Control Protocol

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW

Congestion control is used to manage the flow of data and prevent network congestion when there is too much data being transmitted over a network and network resources, such as bandwidth, processing power and buffer space, are overwhelmed. It is a critical component of modern network protocols such as TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). The main goal of congestion control is to regulate the rate of data transmission to avoid congestion by monitoring the network's congestion state and adjusting the rate at which data is transmitted accordingly.

Our work focuses mainly on TCP congestion control schemes. Some popular existing schemes include: Slow Start, Fast Recovery, Fast Retransmit, Vegas, etc. These congestion control schemes are implemented in modern TCP implementations, such as TCP New Reno and TCP Cubic. Each scheme has its strengths and weaknesses, and the choice of scheme depends on the specific application and network environment. While these schemes are effective in managing congestion and optimising network performances, they are not without their disadvantages. Some of their disadvantages are slow response to congestion. This is particularly true for slow start, which can take a long time to increase the sending rate of data. They may not efficiently utilise available network bandwidth. This is because they do not take into account the available bandwidth and may send data at a rate that is lower than the available bandwidth. Unfairness is another major problem associated with TCP congestion control schemes. They can be unfair to certain users or applications. For example, AIMD congestion control may allocate more bandwidth to a

high-bandwidth application, while a low-bandwidth application may receive less bandwidth. Existing schemes may be prone to bursts of data transmission, which can cause network congestion. This is particularly true for congestion avoidance, which can cause large fluctuations in the sending rate of data. Overall, while existing TCP congestion control schemes are effective in managing network congestion, they have their limitations.

Machine learning-based congestion control (MLCC) refers to the application of machine learning techniques to improve congestion control algorithms in computer networks. MLCC has the potential to improve network performance, especially in dynamic and heterogeneous networks, by adapting to changing network conditions and optimising network performance. MLCC has been shown to perform better than regular congestion control in certain scenarios, such as when the network conditions are highly variable or when there are multiple traffic flows with different characteristics. It is also better at handling non-congestion-related packet losses, such as due to link failures or retransmission timeouts. MLCC also has its own limitations. It needs large amounts of training data and there is a risk of overfitting to specific network conditions. Supervised and unsupervised machine learning congestion algorithms are based on real-time network states to make control decisions instead of using predetermined rules. But they are trained offline and are not capable of classifying realistic wireless and congestion loss. When the topology and parameters of a network change, a new model needs to be trained. Additionally, MLCC may be more computationally intensive than regular congestion control, which could lead to higher latency or reduced throughput.

Reinforcement learning-based congestion control (RLCC) has the potential to improve network performance by dynamically adapting to changing network conditions and optimising network performance. Reinforcement

learning has been used to address decision-making issues. Reinforcement learning based algorithms can deal with realistic congestion in networks with dynamic and sophisticated state space and have higher online learning capability. Due to RL-based congestion control, systems gain competitive performance and adaptability benefits over traditional approaches by enabling strong learning capabilities. The downside is that RL based systems are prone to adversarial attacks. Adversarial attacks in congestion control refer to deliberate attempts to manipulate or disrupt the performance of congestion control algorithms in computer networks. The goal of an adversarial attack in congestion control is to cause the network to become congested or to slow down, which can negatively impact network performance. It can be achieved by introducing fake or malicious traffic to deceive the congestion control algorithm into thinking that there is congestion and manipulating network parameters or feedback mechanisms to cause the congestion control algorithm to make incorrect decisions. Adversarial attacks on congestion control algorithms are a concern because they can be used to disrupt network operations or to cause denial-of-service (DoS) attacks. For example, an attacker might use an adversarial attack on a congestion control algorithm to disrupt the operations of a critical network or to prevent legitimate traffic from getting through. The attacker cannot directly receive the input states in congestion control settings that are exclusively accessible to the agents, in contrast to the most recent adversarial assaults on pictures where an attacker may quickly gain the input states to add perturbations. So, to attack RL based congestion control systems, it is necessary to estimate states of the target agent, craft adversarial perturbations, and apply the generated perturbations in an automated fashion. A state estimation technique can be used to infer the most significant statistic, i.e., RTT, and experimentally determine other statistics connected with RTT in order to estimate the input state of the victim agent. By deliberately inserting packets into the bottleneck link or arbitrarily omitting packets, a perturbation embedding

strategy can be provided to subtly alter the target RL agent's observations. As a result, the RL agent may comprehend a disrupted network environment using our method. An adversarial perturbation generation model can be used to purposefully inject or remove network packets as opposed to doing so at random. Such well curated adversarial attacks affect the target agent's decision making process. The average throughput of the network reduces and the latency and packet loss rate of the network increases thereby breaking the congestion control made by the RL model.

1.2 OVERALL OBJECTIVE

Congestion is a persistent issue in computer networks that can lead to packet loss, decreased throughput and increased latency. There are several traditional congestion control schemes in use today that use simplistic algorithms that are not adaptive to dynamic network conditions, thereby resulting in inefficient utilisation of network resources. Other disadvantages of existing congestion control schemes include limited scalability, inefficient resource allocation, lack of flexibility and vulnerability to attacks such as denial-of-service or traffic shaping attacks, which can lead to network instability and performance degradation. Therefore, there is a need to develop a machine learning-based congestion control mechanism that can adapt to a network's changing conditions and optimise the network's performance while ensuring fairness among different flows.

Reinforcement learning was chosen for this application since it possesses several advantages over traditional machine learning-based congestion control such as real-time adaptability, ability to handle unexpected situations and scalability. The development of such a reinforcement learning-based congestion control mechanism can improve the efficiency and reliability of computer

networks, leading to better user experiences and increased productivity. The developed congestion control scheme is also tested against adversarial attacks, a type of cyber attack that exploits vulnerabilities in machine learning models by deliberately introducing small, targeted perturbations to input data. This is done in order to compare the congestion control models.

1.3 PROBLEM STATEMENT

Congestion is a persistent issue in computer networks that can lead to packet loss, decreased throughput and increased latency. There are several traditional congestion control schemes in use today that use simplistic algorithms that are not adaptive to dynamic network conditions, thereby resulting in inefficient utilisation of network resources. Other disadvantages of existing congestion control schemes include limited scalability, inefficient resource allocation, lack of flexibility and vulnerability to attacks such as denial-of-service or traffic shaping attacks, which can lead to network instability and performance degradation. Therefore, there is a need to develop a machine learning-based congestion control mechanism that can adapt to a network's changing conditions and optimise the network's performance while ensuring fairness among different flows. Reinforcement learning was chosen for this application since it possesses several advantages over traditional machine learning-based congestion control such as real-time adaptability, ability to handle unexpected situations and scalability. The development of such a reinforcement learning-based congestion control mechanism can improve the efficiency and reliability of computer networks, leading to better user experiences and increased productivity. The developed congestion control scheme is also tested against adversarial attacks.

1.4. CHALLENGES & APPLICATIONS

1.4.1. Challenges

1. RL algorithms need to generalise well to diverse network conditions and traffic patterns.They should be able to adapt to varying network topologies, link capacities, and traffic loads.
2. One of the major challenges is scaling RL-based congestion control algorithms to large networks.Traditional RL algorithms often require extensive exploration and interaction with the environment, which can be computationally expensive and time-consuming.
3. Designing appropriate reward functions is a crucial challenge in RL-based congestion control.The reward function should incentivize desirable behaviour, such as maximising throughput or minimising delay, while discouraging harmful actions that could lead to congestion or network instability.
4. RL algorithms require significant amounts of training data to learn effective policies. However, collecting real-world training data for congestion control can be challenging due to the complexities and dynamics of network environments.
5. Congestion control algorithms need to adapt to changing network conditions in real-time.RL algorithms should be capable of quickly adapting to variations in network dynamics, such as fluctuating traffic loads or link failures, to maintain optimal performance.

1.4.2. Applications

1. RL-based congestion control can be applied to manage congestion in data centre networks, where a large number of servers are interconnected. By using RL agents to control traffic flows within the data centre, it is possible to optimise resource allocation, reduce congestion, and improve the overall quality of service for applications running in the data centre environment.

2. RL-based congestion control can be applied to manage traffic congestion in autonomous vehicle networks.RL agents can learn to optimise vehicle routing, speed control, and intersection management to minimise traffic flow, and enhance overall traffic efficiency.

1.5 ORGANISATION OF THE THESIS

The outline of the thesis is as follows:

Chapter 2 details the summary of related work in the area of heuristic-based, machine learning-based and reinforcement learning-based congestion control algorithms. Chapter 3 illustrates the architecture of the system along with individual module details. Chapter 4 discusses the performance of the congestion control models and the metrics used to evaluate the model. A comparative analysis of our model and traditional congestion control models is also included. Chapter 5 concludes this thesis by summing up the project and provides proposals for future work to improve upon our work.

CHAPTER 2
SUMMARY OF THE RELATED WORK

The problem of network congestion can be solved through different approaches. Heuristic-based congestion control is a widely used approach where the congestion control algorithms rely on predefined rules or heuristics to estimate the level of congestion in the network and adjust the transmission rate accordingly. These rules are often based on observations and experience, and they are designed to be simple and efficient. Examples of heuristic-based congestion control algorithms include TCP Vegas, TCP Westwood, and TCP Friendly.

In contrast, machine learning-based congestion control algorithms use machine learning techniques to learn the optimal transmission rate based on feedback from the network. These algorithms can adapt to changing network conditions and can learn from experience to optimise their performance. Examples of machine learning-based congestion control algorithms include Reinforcement Learning-based Congestion Control (RLCC), Deep Reinforcement Learning-based Congestion Control (DRLCC), and Neural Adaptive Video Streaming (NAVS).

2.1 Heuristic-based congestion control algorithms

TCP LoLa is delay-based congestion control scheme, Hock et al., (2017), that supports both, low queuing delay and high network utilisation in high speed wide-area networks. It attains flow rate fairness independent of the round-trip times of different flows in the network. Similar to other congestion control schemes, TCP Lola has the following main states: slow start, cubic increase and

fair flow balancing. It is shown to achieve high link utilisation and attains convergence to fairness even among flows with different round-trip times, due to its novel mechanism called “fair flow balancing”. Additionally, it avoids packet losses and is scalable from lower speed networks. It has been implemented as a Linux kernel module and has been evaluated in a physical testbed. TCP LoLa has also been shown to keep the delay if multiple flows are started consecutively and run in parallel.

BBR (Bottleneck Bandwidth and Round-trip propagation time) is a congestion control algorithm developed by Google (Cardwell et al., 2017). It is designed to improve network performance by achieving higher throughput and lower latency while avoiding congestion and packet loss. BBR works by estimating the available bandwidth and the round-trip time of the network path and adjusting the transmission rate accordingly. It does this by continuously measuring the number of packets in flight and the time it takes for the packets to traverse the network path. It builds a model using the network's delivery rate and RTT to respond to actual congestion, rather than packet loss. BBR uses two main modes of operation: the bandwidth probing mode and the application-limited mode. In the bandwidth probing mode, BBR probes the network to determine the maximum achievable bandwidth, and in the application-limited mode, BBR limits the transmission rate to the rate at which the application can consume data. It has been shown to improve network performance in terms of throughput, latency, and fairness compared to traditional congestion control algorithms, such as TCP Reno and TCP Cubic.

2.2 Machine learning-based congestion control algorithms

The authors Han et al., (2019), have attempted to solve the performance degradation due to random loss by proposing a new loss discrimination algorithm (LDA) which applies a machine learning technique. A multi-layer perceptron is trained by a supervised learning technique that uses the minimum RTT and sRTT value as input and random or congestion loss as output. The ML-LDA learning is performed by collecting data for various network situations such as different bottleneck bandwidth and delay. The algorithm adjusts the congestion window size appropriately by distinguishing packet loss due to network congestion and packet loss due to channel error in order to solve the problem of deterioration of congestion control performance. The results show that congestion control performance is improved in wireless network environments with high loss rate.

The authors Kim et al., (2020) have improved the inter-protocol fairness for BBR congestion control using an opponent congestion control identification model of BBR with machine learning. Severe throughput imbalances have been reported when BBR coexists with loss-based congestion control in a small buffer. The proposed model is based on a decision tree classifier that can identify the opponent congestion control. It has been shown to achieve higher inter-protocol fairness than BBR in competition with CUBIC. The model was only trained for TCP CUBIC CC and its performance against other CC algorithms is unknown.

The authors Li et al., (2019) have proposed a machine learning-based congestion control strategy AdaBoost-TCP that is shown to perform well in highly dynamic situations such as satellite networks where frequent switching

of links leads to in-stability of the connection. AdaBoost-TCP constructs an adaptive Boost recognition model that can effectively classify the packet loss type in the satellite network. The receiver uses the model to differentiate the type of lost packets and combines the ECN flag to transmit the result to the sender. The sender adopts adaptive congestion control measures according to the type of packet loss obtained. Ada-Boost-TCP is shown to have better classification speed and higher efficiency without increasing network load. AdaBoost works by training different weak classifiers for the same training set, and then combines these classifiers to form the final strong classifier. The authors Huang et al., (2022) proposed a TCP CC algorithm, CFCC based on the Proximal Policy. The algorithm model is based on CNN and F-CNN (fuzzy based CNN) and uses network features such as throughput and round-trip time for training.

2.3 Reinforcement Learning-based congestion control algorithms

An RL-based algorithm for network congestion control in datacenters (Tessler et al., 2022) with the aim of generalising to different configurations of real-world datacenter networks has been attempted. A policy gradient algorithm that leverages the analytical structure of the reward function to approximate its derivative and improve stability is also proposed. An Analytic Deterministic Policy Gradient (ADPG) scheme that makes use of domain knowledge to estimate the gradient for a deterministic policy update was presented. As this task lacks a ground truth reward function, they show a fitting reward function and show that this reward function, in a multi-agent partially-observable setting, leading to convergence to a global optimum. An RL environment, based on a realistic networking simulator based on OMNeT++ (Varga 2002), emulates the behaviour of state of the art hardware deployed in current data centers: ConnectX-6Dx Network Interface Card (NIC). In addition, the generalisation

and robustness of the agent is tested by porting it to real hardware and evaluating it there. The results indicate that Analytic Deterministic Policy Gradient (ADPG), learns a robust policy and outperforms the current state-of-the-art methods deployed in real data centers. A novel framework for RL-based congestion control design based on Performance-oriented congestion control called Aurora (Jay et al., 2019) has been proposed. It employs deep RL to generate a policy for mapping observed network statistics such as latency and throughput to choices of rates. Aurora has been shown to outperform state-of-the-art congestion control schemes such as BBR, Copa and RemyCC even with fairly limited training.

2.4 Adversarial attacks on reinforcement learning systems

In reinforcement learning, an agent learns to make decisions by interacting with an environment and receiving feedback in the form of rewards. Adversarial attacks can target the agent's policy, which is a function that maps the agent's observations to actions. One type of adversarial attack in reinforcement learning is the reward shaping attack, where an attacker changes the reward function to encourage the agent to take actions that benefit the attacker. Another type of adversarial attack in reinforcement learning is the observation poisoning attack, where an attacker manipulates the agent's observations to cause it to make incorrect decisions.

Deep Q-Networks (DQNs) have been shown to be vulnerable to adversarial input perturbations (Behzadan et al., 2017). A novel attack is presented where the attacker’s goal is to perturb the optimality of actions taken by a DQN learner. In this model, the attacker has no direct influence on the target’s architecture and parameters, including its reward function and the optimization mechanism. The only parameter that the attacker can directly manipulate is the

configuration of the environment observed by the target. To avoid detection and minimise influence on the environment's dynamics, a constraint has been imposed on the attack such that the magnitude of perturbations applied in each configuration must be smaller than a set value. The results confirm the efficacy of the proposed attack mechanism, and verify the vulnerability of Deep Q-Networks to policy induction attacks.

CHAPTER 3
SYSTEM DESIGN

3.1 ARCHITECTURE DIAGRAM

The framework represented in Figure 3.1 outlines the approach to building an RL based congestion control scheme. The RL environment consists of creating a dumbbell network topology, a state gatherer which sends the network parameters to the RL agent, an actuator which collects and performs the actions (adjust congestion window) sent by the agent, and a reward generator which calculates the reward for the performed actions. The RL agent module consists of a policy function which maps the gathered state to a desired action(increase/decrease congestion window) and a learning algorithm (DQN and PPO) that help in learning the optimal policy function. The attacker module collects the state from the environment, perturbs it, and takes action on the perturbed state. The evaluation module compares the performance of the RL agent with a traditional CC algorithm like NewReno and the performance degradation of the agent before and after an attack.

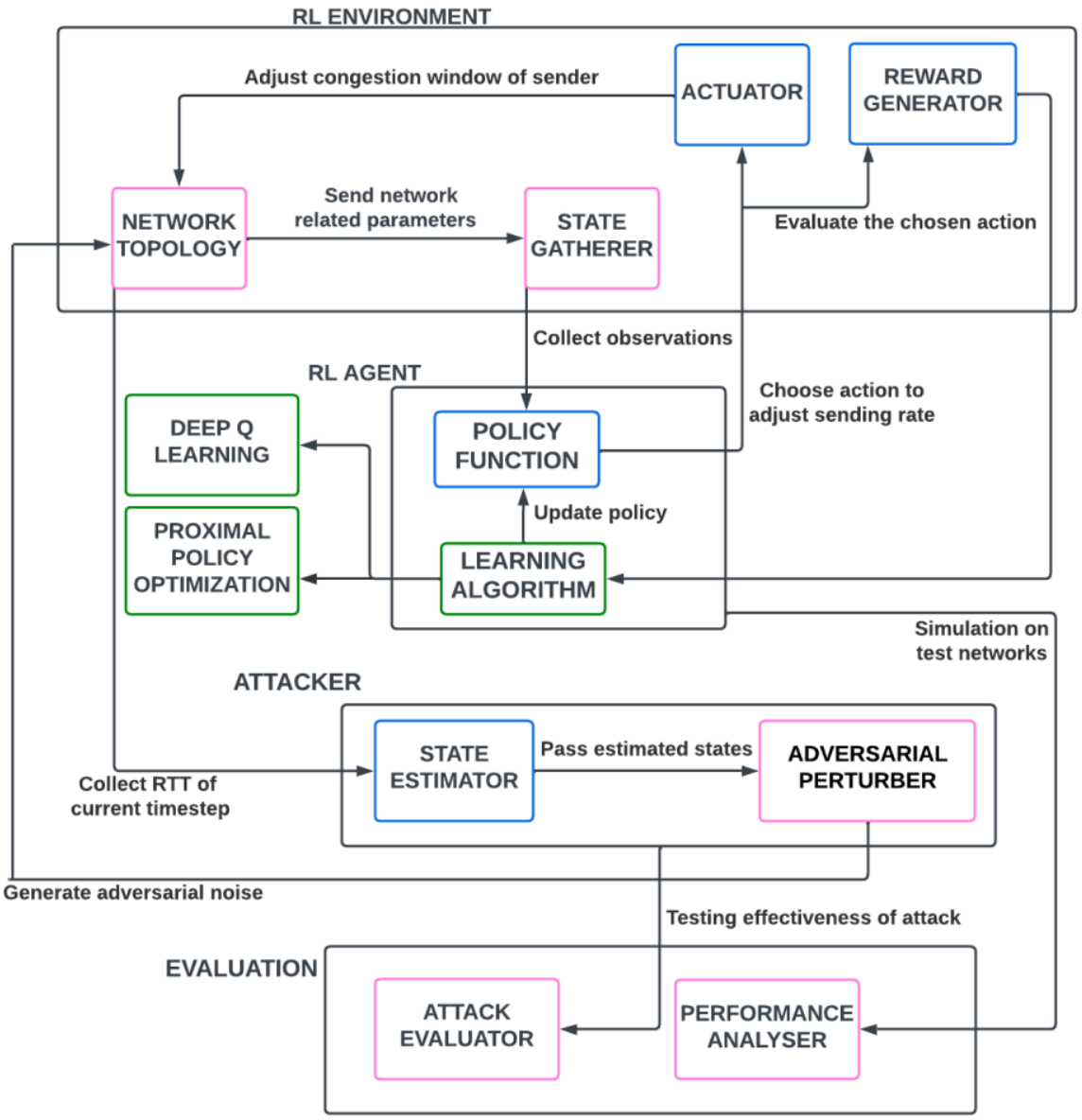

Figure 3.1: Architecture Diagram

3.2 DETAILED MODULE DESIGN

3.2.1 MODULE 1 - Environment Setup

In RL, the environment refers to the external system or process with which an RL agent interacts to make decisions. The environment defines the state of the system, the set of possible actions that the agent can take, and the rewards that the agent receives for each action. In RL, for TCP congestion control, the environment is the network that the TCP flows are traversing. NS-3 simulator is used to generate traffic and also simulate the environment. Figure 3.2 describes the environment setup.

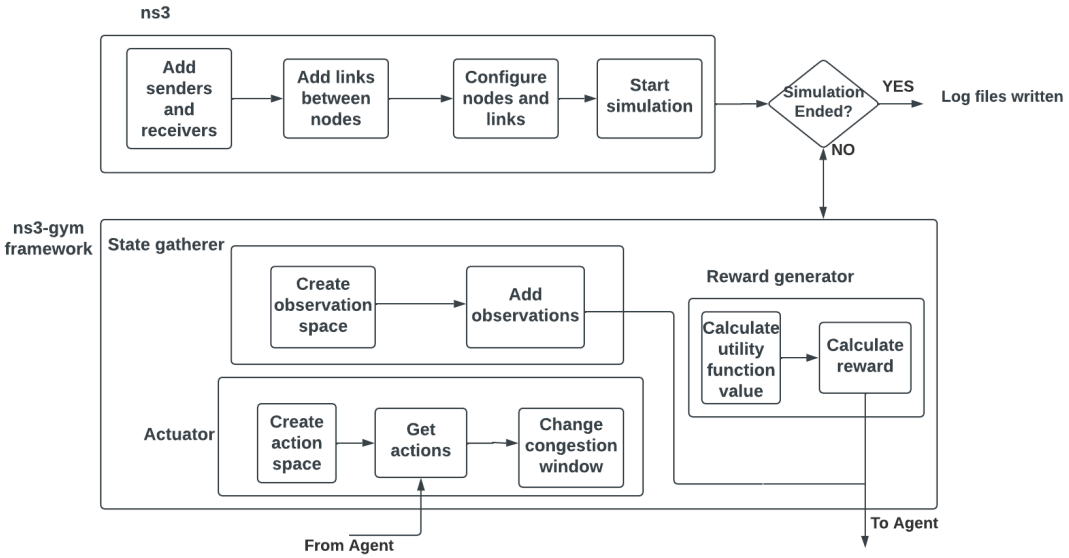

Figure 3.2: Environment setup module

INPUT	:	Modules offered by ns3 to create a topology
OUTPUT	:	A simulated network topology with the creation of state, action, and reward handlers for the agent to interact with

i) Creation of network topology:

A dumbbell topology is created where senders are present on one side and the receivers on the other. A bottleneck link exists between them through which packets are sent and received. Steps to create such a network using ns3 are as follows:

- Specify the number of senders and receivers. Senders are left leaves, and receivers are right leaves.
- Mention the bottleneck bandwidth and delay. Since an RL based model is used to control congestion in the bottleneck, these parameters can be varied to test the models.
- TCP, IP stacks are set up in senders and receiver sides.
- The size of packets in bytes that can be transmitted is set. It's basically the maximum transmission unit(MTU) of the network.
- Allocate buffer space for the routers to process packets.
- Routing tables are initialised to handle the routing of packets.
- IP addresses are assigned to all the network components in our topology.
- Create application containers in the leaves and assign port numbers which enables the creation of sockets between source and sink nodes.

Algorithm

1. *Fix the number of senders and receivers*
2. *Create links between nodes*
3. *Fix the bottleneck bandwidth and total delay*
4. *Configure nodes and routers*
5. *Assign IP addresses and port numbers to all nodes*
6. *Fix the duration for the traffic flow*
7. *Start the simulator and inspect packets*

ii) State Gatherer:

ns3-gym is used as a framework for integrating the NS-3 simulator coded in C++ and the RL agent in Python. This interface takes care of the management of the NS-3 simulation process life cycle as well as delivering state and action information between the Gym agent and the simulation environment.

The observation space includes important parameters of topology like,

- Slow start threshold
- Congestion window
- Segment size
- Total number of bytes in transit
- Average number of bytes in transit
- Total number of ACKed segments
- Average number of ACKed segments
- Average RTT
- Min RTT
- Average inter-transmission time
- Average inter-receiving time
- Throughput

The “Box” space is used for representing observations. It is a vector or matrix of numbers of a single type with values bound between the low and high limits. An example definition of such a space using ns3-gym is,

```
uint32_t parameterNum = 16;
float low = 0.0;
float high = 1000000000.0;
std::vector<uint32_t> shape = {parameterNum,};
std::string dtype = TypeNameGet<uint64_t> ();
Ptr<OpenGymBoxSpace> box = CreateObject<OpenGymBoxSpace> (low,
high, shape, dtype);
```

Here, "parameterNum" is the total number of parameters in the observation space, and the range of the values lies between "low" and "high." During every step of execution, the framework collects the current state of the environment by calling the “GetObservation" function. The values of the network parameters mentioned above are added to the box space defined. An example for adding Average RTT to our space,

```
Time avgRtt = Seconds(0.0);
if(m_rttSampleNum) {
    avgRtt = m_rttSum / m_rttSampleNum;
}
box->AddValue(avgRtt.GetMicroSeconds ());
```

Here, the sum of RTTs is divided by the number of packets that were sent to calculate the average RTT, and it is added to the observation space.

The NS3-Gym framework delivers the collected environment’s state to the agent, which in turn sends the action to be executed.

Algorithm

1. *Choose the type of space needed to store the observations-box, dict, tuple or discrete*
2. *Fix the number of network parameters to store*
3. *Define the method for calculation of the parameters*

iii) **Actuator:**

A "Box" space is defined to encode the actions made by the agent. The agent returns the new congestion window and slow start threshold to the environment, and the actuator replaces the old values with the new ones.
An example definition of an action space is as follows,

```
uint32_t parameterNum = 2;
float low = 0.0;
float high = 65535;
std::vector<uint32_t> shape = {parameterNum,};
std::string dtype = TypeNameGet<uint32_t> ();
Ptr<OpenGymBoxSpace> box = CreateObject<OpenGymBoxSpace> (low,
high, shape, dtype);
```

We create a "Box" space of 2 dimensions, cwnd and ssthresh, which can take values in the range [0, 2^16-1].

Now, to execute the action, the "ExecuteActions()" API provided by ns3-gym is called.

```
m_new_ssThresh = box->GetValue(0);
m_new_cWnd = box->GetValue(1);
```

At the end of a timestep (i.e., a predefined interval, say every 100 ms), the old values of the congestion window and slow start threshold are updated with the new ones.

Algorithm

1. *Choose the type of space to store the actions*
2. *Allocate space for storing actions-CWND and SSTHRESH*
3. *Receive actions sent by the agent*
4. *For all the active TCP socket states (tb), do:*

```
tb->Cwnd=m_new_Cwnd
tb->ssThresh=m_new_ssThresh
```

iv) **Reward Generator:**

For each action selected by the agent, the environment provides a reward which is usually scalar value. The agent has an objective of choosing those actions which gives the maximum expected reward sum at the end of the process. An utility function which takes into consideration the throughput, bandwidth and end-to-end delay is used for calculation of reward.

The utility function is given by Eqn 3.1,

$$U = \log(t /B) - \delta 1.\log(d) \quad (3.1)$$

where t denotes end-to-end throughput, d denotes end-to-end delay and B denotes bandwidth. The difference in utilities is given by Eqn 3.2,

$$\Delta_t = U_t - U_{t-1} \quad (3.2)$$

where t is the current time step.The reward is calculated based on Eqn 3.3.

$$\text{Reward(s)} = \begin{cases} 10, & \Delta_t \geq 0; \\ 2, & 0 \leq \Delta_t < 1; \\ -2, & -1 \leq \Delta_t < 0; \\ -10, & \Delta_t < -1; \end{cases} \tag{3.3}$$

The ns3-gym's "GetReward()" does all the above computation and returns the calculated reward as output to the agent.

Algorithm

1. *Initialise old_U=0 ,reward=0*
2. *Calculate delay=Current RTT-Minimum RTT*
3. *Calculate segmentsAckedSum as the sum of segments that received ACK's*
4. *Throughput=segmentsAckedSum/Bandwidth*
5. *utility_value=log(throughput)-log(delay)*
6. *delta=utility_value-old_U*
7. *If delta>=0 then reward=10*
8. *Else if delta>=0 and delta<1 then reward=2*
9. *Else if delta>=-1 and delta<0 then reward=-2*
10. *Else if delta<-1 then reward=-10*

3.2.2 MODULE 2 - Agent Creation

The agent continuously collects network states by monitoring network environments. It leverages a decision-making policy to take corresponding actions based on a given state. To adapt to changes in the network environment, the RL agent iteratively updates the policy in each time period. It uses a learning algorithm to update the policy based on the state, action, and reward obtained in a particular time step. The learning objective is to maximise the expected cumulative discounted future reward. Figure 3.3 describes the DQN agent creation.

INPUT	:	State information from environment
OUTPUT	:	The trained agent which can adapt to diverse network changes

DEEP Q-LEARNING ALGORITHM

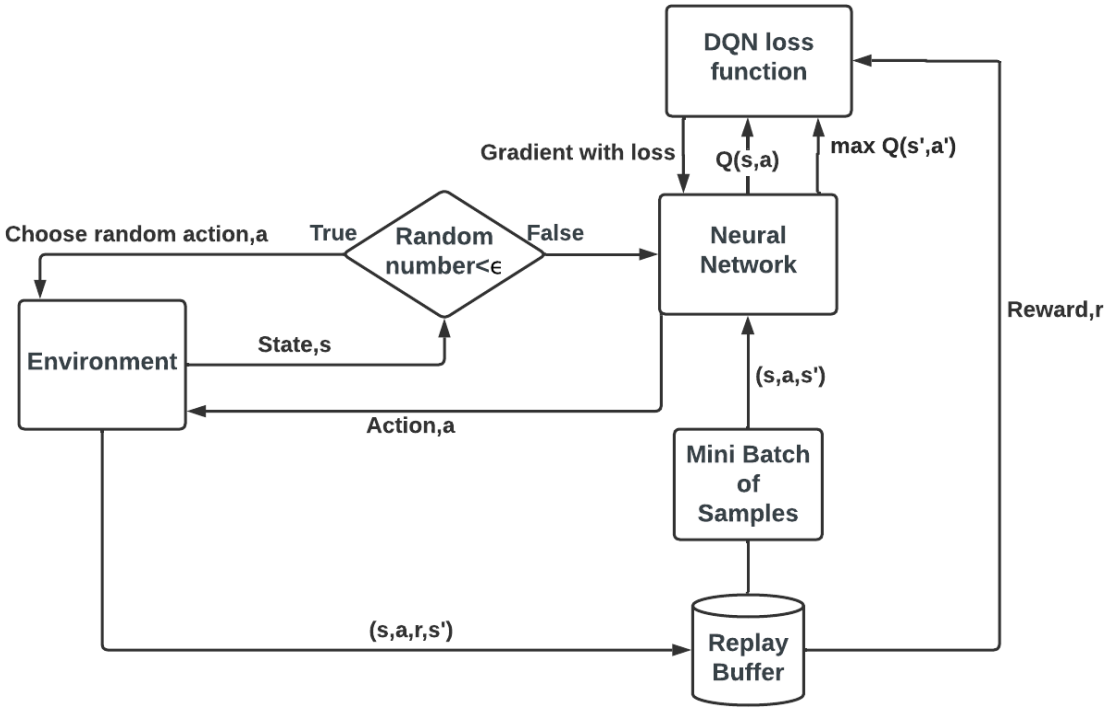

Figure 3.3: DEEP Q-LEARNING ALGORITHM

i) Policy function:

The agent adopts a decaying epsilon-greedy strategy as in Eqn 3.4 to balance exploration and exploitation, i.e., choosing a random action with probability epsilon ϵ, and a greedy action with probability 1-ϵ.

We set the epsilon value to 0.1.

$$\text{Policy(s)} = \begin{cases} a_{random}, & p < \epsilon, \\ arg\,max_a\, Q(s,a), & p \geq \epsilon. \end{cases} \tag{3.4}$$

Algorithm

1. *rand_num=random()*
2. *If rand_num<epsilon:*
   pull random action
3. *Else:*
   Pull current-best action

ii) Learning method:

Q-Learning is not suitable in cases where the state and action spaces are very large. So, deep Q learning, which uses deep neural networks to approximate values is used.The basic working step for deep Q-learning is that the initial state is fed into the neural network, which returns the Q-value of all possible actions as an output. The neural network architecture in use has one input layer and one hidden layer with the number of neurons equal to the size of the state space. A linear network space is used as an output layer, which maps the state space to the action space. “ReLu” activation is used in the input and hidden layers.

The Q-value is updated iteratively according to Eqn 3.5,

$$Q_{i+1}(s_t,a;\theta)=E[r_{t+1}+\gamma.\max_{a'}(Q_i(s_{t+1},a';\theta))|(s_t,a_t;\theta)] \quad (3.5)$$

θ in Eqn 3.5 refers to the neural network weights. These weights are updated to find the optimal Q function. γ is the discount factor which is a measure of how much importance is given to the distant future rewards relative to the immediate future. s_t refers to the current state, a refers to the current action taken and r_{t+1} is the reward obtained. s_{t+1} is the next state and a' is the action which gives maximum discounted reward to the current state s_t.i refers to the current iteration and $i+1$ is the next iteration.

The squared-error loss function is given by Eqn 3.6,

$$\text{Loss}=E[r_{t+1}+\gamma.\max_{a'}(Q(s_{t+1},a';\theta))-Q(s_t,a;\theta)]^2 \quad (3.6)$$

After the loss is calculated, the neural network weights are updated via gradient descent and backpropagation. This process is repeated till the loss is minimised. A technique called "Experience Replay" is used to train the DQN. The idea is that the agent stores all its experiences in a memory buffer called a replay memory buffer. At time step 't', the experience(e_t) in Eqn 3.7 is a tuple containing the current state of the environment, the chosen action, the reward and the next state of the environment.

$$e_t = (s_t,\ a_t,\ r_{t+1},\ s_{t+1}) \quad (3.7)$$

Experience Replay technique samples observations randomly from the buffer to break the correlations that exist when consecutive samples are used and improves the efficiency of learning.

Algorithm

1. *Initialize replay buffer D*
2. *Model a neural network with random weight vector θ*
3. *for each iteration do:*
   - *3.1. Get state s from the environment*
   - *3.2. for each episode do:*
     - *3.2.1 With probability ϵ, choose a random action;*
       otherwise choose $a_t = argmax_a\ Q(s, a; \Theta)$
     - *3.2.2. Update congestion window using a_t and send it to*

the environment

3.2.3. Receive reward r_{t+1} and next state s_{t+1}

3.2.4. Store experience $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ in buffer D

3.2.5. Sample a mini-batch of experiences

$e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ from D

3.2.6. Compute $Q_{target}=r_{t+1}+\gamma.max_{a'}(Q(s_{t+1},a';\theta))$

3.2.7. Compute MSE loss between Q_{target} and $Q(s_t,a_t)$

3.2.8. Update weights θ by gradient descent

PROXIMAL POLICY OPTIMIZATION(PPO) ALGORITHM

PPO is a typical policy-based RL congestion control system to illustrate technical details. It employs deep RL to learn congestion control policy that decides the sending rate of the next time period according to network states. It changes the action of sending rate every Monitor Interval (MI) based on network states of several MIs in an observing window. Here, MIs are continuous equal time periods, whose length is normally one to two RTTs. In the beginning of each MI, the sender can adjust its sending rate x(t) , which then remains fixed within each MI.Figure 3.4 describes the PPO agent creation.

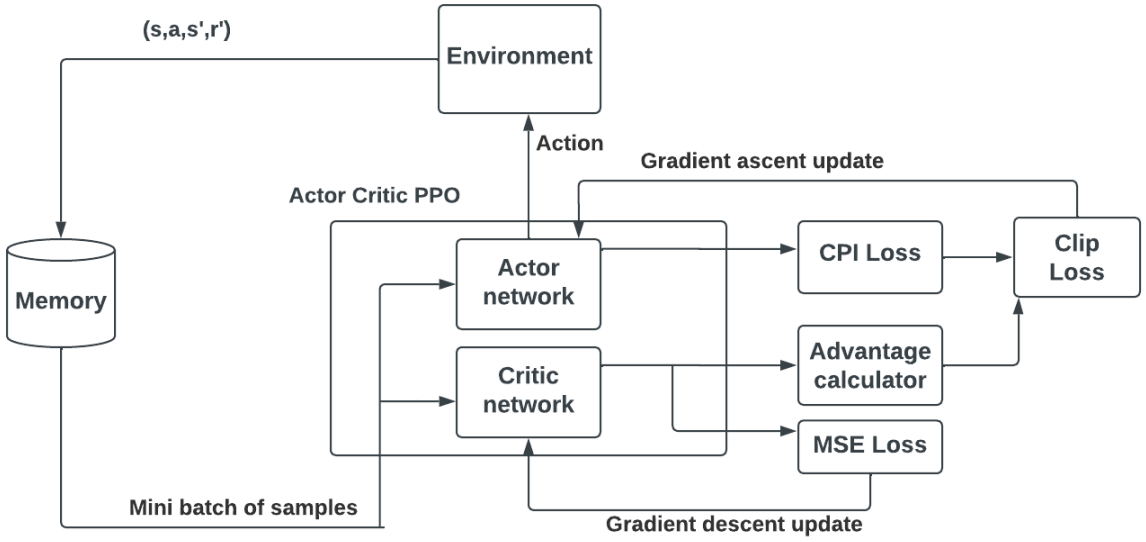

Figure 3.4: PROXIMAL POLICY OPTIMIZATION(PPO) ALGORITHM

i) Policy function:

$$\text{Policy}(x_t, \square) = \begin{cases} a_{t-1} * \left(1 + \beta x_t\right), & x_t > 0, \\ a_{t-1} / \left(1 + \beta x_t\right), & x_t < 0. \end{cases} \quad (3.8)$$

Here, a_{t-1} in Eqn 3.8 denotes the sending rate at the (t − 1)th MI; x(t) denotes the state value which is calculated by using the state vector st, and β is a scaling factor.

ii) Learning algorithm:

Proximal policy based optimization (PPO) is an on-policy algorithm, meaning it uses the current policy to collect experience data, and updates the policy parameters based on that experience. The key idea behind PPO is to optimise the policy while putting a constraint on the size of the update step, to ensure that the updated policy does not deviate too far from the original policy. It uses a trust region constraint to bound the change in the policy.

A PPO agent adopts an actor-critic (AC) architecture and trains two models, i.e., the actor model and the critic model. The actor model, which is the policy structure, performs the task of learning what actions to take under a particular observed state. The critic model learns the estimated value function that evaluates the actions taken by the actor model. PPO uses a replay buffer to learn efficiently from past samples. States, actions, rewards, probabilities of choosing an action (to create the policy distribution), values, and rewards are stored in the buffer. The advantage value is an important factor in the PPO

algorithm because it helps the agent learn more efficiently by focusing on actions that are more likely to lead to good outcomes.

$$A_t = r_{t+1} + \gamma . V(s_{t+1}; \theta_v) - V(s_t; \theta_v) \tag{3.9}$$

In Eqn 3.9, A_t is the advantage value, γ is the discount rate, θ_v is the weights of the critic network and $V(s_t)$ is the value function computed by the critic. Regarding loss functions, for an actor network a loss called CPI (Conservative Policy Iteration) is calculated as in Eqn 3.10. It is the expectation ratio between the policy under old parameters to the policy under new parameters multiplied by the advantage value.

$$L_{CPI}(\theta) = E\left[\frac{\Pi_\theta(a_t|s_t)}{\Pi_{\theta_{old}}(a_t|s_t)} A_t\right] = E[r_t(\theta) A_t] \tag{3.10}$$

The calculated weighted probability value($r_t(\theta)$) would be larger which results in large parameter updates to the model. To avoid this, PPO adds one more additional parameter called epsilon. With the help of this ϵ, the ratio is clipped between 1-ϵ to 1+ϵ, according to Eqn 3.11.

$$L_{CLIP}(\theta) = E[min(r_t(\theta) A_t, clip(1-\epsilon, 1+\epsilon, r_t(\theta)) A_t] \tag{3.11}$$

The critic loss is the MSE between the predicted value function and the true value function.The predicted value function also called as “returns” is calculated as the sum of advantages and values of a batch of samples. The weights of the actor and critic networks are updated using the above losses respectively.

Algorithm

1. *Initialize buffer D*
2. *Model the actor network θ and critic network θ_v*
3. *for each iteration do:*
    - *3.1. Get state s from the environment*
    - *3.2. for each episode do:*
        - *3.2.1. Pass s to actor network to get action a_t using current policy Π.*
        - *3.2.2. Update congestion window using at and send it*
        - *3.2.3. Receive reward r_{t+1} and next state s_{t+1}*
        - *3.2.4. Store experience $e_t=(s_t, a_t, r_{t+1}, s_{t+1})$ in buffer D*
        - *3.2.5. Sample a mini-batch of experiences $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ from D*
        - *3.2.6. Critic network finds the advantage of the Sample using $A_t=r_{t+1}+\gamma.V(s_{t+1};\theta_v)-V(s_t;\theta_v)$*
        - *3.2.7. Actor finds $L_{CLIP}(\theta)$*
        - *3.2.8. Critic finds MSE between true and predicted values*
        - *3.2.9. Actor updates weights θ using gradient ascent*
        - *3.2.10. Critic updates weights θ_v using gradient descent*

3.2.3 MODULE 3 - Adversarial Attacker

To launch adversarial attacks, an attacker who impersonates himself as legitimate sender does the following steps.

- Collect the observations received by the agent from the environment
- Generate adversarial perturbation based on the received observations

- Apply the generated perturbation to the observation before it is collected by the victim agent.

INPUT	:	Observation inferred from the environment
OUTPUT	:	The corrupted observation makes the agent take sub optimal actions

i) State Estimator:

The attacker infers the most significant statistic, i.e., RTT, and empirically sets other statistics based on it. This is because RTT is the most significant signal indicating how congested the current network is in a congestion control system. The other network statistics are like,

a) Inter-packet sending time

b) Inter-ACK arrival time

c) Latency ratio

d) Latency gradient

are set using inferred RTT as an estimation value.

The sending ratio is set using the latency ratio as an empirical value. Higher the latency ratio,the more congested the current network, and the more likely packet loss occurs.

$$Sending\ Ratio = \lambda * Latency\ Ratio + b \tag{3.12}$$

'λ' in Eqn 3.12 is a scaling factor. 'λ' is set to 0.027 and 'b' is set to 0.973.

The congestion window(cwnd) and slow start threshold (ssthresh) are simply set to the initial value, i.e., the initial ssthresh and cwnd.

ii) Adversarial perturber:

To construct the attack, an adversarial perturbation generation algorithm is used to corrupt the input observations and fool the agent into taking wrong actions. The attacker needs access to the victim model to get the trained target policy, Policy(), and trained target critic function, Critic() function (for policy gradient based algorithms) and Q-values (for value based algorithms). To achieve this, the attacker trains a replica of the target agent and exploits the transferability property of adversarial examples.

Algorithm

Procedure Perturbation():

1.Deploy a RL agent on a client

2.Train the agent using either "DQN" or "PPO" learning algorithms

3.If "DQN" algorithm was used:

Q_model=trained_Q_model()

4.If "PPO" algorithm was used:

Policy=trained_policy()

Critic=trained_critic()

5. Adversarial attack magnitude(ϵ)=random(0,1)

6. Number of times to sample noise(n)=10

7. α=-1

8. β=1

9. While True do:

10. Initialize current_state from environment

11. If "PPO" is used:

12. A=Policy(current_state),

V=Critic(current_state,A)

13. $n_{adv}=0$

14.for i=1 to n do:

15.$n_i=Uniform(\alpha,\beta)$

16.$s_i=s+\epsilon * n_i$

17.$A_{adv}=Policy(s_i)$

18. $V_{adv}=Critic(s_i,A_{adv})$

19.If $V_{adv}< V$ then

$V= V_{adv}$

$n_{adv}=n_i$

20.Else if "DQN" is used:

21.A=Q_model(current_state)

22.$n_{adv}=0$

23.for i=1 to n do:

24.$n_i=Uniform(\alpha,\beta)$

25.$s_i=s+\epsilon * n_i$

26.$A_{adv}=Q\_model(s_i=)$

27.If $A_{adv}< A$ then

28.$A=A_{adv}$

29.nadv=ni

30.start_time=get_current_time()

31.end_time=start_time+perturbation_interval

32.While start_time <=end_time:

33.Add_Perturbation(nadv)

34.start_time=get_current_time

Procedure Add_Perturbation(nadv):

1.inject_rate=bottleneck_bandwidth

2.Initialize loss_rate

3. If $n_{adv}>0$ then

3.1.sending_rate=inject_rate

3.2.update_sending_rate(sending_rate)

4.Else If $n_{adv} < 0$ *then*

4.1.packet_drop_rate=loss_rate

4.2.update_drop_rate(packet_drop_rate)

5.Else

5.1.Do-nothing

3.2.4 MODULE 4 - EVALUATION:

The evaluation procedure is divided into 2 classes:

i) Comparison of DQN and PPO learning algorithms by varying bandwidth, latency,number of senders, queue size (Performance Analyser).

ii) Evaluation of the effectiveness of attack on a RL based CC system by comparing increase in RTT rate before and after an attack is conducted (Attack evaluator).

INPUT	:	Trained DQN and PPO based learning agents
OUTPUT	:	Graphs depicting the model's robustness towards change in bandwidth, delay and number of users

i) PERFORMANCE ANALYSER:

The robustness of the model is evaluated by testing the model's behaviour when bandwidth, latency, and number of leaves vary far outside of our training conditions.

Bandwidth sensitivity: The models (DQN, PPO) will be trained on links with bandwidth of 10 Mbps, but a reasonable congestion control system should

operate in a much wider range. The standard link configuration is changed and the bandwidth is configured differently for each test, ranging from 5 to 20 Mbps.

Latency sensitivity: To test the robustness of the trained models on links with different latency, the latency is operated between 60 ms and 240 ms.

Number of leaves: The number of senders is varied in the range [3,6].

ii) ATTACK EVALUATOR:

The bandwidth of the bottleneck link is varied from 10 Mbps to 20 Mbps and RTT is fixed as 30 ms. Ideally, after an attack has taken place, the agent makes poor actions at each state and thereby the congestion window will not be set to its optimum value at every timestep. This can increase the RTT in the environment.

Round-Trip Time: The evaluation is done on the network latency with different bandwidths and attack magnitude ϵ. RTT is highly impacted by the link bandwidth. The smaller the bandwidth, the larger the RTT under attack.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1. TOPOLOGY DESCRIPTION:

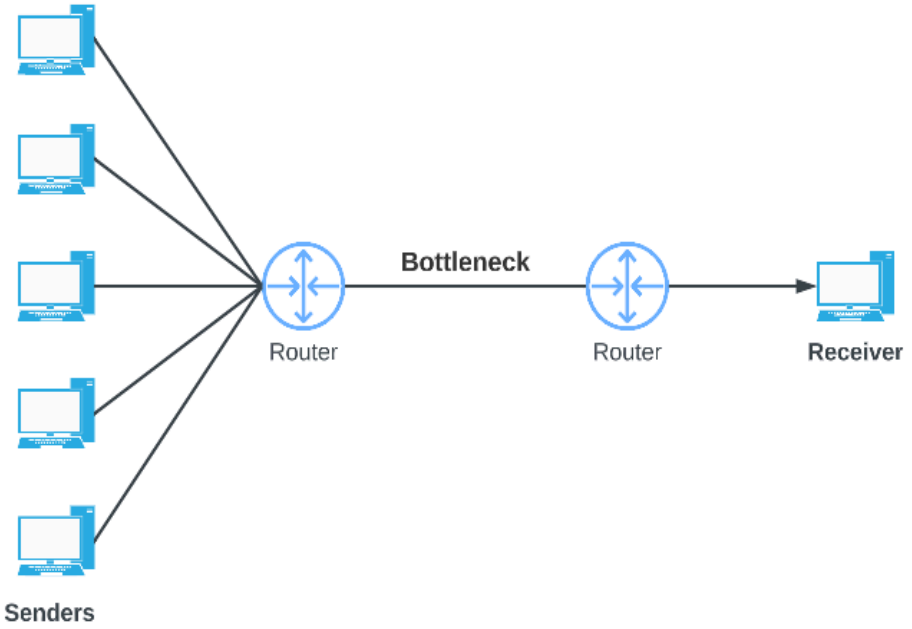

Figure 4.1: Dumbbell Topology with five senders

As per Figure 4.1, a dumbbell topology with 5 senders who send data packets to a receiver via two routers in between is created using NS3 for training the agent to learn the environment.

4.2 RESULTS

After training the DL and PPO agents,the performance of the models on different test scenarios are evaluated. Some sample scenarios on which the the models were tested is given below,

SCENARIO 1:

Table 4.1: Testing scenario network parameters

Number of leaves	**5**
Bottleneck bandwidth	**20 Mbps**
Access bandwidth	**50 Mbps**
Bottleneck Delay	**0.01 ms**
Access Delay	**20 ms**

Deep Q Learning Agent

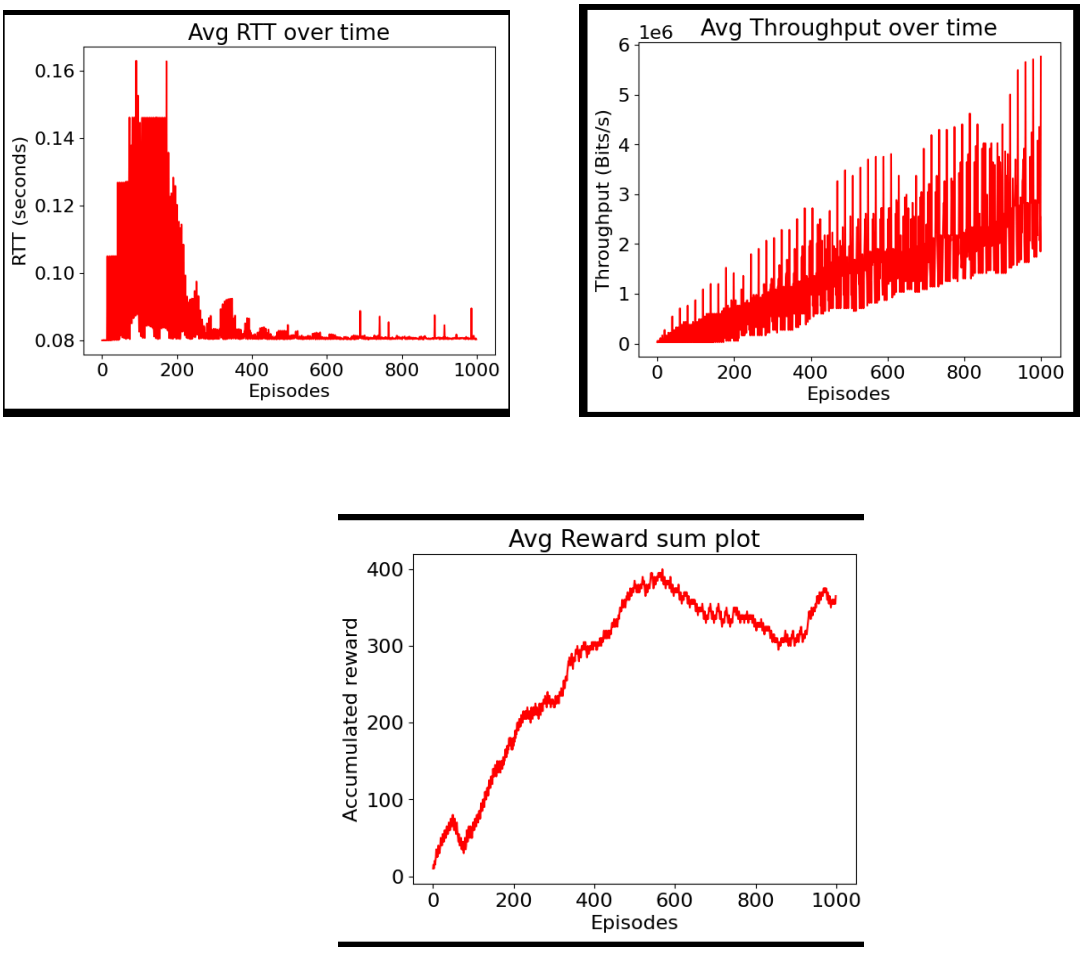

Figure 4.2: Trained DQN model Average RTT(top left), Throughput(top right) and Reward(down)

As per Figure 4.2, after 1000 episodes the agent achieved a minimum RTT of 0.08 seconds. The minimum RTT of the scenario as in Table 4.1 is calculated as (access delay * 4) =20*4=80 ms or 80s. 300 episodes were needed to converge to the minimum RTT. The throughput shows an increasing trend over the course of time. The accumulated reward is also a positive function and stays in the range [300,400] after 600 episodes.

PPO Agent

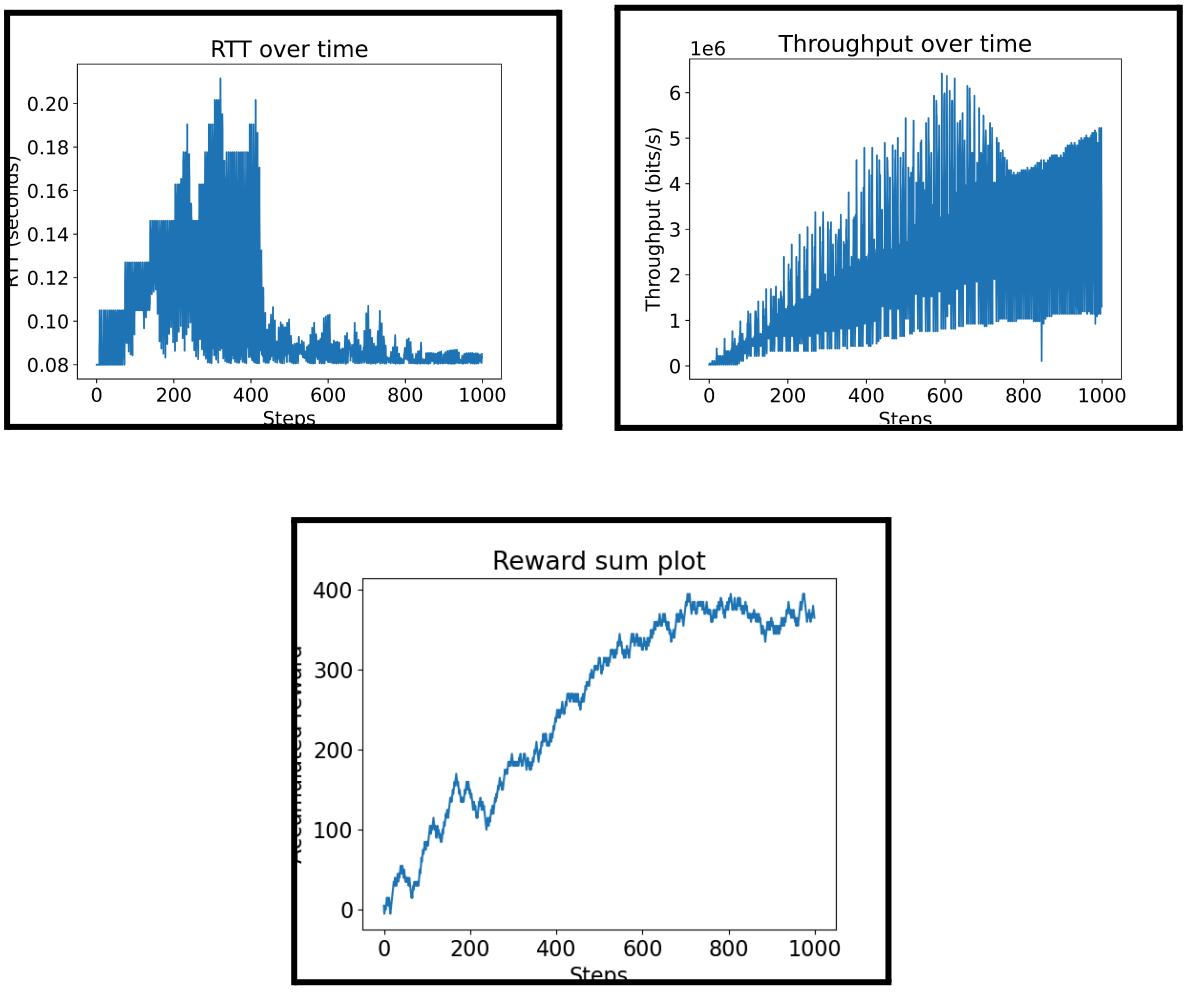

Figure 4.3: Trained PPO model Average RTT(top left), Throughput(top right) and Reward(down)

As per Figure 4.3, after 1000 episodes, the agent achieved a minimum RTT of 0.08 seconds, but the time of convergence taken by the PPO agent is longer compared to the DQN agent. The PPO agent takes nearly 500 episodes to converge to the minimum RTT. The throughput graph after 700 episodes tries to stabilise in the range [1,5] Mbps. The accumulated reward function is an increasing function.

4.3 TEST CASES

A collection of test cases was used to ensure the robustness of the proposed system and to ensure its proper working. A summary of the test cases used is tabulated in Table 4.2

Table 4.2: Test cases used

TC_ID	OBJECTIVE	INPUT	OUTPUT
TC_01	Network environment is setup with proper topology and parameters	Required topology and configurations	Simulated ns3 environment with specified topology and network parameters, such as bandwidth, segment size, etc. as well as environment hyperparameters such as reward function.
TC_02	Connected RL agent and the environment through NS3-Gym interface	Configurations for the interface and the agent to interact with the environment	RL agent is able to interact with the environment and take actions based on the states and rewards received from the environment

TC_03	RL algorithm/model is trained with good performance	Model architecture, environment data flow and hyperparameters	Trained model with high accuracy, robustness and generalizability
TC_04	RL algorithm/model performs well under various scenarios	Variation in environment and parameters	Model performs well under varying conditions, such as inconsistent network conditions, frequent congestion conditions, etc.
TC_05	Adversarial attack of high performance	Attack algorithm and RL CC-network setup	Attack degrades the performance of the RL CC network by a considerable amount.

4.3.1. Test Case Results

TC_01

Input: Creation of dumbbell topology with 5 senders.

Output:

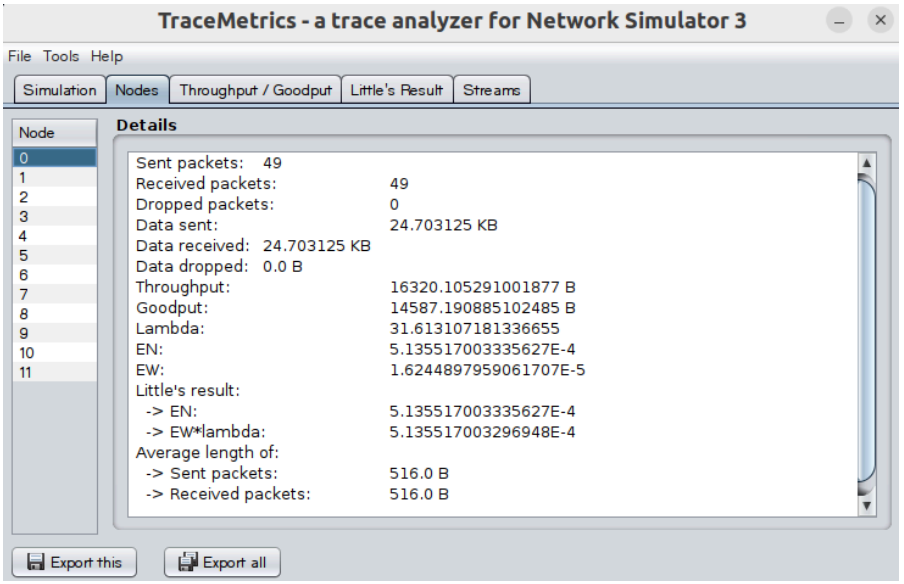

Figure 4.4: Tracemetrics analyzer

As per Figure 4.4, the number of packets sent, received, throughput, goodput, etc. of the nodes present in the topology are displayed by the tracemetrics tool by the end of the simulation.

TC_02

Input: Configuration of ns3-gym interface to enable communication between agent and simulator

Output:

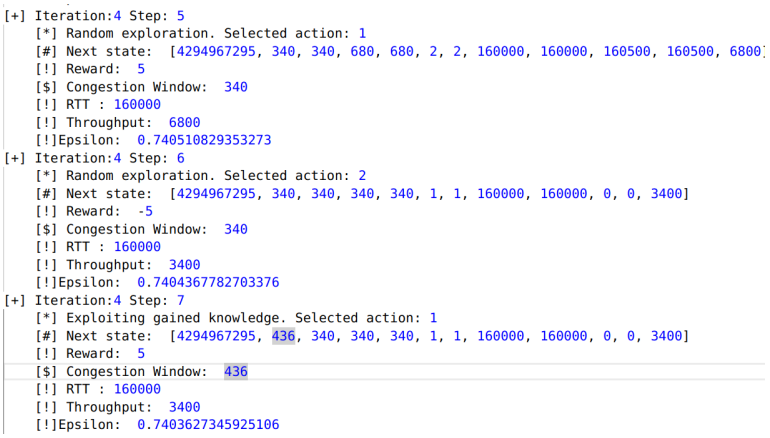

Figure 4.5: Log file of actions made by agent

As per Figure 4.5, the action chosen by the agent, next state information, reward received, RTT and throughput of the network are logged on to a file after every step of an iteration.

TC_03

Input: Building Deep Q Learning and Proximal Policy Optimization based RL agents.

Output: Reaching optimal RTT and bandwidth corresponding to the topology characteristics.

TC_04

Input: Changing the bottleneck bandwidth and delay in the topology.

Output: Graphs of RTT and throughput.

TC_05

Input: Sub-optimal action generation using adversarial attack.

Output: Increase in RTT during the training process.

4.4 PERFORMANCE METRICS

The metrics for evaluation are throughput, round trip time and reward.

1) **Throughput**: It refers to the rate of communication delivery over a communication channel, and is given by Eqn (4.1)

$$\text{Throughput} = \frac{Number\ of\ Acked\ segments * Segment\ size}{Total\ time\ elapsed} \quad (4.1)$$

2) **Round Trip Time(RTT)**: It is the amount of time taken for a message to be sent plus the amount of time taken for the acknowledgment of a message to be received, and is given by Eqn (4.2)

$$RTT = 2 * (2 * Access\ Delay + Bottleneck\ Delay) \quad (4.2)$$

3) **Reward:**

$$Reward = \log(t/B) - 0.01.\log(d) \quad (4.3)$$

In Eqn (4.3), t is the throughput, B is the bandwidth, and d is the delay.

4.5 COMPARATIVE ANALYSIS

Comparative analysis is done between the trained DQN and PPO agents with existing CC algorithms. The Round Trip Time (RTT) is used as a metric to distinguish the performance of the algorithms.

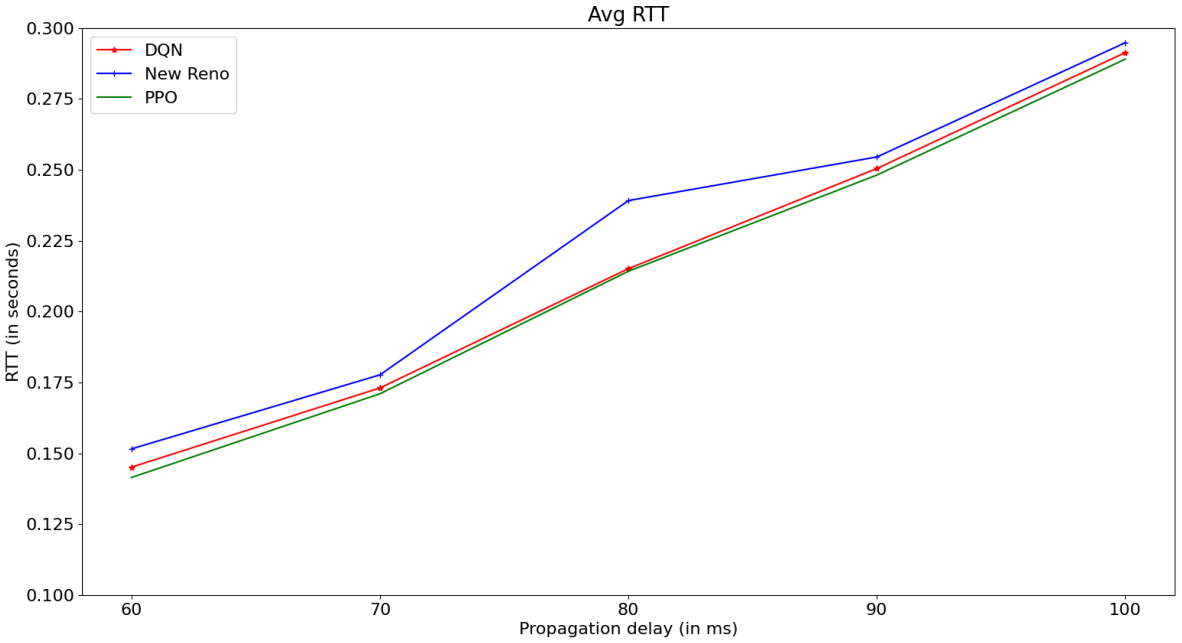

Figure 4.6: Average RTT for increased delay

As per Figure 4.6, the NewReno algorithm has a higher RTT compared to DQN and PPO agents. This is because the rule-based congestion control scheme of NewReno worsens the RTT when the environment changes dynamically. The PPO agent achieves less RTT compared to DQN. This is due to the actor-critic model of PPO, where the action generation and reward calculation are handled by two different neural networks, which increases the accuracy of taking optimal actions.

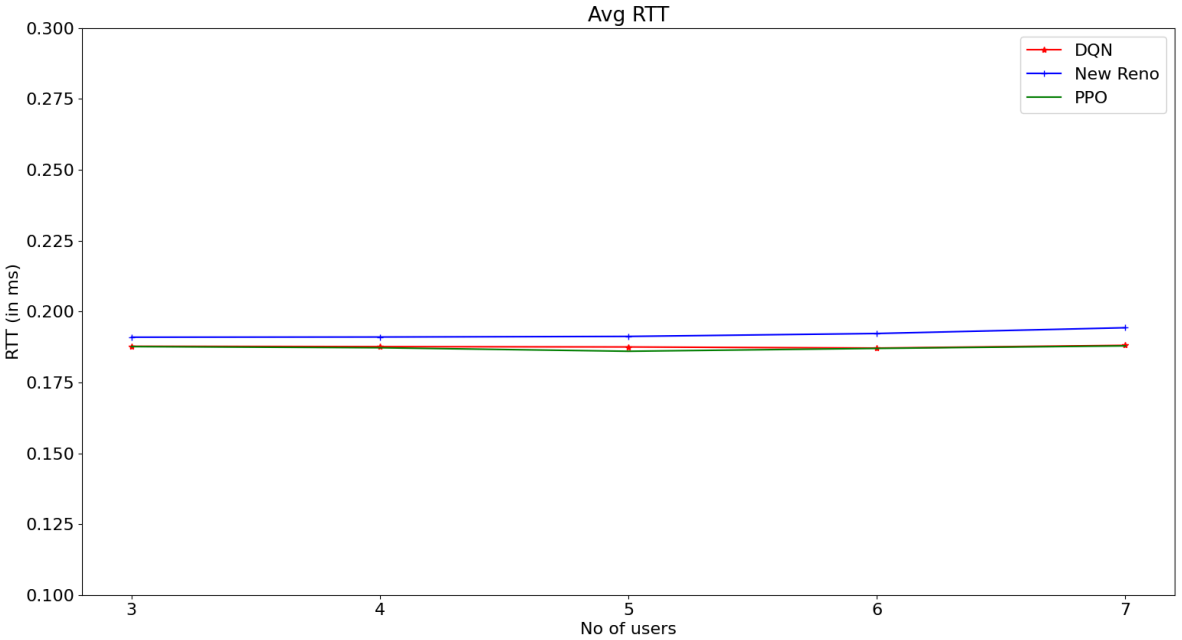

Figure 4.7: Average RTT for increased users

As per Figure 4.7, in the case of NewReno, the average RTT is slightly higher than those obtained by DQN and PPO as the number of users increases.

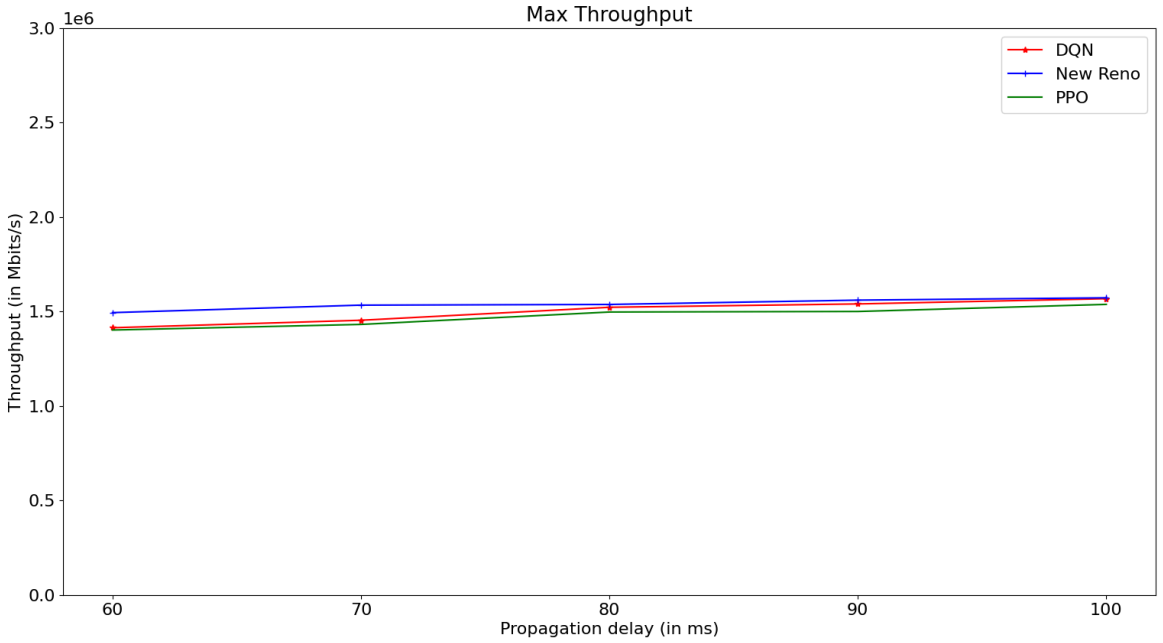

Figure 4.8: Maximum Throughput for increased delay

As per Figure 4.8, the NewReno algorithm achieves better throughput compared to the RL agents till 80ms. Then, the DQN and PPO agents try to match the

maximum throughput obtained by NewReno. The DQN performs comparatively better than the PPO in terms of achieving the maximum throughput under varied conditions.

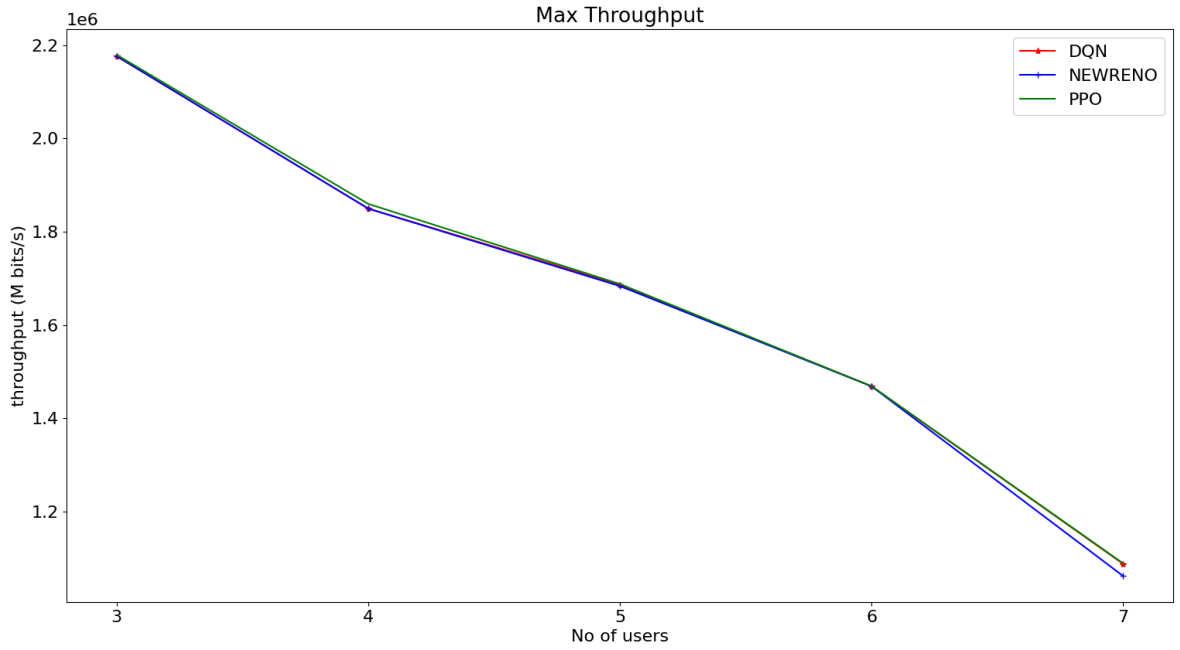

Figure 4.9: Maximum Throughput for increased users

As per Figure 4.9, maximum throughput for DQN, PPO and NewReno decreases as the number of users increase because the bandwidth of the network will be evenly divided across all the users.

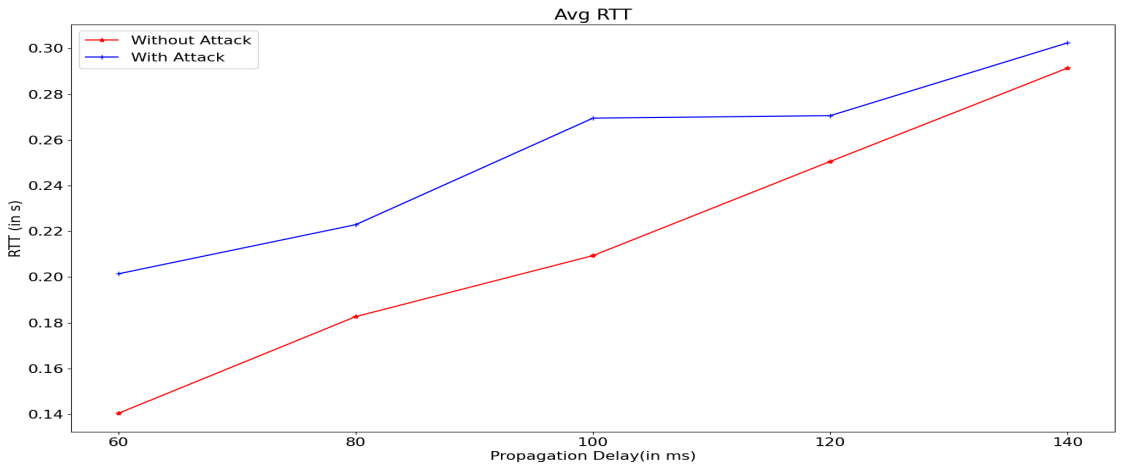

Figure 4.10: Plots of RTT before and after attack

As per Figure 4.10, the rate of increase after an attack has taken place, is more than the rate of increase without performing any attack. This is because as states are perturbed during an attack,the agent takes sub-optimal actions at each timestep, which increases the delay of the entire network.

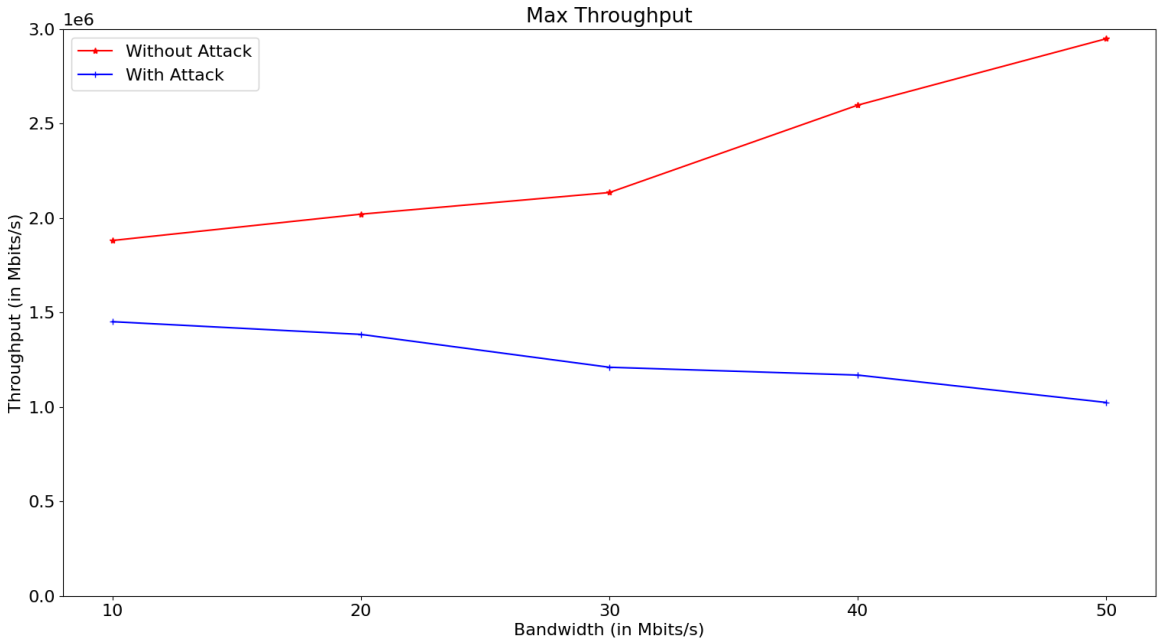

Figure 4.11: Plots of throughput before and after attack

As per Figure 4.11, the rate of decrease in throughput after an attack has taken place is greater than the rate of decrease without performing any attack.

CHAPTER 5
CONCLUSION AND FUTURE WORK

5.1. CONCLUSION

Reinforcement learning based congestion control scheme adaptively adjusts to different network conditions and operates at the optimal congestion window to achieve minimum delay and maximum throughput. It overcomes the weakness of rule-based congestion control schemes which are dependent on network conditions to perform well.

5.2. FUTURE WORK

Dumbbell topology was used to train and test the models. Evaluation of the models on other topologies like star, hub, and mesh topologies can be considered to improve the learning capability of RL-based congestion models. Testing the trained agents on real world emulated networks can be considered a viable direction to prove the effectiveness of RL in congestion control.

REFERENCES

[1] C. Berner et al. (2019), “Dota 2 with large scale deep reinforcement learning”, arXiv:1912.06680.

[2] C. Tessler, Y.Shpigelman, G.Dalal, Amit Mandelbaum, Doron Haritan Kazakov, Benjamin Fuhrer, Gal Chechik, and Shie Mannor (2021), “Reinforcement Learning for Datacenter Congestion Control” SIGMETRICS Perform. Eval, pp. 43–46.

[3] H. Jiang et al. (2020), “When Machine Learning Meets Congestion Control: A Survey and Comparison” arXiv.

[4] N. Jay, N. H. Rotman, B. Godfrey, M. Schapira, and A. Tamar (2019), “A Deep Reinforcement Learning Perspective on Internet Congestion Control”, in Proceedings of the 36th International Conference on Machine Learning, ICML, vol. 97, pp. 3050–3059.

[5] K. Han, J. Y. Lee and B. C. Kim (2019), "Machine-Learning based Loss Discrimination Algorithm for Wireless TCP Congestion Control," 2019 International Conference on Electronics, Information, and Communication (ICEIC), pp. 1-2.

[6] M. Bachl, T. Zseby, and J. Fabini (2019), “Rax: Deep reinforcement learning for congestion control,” in Proc. IEEE Int. Conf. Commun. (ICC), pp. 1–6.

[7] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson (2017), “BBR: Congestion-based congestion control,” Commun. ACM, vol. 60, no. 2, pp. 58–66.

[8] N. Li, Z. Deng, Q. Zhu and Q. Du (2019), "AdaBoost-TCP: A Machine Learning-Based Congestion Control Method for Satellite Networks," 2019 IEEE 19th International Conference on Communication Technology (ICCT), pp. 1126-1129.

[9] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami (2016), “The limitations of deep learning in adversarial settings,” in Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P),

pp. 372–387.

[10] P. Gawlowicz and A. Zubow (2019), "ns-3 meets OpenAI Gym: The Playground for Machine Learning in Networking Research", MSWiM.

[11] S. Emara, F. Wang, B. Li and T. Zeyl (2022), "Pareto: Fair Congestion Control With Online Reinforcement Learning," in IEEE Transactions on Network Science and Engineering, vol. 9, no. 5, pp. 3731-3748.

[12] T. Chen, J. Liu, Y. Xiang, W. Niu, E. Tong, and Z. Han (2019), “Adversarial attack and defense in reinforcement learning-from AI security view”, Cybersecurity, vol. 2, 12.

[13] V. Behzadan and A. Munir (2017), “Vulnerability of Deep Reinforcement Learning to Policy Induction Attacks”, arXiv.

[14] W. Li, F. Zhou, K. R. Chowdhury, and W. Meleis (2019), “QTCP: Adaptive congestion control with reinforcement learning,” IEEE Trans. Netw. Sci. Eng., vol. 6, no. 3, pp. 445–458.

[15] Z. Yang, J. Cao, Z. Liu, X. Zhang, K. Sun and Q. Li (2022), "Good Learning, Bad Performance: A Novel Attack Against RL-Based Congestion Control Systems," in IEEE Transactions on Information Forensics and Security, vol. 17, pp. 1069-1082.