Master's Thesis

A Study on Node Selection Algorithm Based on Dueling Deep Q-Network Model in Unstable P2P Network Environments

SHAN GAO
Program of Information Science and Engineering
Graduate School of Science and Technology
Nara Institute of Science and Technology

Supervisor: Prof. Professor Hajimu IIDA
(Division of Information Science)

Submitted on January 30, 2024

A Master's Thesis
submitted to Graduate School of Science and Technology,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
MASTER of ENGINEERING

SHAN GAO

A Study on Node Selection Algorithm Based on Dueling Deep Q-Network Model in Unstable P2P Network Environments[*]

SHAN GAO

Abstract

In recent years, Peer-to-Peer (P2P) file sharing has become a mainstream method for file sharing and is widely used over Wide Area Networks (WAN), with systems like BitTorrent being notable examples. However, rapid and unstable changes in WAN often result in underutilization of resource nodes. To address this issue, this study proposed a modified node selection algorithm based on a dueling Deep Q-Network (DQN) model, which considers real-time states of resource node utilization. The central server of the hybrid P2P file sharing system maintains node information and collects real-time states such as CPU, memory utilization, and network latency. The modified dueling DQN model ranks nodes by estimated download time, and data is downloaded from the node with the minimum download time. The results demonstrate that the proposed dueling DQN model reduces the download time by 20% compared to the random selection algorithm, and achieves almost equal download time with Tit-for-Tat (TFT) algorithm, while the proposed algorithm can adapt to the network environment faster than the TFT algorithm when the network latency fluctuates significantly.

Keywords:

Peer-to-peer, Node selection, Reinforcement learning, Dueling-DQN

[*]Master's Thesis, Graduate School of Science and Technology, Nara Institute of Science and Technology, January 30, 2024.

Contents

List of Figures

List of Tables

1. Introduction

In today's digital era, file sharing has become an important part of daily life. Peer-to-Peer (P2P) [1] file sharing technology, recognized as an important method of data exchange, has had a far-reaching impact on the development of the Internet. This technology facilitates the sharing of files directly between individuals' computers, bypassing traditional centralized servers. As a result, it has become a backbone for a variety of Internet applications, ranging from media distribution to software updates.

Originally, this technology was designed to mitigate the challenges of bandwidth and storage limitations faced by centralized servers, particularly in handling massive data volumes. The emergence of Napster [2], one of the earliest P2P file sharing services, marked the birth of this technology. Subsequently, numerous other platforms such as BitTorrent [3] have emerged, each contributing to the ongoing evolution and enhancement of P2P technology.

The core of P2P file sharing technology lies in a decentralized network architecture. In this architecture, each participating network node (Peer) functions both as a client and a server. This means that each node can exchange files directly with other nodes without going through a central server. This approach not only improves the efficiency of data transfer but also increases the scalability and fault tolerance of the network, making it robust against various network issues.

Over time, P2P technology has expanded from its initial use as a file sharing service into a variety of domains. For example, it plays an important role in areas such as digital currencies (e.g., Bitcoin [4]), online video streaming [5], and distributed computing [6]. In the context of digital currencies, for instance, P2P technology provides the backbone of the blockchain network, ensuring secure and decentralized transaction logging. The decentralized nature of P2P networks provides greater security and operational efficiency across these diverse applications.

One critical consideration in all these various P2P applications is the data sharing strategy, particularly in selecting which nodes to engage with for optimal file sharing. For example, in BitTorrent technology, choosing which peers to share files with is crucial. But how do we select nodes fairly while ensuring the highest download efficiency and resource utilization [7]? A robust node selection algorithm is essential. Imagine if we had something similar to AlphaGo [8], a powerful intelligent system, to assist in choosing the optimal nodes. Such an advancement could significantly enhance the performance of many P2P applications, not just those limited to file sharing.

In order to address the above practical problems, this study focuses on optimizing the node selection algorithm for hybrid P2P structure [9] file sharing systems, which is widely used in BitTorrent download technology. The hybrid P2P architecture combines the advantages of both centralized and decentralized models, providing a flexible, high-speed, reliable, and secure solution for file sharing. With its flexibility and scalability, it can handle large-scale file sharing demands. In a hybrid P2P file sharing system, the node selection algorithm determines the efficiency and performance of the file transfer process.

The most commonly used node selection algorithm in these systems is Tit-for-Tat (TFT) [10], which selects nodes based on their past interactions. For example, if a node has transmitted the most data to a client in the past 20 seconds, the client will prefer to share data with that node (neighbor node) in the next interaction. However, TFT may not be effective when network conditions change rapidly. If file upload efficiency from a node suddenly drops due to network congestion before selecting that node for data sharing in the next selection, the node should avoid being selected as a neighbor node for data sharing.

To address these limitations, this study proposes a neighbor selection algorithm based on a modified dueling Deep Q-Network (DQN) model [11]. The model is deployed on a central server of a hybrid P2P file sharing network and continuously gathers data from the nodes in real-time, such as CPU status, memory usage, and network latency. It then evaluates and scores each node, guiding the decision-making process for selecting neighbor nodes for file sharing. The effectiveness of this proposed algorithm was evaluated against the traditional random selection approach and the TFT algorithm.

The remainder of this paper is organized as follows: The related work studied by past scholars will be introduced in Chapter 2. Then, in Chapter 3, the methods used in the study will be presented, including a detailed description of the model used, the setup of the P2P environment, and how the reinforcement learning model learns. Chapter 4 will present the evaluation and results of the experiments, such as comparing the download times and low-latency node selection rates between the TFT selection algorithm and a random selection algorithm. Chapter 5 will discuss in detail the results presented in Chapter 4 and analyze possible reasons. Chapter 6 will summarize the achievements of this experiment.

2. Related work

In recent years, a large number of researchers have conducted extensive studies to improve the P2P file sharing experience. How to enhance node resource utilization through efficient neighbor selection is a critical issue. The first section of this chapter will primarily focus on research related to the topology of P2P network nodes, employing traditional methods to improve resource utilization. The second section will delve into studies that utilize cutting-edge AI models to investigate the relationships between nodes and further improve resource utilization. The final section will outline the main focus of this study.

2.1 Traditional Network Topology Research Methods

In 2004, Vivek and Paul [12] from Cornell University explore random graph construction and node selection in P2P and overlay networks. They propose various techniques, focusing on random walks, and assess them based on practical criteria like simplicity, heterogeneity support, selection quality, efficiency, scalability, load balance, and robustness. The paper concludes that no single method excels in all criteria, but their novel approach generally performs best. Limitations include potential ineffectiveness in specific scenarios, additional complexity in handling network dynamics, and limited adaptability to different network environments. Further practical testing and optimization are suggested.

In 2005, the study by Adler et al [13]. explores optimal peer selection in P2P networks for downloading and streaming, introducing a pricing model where server peers charge based on download specifics. For downloading, it optimizes to minimize delay within a budget, also discussing Nash equilibrium for server pricing. In streaming, the focus is on minimizing costs while considering potential server failures. The methodologies developed are versatile for various P2P resource economy scenarios. However, the study assumes guaranteed delivery rates and overlooks the practicalities of implementation in detailed network models. It also simplifies streaming by assuming no initial client buffering, which may not align with all real-world applications.

In 2006, Vivek and Paul explore random graph construction and node selection in unstructured P2P networks, with a focus on accommodating node heterogeneity [14]. Their work introduces and evaluates various techniques, both novel and modified existing ones, against practical deployment criteria like simplicity, heterogeneity support, and load balance. The study highlights SwapLinks, a new graph construction method, for its practicality and simplicity, making it a standout choice. Through simulations, the authors demonstrate the effectiveness of their approaches, particularly SwapLinks, in creating efficient, scalable, and robust P2P networks. However, challenges remain in adapting these methods to highly dynamic network conditions and in managing the overhead associated with maintaining network structure amidst frequent node changes. This study offers valuable insights into designing versatile and practical solutions for unstructured P2P network applications.

Basing on 2-dimensional virtual network coordinates, Duan [15] proposed a novel scheme to improve the proximity property of structured P2P network in 2006. The results showed the proposed scheme reduced the latency between neighboring nodes and added only a modest additional search overhead. However, Duan's scheme performs well only when there are large scale nodes.

In 2008, Steele [16] present a novel load balancing mechanism for P2P networks, eliminating the need for global capacity knowledge by autonomously computing selection parameters at each peer. This parameter-free approach efficiently manages heterogeneous peer selection, crucial for maintaining balance in heavily loaded networks. The method adaptively adjusts the number of request attempts based on system load, ensuring high request-response rates even under heavy load. Implemented over the Swaplinks algorithm, their solution demonstrates significant improvements in load distribution and system utilization. While effective in tested scenarios, the method's performance may vary in different network conditions, and its computational demands could be challenging in larger or more dynamic networks. This research contributes significantly to the development of more autonomous and resilient P2P systems, particularly in managing load balance and resource allocation.

In 2009, Salhi [17], Sbai, and Barakat explore neighborhood selection in mobile P2P networks, particularly examining BitTorrent's performance in MANETs. Their study reveals that node mobility inherently enhances piece diversity, making it unnecessary to connect with distant nodes for this purpose. This contrasts with fixed networks, where diversification efforts are essential. Conducted through NS-2 simulations, the research shows that limiting neighborhood scope in mobile networks leads to better performance, as mobility reduces the need for additional diversification strategies. The findings indicate that in mobile environments, focusing on nearby peers is more efficient, as mobility itself facilitates sufficient piece diversity and sharing opportunities. This insight is crucial for optimizing file sharing applications in mobile networks, suggesting a shift from traditional approaches used in fixed networks. However, the study's applicability might be limited to specific network conditions and primarily to the BitTorrent protocol.

The Adaptive Gnutella Protocol (AGP) [18] proposed by Pogkas, Kriakov, Chen, and Delis, targets two critical issues in P2P networks: efficient information discovery and authentic data acquisition. AGP evaluates peers based on their service contributions, clustering them by reputation and shared content. This clustering forms a network topology centered around collaborative, reputable nodes. The protocol effectively identifies and sidelines harmful peers and free-riders, enhancing the speed and quality of information discovery and search results. AGP integrates a reputation system, content exchange, and topology adaptation modules, tested using PeerSim simulations. These tests showed notable improvements in network structure, query efficiency, and search quality. However, the protocol's complexity and the need for ongoing peer evaluation might pose challenges in larger, more dynamic networks. AGP represents a significant step in optimizing P2P file sharing by fostering a network of trusted, cooperative participants.

In 2018, Gui [19] used greedy algorithm to translate the overall optimization into multiple local optimal problems, and to quickly select service nodes. The final simulation results show that the node selection strategy based on Gui's greedy algorithm can effectively improve the overall performance of P2P streaming media system. However, the decision factors of the node selection strategy of this research are not comprehensive enough, for example, node bandwidth should also be taken into account.

In 2024, a novel approach(MSLT) [20] for enhancing transaction performance in blockchain P2P networks through multi-scale node management was proposed. This method organizes nodes into different scales, reducing redundancy and optimizing traffic flow. MSLT employs a node updating mechanism based on transmission speed, prioritizing faster nodes to improve overall data transmission rates. The model's hierarchical structure facilitates efficient data propagation across the network, with transactions and blocks transmitted to neighbor nodes at suitable scales. Comparative analysis indicates that MSLT outperforms existing models in terms of transmission efficiency, network utilization, and transaction throughput. However, the complexity of the model and the need for dynamic node management pose challenges for network maintenance and scalability. Further, the model's reliance on multi-scale management necessitates robust security measures to ensure network stability and protect against potential disruptions or attacks.

2.2 Novel AI Modeling Research Methods

Koo [21] proposed a GA-based neighbor-selection strategy for hybrid P2P system suitable for large content delivery in 2006, and they have shown through computer simulations that their proposed strategy can increase the content availability from immediate neighbors and thus improve system throughput significantly without sacrificing delivery efficiency. However, the experiment assumes that all nodes honestly return the amount of data they receive. Therefore, it may not perform as expected in a real-world environment.

In 2007, Robert [22] employ machine learning feature selection in a novel manner: to reduce communication cost thereby providing the basis of an efficient neighbor selection scheme for P2P overlays. However, the experiment only used open source datasets for training and testing and there is no confirmation of the performance of this strategy under real environment operation.

In 2012, Izhak-Ratzin [23] proposed a BitTorrent-like protocol based on an online learning (reinforcement learning) mechanism, which can replace the peer selection mechanisms in the regular BitTorrent protocol. The results show that the proposed protocol provides several improvements in terms of the stability of the peer selection mechanism, collaboration among high capacity peers, system fairness, the robustness of the network against noncooperative behaviors, and downloading rates. However, the research did not investigate the robustness of the reinforcement learning strategy-based system.

Naito [24] proposed a method to acquire a nearly optimal peer selection strategy from long-term observations through Deep Q-Network which is a variant of the Q-learning enhanced by deep neural networks. The result indicates that it realize a short download time compared with strategies adopted in BitTorrent. But the environment of this experiment is not realistic enough, for example, the bandwidth of nodes should be different and the delay between nodes should exist.

2.3 Focus of the Proposed Method

Studies based on traditional network topology methods have overly focused on the impact of network structure on node utilization, neglecting the actual states of nodes, such as network latency, CPU, and memory utilization. This oversight leads to resource underutilization in constantly changing networks.

Studies employing AI modeling methods aim to predict node behavior, potentially maximizing resource utilization in most scenarios. However, most of these studies rely on simulations and have not been tested in real-world scenarios. Additionally, some experiments fail to consider the latency between nodes in real environments.

This study is based on a hybrid P2P network structure and involves real-time collection of data such as network latency, CPU, and memory usage from each node. Using AI models, it predicts the current state of each node to maximize the selection rate of low-latency nodes, ultimately achieving maximal resource utilization of nodes.

3. Method

This chapter describes the methodology adopted in this research, which is organized into four sections. First, an overview of the proposed approach is introduced. Then, the design of the model, including its architectural choices and underlying rationale, is elaborated. Subsequently, the training process of the model is described. Finally, the implementation details and experimental environment are provided, explaining the hardware and software configurations and tools used.

3.1 Approach

Figure 1 presents an overview of the proposed system. The proposed dueling-DQN model operates on a central server, which compiles real-time data regarding the status of peer-to-peer (P2P) nodes, including CPU utilization, memory usage, and network latency. When a node initiates a download request, the server feeds the model real-time data from all nodes and outputs the ID of the optimal node for resource sharing to the requesting node. The requesting node receives this ID and directly exchanges data with the selected node in a peer-to-peer manner.

This research modifies the original dueling-DQN model to develop a system that identifies the optimal node for data exchange, utilizing information aggregated by the central server upon the request of any node. By incorporating dynamic parameters, such as CPU utilization, memory usage, and network latency, into the decision-making process, the proposed model aims to optimize resource distribution and improve the efficiency of P2P file sharing.

To generate a dataset for training the model, a virtual environment was constructed to simulate P2P file sharing scenarios. Within this controlled setting, a dataset reflecting various network conditions and node states was generated. This dataset was subsequently used to train the modified dueling-DQN model. Utilizing the virtual environment and the generated dataset allows for a comprehensive assessment of the model's effectiveness in selecting the optimal node for data transfer. Such an approach ensures efficient resource utilization and minimizes latency within the network.

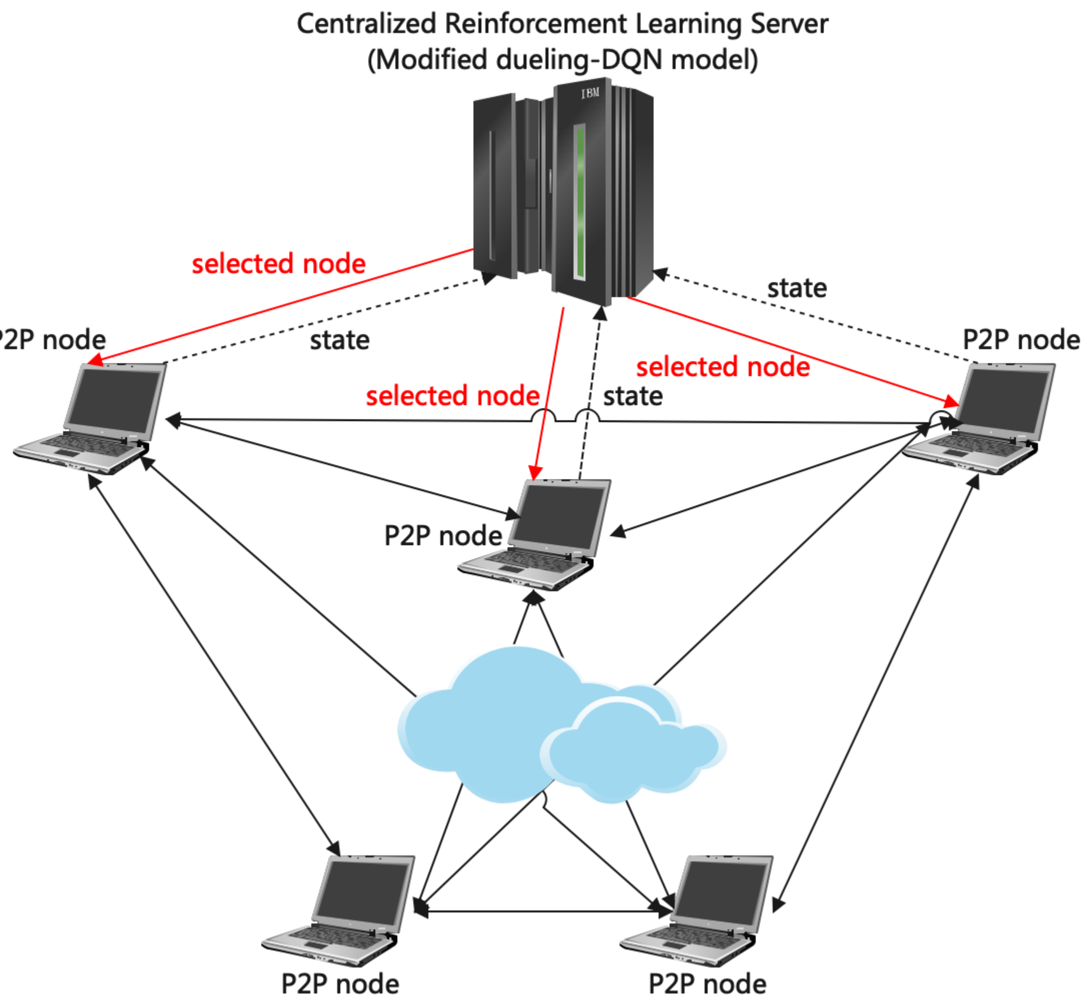

Figure 1. The proposed P2P system

3.2 Model Design

To introduce the proposed model, it is necessary to first understand Reinforcement Learning(RL) [25]. RL is a type of machine learning where an agent learns to make decisions by performing actions in an environment to achieve some goal. The agent receives feedback in the form of rewards or penalties as it interacts with the environment. The objective of the agent is to learn a strategy, or policy, that maximizes the cumulative reward over time.

DQN [26] is one of the foundational methods of RL and it combines traditional Q-Learning, a popular RL algorithm, with deep neural networks. Dueling-

DQN is an extension of the DQN algorithm and the main differences between dueling-DQN and traditional DQN lie in their network structures and methods of estimating the value function, as shown in Figure 2 [11]. Dueling-DQN architecture splits the network into two branches, one estimating the state value and the other estimating the advantage function of actions to indirectly estimate the Q-value of each action.

Both branches, the value network and the advantage network, share the initial layers, which extract features from the input states. However, they diverge in later layers to estimate the state value and the advantage for each action separately. The value network is responsible for predicting a value function, which represents the expected reward from a given state. This facilitates an understanding of the intrinsic value of being in a particular state, independent of the actions taken. The advantage network, on the other hand, estimates the relative benefit of each action by computing the difference between the predicted values and the state values. It enables the users or clients to evaluate the importance of each action in a given state. The output of the state value and the advantage value of the action are combined to obtain the Q-values for each action.

This architecture enables dueling-DQN to often learn more efficiently and with greater stability across various environments compared to traditional DQN, and it performs better in scenarios where state evaluation is more critical than action selection.

The proposed modified dueling-DQN borrows ideas from the original dueling-DQN in terms of state value and action advantage value structure, as shown in Figure 3. However, while the original dueling-DQN was primarily used for gaming applications, utilizing three channels for better game screen recognition, the proposed P2P node selection scenario requires only one input matrix; thus, a single channel was used. At the same time, we adjusted the input and output of the model to the corresponding size for our experimental use case.

The states of all nodes for the last 20 seconds are fed into the proposed model in real time, and the proposed model outputs a score for each node to measure the current state of each node. When only 3 P2P nodes are present, the input state matrix is shown in Fig 4. The first row of each node's data is the time-series data of the node's network latency, the second row is the time-series data of the

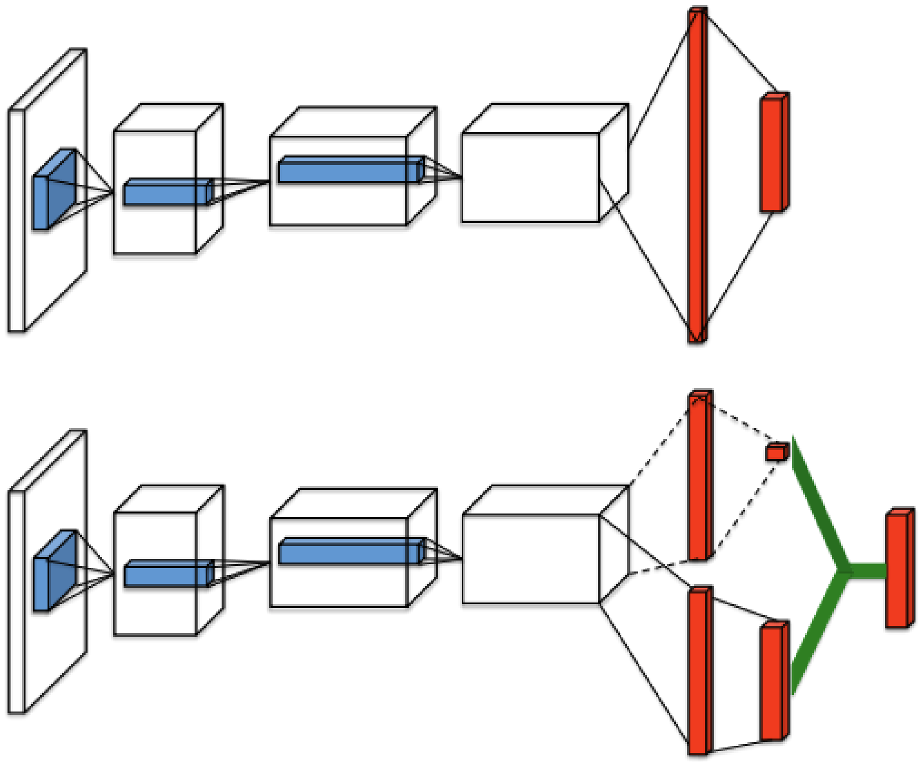

Figure 2. Structural differences between single stream Q-network (top) and dueling Q-network (bottom)

node's CPU utilization, and the third row is the time-series data of the node's memory utilization. The number of columns in the input matrix is 10, indicating that the proposed model takes the state of these nodes for the last 10 seconds. In other words, when 20 nodes are present and take the state of the last 10 seconds, the data has 20 x 3 rows and 10 columns. Then the size of the input state matrix is 60 × 10.

In the proposed model, first layer uses a 3×3 kernel to move 1 unit in the column direction and 3 units in the row direction to reduce the input state matrix from 60×10 to 20×8. The second layer uses only a 1×3 kernel to move 1 unit in both the row and column directions to reduce the matrix size to 20×6. It then expands into a fully-connected layer, which immediately splits into two parts: the first, colored green, representing the state values, and the second, colored blue,

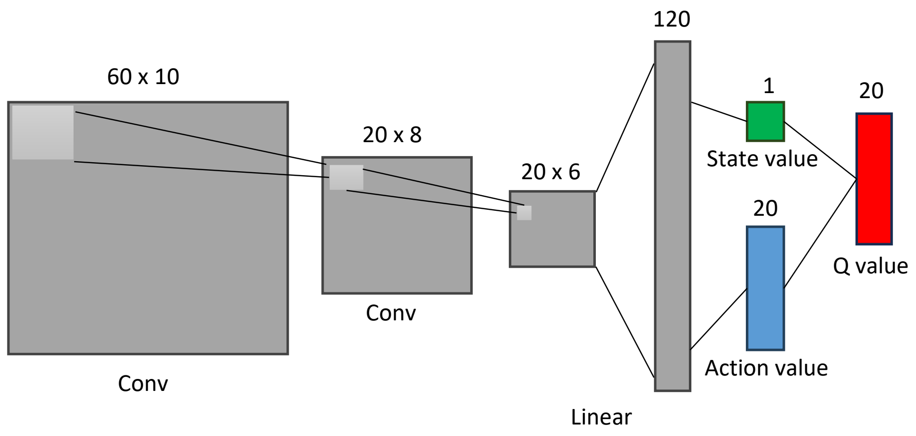

Figure 3. The proposed modified dueling-DQN structure

representing the action advantage values. Finally, we output the Q-value based on the combination of the state value and the action advantage value in the last layer, which can also be termed as the reward.

In order to explain the calculation of the Q-value, the notation used throughout this study will be introduced. The symbols shown in Table 1 represent the variables involved in the Q-value calculation. The action represents which node has been selected. The state denotes the current state of the P2P environment. The value function is used to evaluate the current P2P environment. The ac-

Table 1. The symbols and notations used in the paper

Symbols	Description
a	Action
s	State
V	Value function
A	Action advantage function
Q	Q-value function
ϵ	Epsilon

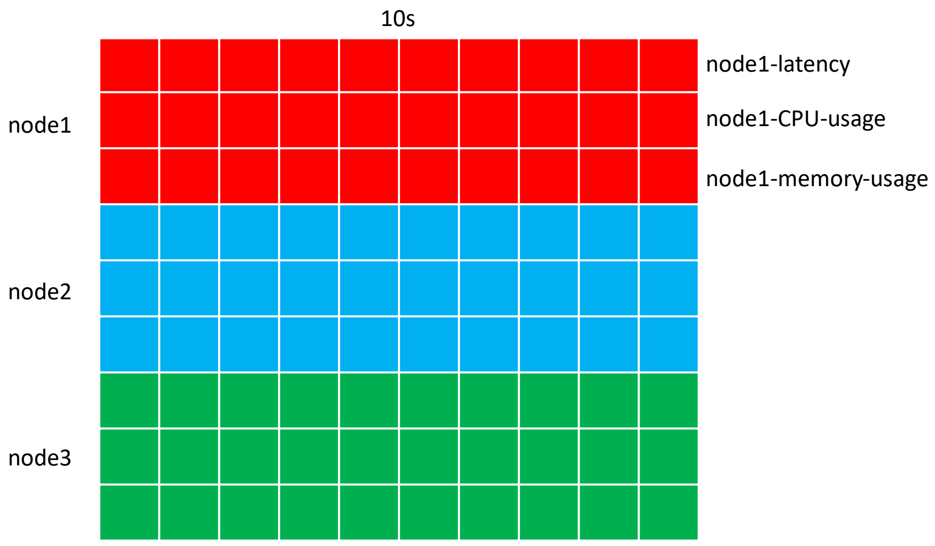

Figure 4. A sample input state matrix when using 3 P2P nodes

tion advantage function is used to assess the expected reward from selecting each node. The Q-value function is used to combine the value function and action advantage function and give the estimated Q-value. Epsilon is used to determine whether the selection strategy follows the model's decision or a random strategy. For example, when epsilon is set to 0.9, nine out of ten experiments will select nodes based on the decision of the reinforcement learning model. One out of ten experiments will select a node at random.

The formula for calculating the Q-value of the dueling-DQN neural network is given in Equation 1 below. In this equation, $V(s)$ is a function that calculates the value of state s and represents the expected reward in state s. $A(s, a)$ is the action advantage value function of the state-action (node selection) pair s, a and represents the value of action a in state s. $\frac{1}{N}\sum_{a'} A(s, a')$ is the average value of all possible actions and is used to reduce the bias of overestimating or underestimating the value of the action. $Q(s, a)$ denotes the Q-value for action a in state s by combines calculated $V(s)$ and $A(s, a)$ and $\frac{1}{N}\sum_{a'} A(s, a')$.

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{N}\sum_{a'} A(s, a') \tag{1}$$

3.3 Training Process

The model is trained in a P2P network scenario consisting of 20 nodes, partially subjected to a latency of 50ms. The selection of a 50ms delay as the evaluation condition is rooted in its representation of typical average latency within a single continent, providing a realistic benchmark for network performance. Although the network size of 20 nodes might seem modest compared to real-world networks, it was intentionally chosen to enable simulations within the constraints of limited computational resources. The chosen network size is considered optimal for initial evaluations, offering a practical compromise between manageability and the complexity needed to explore the dynamics and interactions characteristic of P2P networks. This allows for a detailed analysis and comparison of the proposed model with baseline algorithms, such as the random selection and TFT algorithms, without the extensive computational demand of large-scale network simulations.

The model training process is described below, as shown in Process 1. A file of 256MB size is first placed at each node. These files are manually assigned to avoid resource overlap. Then, the reinforcement learning server and all the P2P nodes are launched. The P2P nodes form their own network and inform the reinforcement learning server of their real-time state, while each node spontaneously requests to download these files, deletes the downloaded files after download, and repeats the request. Delays of 50ms are randomly added to half of the nodes using the tc command, and then the number of times the file has been downloaded after this delay is added is counted. After every 1,000 downloads, the added delays are removed, and delays of 50ms are randomly added again to half of the nodes.

Five trials of the experiment was conducted, in which the node repeatedly downloads the file 10,000 times. For each download, the current state of all nodes in the P2P environment and the node ID selected for the current download, as

Process 1 Model Training Process

```
1: Start RL server
2: loop
3:     Measure the latency of each node with ping protocol
4:     if Receive a download record from a node then
5:         Store the download record
6:         Update the download count of the file
7:         if Download count reaches 101 then
8:             Use the last 100 records to train the model
9:         end if
10:        if Download count reaches 1000 then
11:            Reallocate network latency to half of the nodes
12:        end if
13:        if Download count reaches 5000 then
14:            Update epsilon from 0.5 to 0.9
15:        end if
16:    else
17:        Receive and store the CPU and memory usage from a node
18:    end if
19: end loop
```

Process 2 Node Request Process

```
1: Start all nodes
2: Wait for 11 seconds
3: loop
4:     Continuously send CPU and memory usage data to the RL server
5:     Request to download a file from the RL server
6:     Receive a node ID from the RL server
7:     Connect to the node with the received ID and start downloading the file
8:     After downloaded, send the selected node ID and download time to the
       RL server
9: end loop
```

well as the time spent downloading the file and the state of all nodes in the P2P environment after the download were recorded. It is important to note that the CPU and memory usage of these nodes as well as the download records are uploaded by the nodes themselves, as shown in Process 2.

The epsilon parameter of the reinforcement learning model is set to 0.5 for the first 5,000 file downloads, and then adjust the epsilon to 0.9 for the next 5,000 downloads. The epsilon represents the probability that the model's decision will be followed. Epsilon should not be set too high in the early training stage when the model has not yet converged, as this will make it more difficult for the model to converge.

The proposed model is trained with 100 data inputs after every 101 file downloads. This means that the model was trained 99 times when the file was downloaded 10,000 times, each time using 100 real-time data for training. The RMSprop [27] optimizer was used to update the neural network, and the learning rate was set to 0.01. The collected real-time data are assembled into a state matrix to train the model at each training session.

3.4 Implementation and Environment Setup

To realize the proposed system, the dueling-DQN model was developed using the PyTorch framework, an advanced open-source machine learning library, incorporating open-source resources from labml.ai [28], which is renowned for offering tools and libraries that facilitate research and development in artificial intelligence and machine learning. The central server was constructed utilizing the flask framework [29], which is distinguished by its lightweight architecture and its facilitation of seamless integration with web applications. P2P nodes were developed by employing the go-libp2p [30], an open-source library highly regarded for its comprehensive suite of networking and data transfer capabilities essential for the development of P2P-based decentralized applications. Additionally, a custom plugin was developed utilizing the gopsutil library [31] for the P2P nodes, enabling these nodes to monitor and report their real-time memory and CPU usage to the reinforcement learning server.

In this study, an experimental environment was constructed on a virtual machine cluster, and two experiments, which are described in later chapters, were

Table 2. Specifications of the Virtual Machine Cluster

Component	Specification
Virtualization Platform	VMware ESXi 6.7.0
Number of Nodes	5
CPU per Node	2 x Intel Xeon Silver 4208
Memory per Node	96GB
Network Interface Card	Intel Ethernet Controller 10G X550

Table 3. VM Configuration

Role	CPU cores	Memory	Amount
P2P node	2 Cores	4 GB	10 VMs
Reinforcement learning node	8 Cores	32 GB	1 VM

conducted. In the first experimental environment, two nodes were run under each VM, totaling 20 nodes, and in the second, a single node was run under each VM, totaling 10 nodes. Additionally, one VM was run to act as the central server. The hardware specifications of the virtual machine cluster and the configurations of VMs used in the experiments are shown in Table 2 and 3. The reinforcement learning service runs on an 8-core, 32GB VM.

4. Evaluation

In this chapter, the proposed model is evaluated in both 10-P2P-node and 20-P2P-node environments, with added partial delays of 10ms, 50ms, and 100ms. The model's performance in identifying low-latency nodes is compared with the random selection algorithm and the TFT algorithm in each scenario. The variation in the number of P2P nodes across these environments is designed to explore whether the model can accurately identify low-latency nodes when there are few candidate nodes. Additionally, introducing different delays aims to test whether the proposed model can maintain consistent performance under various network latency conditions.

4.1 Metrics of Evaluation

In the evaluation, each algorithm, including the proposed method, is evaluated based on two metrics. The first metric is the average download time, calculated by averaging the download times of all files in the evaluation scenarios. The second metric is the low-latency node selection rate, which measures the rate of low-latency nodes preferentially selected within the network environment. Since the evaluation environment introduces random delays to some network nodes, this metric evaluates how effectively each algorithm avoids nodes with large delays. These metrics facilitate the evaluation of the performance of each node selection algorithm.

The average download time (lower is better) is calculated as follows:

$$\bar{t} = \frac{1}{N} \sum_{i=1}^{N} t_i, \tag{2}$$

where N represents the total number of files and t_i represents the time taken to download the ith file. This formula sums up the download time for each round of file downloading to calculate the average download time.

The low-latency node selection rate for each round of file downloading is calculated as follows:

$$rate_i = \frac{\text{\# of selected low-latency nodes}}{\text{\# of all selected nodes}}. \tag{3}$$

The denominator represents the number of times each node is selected as a file sharing object, while the numerator represents the number of low-latency nodes selected up to the current round. Subsequently, the average selection rate of low-latency nodes (higher is better) is calculated with:

$$\overline{rate} = \frac{1}{N}\sum_{i=1}^{N} rate_i, \tag{4}$$

where N represents the total number of files.

4.2 Evaluation on a network of 20 nodes

The experiments in this section were conducted in a 20-node environment, and file download times and the selection rates of low-latency nodes were collected under conditions where half of the nodes had latency of 10ms, 50ms, and 100ms, which represent low, medium, and high latency network environments respectively. This allows testing the proposed model's ability to correctly identify low-latency nodes and increase resource utilization under various network latency conditions.

4.2.1 Evaluation with 10ms Latency Introduced

Figure 5 shows the time taken to transfer a single 256MB file in a P2P environment with 20 nodes, under a 10ms latency for half of the nodes, using three different strategies. The horizontal axis represents the number of downloads, and the vertical axis shows the time taken to download (less is better). The random selection strategy takes the longest time, but the proposed algorithm nearly achieves the same download time as TFT, although there is still a gap on average.

Figure 6 displays the rate of selecting low-latency nodes during the transfer of a single 256MB file in a P2P environment with 20 nodes, where half of the nodes have a 10ms latency, under three different strategies. The x-axis represents the number of rounds downloaded and the y-axis shows the cumulative percentage of the selection rate of the low latency node selection at that point. Since half of the nodes are low-latency, the random selection algorithm still hovers around 0.5. TFT can reach up to nearly 0.7 in its selection rate of selecting low-latency nodes, while the proposed model fluctuates between 0.6 and 0.7. The average

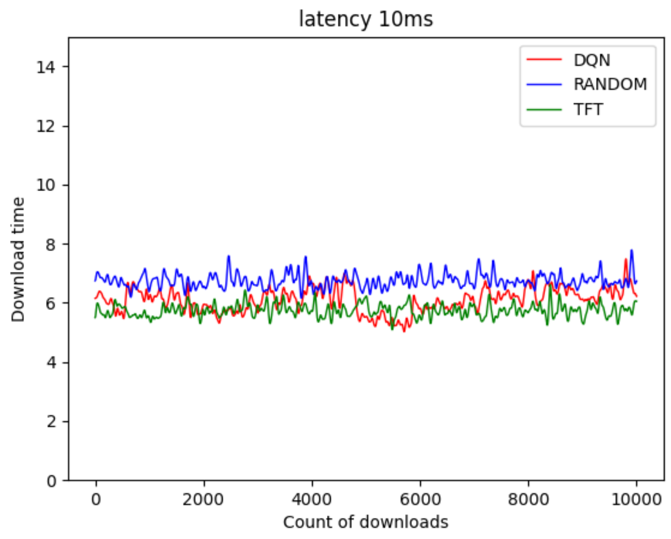

Figure 5. Results of three strategies for downloading a 256MB file with 20 nodes and half of the nodes with 10ms latency

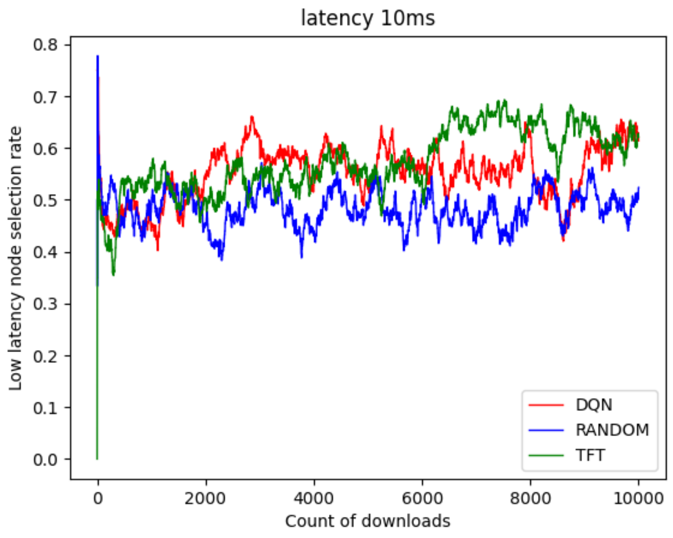

Figure 6. Rate of selecting a low-latency node in each round of downloads within a P2P environment of 20 nodes at 10ms latency for half of the nodes under three different strategies

Table 4. Download time and low latency node selection rate (# of nodes: 20, latency: 10ms)

Algorithm	Avg download time	Avg selection rate
TFT	5.74s	57%
DQN	6.09s	55%
RANDOM	6.74s	48%

download time and the average selection rate of low latency nodes are shown in Table 4.

4.2.2 Evaluation with 50ms Latency Introduced

The comparison of the download time of a 256MB file under a 50ms latency for half of the nodes is shown in Fig 7. The results demonstrate that our proposed DQN model reduces the download time by 20% compared to the random selection algorithm, is slightly slower than the TFT algorithm by 10% between 5,000 and 7,000 downloads, but achieves almost equal download time after 7,000 downloads.

A comparison of the three selection algorithms in terms of their ability to choose low-latency nodes in each round of download is also conducted. The results are shown in Fig 8. Since we set half of the nodes to have a delay of 50ms in our experiments, the selection rate of the random selection algorithm fluctuates around 0.5. And the average selection rate of low latency nodes of the TFT algorithm is around 0.8, while the average selection rate of low latency nodes of our proposed DQN model is around 0.7, as shown in Table 5. Since we set epsilon to 0.5 for the first 5,000 downloads, the performance of our DQN method is almost the same as random, with occasional fluctuations. However, when we adjusted the epsilon to 0.9 after the first 5,000 downloads, the selection rate of the selected low-latency nodes in our model showed an increasing trend.

4.2.3 Evaluation with 100ms Latency Introduced

Figure 9 shows the time taken to transfer a single 256MB file in a P2P environment with 20 nodes, where half of the nodes have a 100ms latency, under three

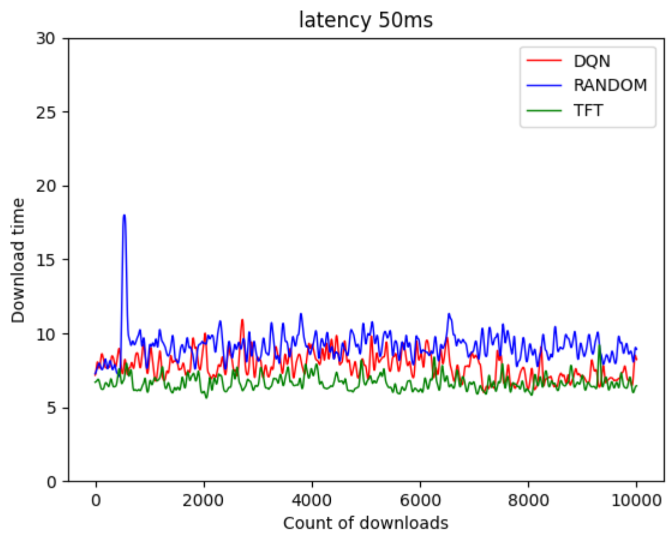

Figure 7. Results of three strategies for downloading a 256MB file with 20 nodes and half of the nodes with 50ms latency

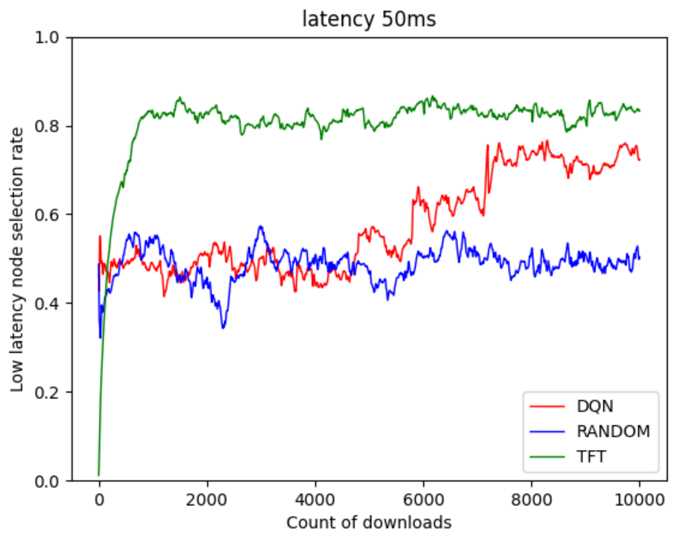

Figure 8. Rate of selecting a low-latency node in each round of downloads within a P2P environment of 20 nodes at 50ms latency for half of the nodes under three different strategies

Table 5. Download time and low latency node selection rate (# of nodes: 20, latency: 50ms)

Algorithm	Avg download time	Avg selection rate
TFT	6.65s	80%
DQN	7.78s	70%
RANDOM	9.16s	49%

Table 6. Download time and low latency node selection rate (# of nodes: 20, latency: 100ms)

Algorithm	Avg download time	Avg selection rate
TFT	11.46s	74%
DQN	12.53s	64%
RANDOM	14.66s	47%

different strategies. Although the random strategy takes the longest time, our proposed algorithm nearly achieves the same download time as TFT.

Figure 10 illustrates the selection rate of selecting low-latency nodes during the transfer of a single 256MB file in a P2P environment with 20 nodes, where half of the nodes have a 100ms latency, under three different strategies. Since half of the nodes are low-latency, the random selection algorithm still hovers around 0.5. TFT performs very well under high latency, almost reaching a selection rate of 0.8 for low-latency nodes. The proposed model also performs well under high latency, achieving a selection rate of 0.6-0.7 for low-latency nodes. Although it still falls short of TFT, it is better than the random selection algorithm. The average download time and the average selection rate of low latency nodes are shown in Table 6.

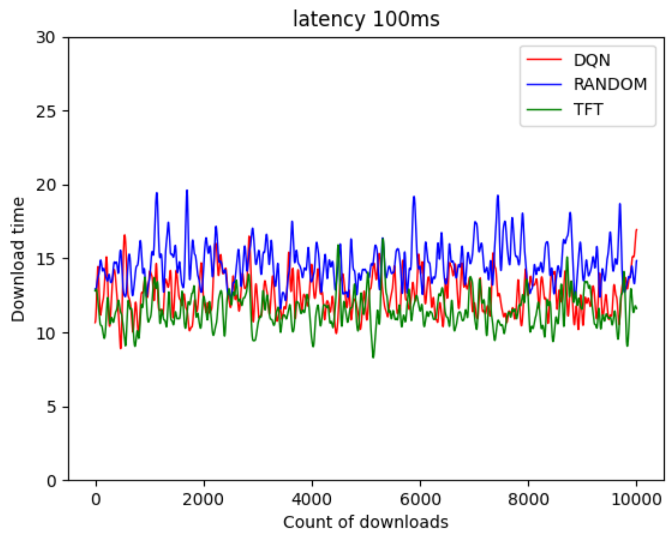

Figure 9. Results of three strategies for downloading a 256MB file with 20 nodes and half of the nodes with 100ms latency

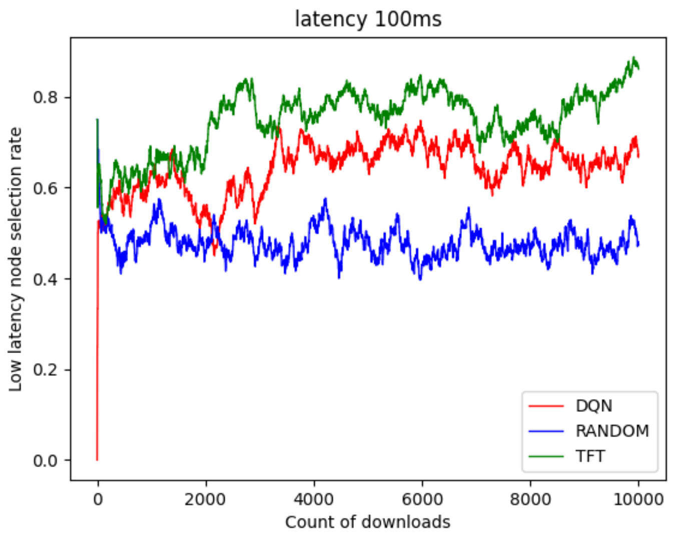

Figure 10. Rate of selecting a low-latency node in each round of downloads within a P2P environment of 20 nodes at 100ms latency for half of the nodes under three different strategies

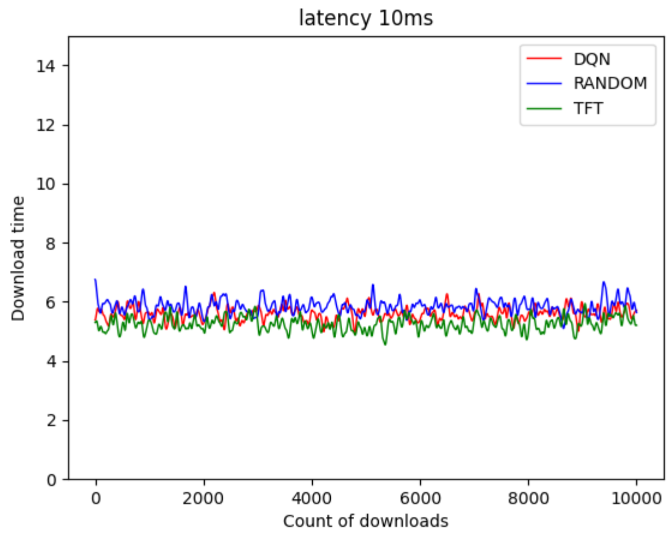

Figure 11. Results of three strategies for downloading a 256MB file with 10 nodes and half of the nodes with 10ms latency

4.3 Evaluation on a network of 10 nodes

The experiments in this section were conducted in a 10-node environment, and file download times and the selection rates of low-latency nodes were collected under conditions where half of the nodes had delays of 10ms, 50ms, and 100ms, respectively.

4.3.1 Evaluation with 10ms Latency Introduced

Figure 11 shows the time taken to transfer a single 256MB file in a P2P environment with 10 nodes, under a 10ms latency for half of the nodes, using three different strategies. Although the random strategy takes the longest time, the download times for nearly all three strategies hover around 5-6 seconds. This might be because the 10ms latency between nodes has almost no impact on the network.

Figure 12 shows the selection rate of selecting a low latency node for each round of downloads. the TFT algorithm has the highest selection rate of low latency nodes, the random selection algorithms and proposed model fluctuate

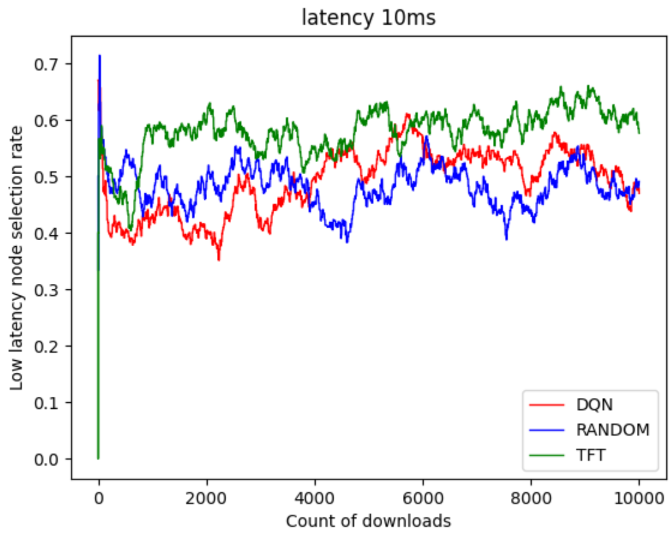

Figure 12. Rate of selecting a low-latency node in each round of downloads within a P2P environment of 10 nodes at 10ms latency for half of the nodes under three different strategies

Table 7. Download time and low latency node selection rate (# of nodes: 10, latency: 10ms)

Algorithm	Avg download time	Avg selection rate
TFT	5.24s	57%
DQN	5.59s	49%
RANDOM	5.84s	48%

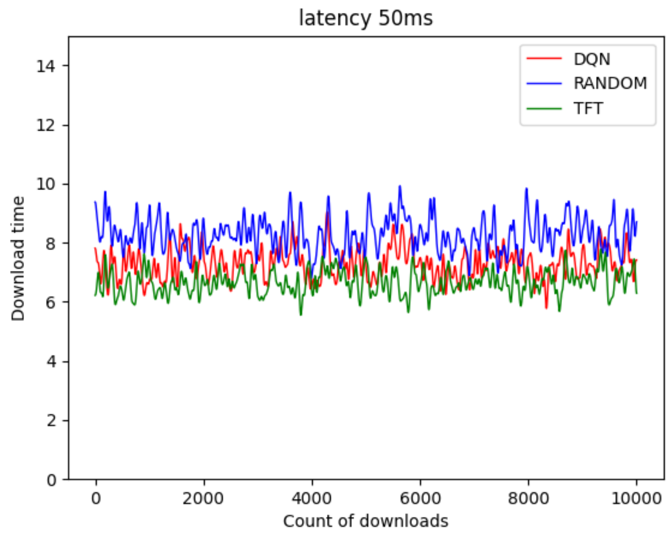

Figure 13. Results of three strategies for downloading a 256MB file with 10 nodes and half of the nodes with 50ms latency

around 0.5. The average download time and the average selection rate of low latency nodes are shown in Table 7.

4.3.2 Evaluation with 50ms Latency Introduced

Figure 13 illustrates the time taken to transfer a single 256MB file in a P2P environment with 10 nodes, under a 50ms latency for half of the nodes, using three different strategies. The random strategy takes the longest time, but under a 50ms delay, our proposed algorithm performs only slightly worse than the mainstream TFT.

Figure 14 shows the rate of selecting low-latency nodes in the transfer of a single 256MB file within a P2P environment of 10 nodes, where half of the nodes have a 50ms latency, under three different strategies. Since half of the nodes are low-latency nodes, the selection rate for the random selection algorithm still hovers around 0.5. The TFT algorithm still selects more low-latency nodes, and while the proposed model outperforms the random selection algorithm, it still falls short of TFT. The average download time and the average selection rate of low latency nodes are shown in Table 8.

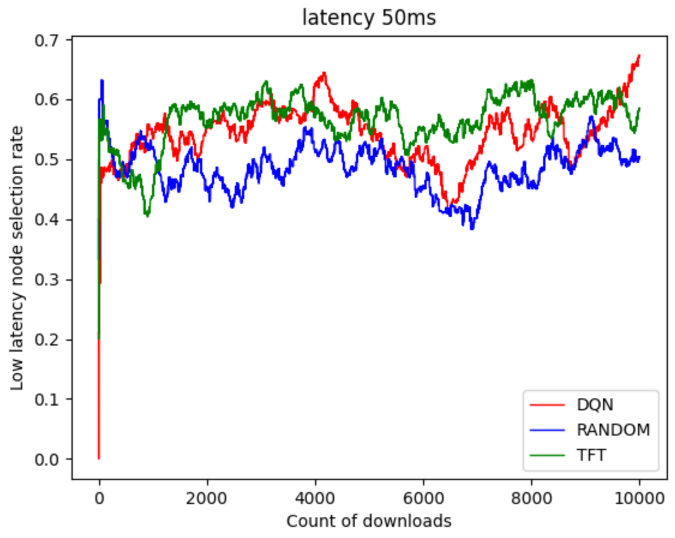

Figure 14. Rate of selecting a low-latency node in each round of downloads within a P2P environment of 10 nodes at 50ms latency for half of the nodes under three different strategies

Table 8. Download time and low latency node selection rate (# of nodes: 10, latency: 50ms)

Algorithm	Avg download time	Avg selection rate
TFT	6.65s	56%
DQN	7.28s	54%
RANDOM	8.26s	48%

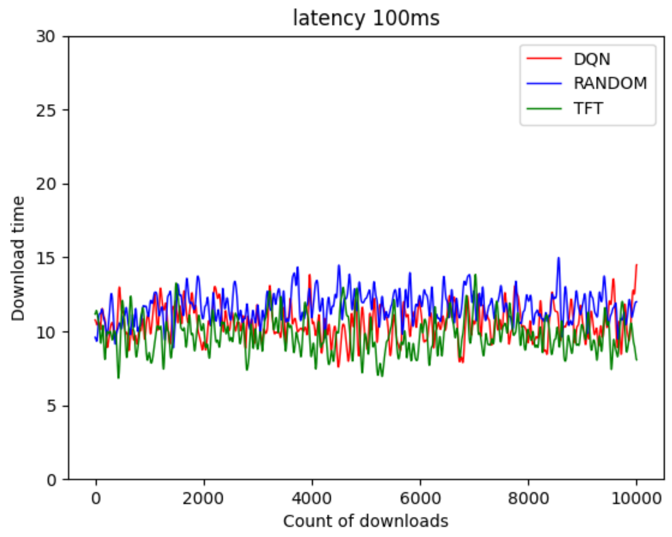

Figure 15. Results of three strategies for downloading a 256MB file with 10 nodes and half of the nodes with 100ms latency

4.3.3 Evaluation with 100ms Latency Introduced

Figure 15 shows the time taken to transfer a single 256MB file in a P2P environment with 10 nodes, where half of the nodes have a 100ms latency, under three different strategies. We can see that due to the significant network latency, the fluctuation in download times also becomes quite large. However, the download time with the TFT algorithm is still less than that of the proposed model and the random selection algorithm. At the same time, the proposed model manages to outperform the random selection algorithm most of the time and achieves almost the same download time as TFT.

Figure 16 illustrates the probability of selecting low-latency nodes in the transfer of a single 256MB chunk within a P2P environment of 10 nodes, where half of the nodes have a 100ms latency, under three different strategies. Since half of the nodes are low-latency, the random selection algorithm still hovers around 0.5. However, TFT is more sensitive to nodes with higher latency, as nodes with fewer data transferred in a given time can be quickly identified as high-latency nodes. Meanwhile, the proposed model also manages to select low-latency nodes with a rate of 0.6, compared to the random selection algorithm which remains

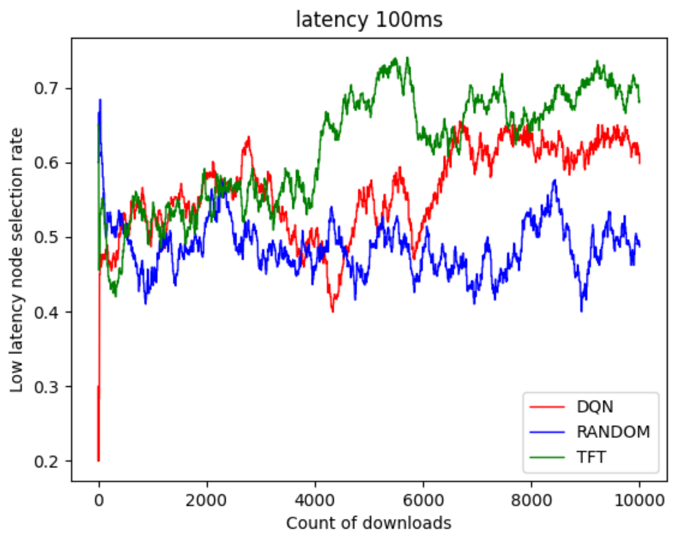

Figure 16. Rate of selecting a low-latency node in each round of downloads within a P2P environment of 10 nodes at 100ms latency for half of the nodes under three different strategies

Table 9. Download time and low latency node selection rate (# of nodes: 10, latency: 100ms)

Algorithm	Avg download time	Avg selection rate
TFT	9.76s	62%
DQN	10.53s	56%
RANDOM	11.66s	48%

at 0.5. The average download time and the average selection rate of low latency nodes are shown in Table 9.

It should also be noted that when a node requests to download a file, it first needs to send a request to our central server to obtain the ID of the optimal node under the current environment. Upon receiving the request, the central server inputs the current environmental states(like CPU, and memory usage) into the model, and the average time taken to produce the required output is approximately 0.0005 seconds. However, TFT and random selection algorithms do not need this computational time to obtain the optimal node. Their decision making is done at the node itself.

4.4 Adaptation of nodes when the network state changes

To examine how the proposed method and the TFT algorithm react to dynamic changes in network conditions, we separately analyze the hit rates of low-latency nodes in the first 20 rounds after every 1,000 rounds of delay redistribution. For example, Fig 17 shows the hit rate of low latency nodes from rounds 5001 to 5020. We observe that the DQN model shows a small upward trend after we redistribute the network delay between nodes, whereas the TFT algorithm shows a downward trend of varying degrees although it does not last long. The short-term downward trend is especially noticeable in the results of the TFT algorithm shown in Fig 18 and Fig 20 and there is also a minor downward trend in Fig 19 and Fig 21.

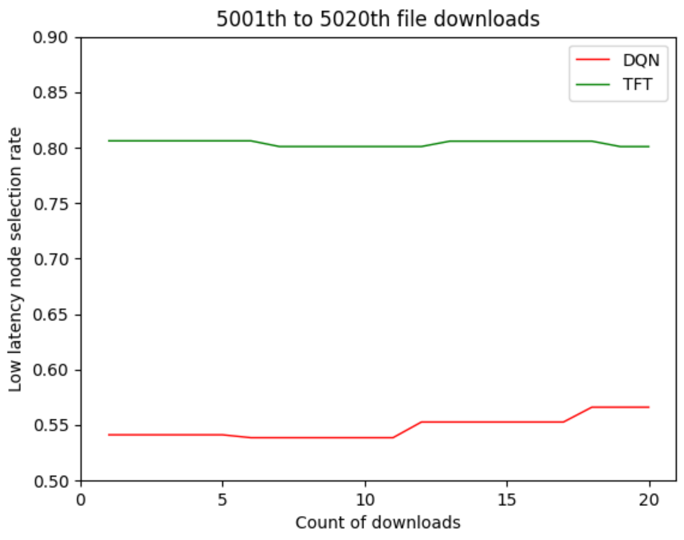

Figure 17. The first 20 downloads starting from the 5,000th download

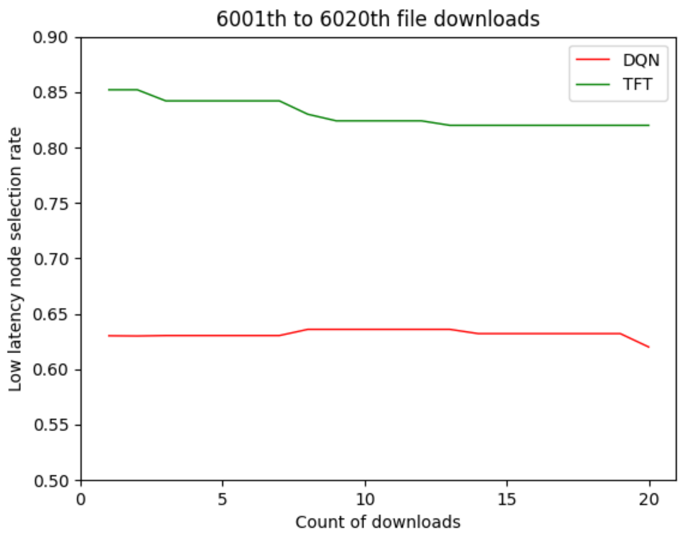

Figure 18. The first 20 downloads starting from the 6,000th download

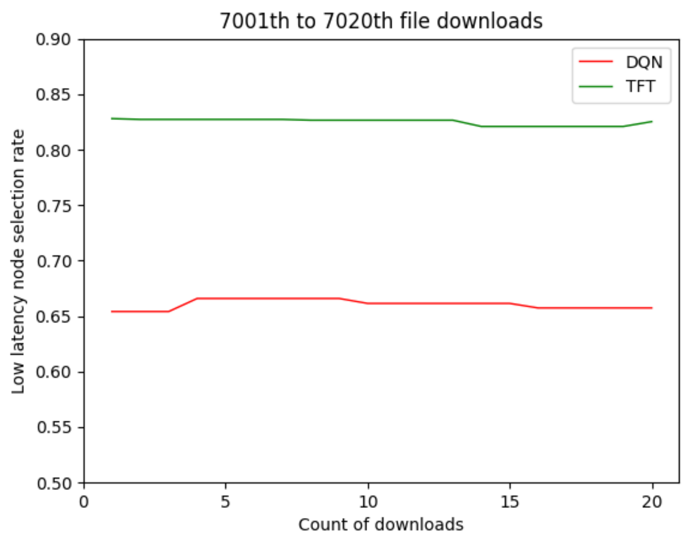

Figure 19. The first 20 downloads starting from the 7,000th download

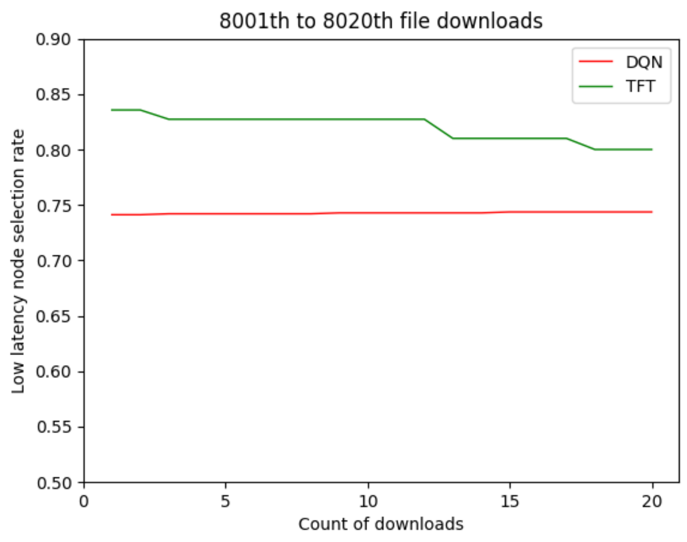

Figure 20. The first 20 downloads starting from the 8,000th download

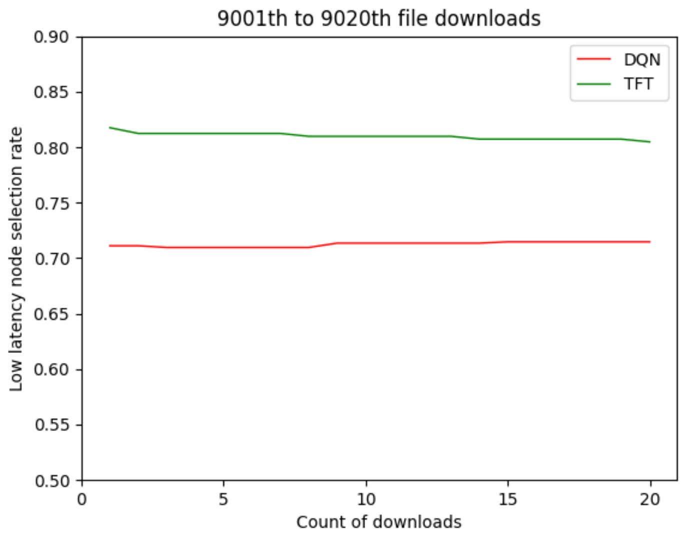

Figure 21. The first 20 downloads starting from the 9,000th download

5. Discussion

This chapter will discuss the results obtained under different scenarios, including those with 10 nodes and 20 nodes, as well as various latency conditions. This chapter will then also delve into the advantages of the proposed algorithm and its limitations, along with areas where improvements can be made.

5.1 Discussion on DQN Algorithm Performance Analysis in a 20-Node Environment

In this low-latency (10ms) scenario, the negligible differences between nodes due to the very low delay result in similar performances for the DQN algorithm (55% selection rate), TFT (57% selection rate), and random selection (48% selection rate). This similarity suggests that in extremely low-latency network environments, the advantages of the DQN algorithm are not pronounced. Regarding download speeds, the DQN algorithm (6.09 seconds), TFT (5.74 seconds), and random selection (6.74 seconds) also show close performances. This further indicates that in a low-latency environment, the impact of latency differences on download speed is minimal. It raises an important consideration: in such environments, the complexity and sophistication of the DQN algorithm might not translate into significant performance gains, as the latency factor becomes less critical compared to other network attributes.

At a 50ms medium latency, the DQN algorithm achieves a 70% selection rate, significantly better than the random selection algorithm's 49%, but slightly inferior to TFT's 80%. This indicates that the DQN algorithm can to some extent discern and select low-latency nodes, yet there is room for improvement. The gap between DQN and TFT suggests that while DQN is effective in identifying lower-latency nodes compared to random selection, it may not be as adept at consistently pinpointing the optimal nodes as TFT. In terms of download speed, the average time for the DQN algorithm is 7.78 seconds, faster than the random selection's 9.16 seconds but slower than TFT's 6.65 seconds. This performance gap could be attributed to the slightly lower accuracy of the DQN algorithm in selecting low-latency nodes. It highlights a potential area for enhancement in the DQN algorithm, particularly in its ability to make more precise selections under varying network latencies.

In a high-latency environment of 100ms, the DQN algorithm's selection rate stands at 64%, still outperforming the random selection's 47% but falling short of TFT's 74%. This disparity might be due to the more pronounced differences in node latency at higher levels, where TFT's method of identifying low-latency nodes becomes increasingly effective. The average download time for the DQN algorithm is 12.53 seconds, quicker than random selection's 14.66 seconds but not as fast as TFT's 11.46 seconds. This underscores the importance of accurate node selection in high-latency environments to improve download efficiency. The DQN algorithm, while better than random selection, shows a notable gap compared to TFT, suggesting that its method of node selection, although effective, might require further refinement to handle higher latency variations more efficiently.

5.2 Discussion on DQN Algorithm Performance Analysis in a 10-Node Environment

In a smaller network of 10 nodes, the minimal impact of a 10ms delay on node selection results in nearly identical performance across the DQN algorithm (49% selection rate), TFT (57% selection rate), and random selection (48% selection rate). This uniformity in performance across different algorithms suggests that in such low-latency conditions, the latency factor is not a significant determinant in node selection efficiency. The download speeds, with the DQN algorithm at 5.59 seconds, TFT at 5.24 seconds, and random selection at 5.84 seconds, are also closely matched. This further reinforces the notion that in environments where latency is minimal, its impact on overall network performance, particularly download speed, is limited. It implies that in such scenarios, the advanced capabilities of the DQN algorithm might not provide a substantial edge over simpler algorithms like random selection or TFT.

At a 50ms delay, the DQN algorithm's selection rate improves to 54%, surpassing random selection's 48% but still trailing behind TFT's 56%. This performance in a smaller network environment suggests that the DQN algorithm's effectiveness in distinguishing between low and high-latency nodes is somewhat constrained by the reduced number of nodes. The smaller pool of nodes might limit the algorithm's ability to leverage its full potential in identifying the most optimal nodes for each task. The average download time for the DQN algorithm in this scenario is 7.28 seconds, slightly slower than TFT's 6.65 seconds but faster than random selection's 8.26 seconds. This indicates that while the DQN algorithm does improve download efficiency compared to random selection, its performance gap with TFT highlights the need for further refinement in environments where the choice of nodes becomes more critical due to increased latency.

In a high-latency setting of 100ms, the DQN algorithm's selection rate is 56%, better than random selection's 48% but not as high as TFT's 62%. This suggests that in environments with significant latency, the ability of the DQN algorithm to consistently choose the lowest latency nodes is somewhat less effective compared to TFT. The higher latency environment amplifies the importance of accurate node selection, where TFT seems to have an edge. The DQN algorithm's average download time in this context is 10.53 seconds, quicker than random selection's 11.66 seconds but lagging behind TFT's 9.76 seconds. This disparity in download speed underlines the critical role of precise node selection in high-latency environments and points to potential areas for improvement in the DQN algorithm's decision-making process.

5.3 Advantages of the proposed DQN algorithm

Through a detailed analysis of the DQN algorithm's performance in different network environments, it is evident that the DQN algorithm has a clear advantage over random selection in medium and high-latency environments, particularly when there is a larger number of nodes. However, compared to the TFT algorithm, there is room for improvement in the DQN algorithm's accuracy in selecting low-latency nodes. These findings suggest that in future research, further optimization of the DQN algorithm is possible, especially in high-latency environments, to enhance its accuracy in selecting low-latency nodes. Additionally, considering the performance differences in various network environments, the adjustments and optimizations of the DQN algorithm should be tailored to specific network conditions and the number of nodes. With these improvements, the DQN algorithm is expected to achieve better performance in a wider range of application scenarios.

5.4 Discussion on the Limitations and Improvement Directions of the DQN Algorithm

5.4.1 Performance Bottleneck in Large-Scale Requests

The DQN algorithm requires additional computation time for decision-making, which can become a significant bottleneck in scenarios with large-scale requests. This computational overhead is particularly noticeable when the algorithm needs to process a vast number of nodes or complex network states, leading to delays that can impact the overall efficiency of the system.

To mitigate this issue, adopting more efficient model architectures that streamline the decision-making process can be beneficial. These architectures could include more advanced neural network designs that require fewer computational resources for the same level of decision-making.

Implementing distributed computing systems can also play a crucial role. By distributing the computational load across multiple servers or nodes, the decision-making process can be parallelized, significantly reducing the time taken for each decision. This approach not only speeds up the processing time but also scales better with increasing network size and complexity.

5.4.2 Adaptation to Changing Node Numbers

The DQN model currently needs structural adjustments when the number of nodes in the environment changes, which limits its flexibility. This rigidity means that the model might not perform optimally when the network topology changes, such as when nodes are added or removed during operation.

Future research could explore more adaptable model designs. One promising direction is the use of Graph Neural Networks (GNNs), which inherently possess the ability to handle varying sizes and structures of networks. GNNs can dynamically adjust to different network topologies, making them ideal for environments where node numbers fluctuate.

Another approach could be the development of modular DQN models that can dynamically reconfigure themselves based on the current network state. This would allow the model to maintain high performance even as the network changes.

5.4.3 Enhancing Responsiveness to Network Variations

While the DQN algorithm performs well in certain environments, its adaptability to different network states needs improvement. In dynamic networks, where conditions such as latency and node availability can change rapidly, the algorithm must quickly adjust its strategies to maintain efficiency.

5.4.4 Incorporating Complex State Representations and Reward Mechanisms

Introducing more complex state representations can provide the algorithm with a deeper understanding of the network, enabling more informed decision-making. This could involve integrating additional network metrics or real-time data into the algorithm's input.

Advanced reward mechanisms can be designed to better capture the nuances of different network states, guiding the algorithm towards more optimal decisions. These mechanisms could be tailored to prioritize certain aspects of network performance, such as latency, throughput, or reliability.

5.4.5 Balancing Real-Time Response and Decision Accuracy

In real-world applications, finding a balance between the algorithm's real-time responsiveness and decision accuracy is crucial. This involves optimizing the algorithm to make quick decisions without significantly compromising on the quality of these decisions.

Enhancing real-time performance might involve optimizing the algorithm's code for faster execution or utilizing more powerful hardware. Improving decision accuracy could be achieved through more precise data collection and processing, ensuring that the algorithm has access to the most relevant and up-to-date information.

5.4.6 Challenges in Large-Scale Deployment

Deploying the DQN algorithm in large-scale network environments poses challenges in terms of computing resource allocation and management. In such scenarios, the algorithm must efficiently handle a vast number of nodes and potentially complex network dynamics without overloading the computational resources.

Exploring the use of cloud computing resources or edge computing technologies can help distribute the computational load. Cloud computing can offer scalable resources as needed, while edge computing can bring computation closer to where data is generated, reducing latency and improving response times.

5.4.7 Importance of User Experience

In P2P networks, the end-user experience is a critical metric for the success of any algorithm. It's essential to regularly collect user feedback to understand how the algorithm impacts the user experience.

Based on this feedback, algorithm parameters can be adjusted to ensure higher user satisfaction. This might involve tweaking the balance between different performance metrics or introducing new features to better align with user preferences and requirements.

5.5 Limitations of the proposed model and possible future improvements

While the DQN algorithm shows promise in certain network environments, its limitations in computational efficiency, model flexibility, and adaptability to varying network states present areas for improvement. Addressing these challenges through advanced model architectures, distributed computing, and enhanced learning mechanisms can lead to a more robust and efficient algorithm. By continually refining the algorithm and tailoring it to the specific needs of different network environments, the DQN algorithm can achieve broader applicability and improved performance, ultimately enhancing the user experience in P2P networks.

5.5.1 Improvements to the Experimental Setup

The current experimental setup may not reflect a realistic environment, which suggests that enhancing this aspect could potentially improve the algorithm's performance. Although the current model includes CPU and memory usage in its inputs, the experiments only control latency, without considering scenarios where each node is under high load. Designing experiments that account for these situations could lead to improved performance under conditions where node load impacts data exchange performance. Furthermore, the file size being exchanged is fixed at 256MB in the experiment, indicating room for improvement. By adjusting the file sizes used in model training to reflect the distribution found in realistic P2P networks, the model might be able to develop more advanced strategies suitable for real-world environments. For example, it could learn to fetch smaller files from distant nodes despite higher latency, considering load balancing. To achieve a more realistic experimental setup, it would be beneficial to simulate the frequency of requests from each node and the latency between nodes based on a realistic network model. Incorporating simulations that reflect actual network behavior would enable the experiments to better represent and adapt to real-world conditions.

Additionally, the current experimental setup has limitations because it relies on a fixed single scenario for training. In this scenario, the model is trained using the records of the past 100 downloads every 100 downloads, with latency being reassigned every 1000 downloads, and the epsilon parameter being changed after 5000 downloads. However, considering that these parameters can impact the accuracy of the trained model, it is believed that tuning, such as experimenting with various combinations of parameters, is necessary. This approach would help in identifying optimal settings that enhance model performance and would illustrate the need for a more flexible and adaptive experimental framework.

5.5.2 Improvements to the Latency Measurement Between Nodes

The current methods for measuring and collecting latency face scalability issues. As the number of nodes in a network increases, it becomes impractical to measure the latency between all pairs of nodes due to the exponential growth in the number of measurements needed. One potential solution to this problem is to divide the network into several smaller regions and then measure the latency only between representative nodes within each region. This approach, however, raises several important considerations. It necessitates decisions about how to categorize nodes into groups, determining the optimal size and number of these groups to ensure that the measured latencies accurately reflect the network's performance.

6. Conclusion

In this study, we introduce an innovative approach to optimizing peer-to-peer (P2P) file sharing networks through a modified dueling-Deep Q-Network (DQN) model. This model is specifically designed to enable nodes within a P2P network to make more efficient decisions when selecting peers for downloading files. The core of this approach lies in its ability to dynamically adapt to the ever-changing conditions of the network, a significant advancement over traditional methods.

The proposed method has been rigorously compared with two conventional algorithms: the random selection algorithm and the Tit-for-Tat (TFT) algorithm. The random selection algorithm, as its name suggests, chooses nodes randomly without considering their current or past performance. This approach, while simple, often leads to suboptimal download speeds and inefficiencies, especially in a network where node performance can vary widely.

The TFT algorithm, on the other hand, represents a more strategic approach. It selects nodes based on historical performance data from a defined previous time period, such as the past 20 seconds. This method allows for a more informed selection process compared to random selection. However, its reliance on historical data can be a drawback. If a node's state changes significantly during the considered time window, the TFT algorithm's decision may no longer be optimal. Despite this limitation, TFT has a mechanism to quickly adjust its strategy once it detects a significant slowdown in the node's performance, showcasing its adaptive nature.

The proposed dueling-DQN model takes this adaptability a step further. Unlike TFT, which relies on historical data, the dueling-DQN model bases its decisions on the real-time state of the nodes. This real-time analysis allows for rapid adaptation to the current conditions of the network, ensuring that the node selection for file downloads is as efficient as possible at any given moment. This immediacy in decision-making is particularly advantageous in a P2P environment where node states can fluctuate rapidly due to varying bandwidths, user activity, or network conditions.

Moreover, the integration of AI into the selection process, as demonstrated by the dueling-DQN model, opens up new avenues for future development in P2P networks. This approach moves beyond traditional selection algorithms, leveraging the power of artificial intelligence to enhance network performance. By continuously learning and adapting to the network's state, AI-driven models like the dueling-DQN can potentially revolutionize how file sharing networks operate, leading to faster download speeds, more efficient network utilization, and an overall better user experience.

In conclusion, the modified dueling-DQN model presents a significant advancement in P2P file sharing technology. Its ability to make real-time, informed decisions about node selection stands in contrast to the limitations of traditional algorithms. While it still has some gaps to bridge when compared to the TFT algorithm, this model represents a novel attempt in the field, offering fresh perspectives and laying the groundwork for future research. As P2P networks continue to evolve, the incorporation of AI and machine learning models like the dueling-DQN offers a promising direction for enhancing the efficiency and effectiveness of these networks. This innovative approach not only addresses current challenges but also opens up new possibilities for the continued evolution and improvement of P2P file sharing systems.

Acknowledgements

Completing this thesis would not have been possible without the collective support and encouragement of the entire NAIST. I am deeply grateful for the nurturing academic environment provided by the university, which has been instrumental in my personal and professional development.

I would like to express my deepest gratitude to Professor Hajimu Iida for not only providing me with the opportunity to conduct this research but also for his continual concern about my adaptation to life in Japan. His invaluable support and recommendations played a crucial role in securing my internship at Toshiba, an experience that has been both enriching and instrumental in my professional growth.

I would also express my sincere gratitude to Professor Fujikawa Kazutoshi for his invaluable insights during the intermediate presentation of my research, particularly his suggestion to test the model in various node environments. This advice not only enhanced the robustness of my work but also helped me identify potential issues and weaknesses in the methodology I proposed. His guidance has been instrumental in refining my approach and advancing my understanding of the subject.

Special thanks are also due to Professor Kohei Ichikawa. In the early stages of my research, Professor Ichikawa attentively listened to my project ideas and offered insightful advice on how to refine and advance my research. I am particularly grateful for his guidance when my project encountered significant challenges; his suggestion to incorporate delays into the P2P environment was a turning point, leading to markedly improved results. Furthermore, his valuable feedback on my presentation for the IPSJ-IoT conference in July 2023 was instrumental in my successful participation in this prestigious academic forum. His practical advice upon my arrival in Japan, such as recommending the Ahamo plan, greatly facilitated my daily life.

I extend my heartfelt appreciation to the teaching assistants, especially Kashiwa-sensei, and the secretary, Ogawa-san, of our laboratory. Their assistance and support have been pivotal in making my life at NAIST not only easier but also more enjoyable.

I am also immensely thankful to my fellow lab mates. Their camaraderie, collaborative spirit, and shared insights have greatly enriched my research experience. The stimulating discussions, mutual encouragement, and the collective brainstorming sessions have not only enhanced my academic journey but also contributed to some of the breakthroughs in my research.

Lastly, I would like to acknowledge the financial support from my family, which enabled me to fully focus on my research. This journey has been a blend of academic rigor and personal growth, and I am profoundly grateful to everyone who has been a part of it.

References

[1] Geoffrey Fox. Peer-to-peer networks. *Computing In Science & Engineering*, 3:75–77, 2001.

[2] Anthony J Howe. Napster and gnutella: a comparison of two popular peer-to-peer protocols. *Universidade de Victoria*, 11, 2000.

[3] Johan Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *Peer-to-Peer Systems IV: 4th International Workshop, IPTPS 2005, Ithaca, NY, USA, February 24-25, 2005. Revised Selected Papers 4*, pages 205–216. Springer, 2005.

[4] Joan Antoni Donet Donet, Cristina Pérez-Sola, and Jordi Herrera-Joancomartí. The bitcoin p2p network. In *International conference on financial cryptography and data security*, pages 87–102. Springer, 2014.

[5] Naeem Ramzan, Hyunggon Park, and Ebroul Izquierdo. Video streaming over p2p networks: Challenges and opportunities. *Signal Processing: Image Communication*, 27(5):401–411, 2012.

[6] Dr Brijender Kahanwal and Dr TP Singh. The distributed computing paradigms: P2p, grid, cluster, cloud, and jungle. *arXiv preprint arXiv:1311.3070*, 2013.

[7] Ashwin R Bharambe, Cormac Herley, and Venkata N Padmanabhan. Analyzing and improving bittorrent performance. *Microsoft Research, Microsoft Corporation One Microsoft Way Redmond, WA*, 98052:2005–03, 2005.

[8] Jim X Chen. The evolution of computing: Alphago. *Computing in Science & Engineering*, 18(4):4–7, 2016.

[9] Zhi Peng, Zhihong Duan, Jianbin Qi, Yujia Cao, and En Lv. Hp2p: A hybrid hierarchical p2p network. In *First International Conference On The Digital Society (ICDS'07)*, pages 18–18, 2007.

[10] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop On Economics Of Peer-to-Peer Systems*, volume 6, pages 68–72, 2003.

[11] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference On Machine Learning*, pages 1995–2003, 2016.

[12] Vivek Vishnumurthy and Paul Francis. On random node selection in p2p and overlay networks. *Manuscript, http://www. cs. cornell. edu/People/francis/RandSelection19. pdf*, 4, 2004.

[13] Micah Adler, Rakesh Kumar, Keith Ross, Dan Rubenstein, Torsten Suel, and David D Yao. Optimal peer selection for p2p downloading and streaming. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 3, pages 1538–1549. IEEE, 2005.

[14] Vivek Vishnumurthy and Paul Francis. On heterogeneous overlay construction and random node selection in unstructured p2p networks. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–12. IEEE, 2006.

[15] Haixin Duan, Xiaoxia Lu, Hong Tang, Xiaoyan Zhou, and Zheng Zhao. Proximity neighbor selection in structured p2p network. In *The Sixth IEEE International Conference On Computer And Information Technology (CIT'06)*, pages 52–52, 2006.

[16] Tyler Steele, Vivek Vishnumurthy, and Paul Francis. A parameter-free load balancing mechanism for p2p networks. In *IPTPS*, page 21, 2008.

[17] Emna Salhi, Mohamed Karim Sbai, and Chadi Barakat. Neighborhood selection in mobile p2p networks. In *Algotel*, 2009.

[18] Ioannis Pogkas, Vassil Kriakov, Zhongqiang Chen, and Alex Delis. Adaptive neighborhood selection in peer-to-peer networks based on content similarity and reputation. *Peer-to-peer networking and applications*, 2:37–59, 2009.

[19] Yanbo Gui, Shiwen Ju, and Hongseok Choi. P2p streaming media node selection strategy based on greedy algorithm. In *IKEEE*, volume 22, pages 570–577, 2018.

[20] Longle Cheng, Xiaofeng Li, Haibo Tan, He Zhao, and Bin Yu. Mslt: A scalable solution for blockchain network transport layer based on multi-scale node management. *IEICE Transactions on Communications*, 107(1):185–196, 2024.

[21] Seong-Moo Koo, Krishna Kannan, and Choong Seon Lee. On neighbor-selection strategy in hybrid peer-to-peer networks. *Future Generation Computer Systems*, 22:732–741, 2006.

[22] Robert Beverly and Mark Afergan. Machine learning for efficient neighbor selection in unstructured p2p networks. In *SysML*, volume 7, pages 1–6, 2007.

[23] Raz Izhak-Ratzin, Hyojin Park, and Mihaela Van Der Schaar. Online learning in bittorrent systems. *IEEE Transactions On Parallel And Distributed Systems*, 23:2280–2288, 2012.

[24] Tomoyuki Naito and Satoshi Fujita. Acquiring nearly optimal peer selection strategy through deep q-network. In *2018 Sixth International Symposium On Computing And Networking (CANDAR)*, pages 120–125, 2018.

[25] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[27] Fuzheng Zou, Li Shen, Zhang Jie, Weike Zhang, and Wei Liu. A sufficient condition for convergences of adam and rmsprop. In *Proceedings Of The IEEE/CVF Conference On Computer Vision And Pattern Recognition*, pages 11127–11135, 2019.

[28] Nipun Wijerathne Varuna Jayasiri. labml.ai annotated paper implementations, 2020.

[29] Pallets. flask, https://github.com/pallets/flask.

[30] Libp2p. go-libp2p, https://github.com/libp2p/go-libp2p.

[31] Shirou. gopsutil, https://github.com/shirou/gopsutil.