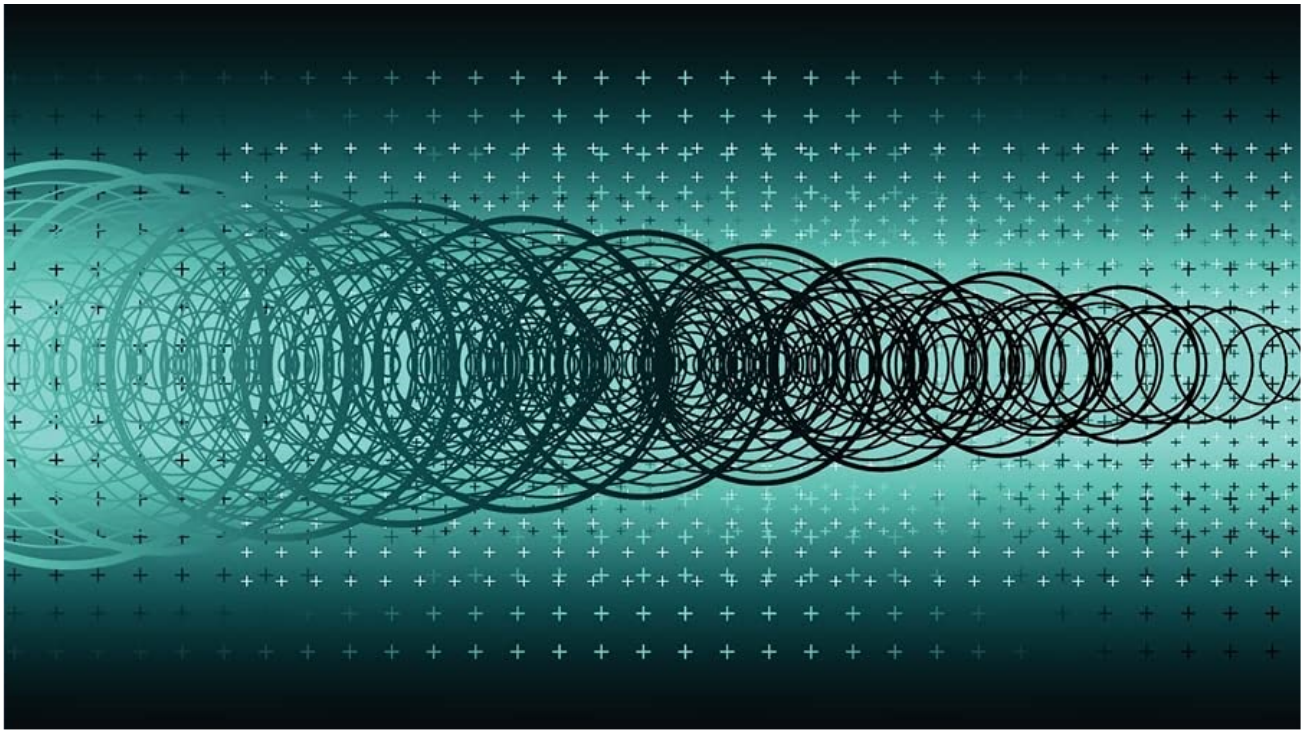
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Improving the Quality of Experience in Real-Time Communication Systems through Data-Driven Bandwidth Estimation with Deep Reinforcement Learning

Master's thesis in Computer Science and Engineering

Wen Xu

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Improving the Quality of Experience in Real-Time Communication Systems through Data-Driven Bandwidth Estimation with Deep Reinforcement Learning

Wen Xu

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Improving the Quality of Experience in Real-Time Communication Systems through Data-Driven Bandwidth Estimation with Deep Reinforcement Learning

Supervisor: Nikita Smirnov, Department of Computer Science, Christian-Albrecht University of Kiel
Advisor: Dr. Sven Tomforde, Department of Computer Science, Christian-Albrecht University of Kiel
Examiner: Dr. Krasimir Angelov, Department of Computer Science and Engineering, Chalmers University of Technology

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of network bandwidth. Source:[1].

Typeset in LaTeX
Gothenburg, Sweden 2024

Improving the Quality of Experience in Real-Time Communication Systems through Data-Driven Bandwidth Estimation with Deep Reinforcement Learning

Wen Xu
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Real-Time Communication (RTC) systems have become increasingly popular, with accurate bandwidth estimation being a critical factor in ensuring Quality of Experience (QoE) for end users. Traditional probe-based and model-based methods for bandwidth estimation have limitations, such as introducing additional overhead or relying on assumptions that may not hold in dynamic network conditions. Data-driven approaches, particularly those using machine learning techniques, have shown promise but may require substantial amounts of labeled data and struggle to adapt to changing network conditions. In this thesis, we propose an offline deep reinforcement learning (DRL) approach for bandwidth estimation in RTC applications. Our method leverages historical network data to train an agent that learns an optimal bandwidth estimation policy without the need for explicit probing or labeled data. This approach expects to offer improved adaptability to dynamic network conditions, reduced overhead, and enhanced accuracy compared to traditional and data-driven methods. We evaluate the performance of our proposed method across various network scenarios. The results reveal valuable insights and highlight the potential of offline DRL for achieving reliable bandwidth estimation in RTC applications. To accommodate reproducibility, we have made our source code publicly available[1].

Keywords: telecommunication, network traffic, artificial neural network, deep reinforcement learning, congestion control, real-time communication

[1] `https://github.com/wenxuhaskell/bwe.git`

Acknowledgements

I wish to express our deepest gratitude to my supervisor Nikita Smirnov and my advisor Prof. Sven Tomforde at University of Kiel for providing me with this thesis project. In particular I want to thank Nikita Smirnov for your incredible support and many insightful technical and non-technical discussions during my thesis work. I would like to thank my examiner Prof. Krasimir Angelov for being a part of the thesis assessment and providing me fruitful feedback throughout the project. I also want to pay special regards to the teachers and professors that i had the fortune of meeting during my studies. Lastly I want to say thanks to my beloved family. Without your support, I would not be able to complete my journey.

Wen Xu
Gothenburg, Sweden
2024-08-28

Contents

List of Figures

List of Tables

List of Abbreviations

AD Action Difference

AI Artificial Intelligence

ANN Artificial Neural Network

BCQ Batch-Constrained Q-learning

CQL Conservative Q-learning

DDPG Deep Deterministic Policy Gradient

DQN Deep Q-Learning Network

DRL Deep Reinforcement Learning

DT Decision Transformer

GCC Google Congestion Control

GRUs Gated Recurrent Units

LSTM Long Short-Term Memory

MDP Markov decision process

MI Monitor Interval

MLP Multi-Layer Perceptron

MSE Mean Squared Error

NN Neural Network

PPO Proximal Policy Optimization

QoE Quality of Experience

RL Reinforcement Learning

RNNs Recurrent Neural Networks

RTC Real-Time Communication

RTT Round Trip Time

SAC Soft Actor-Critic

SGD Stochastic Gradient Descent

TD Termporal Difference

TD3 Twin Delayed DDPG

TD3+BC Twin Delayed DDPG with Behavioural Cloning

UKF Unscented Kalman Filter

WebRTC Web Real Time Communication

1

Introduction

This chapter introduces the background of bandwidth estimation, briefly describes the current state of the field, and highlights the remaining challenges that motivate the proposed approach. It defines the scope of this thesis, formulates key research questions, and outlines the structure of the thesis.

1.1 Background

Real-Time Communication (RTC) applications have gained significant popularity in recent years, driven by the increasing demand for instantaneous information exchange and the proliferation of high-speed Internet connectivity. RTC encompasses a wide range of services, including Voice over IP, video conferencing, online gaming, and remote desktop applications, which require low latency and high-quality data transmission to ensure seamless user experience [2]. One critical aspect that directly impacts the performance of RTC applications is accurate bandwidth estimation.

Bandwidth estimation is the process of determining the available data transfer capacity between two communicating entities in a network [3]. Accurate bandwidth estimation is crucial for several reasons. Firstly, it enables adaptive bitrate streaming, where the quality of the transmitted audio or video is adjusted in real-time based on the available bandwidth, ensuring smooth playback without excessive buffering or quality degradation. Secondly, accurate bandwidth estimation helps in making informed decisions regarding resource allocation, packet scheduling, and congestion control, thereby enhancing the overall Quality of Experience (QoE) for the end-users [4].

The accuracy of bandwidth estimation significantly impacts the performance of RTC applications. Overestimation of bandwidth can lead to network congestion, packet loss, and increased latency. The user experience suffers from frequent resolution changes, video stalls, garbled speech, etc. [4]. However, underestimation of bandwidth can result in sub-optimal resource utilization, leading to lower transmission rates and poor application performance. Therefore, developing accurate and reliable bandwidth estimation techniques is of paramount importance for RTC applications.

Traditional approaches for bandwidth estimation can be classified into two categories: probe-based and model-based methods. Probe-based methods involve sending additional data packets (probes) to estimate the available bandwidth by analyz-

ing the packet transmission and reception characteristics, such as delay, loss, and throughput [5]. While probe-based methods can provide reasonably accurate estimates, they may introduce additional overhead and perturbations in the network, which can negatively impact the performance of RTC applications. Model-based methods, on the other hand, rely on mathematical models of network behavior to estimate bandwidth without sending explicit probes [6]. Although model-based methods do not introduce additional traffic, their performance heavily hinges on the assumptions made about network conditions and the underlying heterogeneous network architecture, which may not always be true in complex dynamic real-world scenarios.

In recent years, data-driven approaches, particularly those that utilize machine learning techniques, have emerged as promising alternatives to estimate bandwidth [7]–[11]. These approaches utilize historical network data to train predictive models that can estimate bandwidth based on various networking features, such as packet inter-arrival time, packet size, and queue length. Although data-driven approaches have shown their strength compared to traditional methods, they often require large amounts of labeled data for training and may struggle to adapt to changing network conditions.

Deep Reinforcement Learning (DRL) has recently gained attention as a potential solution for bandwidth estimation in RTC applications. DRL is a type of machine learning algorithm that combines Artificial Neural Networks (ANNs) with Reinforcement Learning (RL), enabling agents to learn optimal policies by interacting with an environment and maximizing a cumulative reward. Despite its appeal, DRL also has drawbacks to overcome. These disadvantages include the need for continuous online interactions with the environment, which can be time-consuming. Furthermore, the feedback loop between policy learning and data collection in DRL can lead to instability and sub-optimal policies. Lastly, DRL may not be suitable for applications where real-world interactions are sensitive, costly, or safety-critical, as exploration during learning can result in undesirable consequences.

Considering these disadvantages, Offline DRL [12], a machine learning technique that trains agents using static, pre-collected datasets without requiring online environment interactions, becomes a more attractive option for applications like bandwidth estimation in RTC, where data efficiency and stability are essential. In the context of bandwidth estimation, an offline DRL agent can be trained using historical network data to learn the optimal bandwidth estimation policy without the need for explicit probing or labeled data. This approach offers several advantages over traditional and data-driven methods, such as improved adaptability to dynamic network conditions, reduced overhead of probing packets, and potentially enhanced accuracy. However, offline DRL still faces certain challenges, such as the risk for overfitting training data and the difficulty in dealing with out-of-distribution data encountered from real-world deployment [13].

In this thesis, we propose an offline DRL-based approach for bandwidth estimation in RTC applications. Our approach aims to address the limitations of existing methods by leveraging the power of DRL to learn an optimal bandwidth estimation policy

from historical network data. We will evaluate the performance of our proposed approach in various network scenarios and compare it with the result from using traditional approach. The findings of this research have the potential to contribute to the development of more reliable and adaptive bandwidth estimation techniques for RTC applications.

1.2 Objectives

The overarching goal of this thesis is to explore the application of offline DRL for accurately estimating network bandwidth, with the ultimate aim of improving the QoE in video streaming. Specifically, we will experiment with various offline DRL algorithms for bandwidth estimation in RTC systems. The research will involve studying vectorized states to identify effective feature representations. We will implement and evaluate several representative DRL algorithms using various neural network architectures, fine-tuned with different optimization strategies. Additionally, we will assess how the design of reward functions impacts the performance of DRL models.

To concretize these research objectives, we formulate several key research questions:

1. *How do the choices of state representation impact the performance of the DRL model?*
2. *How do the choices of reward function impact the performance of the DRL model?*
3. *How do the choices of learning algorithms impact the performance of the DRL model?*
4. *How do the choices of neural network architectures impact the performance of the DRL model?*

These questions will guide our exploration and evaluation of different methodologies within the scope of this thesis, helping to identify effective strategies for bandwidth estimation in RTC environments.

1.3 Thesis Outline

This thesis is organized into the following chapters:

- **Chapter 1** introduces the problem statement and outlines the research objectives of the thesis.
- **Chapter 2** provides the theoretical background for offline deep reinforcement learning (DRL), covering the fundamental concepts and techniques.
- **Chapter 3** details the formulation of the problem and the methods selected for implementation.

- **Chapter 4** describes the experiments conducted during the project and discusses the results.
- **Chapter 5** summarizes the findings, recommends future directions and concludes the thesis.

2

Theory

This chapter provides a foundation for understanding offline DRL, the approach utilized in this project. We will explore the core concepts that build upon each other, starting with fundamental elements like Markov Decision Processes (MDPs) and progressing towards the specific techniques employed in offline DRL [14].

The chapter covers following topics:

(i) *Reinforcement Learning*

(ii) *Artificial Neural Network and Deep Learning*

(iii) *Deep Reinforcement Learning*

(iv) *Offline Deep Reinforcement Learning*

2.1 Reinforcement Learning

Reinforcement Learning is a branch of machine learning that focuses on solving sequential decision-making problems by training an adaptable agent interacting with an environment on a trial-and-error basis. In RL, an agent learns to interact with its environment by performing actions and observing the consequences of those actions, with the goal of optimizing a long-term objective. Reinforcement learning as an autonomous and self-learning system has been particularly successful in robotic applications and tactic gameplay. Figure 2.1 shows the basic principle of reinforcement learning. This section provides an introduction to the key principles of RL, including MDP, policy and value functions, and popular algorithms such as Q-learning, and policy gradient methods.

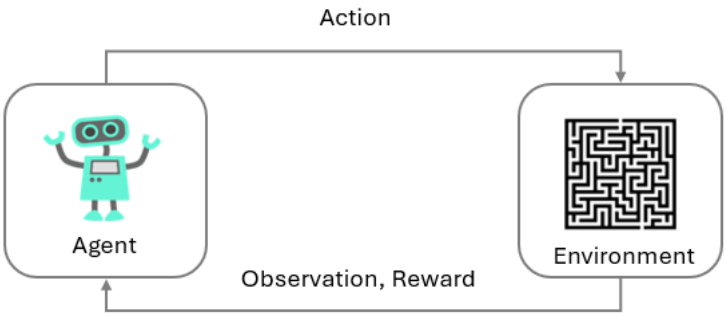


Figure 2.1: An agent observes the environment, takes an action and receives a reward from the environment

2.1.1 Markov Decision Process

The MDP is a mathematical framework used in reinforcement learning that represents an environment in which all aspects are observable.

An MDP is defined by a tuple (S, A, P, R, γ) , where S is a finite set of states s_t, A is a finite set of actions a_t, P is a finite set of state transition probabilities, R is the reward function, $\gamma[0, 1]$ is the discount factor and t is a non-negative integer representing a given time step.

State space $S = s_1, ..., s_n$ is a finite set containing all n possible states. within this definition, the state s_n refers to a specific configuration or observation of the MDP environment. Essentially, the state serves as a complete snapshot, providing all the necessary information for the agent to make informed decisions and anticipate future outcomes.

Action space $A = a_1, ..., a_k$ is a finite set of k possible actions made by an agent operating within the environment. The agent is tasked with selecting an action from a set of available options, with the intention of influencing the transition from one state to another and optimizing the overall reward. This process leads to the transition or state transition which occurs when an agent takes an action, prompting the environment to shift from one state to another. The likelihood of transitioning to a particular state is typically defined by the transition function P.

Transition probabilities $P(s'|s, a)$ describes the probability p for the environment to transit to s' after the agent takes action a at state s at a given time step t

$$p(s'|s, a) = P(S_{t+1} = s'|S_t = s, A_t = a) \tag{2.1}$$

Reward function $R : S \times S \to \mathbb{R}$ outputs a numerical value called reward r_t that an agent receives upon performing a at state s and transits to a new state s' at a given time step t. The reward serves as valuable feedback, signaling the effectiveness of the chosen action. The primary objective for the agent is to maximize the cumulative reward over time.

Discount factor γ is a real value ranging between 0 and 1 that determines the significance of future rewards in comparison to immediate rewards. A higher discount factor puts a greater emphasis on long-term rewards, while a lower factor suggests a preference for short-term gains. This component is essential in striking a balance between the agent's focus on immediate and future rewards when making decisions within the MDP framework.

An Markov Decision Process is founded on the Markov property, which stipulates that the future is only dependent on the present and not on the past. Essentially, this means that once the current state encapsulates all necessary information from past events, knowledge of the history becomes redundant, as the current state provides a sufficient statistical representation of future outcomes. The majority of reinforcement learning problems can be structured within the context of a Markov Decision Process. The following equation describes the Markovian property.

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...) = P(s_{t+1}|s_t, a_t) \tag{2.2}$$

2.1.2 Policy Function

A policy function $\pi : S \times A \to [0, 1]$, also referred to as policy, defines the strategy of a Reinforcement Learning (RL) agent. It is a probability distribution function that maps states to distributions over actions, specifying the behavior of the agent in any given state. The agent selects actions to take by sampling from the policy.

$$A_t \sim \pi(\cdot|S_t) \tag{2.3}$$

In essence, solving a task using reinforcement learning is to find a optimal policy for the RL agent to receive equal or more expected accumulative rewards compared to any other policy.

A policy can be either deterministic or non-deterministic. A deterministic policy always choose a fixed action at each state while a non-deterministic policy samples an action from the distribution of actions at each state according to the policy.

2.1.3 Value Function

A value function plays a central role in reinforcement learning, representing the expected long-term return of an agent following a specific policy. Within a given state, a value function measures the goodness of current policy π. A value function typically employs the discounted factors to compute the weighted sum of future rewards and is given as

$$G_t = r_{t+1} + \gamma r_{t+2} + ... = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}, \quad \gamma \in [0, 1] \tag{2.4}$$

where r_t is the reward at given time step t.

There are two main types of value functions: state-value functions and action-value functions. The state-value function, $V_\pi(s)$ as given below,

$$V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] \tag{2.5}$$

represents the expected value of the cumulative discounted reward with respect to the current state s given that the agent follows the policy π. With an optimal policy π^*, the optimal state-value function becomes

$$V^*(s) = \max_\pi V_\pi(s), \quad \forall s \in S \tag{2.6}$$

As another form of value function, the action-value function

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] \tag{2.7}$$

defines the expected cumulative discounted reward of taking action a according to the policy π at current state s. This is also called Q-function with the optimal Q value as the maximum over all policies defined as below

$$Q^*(s, a) = \max_\pi Q_\pi(s, a) \tag{2.8}$$

2.1.4 Temporal Difference Learning

Temporal Difference (TD) learning represents a class of reinforcement learning algorithms that blend aspects of Monte Carlo methods and dynamic programming to estimate value functions [15]. As depicted in Figure 2.2, TD learning algorithms update value function estimates by leveraging the difference between the predicted and observed rewards, known as the TD error as the definition below.

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2.9}$$

This approach enables TD learning algorithms to learn from incomplete episodes and bootstrap estimates from subsequent time steps, making them more efficient and adaptable.

A basic version of TD learning is called one-step TD, or TD(0) as defined in the pseudo-code in figure 2.2.

2.1.5 Q-Learning

Coined by Walkins [16], Q-learning is a widely used TD learning algorithm employed to estimate action-value functions. Q-learning is an off-policy algorithm and has the ability to learn the optimal policy while following a separate, exploratory policy. The psudo-code in 2.3 shows that Q-learning iteratively refines the action-value function

Algorithm 1 Tabular TD(0) for estimating v_π

Input: Policy π to be evaluated **Parameters:** Learning rate $\alpha \in (0, 1]$

for each episode: **do**
  Initialize S
  while S is not terminal: **do**
    Take action A given by $\pi(a|S)$
    Observe R, S'
    Update $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
    $S \leftarrow S'$
  end while
end for

Figure 2.2: Psudo-code for TD learning, adapted from [15]

based on the maximum estimated action-value for the subsequent state, converging to the optimal policy under certain conditions. This makes Q-learning a versatile and powerful tool for solving various reinforcement learning problems.

Algorithm 2 Q-learning (off-policy TD control) for estimating $\pi \approx \pi_\star$

Parameters: Learning rate $\alpha \in (0, 1]$, small $\epsilon > 0$

Initialize $Q(s, a)$ for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ and $Q(terminal) = 0$
for each episode: **do**
  Initialize S
  while S is not terminal: **do**
    Take action A using a policy derived from Q (ϵ-greedy)
    Observe R, S'
    Update $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
    $S \leftarrow S'$
  end while
end for

Figure 2.3: Psudo-code for Q-Learning, adapted from [15]

2.1.6 Policy Gradient Methods

Policy gradient methods [17] are a class of reinforcement learning algorithms that directly optimize the policy function, π, without explicitly estimating value functions. These methods parameterize the policy using a function approximator, such as a neural network, and update the parameters using gradient ascent to maximize the expected cumulative reward. Policy gradient methods can handle continuous action spaces and are often more sample-efficient than value-based methods. However, they

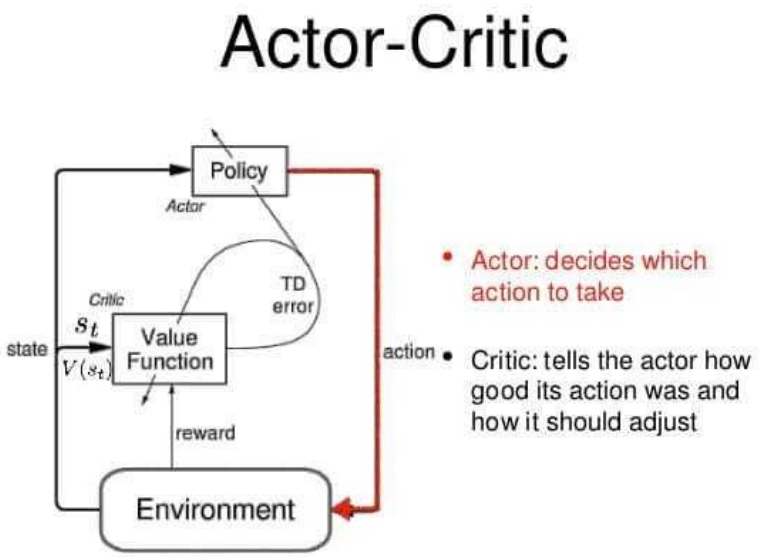


Figure 2.4: Actor-Critic. Source: [15]

can suffer from high variance and may require careful tuning of the learning rate and other hyperparameters.

2.1.7 Actor-Critic

Actor-Critic [15] is a class of methods in reinforcement learning that combines the strengths of value-based and policy-based methods. As shown in figure 2.4, an actor-critic architecture consists of two primary components: the actor and the critic. The actor is responsible for selecting actions based on the current state of the environment, effectively learning a policy that maps states to actions. The critic, on the other hand, evaluates the actions taken by the actor by estimating the value function, which can be either the state-value function $V_\pi(s)$ or the action-value function $Q_\pi(s, a)$. This dual structure allows the actor-critic architecture to leverage the benefits of both components, facilitating more efficient learning and better performance.

One of the key advantages of the actor-critic approach is its ability to handle continuous and high-dimensional action spaces. Unlike purely value-based methods, which typically require discretization of action spaces, the actor can directly output continuous actions, making it well-suited for complex control tasks. The critic provides feedback by estimating the expected return for the actions taken, guiding the actor to improve its policy. This feedback loop helps the actor to converge toward an optimal policy, while the critic refines its value estimates. Additionally, the actor-critic architecture can use techniques like bootstrapping to update the value function, leading to more stable and sample-efficient learning. This architecture forms the basis for many popular reinforcement learning algorithms and is widely

used in applications ranging from game playing to robotic control.

2.1.8 On-policy and Off-policy

Reinforcement learning algorithms can be broadly classified into two categories: on-policy and off-policy, each representing distinct strategies for learning.

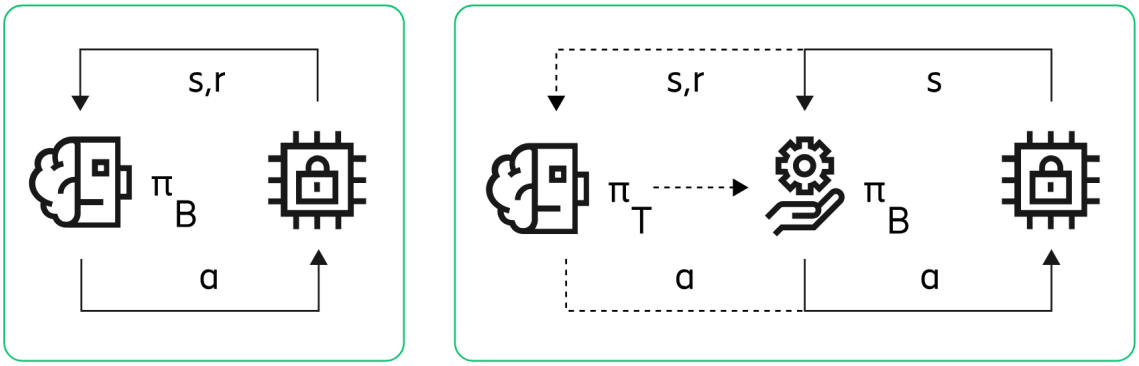


Figure 2.5: On-policy RL iteratively refines a single behavior policy π_B that also generates control actions within the environment. Off-policy RL maintains a behavior policy π_B, and also trains a target policy π_T. Source: [18]

As illustrated in figure 2.5, on-policy algorithms, such as State-Action-Reward-State-Action (SARSA) [15], focus on learning a behavior policy that is actively used during interactions with the environment. These algorithms employ the same policy for both exploration (trying out new actions) and exploitation (using known good actions). They update this policy based on the data gathered from these interactions. This approach ensures that the learned policy directly reflects the behavior of the agent, maintaining consistency between exploration and policy refinement.

In contrast, off-policy algorithms like Q-learning aim to learn a target policy while using a separate behavior policy for exploration. This allows them to learn from data generated by other policies or agents. As a result, off-policy algorithms tend to be more sample-efficient, as they can utilize a wider range of experiences.

This characteristic is especially beneficial in offline reinforcement learning scenarios, where the agent must learn from a fixed dataset without further interactions with the environment. By decoupling the learning process from data collection, off-policy methods offer greater flexibility and efficiency, making them suitable for a broader range of applications.

2.2 Artificial Neural Networks and Deep Learning

Before delving into Deep Reinforcement Learning (DRL), it is essential to provide a brief introduction to Artificial Neural Networks (ANNs) and deep learning, as they form the backbone of DRL algorithms.

2.2.1 Artificial Neural Networks

ANNs are computational models inspired by the structure and function of biological neurons in the human brain. As illustrated in figure 2.6, ANNs consist of interconnected nodes, or artificial neurons that mimics biological neurons, organized in layers: an input layer, one or more hidden layers, and an output layer. Each connection between nodes has an associated weight, which represents the strength of the connection, and each node has a bias term, which adjusts the node's activation threshold.

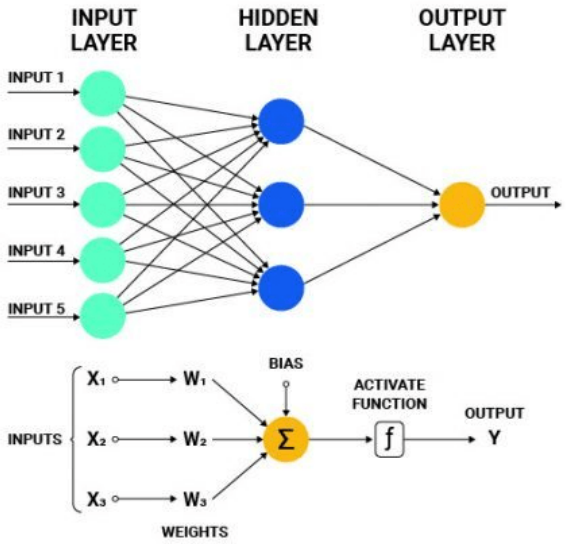


Figure 2.6: An ANN including input, hidden and output layers as well as weights and activation functions.

During the training process, an artificial neural network receives input data that is passed through multiple hidden layers. Each layer consists of nodes, where the input is transformed by applying activation functions. These transformations enable the network to model complex, non-linear relationships in the data. The transformed input progresses through the layers until it reaches the output layer, which generates the final prediction or decision.

To optimize the network's performance, the weights and biases associated with the connections between nodes are adjusted. This adjustment process aims to minimize the loss function, which quantifies the difference between the predicted outputs and the actual target values. Optimization techniques such as stochastic gradient descent (SGD) [19] are employed to iteratively update the weights and biases in the direction that reduces the loss.

A critical aspect of this optimization is backpropagation [20], a method that computes the gradient of the loss function with respect to each weight by propagating the error backward through the network. By updating the weights in accordance with these gradients, the network learns to improve its predictions over time, gradually uncovering intricate patterns and relationships in the training data.

2.2.2 Deep Learning

Deep learning is a subset of machine learning that focuses on training ANNs with multiple hidden layers, also known as deep neural networks. These networks can automatically learn hierarchical representations of data, enabling them to process and analyze complex patterns in high-dimensional input data. Deep learning has revolutionized various domains, including computer vision, natural language processing, and speech recognition, by achieving state-of-the-art performance in numerous tasks.

Various neural network architectures have been developed to address specific challenges in different domains. Some of the most popular architectures include:

- *Multilayer Perceptrons (MLPs)*: MLPs [21] are feedforward artificial neural networks with an input layer, one or more hidden layers, and an output layer. They are fully connected, meaning each node in a layer is connected to every node in the subsequent layer. MLPs are widely used in various applications, such as classification, regression, and function approximation, serving as a foundational architecture for deep learning models.
- *Convolutional Neural Networks (CNNs)*: CNNs [22] are a specialized type of neural network designed for processing data with grid-like topology, such as images. They are particularly effective in capturing spatial hierarchies and patterns through the use of convolutional layers, which apply a series of learnable filters to the input data. These filters help in detecting edges, textures, and other features at various levels of abstraction. The key components of a CNN include convolutional layers, pooling layers, and fully connected layers. Convolutional layers extract feature maps by performing convolutions, pooling layers downsample the feature maps to reduce dimensionality and computational load, and fully connected layers make final predictions based on the extracted features. This architecture allows CNNs to learn and recognize complex patterns and structures in data, making them highly effective for tasks like image classification, object detection, and segmentation.
- *Recurrent Neural Networks (RNNs)*: Recurrent Neural Networks (RNNs) [23] are a class of neural networks designed for sequential data processing, making them well-suited for tasks involving time series, natural language, and other ordered data. Unlike traditional feedforward neural networks, RNNs have a unique architecture that includes loops, allowing them to maintain a memory of previous inputs in the form of hidden states. This structure enables RNNs to capture temporal dependencies and patterns over time, as each input is influenced not only by the current data point but also by the context provided by preceding elements. However, standard RNNs can struggle with long-term dependencies due to issues like vanishing gradients. To address these challenges, more advanced variants like Long Short-Term Memory (LSTM) [24] networks and Gated Recurrent Units (GRUs) [25] have been developed, offering better memory retention and handling of long-range dependencies. RNNs have been widely used in applications such as speech recognition, language modeling, and time series prediction.

- *Transformers*: Transformers [26] are a type of neural network architecture that relies on self-attention mechanisms to process sequential data, such as natural language. Unlike RNNs, which process data sequentially, transformers do not require recurrent connections, allowing them to process input sequences in parallel and capture long-range dependencies more effectively. This architecture has become the foundation for many state-of-the-art models, including BERT [27], GPT [28], and T5 [29], transforming tasks like machine translation, text generation, and beyond.

For more details of deep leaning, please refer to the Deep Learning Book by Ian Goodfellow, et al. [23].

2.3 Deep Reinforcement Learning

DRL is a powerful approach that combines the strengths of deep learning with traditional RL techniques. In DRL, an agent interacts with an environment to learn an optimal policy that maximizes a cumulative reward signal. The agent's policy is represented by a deep neural network, which takes the current state of the environment as input and outputs the appropriate action to take. The neural network is trained using gradient-based optimization methods, such as SGD, to minimize the loss function, which reflects the difference between the predicted and actual rewards. By leveraging the representational power of deep neural networks, DRL algorithms can learn complex patterns and relationships in high-dimensional state spaces, enabling them to tackle challenging problems that were previously infeasible for traditional reinforcement learning methods.

DRL has achieved remarkable success in various domains, including game playing, robotics, and resource management, by enabling agents to learn complex behaviors from high-dimensional input data. In this section, we will discuss some popular deep reinforcement learning algorithms, such as Deep Q-Networks (DQN) [30], Deep Deterministic Policy Gradient (DDPG) [31], and Soft Actor-Critic (SAC) [32], which have significantly contributed to the advancements in the field of reinforcement learning.

2.3.1 Deep Q-Networks

DQN is a pioneering deep reinforcement learning algorithm introduced by Mnih et al. [30]. DQN extends the traditional Q-learning algorithm by using a deep neural network to approximate the action-value function, $Q(s, a)$, which represents the expected discounted return of taking action a in state s and following the optimal policy thereafter. DQN uses a technique called experience replay [33], [34] to stabilize the learning process. Experience replay stores the agent's experiences, represented as transitions (s, a, r, s'), in a buffer. During training, the DQN samples mini-batches of transitions from the buffer to update the action-value function, reducing the correlation between consecutive samples and improving the learning efficiency. Additionally, DQN employs a target network, which is a separate copy of the action-value function with frozen weights. The target network is periodically

updated with the weights of the primary action-value function, providing a more stable learning signal and further enhancing the learning stability.

2.3.2 Deep Deterministic Policy Gradient

DDPG [31] is a powerful reinforcement learning algorithm specifically designed for environments with continuous action spaces. It merges the principles of Q-learning and policy gradient methods, offering a framework that can efficiently learn control policies for complex tasks. DDPG utilizes an actor-critic architecture, where the actor network is responsible for determining the specific action to take given a particular state, while the critic network evaluates the quality of the action by estimating the associated Q-value. This combination allows DDPG to learn both the policy and the value function, making it a versatile and effective algorithm for a wide range of applications.

A notable feature of DDPG is its use of deterministic policies, which simplifies the learning process in continuous action spaces. The algorithm employs techniques such as experience replay and target networks to enhance stability and convergence during training. Experience replay allows the agent to store and reuse past experiences, breaking the temporal correlation between consecutive training samples and improving learning efficiency. Target networks help mitigate the problem of unstable Q-value estimates by providing a slowly updated version of the actor and critic networks, thereby smoothing out updates. DDPG's ability to handle high-dimensional action spaces and learn from complex environments makes it particularly suitable for applications like robotic control, autonomous driving, and other areas requiring precise and continuous action selection.

2.3.3 Soft Actor-Critic

SAC is an off-policy deep reinforcement learning algorithm introduced by Haarnoja et al. [32]. SAC is based on the maximum entropy reinforcement learning framework, which aims to learn a policy that maximizes both the expected return and the entropy of the policy. By maximizing the policy entropy, SAC encourages exploration and learns a more robust and diverse set of behaviors. SAC consists of two main components: a soft state-value function, $V(s)$, and a soft Q-value function, $Q(s, a)$. The soft state-value function represents the expected discounted return and the entropy of the policy, while the soft Q-value function represents the expected discounted return, the entropy of the policy, and the immediate reward of taking action a in state s. SAC maintains two separate Q-value function approximators and uses the minimum value of those two to mitigate the overestimation bias commonly found in Q-learning methods. SAC has been shown to achieve state-of-the-art performance on various benchmark tasks, particularly in continuous control domains.

2.4 Offline Deep Reinforcement Learning

Offline DRL is a variant of deep reinforcement learning that focuses on learning from static, pre-collected datasets without further interaction with the environment. This approach is particularly useful in scenarios where online data collection is costly, time-consuming, or risky, such as real-world robotics, autonomous driving, and healthcare applications. Offline DRL algorithms aim to address the unique challenges associated with learning from fixed datasets, such as distributional shift and overestimation of Q-values. In this section, we will introduce some popular offline deep reinforcement learning algorithms used in this thesis, including Batch-Constrained Q-learning (BCQ) [35], Conservative Q-Learning (CQL)[36], Twin Delayed DDPG with Behavioural Cloning (TD3+BC) [37], and Decision Transformer (DT) [38].

2.4.1 Batch-Constrained Q-learning

BCQ algorithm was introduced by Fujimoto et al. [35]. As the first batch deep reinforcement learning algorithm, BCQ tries to address the challenge of distributional shift in offline reinforcement learning, where the learned policy may generate out-of-distribution actions that are not well-represented in the static dataset. BCQ approaches this issue by learning a generative model of the actions in the training dataset and constraining the learned policy to generate actions that are close to the actions in the dataset.

2.4.2 Conservative Q-Learning

CQL is an offline deep reinforcement learning algorithm introduced by Kumar et al. [36]. CQL focuses on addressing the challenge of overestimating Q-values in offline reinforcement learning, which can lead to suboptimal policies. CQL tackles this by learning a conservative Q-value function that lower-bounds the true Q-value function, ensuring that the learned policy does not overestimate the returns of unseen actions.

2.4.3 Twin Delayed DDPG with Behavioural Cloning

TD3+BC was first introduced by Fujimoto et al. [37] as an extension of the Twin Delayed DDPG (TD3) algorithm [39], which addresses the challenge of overestimation bias in Q-value estimation. TD3+BC incorporates a behavior cloning term in the policy optimization objective meanwhile normalizes the state, encouraging the learned policy to mimic the actions in the dataset while maintaining the benefits of TD3.

2.4.4 Decision Transformer

DT was introduced by Chen et al. [38]. It leverages the power of transformer neural network architecture to learn a sequential decision-making policy from a static dataset. Unlike traditional reinforcement learning methods that rely on value-based

or policy-based approaches, the Decision Transformer adapts the Transformers self-attention mechanism to model the decision-making process directly. By treating the entire sequence of states, actions, and rewards as input, it generates a sequence of actions that optimize long-term rewards. This approach enables the DT to capture complex temporal dependencies and relationships within the sequence data, offering a more flexible and scalable solution to tasks that require sequential sequence modeling and prediction.

3

Methods

In this chapter, we present our methods for utilizing data-driven DRL to address the bandwidth estimation problem in RTC applications. We begin with a comprehensive literature review of existing work in this field. Drawing inspiration from state-of-the-art (SOTA) research and considering the limited feasibility of training an RL agent in real-world network conditions, we opt to employ offline DRL as our approach.

We introduce the selected dataset for offline training and provide a formal MDP formulation of the bandwidth estimation problem, which serves as the foundation for applying DRL. We discuss the motivations and considerations behind our choice of formulation. Furthermore, we describe our process for analyzing and reducing redundant features, facilitating faster training.

This chapter also covers practical aspects of our methodology, such as the choice of algorithms library and the optimization library for hyperparameter fine-tuning.

3.1 Literature Review

More recently, DRL has emerged as a compelling approach for bandwidth estimation in RTC applications. Researchers have been exploring DRL algorithms to tackle the challenges in this domain, with several remarkable approaches proposed.

Fang et al. [40] proposes R3Net which uses a DRL algorithm with a recurrent network architecture to model the temporal dependencies in the network conditions. R3Net defines the state of the RL agent as a 4-dimension vector that includes receive rate, average packet interval, packet loss rate, and average Round Trip Time (RTT). The reward is defined to favor high throughput, less packet loss and low delay. R3Net was trained using simulated network traces. While the result from evaluation in simulation looks promising, the test on real networks leads to poor video quality caused by high packet loss, indicating that R3Net provides non-satisfactory adjustment to dynamic network conditions.

Inspired by the hybrid scheme from [41], HRCC [42] combines heuristics congestion control scheme adapted from Google Congestion Control (GCC) [43] with an RL agent that continuously learn to tune the estimated bandwidth from the heuristic scheme. The RL agent in HRCC is also built upon actor-critic framework. The network consists of a $1D$ convolutional layer and two fully connected layers. By using ReLU [23] as the activation function the actor output a gain coefficient. Meanwhile

the critic uses the same structure as the actor and output a value to supervise the actor.

In contrast to previous work [40]–[42], Zhang et al. [44] proposes a federated DRL framework OnRL for bandwidth estimation of video streaming. Aiming at closing the simulation-to-reality gap, OnRL focuses on online RL learning and design an iterative two-stage learning architecture. The work mainly focuses on system-level improvement but not RL. Relevant aspects of DRL such as different network architectures, choices of algorithms or training methodologies need to be further studied.

While most previous approaches have primarily considered packet-level information in their problem formulations, CLCC [45] proposes an alternative DRL based approach that innovatively incorporates both packet and frame-level information into its MDP design, thereby delivering competitive performance.

Bentaleb et al. [46] introduce a system that leverages both heuristic algorithms and RL techniques to accurately forecast bandwidth needs. By merging rule-based methods with learning-based algorithms, they enhance prediction precision. DRL often demands extensive offline training data but operates with limited online data, resulting in inconsistent or suboptimal performance in the real world. To tackle this, they integrate an adaptive bandwidth prediction selector. The selector initially employs a heuristic-based controller to alleviate the cold start issue and seamlessly shifts to a learning-based controller as the system gathers sufficient input data during the session.

Gottipati et al. [47] introduce Merlin, an offline, data-driven approach for bandwidth estimation in RTC systems. Merlin pioneers the use of offline imitation learning for this task, employing behavioral cloning to replicate the performance of an expert Unscented Kalman Filter (UKF) model using pre-collected examples. This approach enables Merlin to achieve superior subjective quality. However, its performance is inherently limited by the expert UKF's capabilities, and it lacks the ability to explore and adapt beyond the expert's behavior.

The challenges of applying RL to real-world problems, particularly in the context of RTC applications, include the need for continuous agent-environment interaction, the dynamic and unpredictable nature of network conditions, and the potential for sub-optimal decisions that may disrupt live traffic and degrade user experience. Additionally, collecting real-world network data for online DRL is resource-intensive and time-consuming.

Inspired by previous work [47], we are motivated to address these challenges by employing offline DRL, which leverages pre-collected datasets for training. This approach eliminates the need for continuous interaction and reduces the risk of negatively impacting user experience.

3.2 Microsoft Dataset for Bandwidth Estimation

The Microsoft Grand Challenge in Bandwidth Estimation [48], organized by Microsoft Research, aimed to foster innovation and advancement of bandwidth estima-

tion for RTC applications. The competition provided a dataset[1] of approximately 28 gigabytes, containing diverse network traces. Participants were tasked with developing algorithms to accurately estimate available bandwidth based on these traces.

The dataset provided by the challenge consists of log files, each generated during the recording of a Microsoft Teams audio/video call. These calls were conducted between testbed nodes distributed across various geographical locations, capturing a diverse range of networking environments. The dataset includes audio/video calls that utilized different bandwidth estimators, also known as behavior policies. These estimators range from traditional methods, such as Kalman-filtering-based estimators and Web Real Time Communication (WebRTC), to various machine learning policies.

Each log file in the dataset contains multiple data records, referred to as trajectories. A trajectory corresponds to a single leg of an audio/video call and is composed of a sequence of four main components:

1. o_n: 150-dimensional observation vector

   Description This vector is derived from on the packet information received by the client during the audio/video call.

2. b_n: bandwidth estimate

   Description These estimates are predicted by a behavior policy, which is the algorithm used to determine the bandwidth allocation for the call.

3. r_n^{audio}: objective audio quality

   Description This metric represents the received audio quality on a scale of $[0, 5]$, with a score of 5 being the highest.

4. r_n^{video}: objective video quality

   Description This metric represents the received video quality on a scale of $[0, 5]$, with a score of 5 being the highest.

The observation vector at time step n captures the observed network statistics that define the state of the bottleneck link between the sender and receiver. These statistics are collected over the five most recent short-term Monitor Intervals (MI) of 60ms and the five most recent long-term MIs of 600ms. The observation vector consists of 150 dimensions, tracking 15 different network features across the five short-term and five long-term MIs, with each feature having a dimension of 10.

The 15 features and their descriptions are as follows:

1. *Receiving rate*: rate at which the client receives data from the sender during a MI, unit: bps.

2. *Number of received packets*: total number of packets received in a MI, unit: packet.

[1] `https://github.com/microsoft/RL4BandwidthEstimationChallenge`

3. *Received bytes*: total number of bytes received in a MI, unit: Bytes.
4. *Queuing delay*: average delay of packets received in a MI minus the minimum packet delay observed so far, unit: ms.
5. *Delay*: average delay of packets received in a MI minus a fixed base delay of 200ms, unit: ms.
6. *Minimum seen delay*: minimum packet delay observed so far, unit: ms.
7. *Delay ratio*: average delay of packets received in a MI divided by the minimum delay of packets received in the same MI, unit: ms/ms.
8. *Delay average minimum difference*: average delay of packets received in a MI minus the minimum delay of packets received in the same MI, unit: ms.
9. *Packet interarrival time*: mean interarrival time of packets received in a MI, unit: ms.
10. *Packet jitter*: standard deviation of interarrival time of packets received in a MI, unit: ms.
11. *Packet loss ratio*: probability of packet loss in a MI, unit: packet/packet.
12. *Average number of lost packets*: average number of lost packets given a loss occurs, unit: packet.
13. *Video packets probability*: proportion of video packets in the packets received in a MI, unit: packet/packet.
14. *Audio packets probability*: proportion of audio packets in the packets received in a MI, unit: packet/packet.
15. *Probing packets probability*: proportion of probing packets in the packets received in a MI, unit: packet/packet.

The indices (zero-indexed) of features over the five short-term MIs are

$$(feature\ number - 1) \times 10, ..., (feature\ number - 1) \times 10 + 4$$

and the indices (zero-indexed) of features over the five long-term MIs are

$$feature\ number \times 10 - 5, ..., feature\ number \times 10 - 1$$

respectively. For example, the packet arrival rate (feature 1) of the first short-term MI is at index 0, and the packet arrival rate over the first long-term MI is at index 5.

The Microsoft dataset, with its rich set of metrics and extensive data logs captured under both emulated and real-world network conditions, presents an excellent opportunity for applying DRL techniques to estimate bandwidth for RTC applications. Consequently, we chose to use this dataset for training DRL models in our project.

3.3 Problem Formulation

To apply reinforcement learning techniques to the bandwidth estimation problem, we must frame it as a MDP. In our model, the network serves as the environment, with its real-time conditions obtained from the observation vector in the dataset. The RTC application's bandwidth estimation function is represented as our RL agent, which interacts with the network (environment). The bandwidth estimate made by the function (agent) is considered the action to be taken. Lastly, we define the reward as the output of a reward function that takes the network conditions as input. This function assesses the quality of the chosen action based on the current network state. The following definitions outline the components of the MDP for our problem:

- **State (s)**: The state of our RL agent is a real value vector that includes one or more network metrics. In our case, we use the observation vector contained in the training dataset as the state. This vector encompasses commonly used network metrics, such as the current baud rate, packet loss rate, packet delivery delay, and jitter, providing a comprehensive view of the networking condition.
- **Action (a)**: The agent's action represents an estimated bandwidth to be utilized by the RTC application, represented as a continuous real value denoting the target bitrate per second.
- **Reward (r)**: A reward signifies the immediate feedback that the RTC application receives upon applying an estimate. The reward should reflect the application's performance and the user's QoE. It is typically computed through a reward function based on relevant metrics such as network throughput, packet loss rate, and packet transmission delay. During our literature review, we discovered two distinct reward functions from [40] and [49], which we refer to as RW_R3Net and RW_QoE. Both reward functions have elegant design and fine tuned parameters. In this project, we will experiment with both reward functions and introduce a third one called RW_VA_QoE calculated exclusively from the video QoE metric available in the dataset. The definitions of all three reward functions are included in Appendix A.
- **Discount Factor (γ)**: The discount factor γ is defined as a real value ranging between 0 and 1, in line with standard reinforcement learning practices. This factor determines the importance of future rewards relative to immediate rewards.

When using a dataset for offline training, the transition probabilities become constant and can be disregarded, as the agent learns from the fixed transitions in the dataset.

3.4 Choice of Algorithms

We have chosen a number of DRL algorithms for our project: BCQ, CQL, SAC and TD3+BC. Each of these algorithms is designed to address specific challenges

in offline RL. BCQ is notable for its use of a generative model to constrain the action space, ensuring that the agent selects actions similar to those in the dataset, thereby mitigating the issue of distributional shift. CQL, on the other hand, introduces a conservative penalty to the Q-function, which discourages the agent from taking actions that lead to out-of-distribution states, thus enhancing stability and performance. Soft Actor-Critic (SAC) is characterized by its use of entropy regularization, which encourages exploration by maximizing a trade-off between expected return and policy entropy, resulting in more robust off-policy learning and improved sample efficiency. TD3+BC combines the strengths of TD3, which employs twin critics and delayed policy updates to stabilize learning, with behavior cloning to regularize the policy towards the dataset actions, effectively balancing exploration and exploitation in offline settings.

Additionally, we also employ DT in our experiments. The DT stands out as a unique approach within the realm of RL due to its innovative use of transformer architecture, originally designed for natural language processing tasks, to address sequential decision-making problems. Unlike traditional RL methods that rely on value functions or policy gradients, the DT models the entire history of states, actions, and rewards as a sequence, enabling it to capture long-term dependencies, making it particularly suitable for scenarios where online interaction is costly or impractical.

Each of these algorithms offers a distinct approach for learning from offline datasets, offering valuable insights for our experiments.

3.5 Experiments design

In this project, we will evaluate a diverse set of offline DRL algorithms to address the bandwidth estimation problem. One of our goals is to compare the performance of these algorithms comprehensively, while also considering key factors such as data quality. To ensure a fair comparison, we will use the default parameters provided by d3rlpy [50] for each algorithm. Specifically, all algorithms except the DT will utilize a common neural network architecture: a fully connected MLP (figure 3.1) for their actor and critic networks. The MLP consists of three layers, each containing 256 neurons, and outputs estimated bandwidth.

In contrast, the Decision Transformer employs a distinct network architecture based on self-attention mechanisms (figure 3.2), which is initially configured with two attention heads and three layers in our case. This setup will allow us to assess how different architectures and configurations impact the effectiveness of the algorithms in the context of bandwidth estimation.

Based on these considerations, we have designed a series of experiments to systematically evaluate and refine the performance of RL algorithms for the bandwidth estimation problem. Our experimental approach comprises five stages, each focusing on a specific aspect:

1. **Data Pre-processing**: We will begin by examining and filtering out unreasonable outliers or duplicated information from the offline dataset. This results

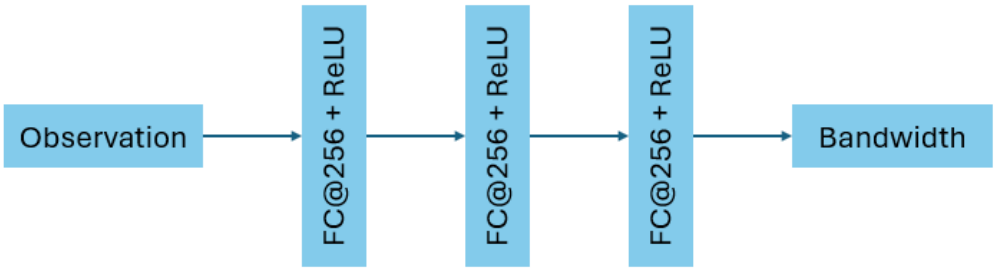


Figure 3.1: A fully connected $256 \times 256 \times 256$ MLP

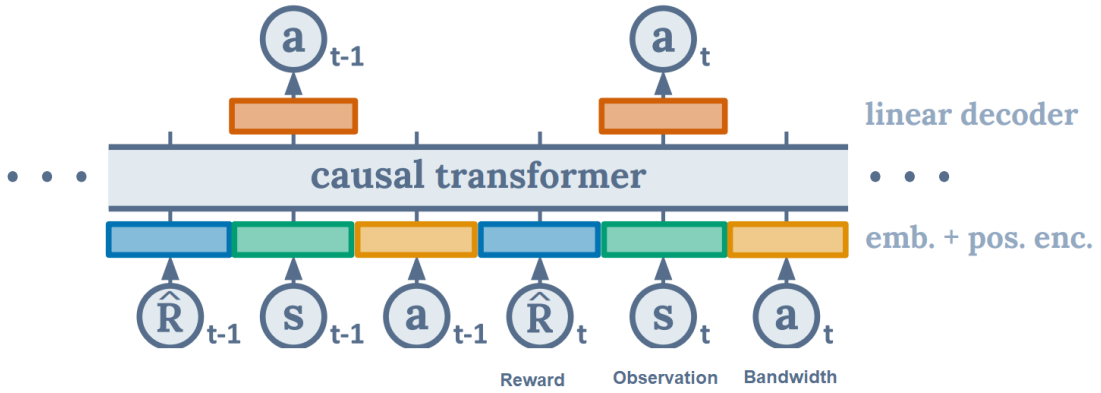


Figure 3.2: DT's attention network, adapted from [38]

in a more compact observation vector and improved computational efficiency.

2. **Reward Function Comparison**: Next, we will compare three reward functions to select the most suitable one for subsequent experiments.
3. **Model Evaluation**: We will then evaluate a wide range of DRL models to identify the most promising ones.
4. **State Reduction**: In the fourth stage, we will delve deeper into the dataset, further reducing the state and assessing the resulting models.
5. **Hyperparameter Optimization**: Finally, we will apply hyperparameter optimization to further assess and refine the selected DRL models, identifying the best model parameters for this project.

This structured approach enables us to thoroughly explore and evaluate the performance of the selected DRL algorithms for bandwidth estimation.

3.6 Training and Evaluation

Given the extensive training dataset, we will utilize the compute cluster at Kiel University to train our models. This high performance cluster allows users to share up to three graphics processing units (GPUs) for parallel machine learning, significantly

accelerating the training process. Leveraging this high-performance computing resource will enable us to handle the large-scale data efficiently and optimize our models for better performance.

3.6.1 Dataset Division

The Microsoft dataset consists of two distinct sets of data logs: a larger set and a smaller set. The larger dataset comprises data logs collected from 18,859 calls under various real-world network conditions, with each log containing approximately 3000 to 4000 records. Notably, none of these records include ground truth information on available networking bandwidth. For this project, we will allocate 10,000 of these data logs for training the RL models, reserving the rest for future usage.

The smaller set of data logs was collected from 9405 emulated test calls with known true capacity of the bottleneck link between the sender and receiver. This subset will be utilized for validating the trained models, enabling an assessment of both our models and the baseline estimator across various aspects, including adaptability to realistic network conditions.

3.6.2 Evaluation and Metrics

Both sets of data logs in the dataset include bandwidth estimations predicted by the built-in estimator of the Real-Time Communication systems during logging. Although the estimator's policies are unknown to us, we can still utilize them as our baseline to benchmark our models.

Accurate bandwidth estimation often results in higher audio and video quality, leading to improved Quality of Experience (QoE)[51]–[53]. Since there is no established assessment model for the quality of audio and video, we will focus on enhancing the prediction accuracy of our models. To this end, inspired by [52], [54], [55], we have selected three metrics for evaluating the prediction accuracy of the model output.

- **Mean Squared Error (MSE)**: MSE is a commonly used loss function that measures the average of the squares of the differences between predicted and actual values. It quantifies the magnitude of errors in predictions, with larger errors being penalized more due to the squaring operation. The MSE is defined as the equation below.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  where n is the number of data points, y_i is the true bandwidth and $\hat{y}_i$ is the predicted value.

- **Prediction Error Rate**: Prediction error rate measures the average derivation between the prediction and the true bandwidth. It is defined as follows,

$$error\_rate = \frac{1}{n}\sum_{i=1}^{n}\min(1, \frac{|\hat{y}_i - y_i|}{y_i})$$

where $\hat{y}_i$ is the prediction from the model and y_i the true bandwidth.

- **Overestimation Rate**: The overestimation rate is a metric that evaluates how much the model overestimates the true link capacity, indicating its conservatism or aggressiveness [54], [56]. Overestimating bandwidth can cause video re-buffering, while underestimation only reduces video quality. RTC systems generally prefer more conservative estimates. The *Overestimation Rate* is defined as follows,

$$over\_rate = \frac{1}{n}\sum_{i=1}^{n}\max(0, \frac{\hat{y}_i - y_i}{y_i})$$

where $\hat{y}_i$ is the prediction from the model and y_i the true bandwidth.

3.7 Toolkits

This section introduces the open source libraries used in this project for implementing the Offline DRL algorithms for bandwidth estimation.

3.7.1 PyTorch

PyTorch [57] is a leading open-source machine learning library developed by Facebook's Artificial Intelligence (AI) Research lab, recognized for its versatility, ease of use, and strong support for deep learning research and development. It is particularly valued for its dynamic computational graph, which allows for real-time model modifications and debugging, making it ideal for research and experimental tasks. Additionally, PyTorch offers a rich ecosystem of tools and extensions that cater to a wide range of applications, including computer vision, natural language processing, and RL. In our project, which involves handling large offline datasets, we leverage PyTorchs distributed data parallel training capabilities to efficiently process the extensive amount of data and train the models within a feasible timeframe.

3.7.2 d3rlpy

d3rlpy [50] is an open-source DRL library specifically designed for offline RL research and applications. The library offers a user-friendly and modular framework for implementing and evaluating offline DRL algorithms. d3rlpy is built on top of the popular deep learning framework PyTorch, ensuring compatibility with a wide range of existing tools and projects.

One of the key advantages of using d3rlpy for this thesis is its ease of use. The library provides a rich set of offline RL algorithms with well-documented and easy-to-understand application programming interfaces, allowing researchers to quickly implement and test various algorithms without delving into the tedious details of

the underlying code. This helps us to focus on the core aspects of our research, such as evaluating and comparing different offline deep reinforcement learning techniques for bandwidth estimation in RTC applications.

Another important benefit of d3rlpy is its optimized run-time performance. The library is designed to efficiently handle large-scale datasets and complex algorithms, ensuring that experiments can be executed in a reasonable amount of time. This is particularly crucial for our research, as we aim to evaluate and compare the performance of various offline RL algorithms on the Microsoft Bandwidth Estimation Challenge dataset, which contains a large collection of network traces, requiring considerably amount of time for model training.

3.7.3 Scikit-learn

Scikit-learn [58], often abbreviated as sklearn, is a popular open-source machine learning library for Python [59]. It offers a wide array of tools for various machine learning tasks, such as classification, regression, clustering, dimensionality reduction, and model selection. Sklearn provides a diverse set of pre-built models, ranging from simple linear and logistic regression to advanced algorithms like Support Vector Machines [60], random forests [61], and Gradient Boosting Machines [62]. Furthermore, it includes utilities for data pre-processing, feature extraction, and model evaluation using metrics like accuracy, precision, recall, and F1-score. In our project, we leverage sklearn's utilities for normalizing input vectors and conducting correlation analysis of features, streamlining our data pre-processing and feature selection processes.

3.7.4 Optuna

Hyperparameters are the configuration variables that govern the behavior and complexity of a learning algorithm, such as learning rates, batch sizes, number of neurons and even neural network architectures. Hyperparameters optimization is a crucial aspect of machine learning and DRL, as it might significantly impact the performance of algorithms and models. However, selecting the optimal set of hyperparameters for a specific problem and dataset can be a challenging and time-consuming task, often requiring extensive experimentation and domain expertise. As we aim to evaluate and compare various offline RL techniques, it is essential to identify the optimal hyperparameter configurations for each algorithm to ensure a fair and accurate comparison.

Optuna [63] is a powerful and flexible open-source software framework for automated hyperparameter optimization. Optuna provides a user-friendly and modular framework for defining and optimizing hyperparameters in machine learning and deep learning models. The library is designed to work seamlessly with popular deep learning frameworks such as TensorFlow [64] and PyTorch, making it an ideal choice for our research on offline DRL.

One of the key motivations for us to choose Optuna is its efficiency in exploring the hyperparameter space. Optuna employs a novel approach called Tree-structured

Parzen Estimator (TPE) for hyperparameter optimization [65], which has been shown to outperform other methods such as grid search and random search in terms of both speed and accuracy. This allows us to efficiently explore the hyperparameter space of the RL algorithms and identify the optimal configurations more quickly, saving computational resources and reducing searching time.

Another motivation for selecting Optuna is its ease of use and flexibility. The library provides a simple and intuitive application programming interface for defining and optimizing hyperparameters, allowing researchers to easily integrate it into their existing workflows. Additionally, Optuna supports various pruning strategies, such as early stopping and median pruning, which can further improve the efficiency of the hyperparameter optimization process by discarding suboptimal configurations.

4

Experiments and Discussions

This chapter presents the experiments and their corresponding results, discussing the key findings obtained throughout our project. Our aim is to provide a comprehensive understanding of the approach employed, the outcomes achieved, and the insights derived from our research.

4.1 Experiment 1: Dataset Pre-processing

The first experiment focuses on applying conventional data pre-processing techniques to the dataset, encompassing the elimination of outliers and the reduction of features.

4.1.1 Removal of Outliers

In the course of this experiment, we examined the dataset and identified unreasonable outliers within the data logs. A notable observation was the presence of a series of records with a constant predicted bandwidth of 20,000 bps at the beginning of each data file. Upon further analysis, we determined that these values did not accurately represent the actual networking conditions. We hypothesized that these values were default estimates generated by the baseline estimator during the initial phase of data logging, before it became fully operational. Consequently, we decided to remove these records from the beginning of every data log file to maintain the integrity and reliability of our dataset.

4.1.2 Features Reduction

Following the initial data cleaning, we proceeded to further inspect the state (observation vector) for additional insights.

The mere examination of the feature set presented in Chapter 3.1.1 already reveals the redundancy and overlapping due to the highly correlated nature of certain features. For example, *Receiving rate* and *Received bytes* in the state appeared to be semantically equivalent but expressed in different units. Furthermore, delay-based features apart from rate-based are also somewhat similar and redundant.

In order to better understand the relationships among all the features, we performed a correlation analysis on all the features contained in the state. The resulting

Feature name	Reference code
Receiving rate	f1
Number of received packets	f2
Received bytes	f3
Queuing delay	f4
Delay	f5
Minimum seen delay	f6
Delay ratio	f7
Delay average minimum difference	f8
Packet interarrival time	f9
Packet jitter	f10
Packet loss ratio	f11
Average number of lost packets	f12
Video packets probability	f13
Audio packets probability	f14
Probing packets probability	f15

Table 4.1: A complete list of features comprising the state, with each feature assigned a unique code for reference purpose.

heatmap, included in figure 4.1, revealed statistical correlations among those features.

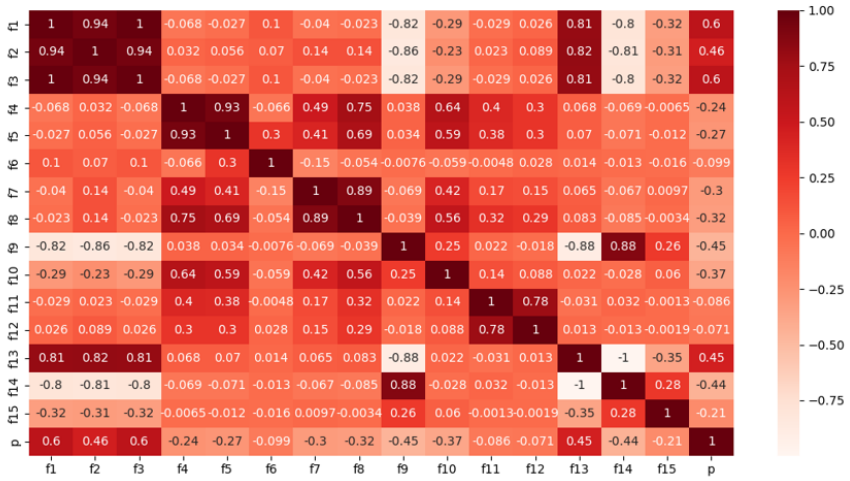


Figure 4.1: Heatmap of correlation analysis of features

Guided by the heatmap's indications, we identified several groups of highly correlated metrics and opted to retain only one metric from each group to eliminate redundancy.

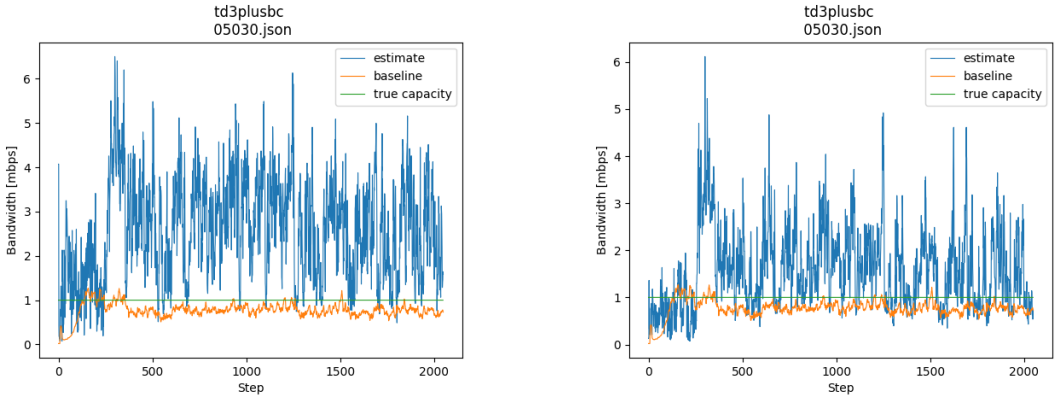
The heatmap reveals multiple groups of features with relatively high correlations. Notably, it validates our earlier observation about the two identical features: receiving rate and received bytes. We then applied the recursive feature elimination

technique to identify potentially redundant features. The results suggest that *Delay*, *Average packet loss*, and *Probing packets probability* can be removed without significantly impacting the model's performance. Considering the reward functions' definition and the analysis above, we decided to eliminate five features: *Received bytes*, *Number of received packets*, *Delay ratio*, *Video packets probability* and *Audio packets probability*. The features list of reduced state is shown as the table 4.2.

Feature name	Reference code
Receiving rate	F1
Queuing delay	F4
Delay	F5
Minimum seen delay	F6
Delay average minimum difference	F8
Packet interarrival time	F9
Packet jitter	F10
Packet loss ratio	F11
Average number of lost packets	F12
Probing packets probability	F15

Table 4.2: The list of features contained in reduced state.

The feature reduction process eliminates redundant or less informative features from the state, resulting in a more compact and efficient training dataset. This optimization leads to a 30% reduction in the size of the training dataset and approximately 20% less training time. Moreover, the feature reduction process enhances the model's overall performance (table 4.3). As shown in Figure 4.2 and Figure 4.3, our experiments revealed that a TD3+BC model trained using the reduced state improved its overall performance, aligning more closely with the baseline curve. These findings highlight the benefits of feature reduction in enhancing both model accuracy and effectiveness.

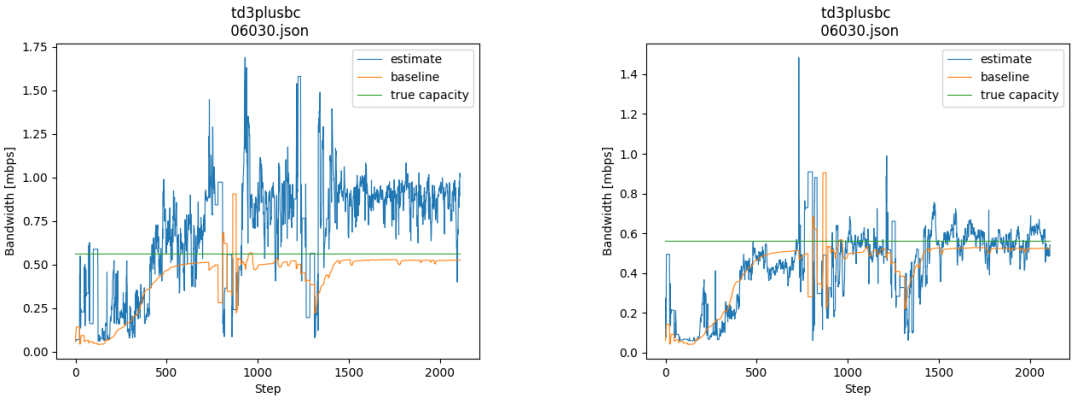


(a) Original feature.

(b) Reduced feature.

Figure 4.2: TD3+BC: original feature vs reduced feature

By streamlining the data through feature reduction, we enhance the overall accuracy and efficiency of our model. Consequently, we will utilize the reduced state



(a) Original feature. (b) Reduced feature.

Figure 4.3: TD3+BC: original feature vs reduced feature. Cont.

Table 4.3: TD3+BC: Original feature vs Reduced feature

Feature	MSE	Prediction Error Rate	Overestimation Rate
Original	1.15	0.45	0.99
Reduced	0.79	0.40	0.30

in subsequent experiments to maintain improved computational performance and model accuracy.

4.2 Experiment 2: Reward Functions Comparison

As mentioned earlier, we have chosen three reward functions, RW_R3Net, RW_QoE and RW_VA_QoE. In this experiment, we compare their performance to determine the most suitable one for subsequent experiments. For each of the reward functions, we employ BCQ algorithms with same configuration and hyperparameters to train a model, allowing us to assess the impact of reward functions on learning performance. Some plots from validation process are included as Figure 4.4,Figure 4.5 and Figure 4.6.

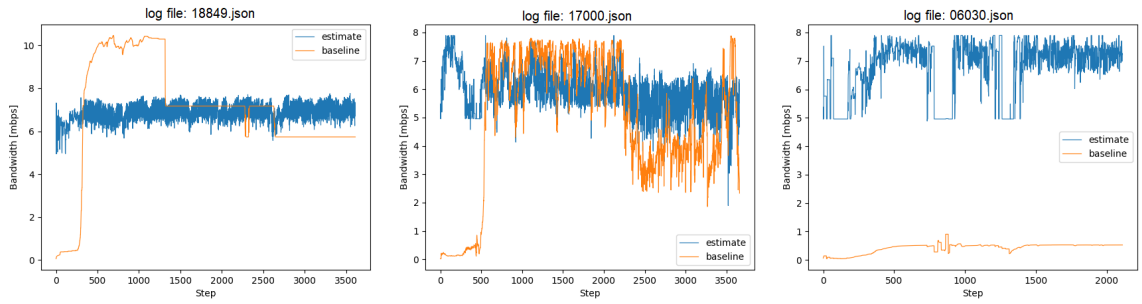


Figure 4.4: BCQ using RW_VA_QoE.

Figure 4.4 indicates that the model trained using reward function *RW_VA_QoE* exhibits significantly worse performance compared to models trained with the other two reward functions. In many cases, the output from the trained model fails to follow the baseline and exhibits substantial drift.

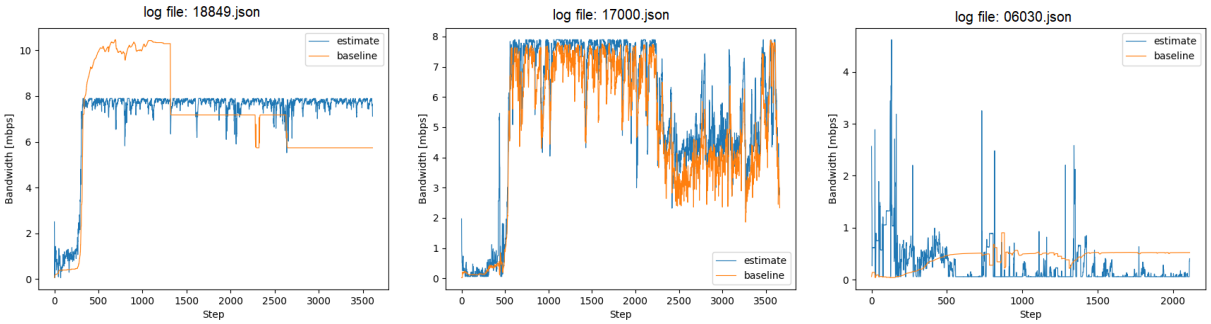


Figure 4.5: BCQ using *RW_R3Net*.

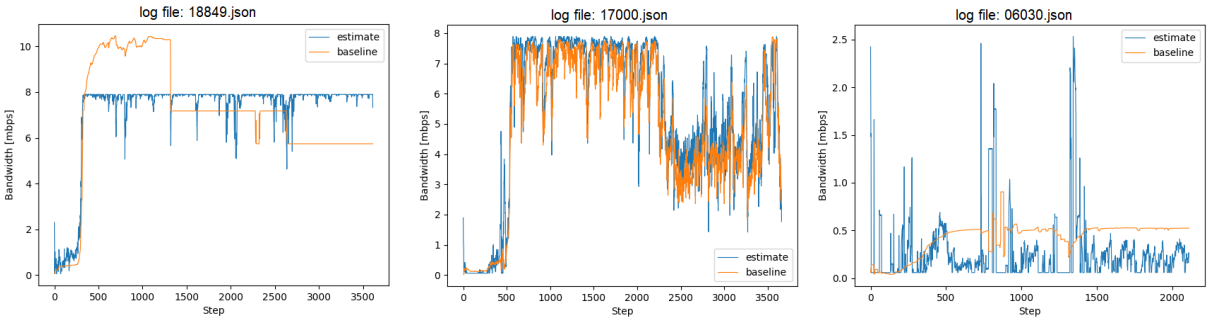


Figure 4.6: BCQ using *RW_QoE*.

In contrast, the models trained with reward functions *RW_R3Net* and *RW_QoE*, using the BCQ algorithm, demonstrate the ability to closely follow the baseline, exhibiting better and similar performance. When comparing these two, we observe that using *RW_QoE* sometimes produces smoother output (log file: 18849.json) 4.6 than using *RW_R3Net*, particularly when the available bandwidth is good. This is most likely because the large coefficient for *Receiving_rate* of *RW_R3Net* encourages more good networking condition than *RW_QoE*.

When the available bandwidth is low (log file: 06030.json), *RW_QoE* also tends to produce smaller outliers than *RW_R3Net*. This could be attributed to the fact that *RW_QoE* includes *packet_jitter* in its definition, which might play a role when network speed is low.

Table 4.4: BCQ: RW_R3Net vs RW_QoE

Reward Function	MSE	Prediction Error Rate	Overestimation Rate
RW_R3Net	1.97	0.64	1.85
RW_QoE	1.63	0.63	1.59

As the evaluation results in table 4.4, using *RW_QoE* can archive better results in all three evaluation metrics.

To further validate and examine the differences between *RW_QoE* and *RW_R3Net*, we conducted the same experiments using a TD3+BC model and a DT model. As shown in Table 4.5 and Table 4.6, the experiment with TD3+BC revealed minimal differences between *RW_QoE* and *RW_R3Net*. However, the DT model with *RW_QoE* demonstrated better prediction accuracy, albeit with slightly higher values for MSE and overestimation rate. Nonetheless, using *RW_QoE* as the reward function consistently resulted in lower prediction errors.

Table 4.5: TD3+BC: RW_R3Net vs RW_QoE

Reward Function	MSE	Prediction Error Rate	Overestimation Rate
RW_R3Net	0.81	0.47	0.28
RW_QoE	0.79	0.40	0.30

Table 4.6: DT: RW_R3Net vs RW_QoE

Reward Function	MSE	Prediction Error Rate	Overestimation Rate
RW_R3Net	2.17	0.53	1.45
RW_QoE	2.41	0.47	1.67

Based on these observations and analysis, we decided to select *RW_QoE* as the reward function for following experiments.

4.3 Experiment 3: Model evaluation

One of the objectives of this project is to evaluate the performance of different offline DRL algorithms when applied to bandwidth estimation. In this experiment, we implemented and tested BCQ, CQL, SAC, TD3+BC, and DT using the same training dataset. We utilized the default hyperparameters provided by the d3rlpy library, maintaining a consistent batch size across all algorithms.

Table 4.7: BCQ vs CQL vs TD3+BC vs DT vs Baseline

Algorithm	MSE	Prediction Error Rate	Overestimation Rate
BCQ	1.63	0.63	1.59
CQL	6.93	0.85	9.65
SAC	15.20	0.72	10.84
TD3+BC	0.79	0.40	0.30
DT	2.41	0.47	1.67
Baseline	0.72	0.13	0.02

From the summary of evaluation results in the table 4.7 and the plot in the Figure 4.7, we observed that TD3+BC exhibits the best overall performance, while BCQ,

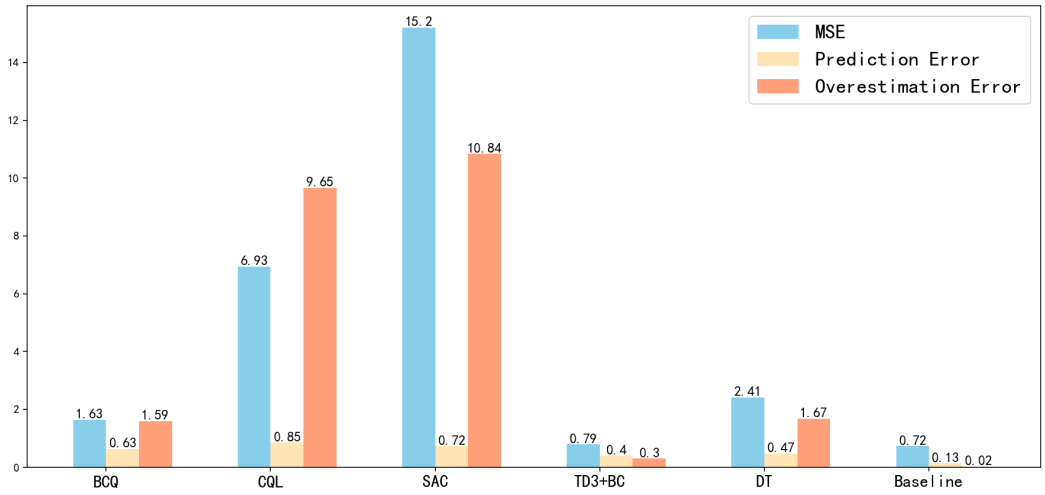


Figure 4.7: Evaluation of all models and baseline.

CQL, and SAC struggle to learn effective models. Notably, DT shows similar metric values to TD3+BC.

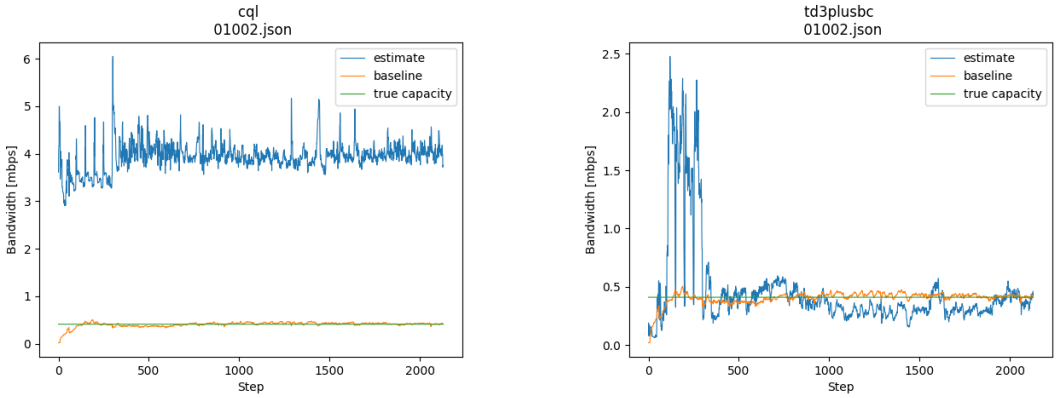


Figure 4.8: CQL vs TD3+BC

Firstly, both CQL and SAC barely replicate the behavior of the baseline estimator present in the dataset (figure 4.8 and figure 4.9). The comparison among BCQ, CQL and SAC presented by figure 4.10 and 4.11 shows that SAC records the highest *MSE* and *Overestimation Rate*, likely due to its use of entropy regularization that encourages exploration, which might exacerbate off-policy evaluation issues and degrade performance. On the other hand, CQL achieves lower values for these criteria, possibly due to the conservative Q-function that constrains its actions.

BCQ, benefiting from a generative model, performs slightly better in learning from the dataset but still suffers from high *Prediction Error Rate*, indicating the model's limitations in this context. Additionally, SAC's predictions exhibit significant variations, potentially caused by its exploration-driven approach.

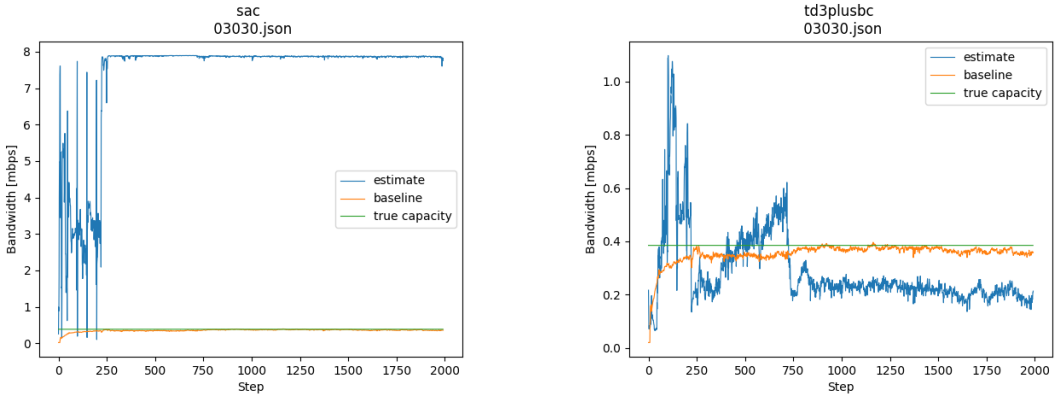


Figure 4.9: SAC vs TD3+BC

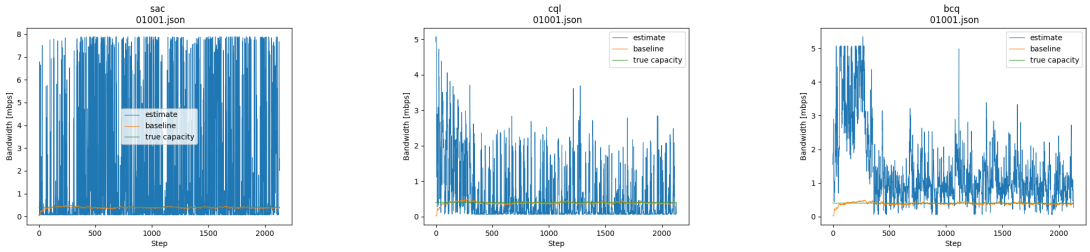


Figure 4.10: SAC vs CQL vs BCQ.

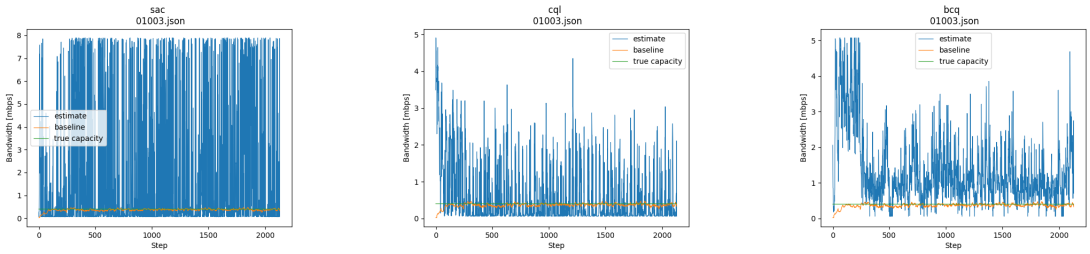


Figure 4.11: SAC vs CQL vs BCQ, cont.

Our findings also indicate that both DT and TD3+BC effectively mimic the behavior of the baseline estimator. However, TD3+BC outperforms in all evaluation metrics. Specifically, TD3+BC closely follows both the baseline and the true capacity (Figure 4.12), attributing to the cloning term incorporated in its design. Conversely, DT excels in producing the majority of predictions with smaller variations, resulting in an averagely smoother prediction curve. This advantage may not be fully captured by the evaluation metrics, as the impact of oscillating values can be somewhat neutralized. Nonetheless, in practice, having a series of predictions with small variations is crucial for real world deployment. A notable drawback of DT, however, is its tendency to overestimate bandwidth, leading to higher *MSE* and *Overestimation Rate*.

Based on the analysis above, we have decided to select TD3+BC for the subsequent experiments, as it is currently the best candidate. Although DT received lower grades with satisfactory performance, we will include it in the next round of experi-

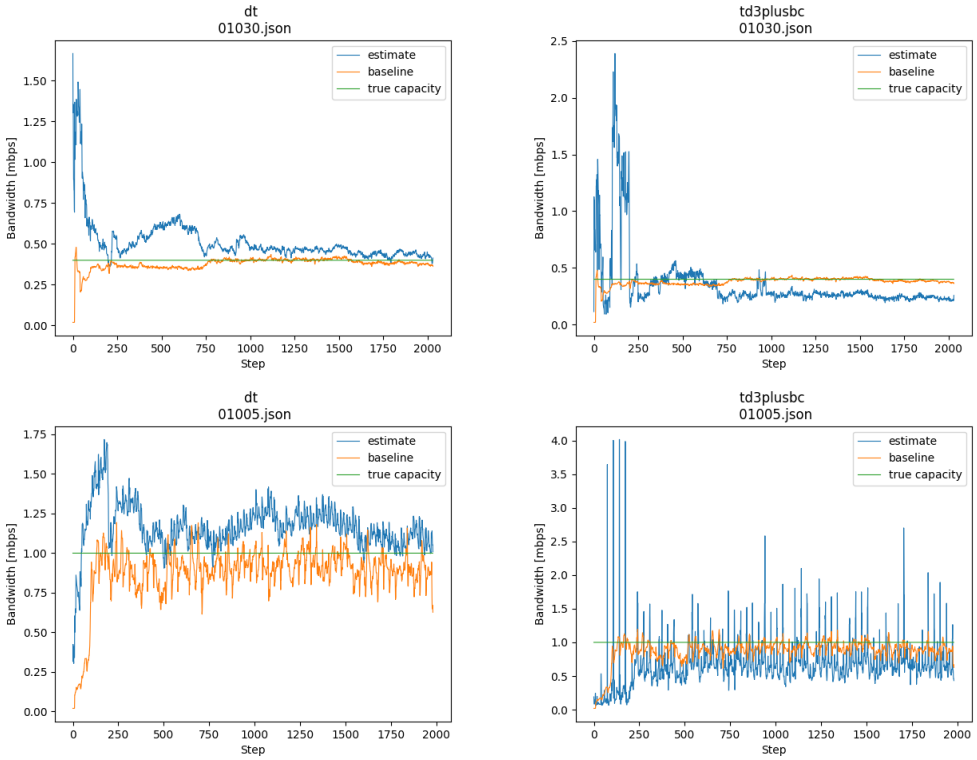


Figure 4.12: DT vs TD3+BC.

ments due to its smoother output, which we find advantageous. Consequently, BCQ, CQL, and SAC will be excluded from further consideration.

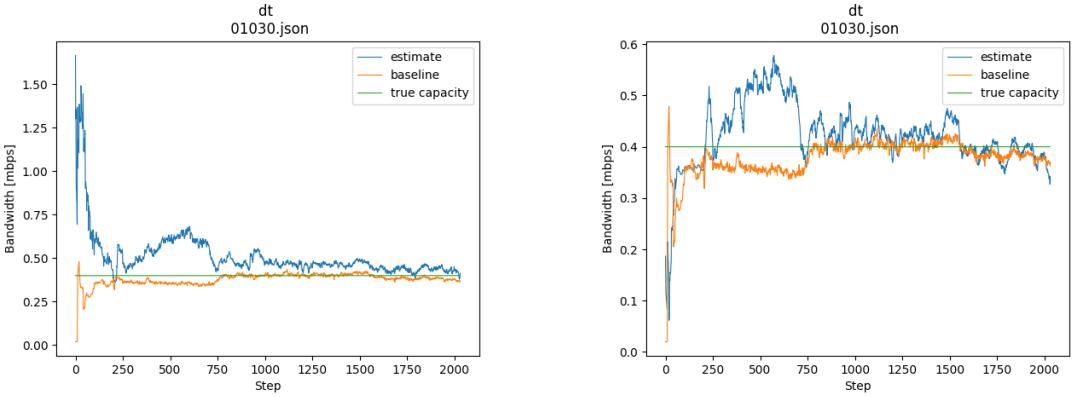
4.4 Experiment 4: State Reduction

Previously, when analyzing the state (observation) vector for potential redundancies, we found that the initial long MI was subdivided into five shorter MIs, all included in the state vector. These were intended to provide a more detailed view of network dynamics. However, each short MI lasted only 60 ms, requiring at least 16 measurements of all metrics per second. Such a high-frequency measurements are often impractical for applications operating under real-world networking conditions. Although real-world training and validation were not planned for this project, we sought to narrow the gap between our research setup and real-world deployment. Thus, we decided to remove these five short MIs from the state as our second attempt of state reduction. After training the DT model with the revised observation (state) vector, we found the evaluation results in Table 4.8 to be quite promising. The *Overestimation Rate* significantly decreased, and the prediction accuracy substantially improved. While the *MSE* slightly increased, the overall model performance notably enhanced when excluding the short MIs from the state.

This improvement was somewhat unexpected, as our initial motivation for revising the state was to prepare for future research rather than to immediately enhance performance. However, a closer examination of the state vector provides a plausible

Table 4.8: Decision Transformer: both MIs vs long MIs

State	MSE	Prediction Error Rate	Overestimation Rate
Short and long MIs	2.41	0.47	1.67
Long MIs	2.84	0.29	0.40

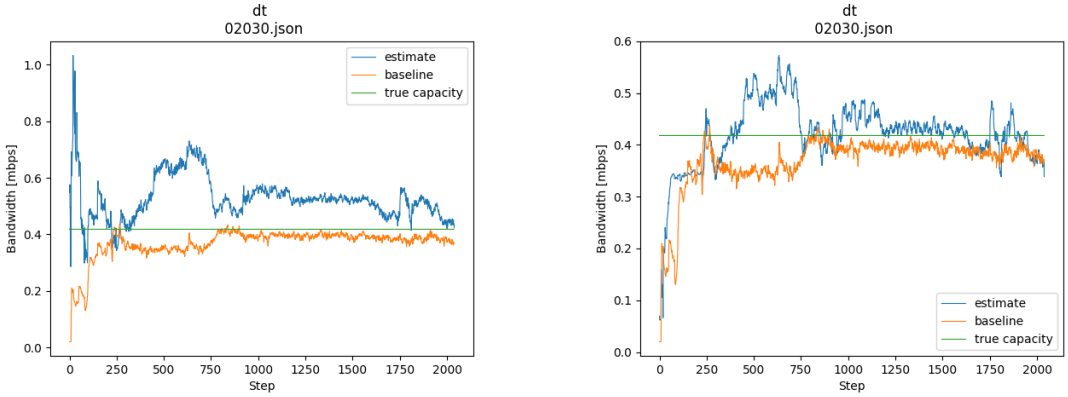


(a) Both short and long MIs. (b) Long MIs.

Figure 4.13: DT: short and long MIs vs long MIs

explanation. The sequence of five short MIs followed by five long MIs, each with identical dimensions, could be misinterpreted as a sequence of ten identical MIs. Removing either the short or long MIs would reduce this ambiguity in the state, thereby improving model performance. This is further supported by the illustration in Figure 4.13 and 4.14, where the model trained using the reduced state produced considerably less overestimation, diminished outliers, and aligned more closely with both the baseline and the true capacity.

In summary, simplifying the state not only makes our research setup more relevant to real-world conditions but also enhances the model's performance by reducing ambiguity in the state representation. As a consequence, we will use only the state



(a) Both short and long MIs. (b) Long MIs.

Figure 4.14: DT: short and long MIs vs long MIs

of long MIs for subsequent experiments.

4.5 Experiment 5: Hyperparameters Optimization

In this experiment, we will focus on hyperparameter finetuning of the algorithms selected from the previous experiment. We discovered that integrating Optuna with the DT algorithms from the offline RL library d3rlpy is impractical due to certain limitations. Therefore, we will use Optuna exclusively for finetuning the hyperparameters of TD3+BC.

4.5.1 Finetuning of TD3+BC

The hyperparameter tuning session can be configured for two different optimization objectives: (i) minimizing Temporal Difference error (TD tuning) and (ii) minimizing the difference between the predicted action and the action in the training data (AD tuning). Throughout this thesis, we will refer to these as TD tuning and AD tuning.

Since all algorithms share a common MLP as the actor network, our finetuning session will leave it unchanged. To avoid an excessive number of parameter combinations, we have selected a small set of common RL parameters for finetuning: τ, γ, α, actor learning rate, critic learning rate, and batch size.

Our finetuning sessions identified two sets of hyperparameters, as shown in the following table 4.9.

Table 4.9: Hyperparameters optimization for TD3+BC

Tuning	τ	γ	α	Actor Learning Rate	Critic Learning Rate	Batch Size
AD Tuning	0.005	0.995	2.5	0.0041	0.0055	512
TD Tuning	0.001	0.999	2.0	0.0047	0.0088	512

We trained two TD3+BC models using those two sets of hyperparameters. The evaluation of these models indicates that TD tuned optimization results in better performing model (Table 4.10).

Table 4.10: AD tuning vs TD tuning for TD3+BC

Tuning objective	MSE	Prediction Error Rate	Overestimation Rate
AD tuning	1.90	0.64	0.84
TD tuning	0.92	0.55	0.21

We observed that the model trained with AD-tuned parameters almost always generates larger outliers compared to the TD-tuned model (Figure 4.15). This is likely because AD tuning focuses more on short-term gains, encouraging the model to closely follow sudden changes in network conditions or baseline predictions. In contrast, TD tuning prioritizes long-term cumulative rewards, which discourages the model from reacting immediately to rapid changes in network conditions.

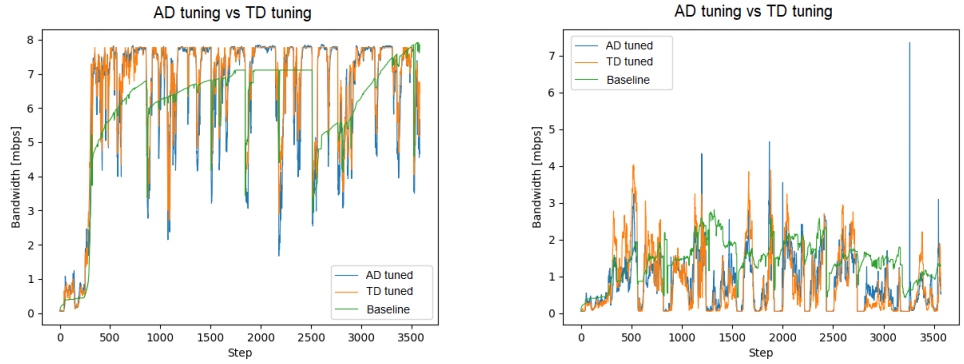


Figure 4.15: Comparing AD tuning vs TD tuning of TD3+BC shows that AD tuned model produces larger outliers.

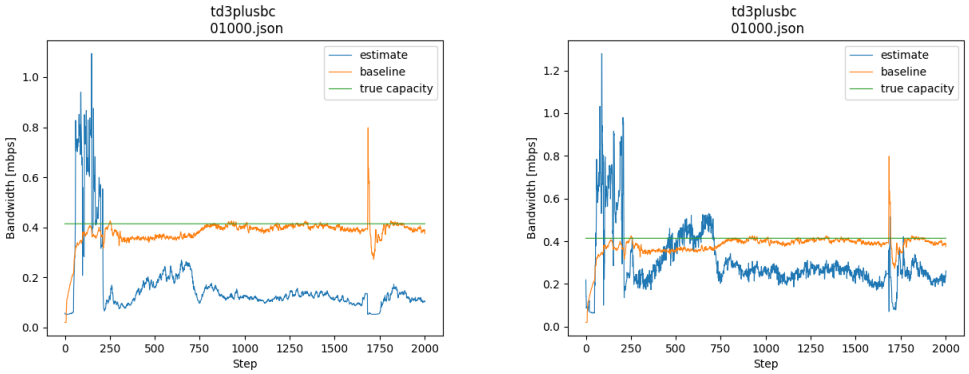


Figure 4.16: TD tuning vs No tuning.

Finally, we compared the model trained with TD-tuned parameters to the model using default parameters. The TD-tuned model performs better in the *Overestimation Rate* metric but worse in the other two metrics (Table 4.11). Additionally, the TD-tuned model produces a smoother prediction curve, driven by its focus on a long-term perspective when responding to sudden changes in network capacity (Figure 4.16). However, this advantage is not reflected in the evaluation metrics. Conversely, the non-tuned model using default parameters predicts bandwidth more accurately, resulting in lower *Prediction Error Rates* and *MSE*. This could be because the default parameters provided by d3rlpy are already well-validated by other research projects and are near optimal.

Table 4.11: No tuning vs TD tuning for TD3+BC

Model	MSE	Prediction Error Rate	Overestimation Rate
No tuning	0.79	0.40	0.30
TD tuning	0.92	0.55	0.21

4.5.2 Finetuning of Decision Transformer

To optimize the DT, we experimented with various combinations of attention heads and layers to assess their impact on model performance. The results are summarized in table 4.12. Upon examining the data, we made several observations.

First, increasing the number of attention heads did not enhance prediction accuracy. Second, the effect of adding more layers depended on the number of attention heads. With two attention heads, adding more layers tended to decrease *Overestimation Rate*. However, this trend reversed with four attention heads.

Generally, a DT with four attention heads performed better overall. Nevertheless, the optimal configuration depended on the specific goal. If minimizing *MSE* and *Prediction Error Rate* is the primary objective, the best configuration is two heads and four layers. Conversely, if reducing *Overestimation Rate* is the priority, a network of four heads and three layers is optimal.

It is important to note that the limited number of combinations tested did not reveal clear monotonic correlations between evaluation metrics and the number of attention heads and layers.

Table 4.12: Optimization of DT

Network configuration	MSE	Prediction Error Rate	Overestimation Rate
2 heads, 3 layers	2.84	0.29	0.40
2 heads, 4 layers	0.75	0.19	0.27
2 heads, 5 layers	4.38	0.32	0.07
4 heads, 3 layers	1.72	0.32	0.05
4 heads, 4 layers	2.06	0.29	0.10
4 heads, 5 layers	3.97	0.33	0.21

4.6 Best Models

In summary, our experiments have identified the top-performing models in this project: a DT with two attention heads and four layers, closely followed by a non-tuned TD3+BC. Compared to TD3+BC, the DT demonstrated a significantly lower *Prediction Error Rate*, while other metrics indicated similar performance (Table 4.13).

Table 4.13: Best models vs Baseline

Model	MSE	Prediction Error Rate	Overestimation Rate
Non-tuned TD3BC	0.79	0.40	0.30
DT with 2 heads, 4 layers	0.75	0.19	0.27
Baseline	0.72	0.13	0.02

When compared to the baseline and as illustrated in Figure 4.17, the DT exhibited comparable performance in terms of *MSE* and *Prediction Error Rate*. However, its

Overestimation Rate remained relatively higher than that of the baseline. This discrepancy may be attributed to the reward function's inability to capture all environmental feedback, thus failing to provide the model with an accurate representation of network conditions.

Additionally, the baseline policy in the dataset remains a black box, potentially consisting of multiple models such as the GCC [43] algorithm coupled with XGBoost [66] or similar, thereby benefiting from the advantages of all components. Lastly, it is important to note that the offline training of our models essentially aims to imitate the baseline behavior present in the dataset. Therefore, when testing with the offline dataset, it is expected that the model performance will be close to but not better than the baseline.

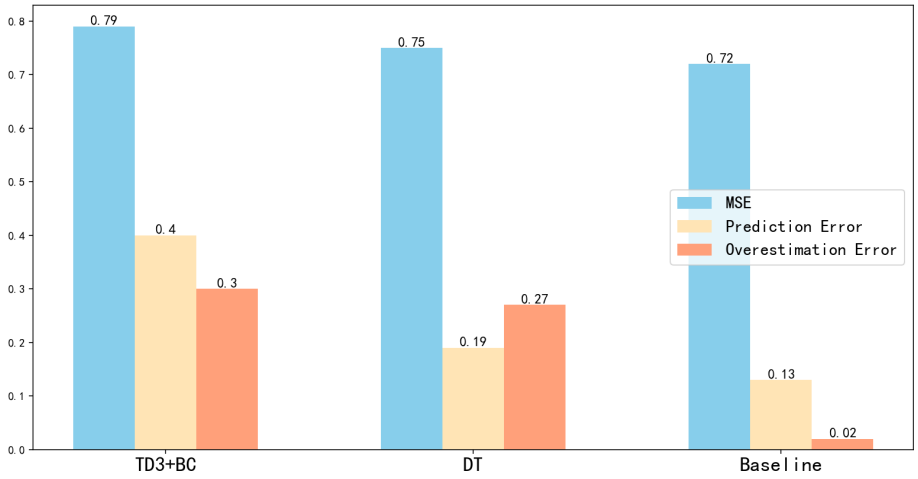


Figure 4.17: Comparing the baseline to the best models identified in this project: a DT with 2 attention heads and 4 layers and a non-tuned TD3+BC.

5

Conclusion

This chapter presents the findings in relation to the established objectives, revisiting the research questions posed at the outset of the thesis and providing comprehensive answers based on our research. It also outlines potential future directions for further exploration in this field. Finally, the chapter concludes with a summary of the thesis work.

5.1 Findings

Our initial findings underscore the critical importance of data engineering as a preparatory step for any machine learning task. By eliminating noisy outliers, we aimed to improve model accuracy. Through correlation analysis of metrics in the state vector, we successfully reduced the dataset size by 30%. This reduction led to a 20% decrease in training time, significantly enhancing computational efficiency. Moreover, the model trained using a streamlined state vector with fewer metrics demonstrated higher prediction accuracy. Our second attempt at compacting the state vector involved removing all shortMIs from the state vector, which resulted in a DT model with even better performance.

The comparison of three reward definitions revealed that the choice of reward function may profoundly impact the performance of RL models. A carefully designed reward definition, such as RW_QoE, showed superior performance in terms of prediction accuracy and stability, particularly under varying network conditions. RW_QoE, which utilizes the most network metrics in its definition, providing a comprehensive understanding of network conditions. In contrast, RW_VA_QoE, which uses only one metric, was incapable of effectively guiding model learning. RW_R3NET, despite using fewer network metrics, performed reasonably well due to a set of optimally validated coefficients.

The evaluation of DRL algorithms indicated that some popular offline DRL algorithms, such as CQL and SAC, might not be well-suited for bandwidth estimation. While BCQ showed some capacity to learn from the dataset, its performance remained insufficient for practical use. In contrast, TD3+BC successfully captured the behavior patterns in the dataset, resulting in superior prediction performance. Unlike in traditional deep learning, hyperparameter tuning of RL parameters for TD3+BC in this context did not yield significant performance improvements. However, notable differences in model behavior were observed when tuning parameters

for different optimization objectives, emphasizing the importance of targeted parameter tuning.

The DT, a novel mechanism for modeling sequential decision-making problems, demonstrated considerable promise in this project. Outperforming other algorithms and closely following the baseline, the DT excelled in generating smoother prediction curves with a low *Prediction Error Rate*, showcasing its ability to capture long-term temporal dependencies through self-attention. We also observed that simply adding attention heads or deepening the layers of attention does not necessarily improve the performance of the DT. This suggests that a more sophisticated approach is required to effectively optimize its architecture.

Although computational efficiency was not primary focus of the project, we observed a major drawback of the DT: its extensive training and inference time. Without precise baseline setup and benchmark, our rough estimate suggests that training a DT takes approximately 200% longer than training any of the other models. Moreover, the time required for the DT to predict an action is roughly four to twenty times longer than that of other models. In contrast, TD3+BC requires the least amount of time for both training and prediction, further enhancing its position as one of the top candidates.

5.2 Answers to Research Questions

Based on the outcomes of this project, we summarize the answers to the research questions as follows:

1. *How do the choices of state representation impact the performance of the DRL model?*
   Answer: Removing highly correlated or redundant information from the state significantly improves the computational effectiveness and model accuracy.

2. *How do the choices of reward function impact the performance of the DRL model?*
   Answer: Generally, a reward function incorporating more network metrics can better guide the model's learning process. However, with optimized coefficients, a reward function with fewer metrics can also archive strong performance.

3. *How do the choices of learning algorithms impact the performance of the DRL model?*
   Answer: A Decision Transformer model demonstrates best performance, while a model trained using TD3+BC performs similarly to the Decision Transformer but with a higher *prediction error rate*.

4. *How do the choices of neural network architectures impact the performance of the DRL model?*
   Answer: We have evaluated two neural network architectures: MLP and self-attention. While both architectures are capable of learning a model, the self-attention architecture is notably better at capturing the hidden sequential

pattern in the training dataset.

5.3 Future Directions

While our work has provided valuable insights into the application of deep reinforcement learning for bandwidth estimation, several avenues for future research remain:

- *Advanced Reward Functions*: Our experiments demonstrated that the design of the reward function significantly impacts model performance. Future work could explore more sophisticated reward functions that better capture the nuances of user experience and network conditions. For instance, incorporating user experience-related metrics, such as video or audio quality, into the reward function may provide more accurate assessments of model performance.
- *Additional Neural Network Architectures*: During the project, we implemented an LSTM as part of a SAC model. However, the model did not learn effectively from the dataset, possibly due to incorrect implementation or limitations of the d3rlpy library. Given that bandwidth estimation involves making predictions based on time series data, experimenting with RNNs like architectures [23] could be a promising direction for improving model performance.
- *Hyperparameter Finetuning of Neural Networks*: In this project, we used a common neural network architecture for the actor network across all algorithms, except for the Decision Transformer. Due to limited computational resources, our hyperparameter optimization focused only on a small set of reinforcement learning parameters, overlooking the parameters of the actor neural network, which may also play a crucial role in the learning process. Fine-tuning these parameters and assessing their impact would be a valuable direction for future research.
- *Enhanced Evaluation Metrics*: One of the evaluation metrics used in this project is MSE, which measures the degree of variation in model predictionsan essential factor for real-time communication applications. In practice, this is often visually represented by the smoothness of the prediction curve. However, these two indicators may sometimes contradict each other. For instance, during the comparison of all models (Table 4.7), despite the DT having a higher MSE value than TD3+BC, our visual inspection of figure 4.12 indicates that DT produces a smoother prediction curve. This suggests that MSE may not adequately capture the smoothness of the prediction curve, partially due to its tendency of penalizing large derivations. Therefore, developing more objective metrics that evaluate the stability and smoothness of model predictions would be valuable for a more comprehensive assessment of model performance.
- *On-Policy Evaluation*: In this project, models were trained and evaluated using an offline dataset, which may lead to a potential mismatch between the actions taken by the agent during evaluation and the fixed actions in the dataset. This issue, known as the "off-policy evaluation" [67] problem, can introduce biases and degrade the performance of the trained policy. Deploying

the model in a real-world networking environment would enable an objective evaluation of its true performance.

- *Further Investigation of Results*: Due to the limited timeframe of this thesis, the analysis of experimental results primarily focused on system-level performance. Although many interesting insights were uncovered, not all of them could be fully understood. For instance, CQL, despite being one of the most promising offline DRL algorithms, exhibited the worst performance during evaluation. A natural next step would involve diving deeper into these to identify and understand the root causes.

These suggested future directions offer a roadmap for enhancing the current understanding on application of DRL in bandwidth estimation, potentially leading to more robust and efficient models.

5.4 Summary of Project

Throughout this thesis, we have explored the application of DRL for bandwidth estimation in real-time communication systems. Our work began with an extensive review of the current state of bandwidth estimation, highlighting the challenges and motivations behind our proposed approach. We defined the scope of our research and formulated key research questions to guide our investigation.

We conducted a thorough study of the bandwidth estimation problem and studied the latest literature. After evaluating relevant work, we decided to leverage DRL with offline training to address this challenge. We provided a rigorous MDP formulation of the problem and implemented three distinct reward definitions. Additionally, we established three metrics for evaluation.

Our workflow began with data engineering, where we pre-processed and cleaned the dataset, removing noisy and redundant information. Next, we compared the performance of the reward definitions to determine which best suits our problems formulation and the dataset. We evaluated a wide range of offline DRL algorithms, including BCQ, CQL, SAC, TD3+BC, and DT. Each algorithm was assessed using a common dataset and default configurations. During this process, we refined the state vector by reducing short MIs, which led to notable improvements in model performance.

Given that Decision Transformer and TD3+BC demonstrated the best performance, we proceeded to fine-tune TD3+BC using an automated hyperparameter tuning toolkit, investigating the impacts of different optimization schemes. We also experimented with various configurations of the attention network in the DT to identify the optimal setup for this project.

Although neither TD3+BC nor DT matched the performance of the baseline in the dataset, our work has provided valuable insights and identified key opportunities for future enhancement. These findings suggest areas where further refinement and innovation could potentially improve model accuracy and robustness, making them more effective for real-time communication systems.

In conclusion, this thesis has yielded significant insights into the domain of bandwidth estimation through the application of offline DRL. The findings presented and the proposed future directions together lay a solid foundation for continued research and development in this burgeoning area.

Bibliography

[1] C. Guerrero Santander, "Available bandwidth estimation: A hidden markov model approach," American English, in *Available Bandwidth Estimation: A Hidden Markov Model Approach*, Lambert Academic Publishing, Oct. 2010, ISBN: 9783843360487.

[2] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, RFC 3550, Jul. 2003. DOI: 10.17487/RFC3550. [Online]. Available: https://www.rfc-editor.org/info/rfc3550.

[3] R. Prasad, C. Dovrolis, M. Murray, and K. Claffy, "Bandwidth estimation: Metrics, measurement techniques, and tools," *IEEE Network*, vol. 17, no. 6, pp. 27–35, 2003. DOI: 10.1109/MNET.2003.1248658.

[4] S. S. Chaudhari and R. C. Biradar, "Survey of bandwidth estimation techniques in communication networks," *Wirel. Pers. Commun.*, vol. 83, no. 2, pp. 1425–1476, Jul. 2015, ISSN: 0929-6212. DOI: 10.1007/s11277-015-2459-2. [Online]. Available: https://doi.org/10.1007/s11277-015-2459-2.

[5] S.-r. Kang, X. Liu, M. Dai, and D. Loguinov, "Packet-pair bandwidth estimation: Stochastic analysis of a single congested node," in *Proceedings of the 12th IEEE International Conference on Network Protocols, 2004. ICNP 2004.*, 2004, pp. 316–325. DOI: 10.1109/ICNP.2004.1348121.

[6] Z. Yuan, H. Venkataraman, and G.-M. Muntean, "Mbe: Model-based available bandwidth estimation for ieee 802.11 data communications," *IEEE Transactions on Vehicular Technology*, vol. 61, no. 5, pp. 2158–2171, 2012. DOI: 10.1109/TVT.2012.2190760.

[7] M. Dong, Q. Li, D. Zarchy, B. Godfrey, and M. Schapira, "PCC: re-architecting congestion control for consistent high performance," *CoRR*, vol. abs/1409.7092, 2014. arXiv: 1409.7092. [Online]. Available: http://arxiv.org/abs/1409.7092.

[8] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri and R. Salakhutdinov, Eds., ser. Proceedings of Machine Learning Research, vol. 97, PMLR, Jun. 2019, pp. 3050–3059. [Online]. Available: https://proceedings.mlr.press/v97/jay19a.html.

[9] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, Los Angeles, CA, USA: Association for Computing Machinery, 2017, pp. 197–210, ISBN: 9781450346535. DOI:

10.1145/3098822.3098843. [Online]. Available: https://doi.org/10.1145/3098822.3098843.

[10] K. Winstein and H. Balakrishnan, "Tcp ex machina: Computer-generated congestion control," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13, Hong Kong, China: Association for Computing Machinery, 2013, pp. 123–134, ISBN: 9781450320566. DOI: 10.1145/2486001.2486020. [Online]. Available: https://doi.org/10.1145/2486001.2486020.

[11] A. Zhou, H. Zhang, G. Su, *et al.*, "Learning to coordinate video codec with transport protocol for mobile video telephony," in *The 25th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '19, Los Cabos, Mexico: Association for Computing Machinery, 2019, ISBN: 9781450361699. DOI: 10.1145/3300061.3345430. [Online]. Available: https://doi.org/10.1145/3300061.3345430.

[12] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *CoRR*, vol. abs/2005.01643, 2020. arXiv: 2005.01643. [Online]. Available: https://arxiv.org/abs/2005.01643.

[13] R. F. Prudencio, M. R. O. A. Maximo, and E. L. Colombini, "A survey on offline reinforcement learning: Taxonomy, review, and open problems," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–, 2024, ISSN: 2162-2388. DOI: 10.1109/tnnls.2023.3250269. [Online]. Available: http://dx.doi.org/10.1109/TNNLS.2023.3250269.

[14] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[15] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018, ISBN: 0262039249.

[16] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992, ISSN: 1573-0565. DOI: 10.1007/BF00992698.

[17] J. Peters, "Policy gradient methods," *Scholarpedia*, vol. 5, no. 11, p. 3698, 2010, revision #137199. DOI: 10.4249/scholarpedia.3698.

[18] D. D. N. Vidit Saxena Burak Guldogan, *On-policy and off-policy reinforcement learning: Key features and differences*, Accessed on July 26, 2024, 2023. [Online]. Available: https://www.ericsson.com/en/blog/2023/12/online-and-offline-reinforcement-learning-what-are-they-and-how-do-they-compare.

[19] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[20] H. J. Kelley, "Gradient theory of optimal flight paths," *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[21] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds.,

vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[25] K. Cho, B. van Merrienboer, C. Gulcehre, *et al.*, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, 2014. arXiv: 1406.1078 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1406.1078.

[26] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762.

[27] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: 1810.04805 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1810.04805.

[28] G. Yenduri, R. M, C. S. G, *et al.*, *Generative pre-trained transformer: A comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions*, 2023. arXiv: 2305.10435 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2305.10435.

[29] C. Raffel, N. Shazeer, A. Roberts, *et al.*, *Exploring the limits of transfer learning with a unified text-to-text transformer*, 2023. arXiv: 1910.10683 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1910.10683.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. arXiv: 1312.5602. [Online]. Available: http://arxiv.org/abs/1312.5602.

[31] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning*, E. P. Xing and T. Jebara, Eds., ser. Proceedings of Machine Learning Research, vol. 32, Bejing, China: PMLR, Jun. 2014, pp. 387–395. [Online]. Available: https://proceedings.mlr.press/v32/silver14.html.

[32] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *CoRR*, vol. abs/1801.01290, 2018. arXiv: 1801.01290. [Online]. Available: http://arxiv.org/abs/1801.01290.

[33] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. Learn.*, vol. 8, no. 34, pp. 293–321, May 1992, ISSN: 0885-6125. DOI: 10.1007/BF00992699. [Online]. Available: https://doi.org/10.1007/BF00992699.

[34] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[35] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," *CoRR*, vol. abs/1812.02900, 2018. arXiv: 1812.02900. [Online]. Available: http://arxiv.org/abs/1812.02900.

[36] A. Kumar, A. Zhou, G. Tucker, and S. Levine, *Conservative q-learning for offline reinforcement learning*, 2020. arXiv: 2006.04779 [cs.LG].

[37] S. Fujimoto and S. S. Gu, "A minimalist approach to offline reinforcement learning," in *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[38] L. Chen, K. Lu, A. Rajeswaran, *et al.*, "Decision transformer: Reinforcement learning via sequence modeling," *CoRR*, vol. abs/2106.01345, 2021. arXiv: 2106.01345. [Online]. Available: https://arxiv.org/abs/2106.01345.

[39] R. F. Muktiadji, M. A. M. Ramli, and A. H. Milyani, "Twin-delayed deep deterministic policy gradient algorithm to control a boost converter in a dc microgrid," *Electronics*, vol. 13, no. 2, 2024, ISSN: 2079-9292. DOI: 10.3390/electronics13020433. [Online]. Available: https://www.mdpi.com/2079-9292/13/2/433.

[40] J. Fang, M. Ellis, B. Li, *et al.*, "Reinforcement learning for bandwidth estimation and congestion control in real-time communications," *CoRR*, vol. abs/1912.02222, 2019. arXiv: 1912.02222. [Online]. Available: http://arxiv.org/abs/1912.02222.

[41] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Classic meets modern: A pragmatic learning-based congestion control for the internet," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '20, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 632–647, ISBN: 9781450379557. DOI: 10.1145/3387514.3405892. [Online]. Available: https://doi.org/10.1145/3387514.3405892.

[42] B. Wang, Y. Zhang, S. Qian, Z. Pan, and Y. Xie, "A hybrid receiver-side congestion control scheme for web real-time communication," in *Proceedings of the 12th ACM Multimedia Systems Conference*, ser. MMSys '21, Istanbul, Turkey: Association for Computing Machinery, 2021, pp. 332–338, ISBN: 9781450384346. DOI: 10.1145/3458305.3479970. [Online]. Available: https://doi.org/10.1145/3458305.3479970.

[43] S. Holmer, H. Lundin, G. Carlucci, L. D. Cicco, and S. Mascolo, "A Google Congestion Control Algorithm for Real-Time Communication," Internet Engineering Task Force, Internet-Draft draft-ietf-rmcat-gcc-02, Jul. 2016, Work in Progress, 19 pp. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-rmcat-gcc/02/.

[44] H. Zhang, A. Zhou, J. Lu, *et al.*, "Onrl: Improving mobile video telephony via online reinforcement learning," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '20, London, United Kingdom: Association for Computing Machinery, 2020, ISBN: 9781450370851. DOI: 10.1145/3372224.3419186. [Online]. Available: https://doi.org/10.1145/3372224.3419186.

[45] H. Li, B. Lu, J. Xu, *et al.*, "Reinforcement learning based cross-layer congestion control for real-time communication," in *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, BMSB 2022, Bilbao, Spain,*

June 15-17, 2022, IEEE, 2022, pp. 1–6. DOI: 10.1109/BMSB55706.2022.9828569.

[46] A. Bentaleb, M. N. Akcay, M. Lim, A. C. Begen, and R. Zimmermann, "Bob: Bandwidth prediction for real-time communications using heuristic and reinforcement learning," *IEEE Transactions on Multimedia*, vol. 25, pp. 6930–6945, 2023. DOI: 10.1109/TMM.2022.3216456.

[47] A. Gottipati, S. Khairy, G. Mittag, V. Gopal, and R. Cutler, *Real-time bandwidth estimation from offline expert demonstrations*, 2023. arXiv: 2309.13481 [cs.NI]. [Online]. Available: https://arxiv.org/abs/2309.13481.

[48] Microsoft, *Microsoft grand challenge*, Accessed on July 11, 2024, 2024. [Online]. Available: https://www.microsoft.com/en-us/research/academic-program/bandwidth-estimation-challenge/links/.

[49] N. Smirnov and S. Tomforde, "Real-time data transmission optimization onă5g remote-controlled units using deep reinforcement learning," in *Architecture of Computing Systems*, G. Goumas, S. Tomforde, J. Brehm, S. Wildermann, and T. Pionteck, Eds., Cham: Springer Nature Switzerland, 2023, pp. 281–295, ISBN: 978-3-031-42785-5.

[50] T. Seno and M. Imai, "D3rlpy: An offline deep reinforcement learning library," *CoRR*, vol. abs/2111.03788, 2021. arXiv: 2111.03788. [Online]. Available: https://arxiv.org/abs/2111.03788.

[51] G. Lv, Q. Wu, W. Wang, Z. Li, and G. Xie, "Lumos: Towards better video streaming qoe through accurate throughput prediction," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 650–659. DOI: 10.1109/INFOCOM48880.2022.9796948.

[52] Y. Sun, X. Yin, J. Jiang, *et al.*, "Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction.," in *SIGCOMM*, M. P. Barcellos, J. Crowcroft, A. Vahdat, and S. Katti, Eds., ACM, 2016, pp. 272–285, ISBN: 978-1-4503-4193-6. [Online]. Available: http://dblp.uni-trier.de/db/conf/sigcomm/sigcomm2016.html#SunYJSLWLS16.

[53] F. Y. Yan, H. Ayers, C. Zhu, *et al.*, *Learning in situ: A randomized experiment in video streaming*, 2019. arXiv: 1906.01113 [cs.NI]. [Online]. Available: https://arxiv.org/abs/1906.01113.

[54] G. Lv, Q. Wu, Y. Liu, *et al.*, "Chorus: Coordinating mobile multipath scheduling and adaptive video streaming," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ser. ACM MobiCom '24, Washington D.C., DC, USA: Association for Computing Machinery, 2024, pp. 246–262. DOI: 10.1145/3636534.3649359. [Online]. Available: https://doi.org/10.1145/3636534.3649359.

[55] J. Jiang, V. Sekar, and H. Zhang, "Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive," *IEEE/ACM Transactions on Networking*, vol. 22, no. 1, pp. 326–340, 2014. DOI: 10.1109/TNET.2013.2291681.

[56] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over http.," in *SIGCOMM*, S. Uhlig, O. Maennel, B. Karp, and J. Padhye, Eds., ACM, 2015, pp. 325–338, ISBN: 978-

1-4503-3542-3. [Online]. Available: http://dblp.uni-trier.de/db/conf/sigcomm/sigcomm2015.html#YinJSS15.

[57] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[58] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[59] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[60] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[61] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, IEEE, vol. 1, 1995, pp. 278–282.

[62] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[63] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2623–2631.

[64] Martín Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[65] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., vol. 24, Curran Associates, Inc., 2011. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc6
Paper.pdf.

[66] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD 16, ACM, Aug. 2016. DOI: 10.1145/2939672.2939785. [Online]. Available: http://dx.doi.org/10.1145/2939672.2939785.

[67] M. Uehara, C. Shi, and N. Kallus, *A review of off-policy evaluation in reinforcement learning*, 2022. arXiv: 2212.06355 [stat.ML]. [Online]. Available: https://arxiv.org/abs/2212.06355.

A

Appendix A

A.1 Reward Functions

This section summarizes the reward functions' definitions used in our project.

A.1.1 RW_R3Net

RW_R3Net is defined as A.1

$$reward = 0.6 \times ln(4R + 1) - D - 10L \tag{A.1}$$

where R is the receive rate in bps, D is the average RTT in seconds, and L is the packet loss rate, at a given time step. The reward formula penalize the packet loss and delay while award more packets at receiver side. Its value falls in the range of $[0, 5]$.

A.1.2 RW_QoE

RW_QoE is defined as a weighted sum of four measurements, namely qoe_{rate}, qoe_{delays}, qoe_{loss}, $qoe_{jitters}$, as in the equation A.2. This reward also falls in the range $[0, 5]$.

$$reward = w_1 * qoe_{rate} + w_2 * qoe_{delays} + w_3 * qoe_{loss} + w_4 * qoe_{jitters} \tag{A.2}$$

where $\sum_{1 \leq i \leq 4} w_i = 1$

The definitions of the measurements above are as following

$$qoe_{rate} = ln((e - 1) \times (\frac{r}{r_{max}} + 1)) \tag{A.3}$$

where r and r_{max} are the receiving rate and maximum receiving rate (bps) respectively.

$$qoe_{delays} = \frac{d_{max} - d}{d_{max} - d_{min}} \tag{A.4}$$

where d is the packet delay and d_{max} and d_{min} are maximum and minimum delay (ms) respectively.

$$qoe_{loss} = 1 - l \tag{A.5}$$

where l is the packet loss ratio.

$$qoe_{jitters} = 1 - 0.04 * \sqrt{\min(j, 625)} \tag{A.6}$$

where j is the packet jitter (ms).

A.1.3 RW_VA_QoE

RW_VA_QoE is introduced by us in this project. As shown in A.7, this reward function uses solely the metric *objective video quality* in the dataset. This metric is represented as a real value between $[0, 5]$, with a score of 5 being the best quality.

$$reward = objective\_video\_quality \tag{A.7}$$

B

Appendix B

Some figures in the previous chapter contain multiple images, making them difficult to read. This section provides high-definition versions of those images for better clarity.

B.1 Images of Figure 4.4

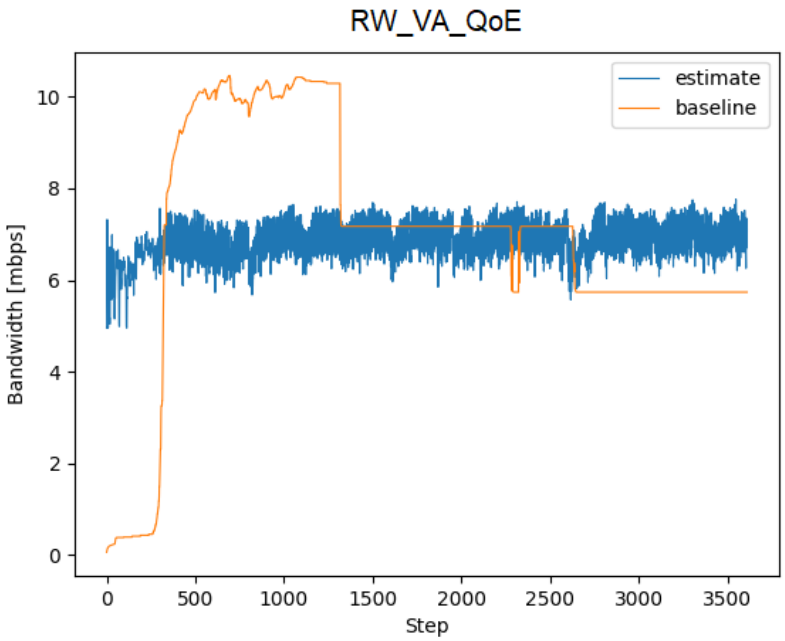


Figure B.1: BCQ using RW_VA_QoE and log file: 18849.json

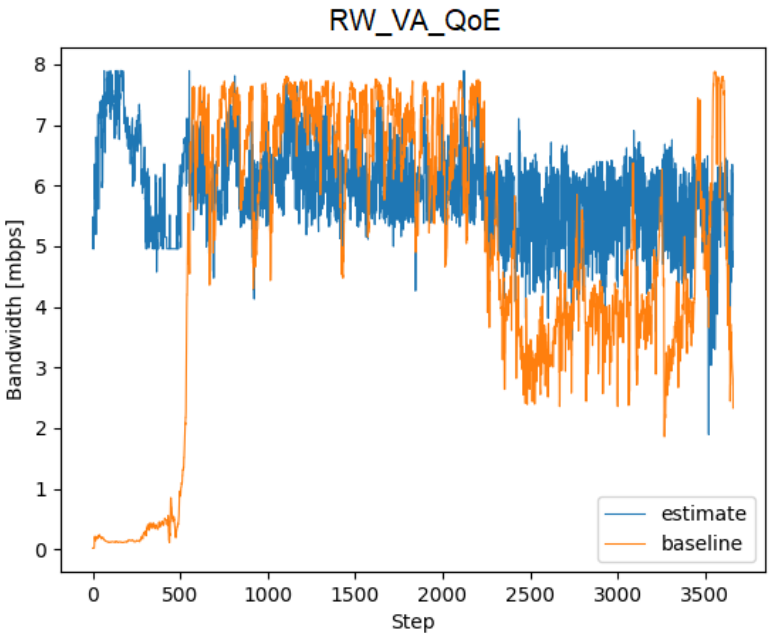


Figure B.2: BCQ using RW_VA_QoE and log file: 17000.json

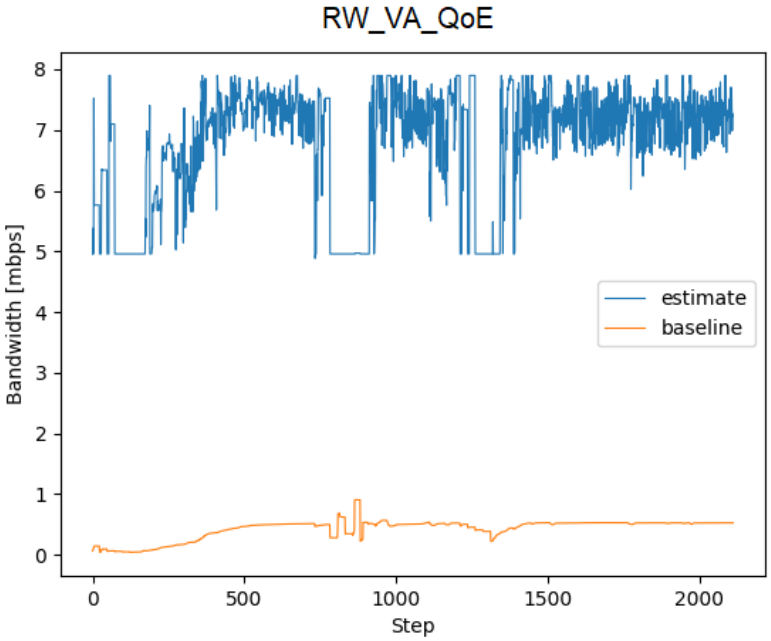


Figure B.3: BCQ using RW_VA_QoE and log file: 06030.json

B.2 Images of Figure 4.5

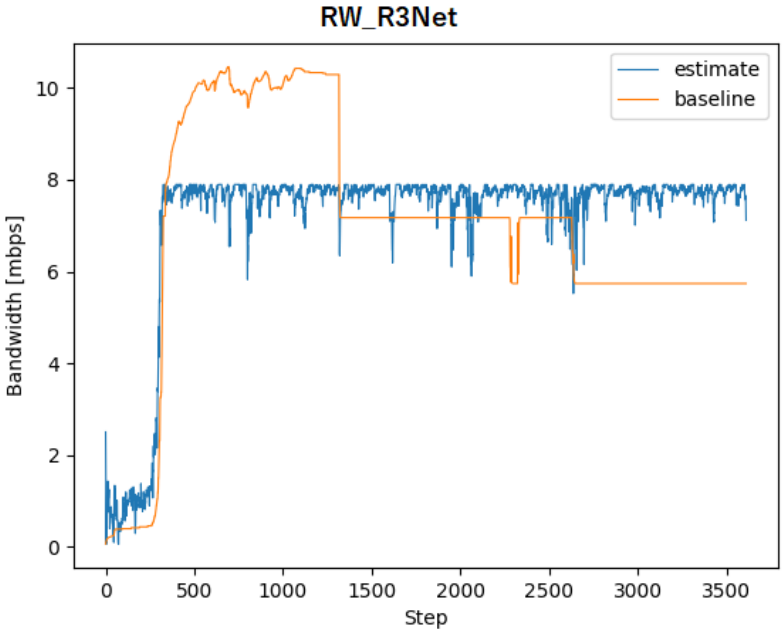


Figure B.4: BCQ using RW_R3Net and log file: 18849.json

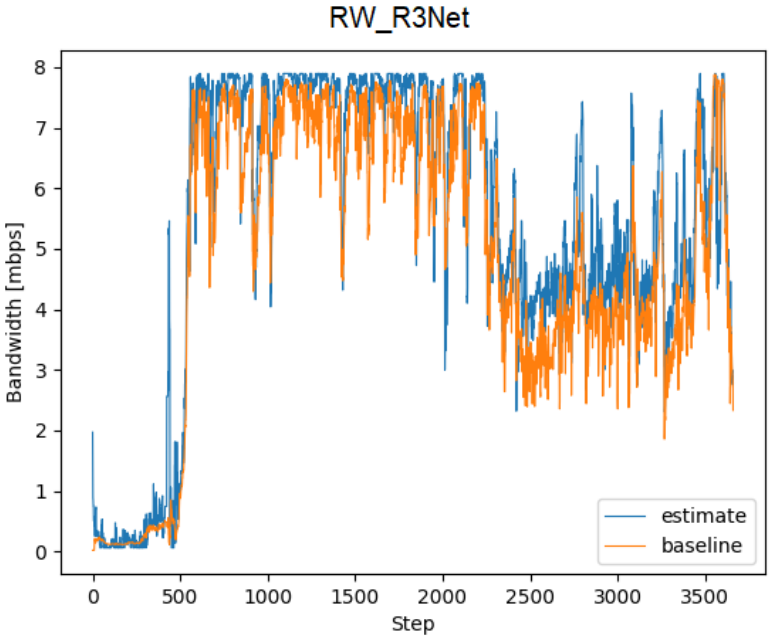


Figure B.5: BCQ using RW_R3Net and log file: 17000.json

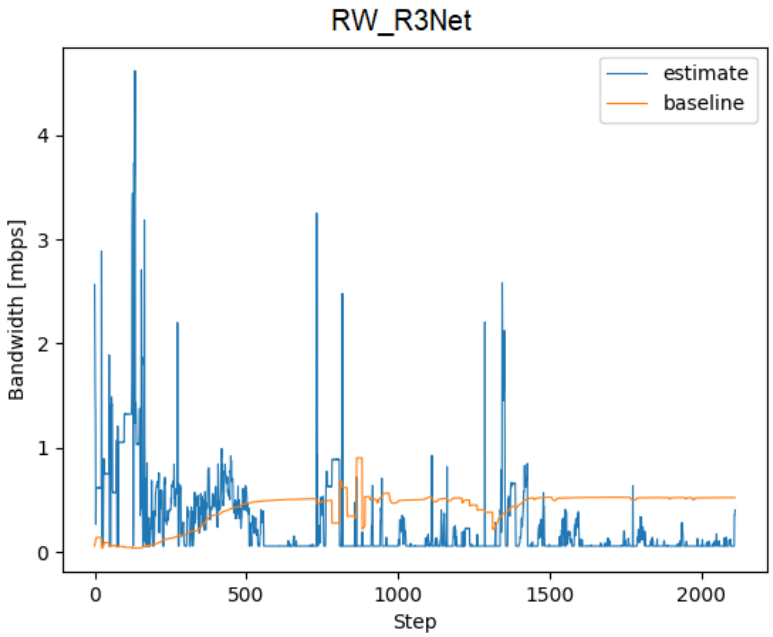


Figure B.6: BCQ using RW_R3Net and log file: 06030.json

B.3 Images of Figure 4.6

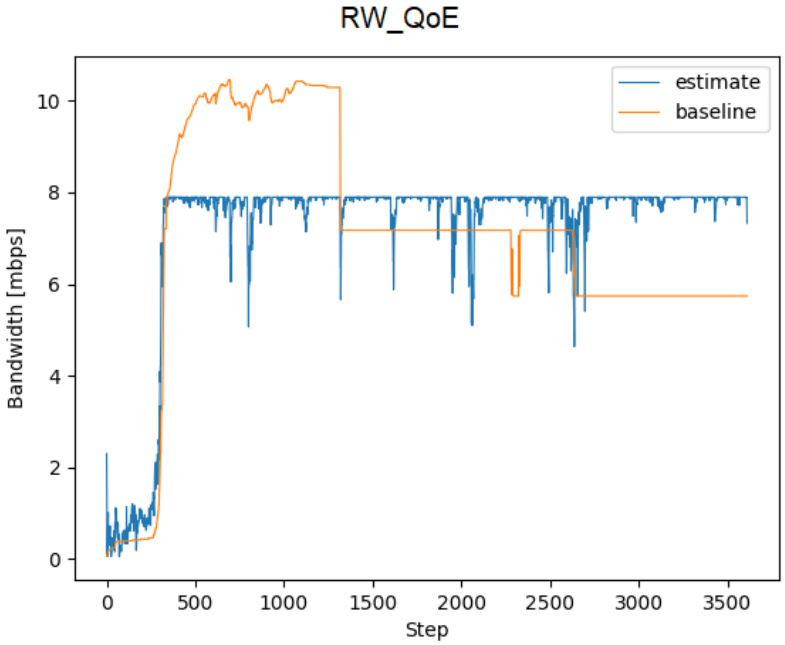


Figure B.7: BCQ using RW_QoE and log file: 18849.json

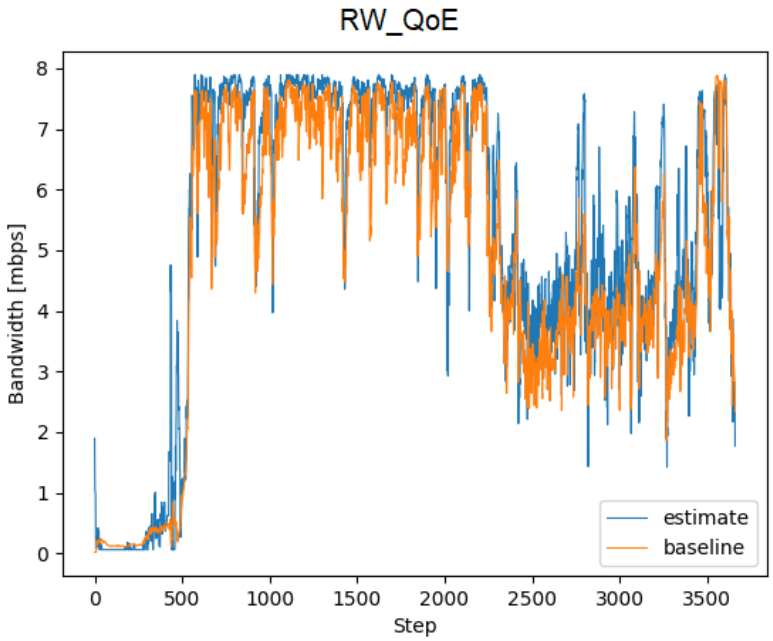


Figure B.8: BCQ using *RW_QoE* and log file: 17000.json

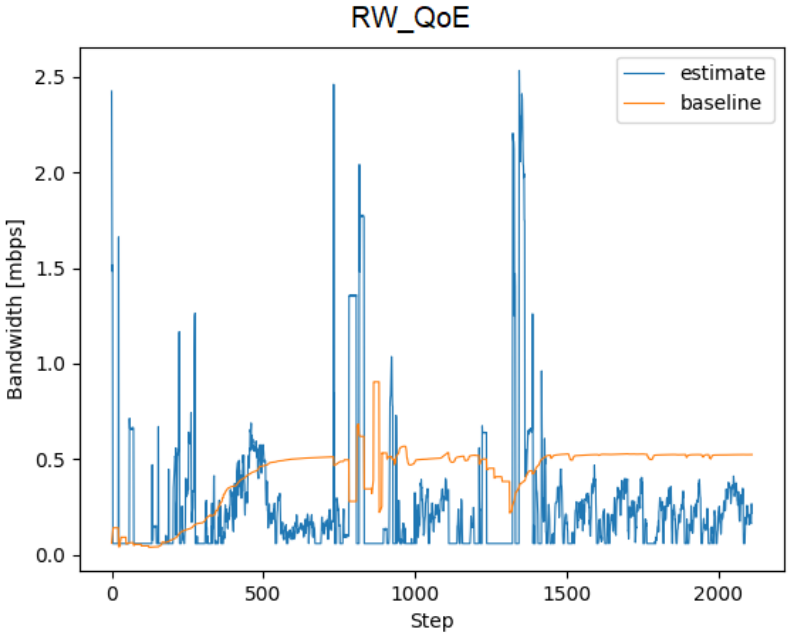


Figure B.9: BCQ using *RW_QoE* and log file: 06030.json

B.3.1 Images of Figure 4.10

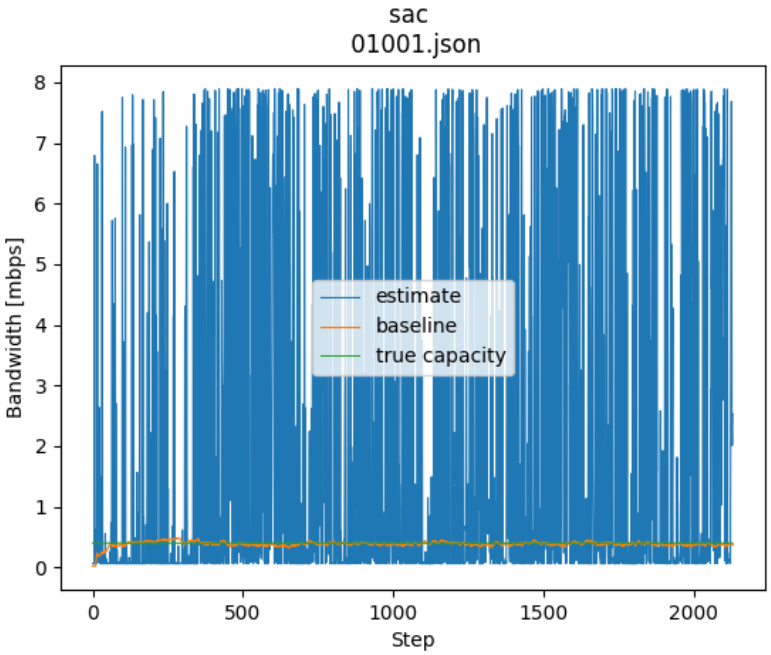


Figure B.10: SAC using *RW_QoE* and log file: 01001.json

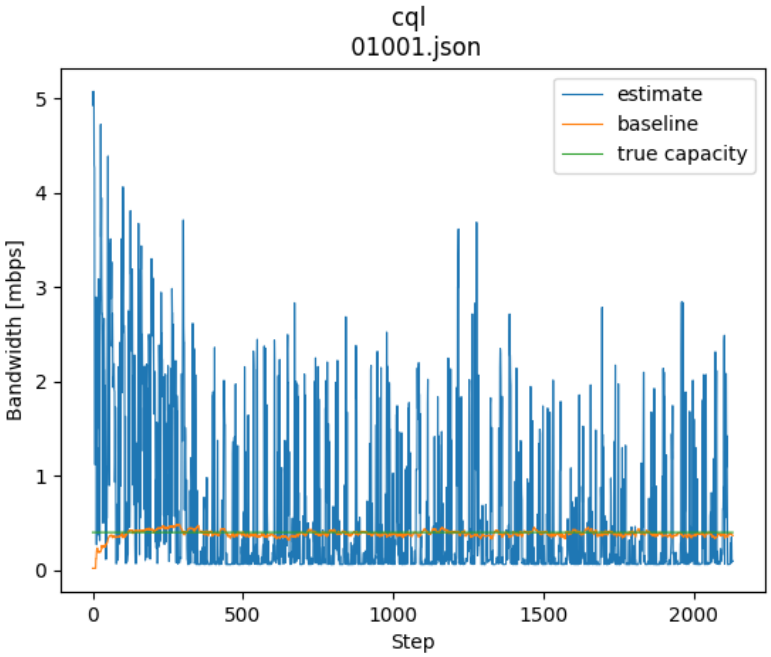


Figure B.11: CQL using *RW_QoE* and log file: 01001.json

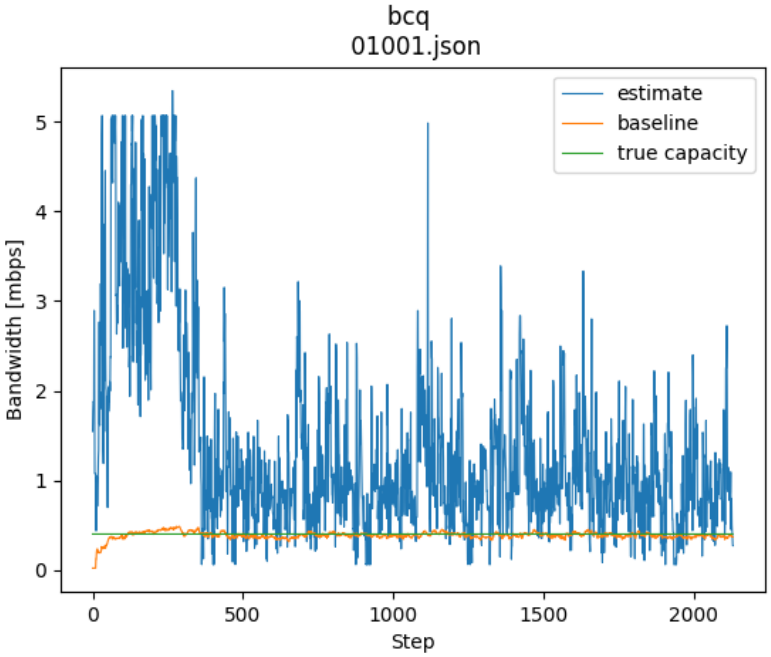


Figure B.12: BCQ using *RW_QoE* and log file: 01001.json

B.4 Images for Figure 4.11

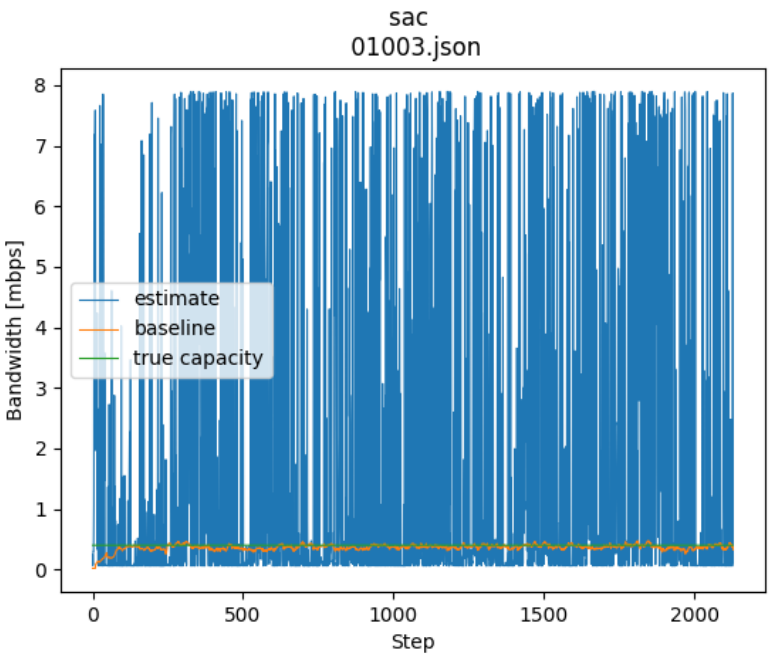


Figure B.13: SAC using *RW_QoE* and log file: 01003.json

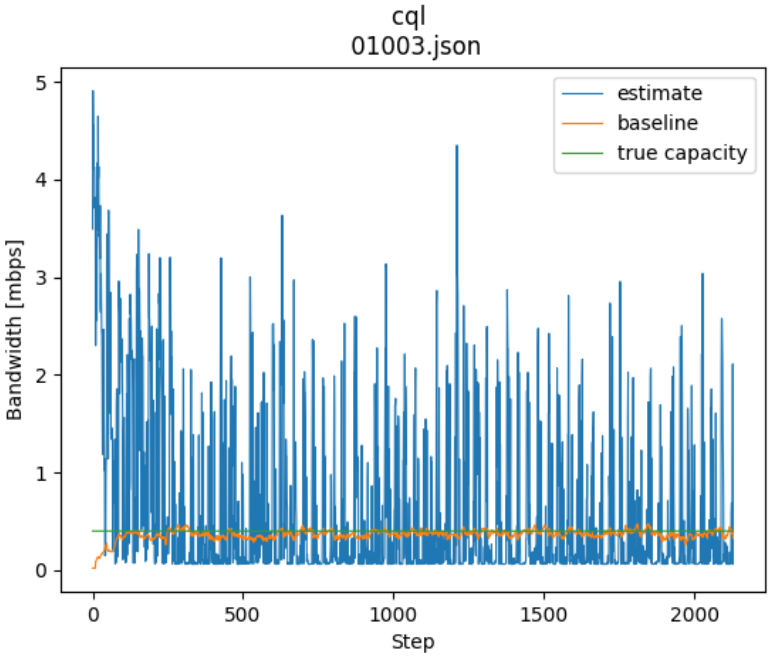


Figure B.14: CQL using *RW_QoE* and log file: 01003.json

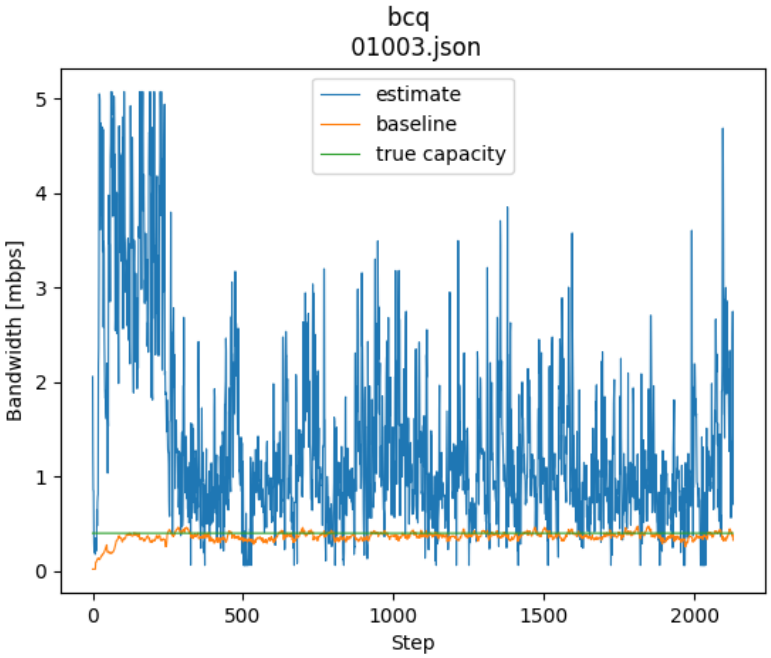


Figure B.15: BCQ using *RW_QoE* and log file: 01003.json