future internet

MDPI

Article

An Intelligent TCP Congestion Control Method Based on Deep Q Network

Yinfeng Wang [1], Longxiang Wang [2,*] and Xiaoshe Dong [2]

[1] College of Software, Shenzhen Institute of Information Technology, Shenzhen 518116, China; 2011100764@sziit.edu.cn
[2] College of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China; xsdong@xjtu.edu.cn
* Correspondence: wlx419@xjtu.edu.cn

Abstract: To optimize the data migration performance between different supercomputing centers in China, we present TCP-DQN, which is an intelligent TCP congestion control method based on DQN (Deep Q network). The TCP congestion control process is abstracted as a partially observed Markov decision process. In this process, an agent is constructed to interact with the network environment. The agent adjusts the size of the congestion window by observing the characteristics of the network state. The network environment feeds back the reward to the agent, and the agent tries to maximize the expected reward in an episode. We designed a weighted reward function to balance the throughput and delay. Compared with traditional Q-learning, DQN uses double-layer neural networks and experience replay to reduce the oscillation problem that may occur in gradient descent. We implemented the TCP-DQN method and compared it with mainstream congestion control algorithms such as cubic, Highspeed and NewReno. The results show that the throughput of TCP-DQN can reach more than 2 times of the comparison method while the latency is close to the three compared methods.

Keywords: congestion control; reinforcement learning; TCP

Citation: Wang, Y.; Wang, L.; Dong, X. An Intelligent TCP Congestion Control Method Based on Deep Q Network. *Future Internet* **2021**, *13*, 261. https://doi.org/10.3390/fi13100261

Academic Editor: Paolo Bellavista

Received: 31 August 2021
Accepted: 30 September 2021
Published: 9 October 2021

1. Introduction

In recent years, China's supercomputers have made great progress. The Sunway TaihuLight and Tianhe have become one of the fastest supercomputers in the world. However, the storage resources are widely dispersed and autonomous among the national supercomputing centers. Large-scale computing applications urgently need a global data space that can support cross-domain unified access, wide area data sharing, storage and computing collaboration. To solve this problem, we built the virtual data space to aggregate storage resources among different supercomputing centers. One challenge of virtual data space is to build a high-performance and reliable network transmission protocol to fit the demand of Giga Bytes or even Tera Bytes scale data migration.

Congestion control is the key technology to achieve efficient and reliable transmission. The virtual data space is built on WAN, which is complex and changeable. Although researchers have proposed a variety of TCP congestion control algorithms in the past 30 years, such as New Reno [1], Vegas [2], Highspeed [3] and cubic [4], etc., these methods are designed according to the characteristics of the specific network environment and can only be worked according to a predefined rule. Therefore, these methods are difficult to adapt to the complex and changeable network environment of virtual data space. Whether the classic new Reno or cubic algorithm, which is currently used as the default TCP congestion control algorithm in Linux operating systems, its core idea is to regulate network congestion parameters based on some predefined rules. These methods are only suitable for scenarios with relatively stable network conditions. If the network conditions change, the traditional methods cannot adapt to the new scenarios. Therefore, researchers proposed

using reinforcement learning (RL) to build a smart congestion control method. As a research hotspot in the machine learning field, reinforcement learning has been widely used in UAV control [5], robot control [6], optimization and scheduling [7,8] and computer games [9]. The basic idea of RL is to construct an agent to interact with the environment, and to learn the optimal strategy by maximizing the cumulative reward obtained by the agent from the environment. Compared with the traditional method, the congestion control method, based on reinforcement learning, has better adaptability and can learn a new strategy from the changed network environment. Li et al. [10] proposed QTCP, which is a TCP congestion control method based on Q-Learning (QTCP). The core idea of Q-learning method [11] is to estimate the Q value of all state-action pairs (s, a). Q represents the expected reward that can be obtained by taking action a from the current state s until the episode ends. If the Q value of all state-action pairs (s, a) is estimated, the optimal congestion control strategy is trivial. It can be achieved by always selecting the action a which can maximize the Q value at every step. However, the state space of TCP congestion control is too large. If all Q values of (s, a) pairs are saved as a two-dimension table in memory, it will overfit the memory. It is difficult to implement and unnecessary in practice. At present, the solution to this problem is to use a function to approximate the Q value table. QTCP uses the Kanerva coding function to approximate the Q value. However, the Q-learning method converges slowly after approximating the Q-value table with functions. Since the purpose of Q-learning method is to find the unbiased Q value of all state action pairs (s, a), the exact value of Q needs to be estimated by iterating on Bellman equation. However, if the Q-value table is approximated by a function, the Bellman equation will be bootstrapped. When the Q-value changes slightly, it may lead to severe oscillations during the training process. Moreover, when the action space of Q-learning method is large, it is easy to converge to a local optimal solution. Therefore, the QTCP does not work very well. Compared with NewReno, QTCP improves the throughput by 59.5%. The results may look good; however, it has been 20 years since the NewReno method was proposed in 1999. The NewReno method has been deprecated in practice. The cubic algorithm, which is used as the default congestion control method in Linux OS, improves the throughput by 200% compared with NewReno. Compared with the cubic method, QTCP has little advantage. To build a more efficient congestion control method by leveraging reinforcement learning, we propose TCP-DQN, which is a congestion control method based on the DQN algorithm, to implement efficient and reliable data migration in virtual data space. DQN [9] is the state-of-the-art method of the Q-learning family. DQN uses experience replay and target network to reduce the correlation between training data, solving the oscillation problem in the traditional Q-learning method. The main contributions of this paper are as follows

(1) We propose TCP-DQN. By leveraging the state-of-the-art DQN algorithm, the training of the congestion control is more stable and faster than the conventional Q-learning method.
(2) We implemented TCP-DQN and compared it with the cubic method, which is the state-of-the-art congestion control algorithm and is used as the default congestion control method in Linux kernel. The results show that TCP-DQN can improve the throughput by 200% compared with cubic.

2. Related Works

2.1. Traditional Congestion Control Protocols

Improving network performance is a hot research topic [12,13]. Over the years, researchers have proposed various congestion control algorithms. They can be roughly divided into four types according to design principles: loss-based, delay-based, capacity-based, and hybrid.

Reno, NewReno, and BIC are representatives of loss-based congestion control mechanisms. They both use packet loss to detect congestion and reduce the congestion window (CWND) to reduce network congestion when congestion is detected, where Reno and NewReno halve CWND only when three duplicate ACKs are detected, while cubic uses the

cubic function to adjust CWND. Loss-based congestion control mechanisms use packet loss as the only signal to detect congestion. When the receive buffer is large, the data will not be lost until the buffer is full. During this period, the sender continues to increase its sending rate. In this case, there is a long time difference between the occurrence of congestion and the awareness of network congestion by the congestion control mechanism.

Delay-based congestion control mechanisms can solve the above problems by using delays instead of packet loss as the congestion signals. Vegas [14] is the representative of delay-based congestion control protocol. When Vegas detects that RTT exceeds the threshold value, it starts decreasing CWND.

Capacity-based congestion control mechanisms use the estimation of link sending capability as the basis for congestion control. Westwood [15] is improved from NewReno, which measures ACK packets to determine an appropriate send speed and adjusts CWND and slow start thresholds.

The Hybrid congestion control mechanism combines the above two congestion control mechanisms to get their own advantages and further improve the congestion control. Compound [16], BBR [17] belong to Hybrid congestion control mechanism.

Each of the above congestion control mechanisms has its own unique attributes, advantages, and disadvantages. However, since traditional congestion control mechanisms use a set of rules to control CWND and other related parameters, it is difficult for traditional congestion control protocols to adapt to the complexity and rapid development of modern networks.

2.2. Reinforcement Learning and Its Applications

Reinforcement learning (RL) is a subarea of machine learning. The basic idea of RL is to construct an agent to interact with the environment and learn the best policy to achieve the goal by maximizing the cumulative reward that the agent receives from the environment. In essence, network congestion control can be regarded as an optimization problem [18–20]. Reinforcement learning has been widely used in many important areas. Y Ji et al. [21] proposed a novel energy management approach for real-time scheduling of an microgrid. Y. Fang et al. [22] proposed a reinforcement learning method to optimize the XSS detection model to defend against adversarial attacks. In the past, researchers have proposed a large number of optimization methods, such as NSGA-III variants [23]. Compared with traditional congestion control algorithms, the congestion control algorithm based on reinforcement learning has better adaptability and can learn new congestion control policies from the network environment independently. Due to this feature, reinforcement learning has become an important tool for network and protocol designers in recent years. The study of congestion control based on reinforcement learning has become a hot spot and has been applied in various areas. For example, Le T. et al. [24] suggests learning channel allocation decisions by using a linear bandit model to minimize total switching costs in multichannel wireless networks. N. Liu et al. [25] uses deep reinforcement learning (DRL) to deal with large and complex state spaces when solving cloud resource allocation and power management problems. Xu et al. [26] presents a DRL-based framework for energy-efficient resource allocation in cloud RAN. Lu et al. [27] propose a reinforcement learning-based decision method for electricity pricing plan selection by smart grid end users. Pérez et al. [28] present a deep reinforcement learning method for energy-conscious optimization of edge computing. Jung et al. [29] propose a deep inverse reinforcement learning method for autonomous driving in an urban environment. Deltetto et al. [30] present a reinforcement learning (rl) control strategy for the participation in an incentive-based demand response program of a cluster of commercial buildings. Fischer et al. [31] use a reinforcement learning method to address the question of whether they can predict reaching movements in a full skeletal model of the human upper extremity. In addition, there are many algorithms based on reinforcement learning to improve the quality of service of network applications. Habachi et al. [32] presented an algorithm based on RL to generate congestion control rules to optimize QoE for multimedia applications. Hemmati

et al. [33] defines the network resource allocation problem as a DEC-POMDP model in a multiuser video stream, and applies a distributed RL algorithm to solve the problem. Hooft et al. [34] uses RL algorithm to improve the QoE of video streams by adaptively changing parameter configurations. Its limitation is the use of a table-based algorithm, thereby limiting their application to large continuous domains. Cui et al. [35] presented Hd-TCP: a custom congestion control algorithm that uses deep reinforcement learning to deal with the poor network experience caused by frequent network switching on high-speed networks from a transport layer perspective.

All above scenarios are task driven. They are designed for specific applications and cannot be applied directly to congestion control problems. QTCP [10] is the first solution to directly apply RL to TCP congestion control protocol design. QTCP continuously updates the values of possible state-action pairs of protocols based on measurements of performance indicators collected from the network environment and uses the Q-Learning algorithm to search for the best action, that is, how to adjust CWND. In a particular state, maximize the long-term return of the sender. However, QTCP may cause repeated oscillations during training, and it is easy to converge to the local optimal solution when the Q-Learning method has a large space of motion.

TCP-Drinc [36] is a model-free intelligent congestion control algorithm based on deep reinforcement learning, which gets experience from past network states and determines how to adjust the CWND based on the experiences. The experimental results show that TCP-Drinc congestion control algorithm achieves a balance between throughput and RTT. It has a more stable and average performance than NewReno, Vegas, and other algorithms, but there is no significant improvement in throughput.

Rax [37] uses online reinforcement learning to maintain the best congestion window based on the received reward from the network environment. It has a lower packet loss rate, but less throughput improvement than Reno, PCC, and other congestion control algorithms. VNE-TD [38] uses TD Learning to solve the Virtual Network Embedding.

Q-learning algorithm is widely used in congestion control. However, when the action space of Q-learning algorithm is large, Q-learning is very easy to converge to the local optimal solution. To solve this problem, we propose a DQN based network congestion control method. Compared with the traditional Q-learning algorithm, DQN converges more quickly, has much better performance.

3. Background

DQN

Q-learning [11] is a value-based reinforcement learning algorithm, which is the basis of DQN. Generally speaking, reinforcement learning interacts and learns from the environment by using an agent. The agent tries to generate a suitable action to maximize the received reward in a particular state, and finally learns the optimal control policy. The core idea of reinforcement learning is trial and error mechanism and policy optimization. Agents improve their policy by trying various actions to learn the good action that can get more reward from the environment.

In Q-learning, a greedy policy is used to generate the action a, which means the agent will always choose the action that can obtain the maximum Q value in each state s. The core idea of Q-learning is the Q(s, a) function. The Q(s, a) function represents the cumulated reward until the current episode ends after choosing an action a in state s. According to the Bellman equation, the Q value can be iteratively calculated by Equation (1).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \quad (1)$$

where α is the learning rate; R is the reward value obtained after performing the action a_t; γ represents the discounting factor, which determines the importance of future rewards. In theory, all Q(s, a) values can be held in a two-dimensional table, and all the exact Q values can be calculated by iterating enough times according to Equation (1). After calculating all Q values in the Q table, the optimal control policy is trivial, which can be implemented by

choosing the action a which can maximize the Q value at each state. However, in practical, the Q table is too large to hold in memory. Therefore, a function is used to approximate the Q table, which means the Q value can be estimated if a (s, a) pair is inputted to the function. The simplest way is using a linear function to approximate the Q table. To get a better result, a non-leaner function such as artificial neural network can be used to approximate Q table. However, with function approximation, the oscillation problem may be occurred during the training process.

As shown in Figure 1, DQN [12] is the development of Q-learning algorithm [13], which is a new algorithm combining deep learning with reinforcement learning. DQN solves the oscillation problem during training by experience replay and dual neural network. Different from Q-learning, the parameters are updated immediately after gaining experience in each step, DQN saves the experience $e_t = (s_t, a_t, r_t, s_{t+1})$ obtained in each step to an experience pool, which is formalized as $D_t = \{e_1, \ldots, e_t\}$. DQN will randomly sample experience from the experience pool to update the parameters. This mechanism has three advantages:

(1) Each experience may be used for multiple parameter updating. Combing with the experience replay, the learning is more effective.
(2) The samples of Q-learning learning are generated in continuous actions, which makes the samples have strong correlation with each other and result in oscillation in training. DQN randomly samples from the experience pool to eliminate the correlation between continuous samples and enhance stability of the training process.
(3) DQN randomly picks samples from the experience pool for training, which avoids the problem that Q-learning may fall into a local optimal solution. In addition, DQN constructs two neural networks with the same structure for learning. They are: Q-network and target-Q network. Samples are generated by the target-Q network, and the Q-network is used for parameter update. After c-time parameter update, the parameters of Q-network are copied into target-Q network. This leads to more stable training because it keeps the target function fixed (for a while). The algorithm of DQN is described as follows (Algorithm 1):

Algorithm 1. DQN

Initialize replay memory D to capacity N
Initialize action value function Q with random weights θ
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1, T **do**
    With probability ε select a random action a_t
    Otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t next state x_{t+1}
    Set $s_{t+1} = s_t, a_t x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
  End for
End for

x_t represents the observation at the time-step t. θ is used as the weights of the neural network in DQN. ϕ is the preprocessing function. The DQN algorithm is originally proposed to play the Atari games. Working directly with raw Atari frames, which are 210×160-pixel images with a 128 color palette, can be computationally demanding, so DQN applies a basic preprocessing step aimed at reducing the input dimensionality. In our implementation, the input data are small. Therefore, we do not' use a preprocess function.

T represents the terminate time-step. The agent will select a random action to explore the environment with probability ε, which can be preset by the user.

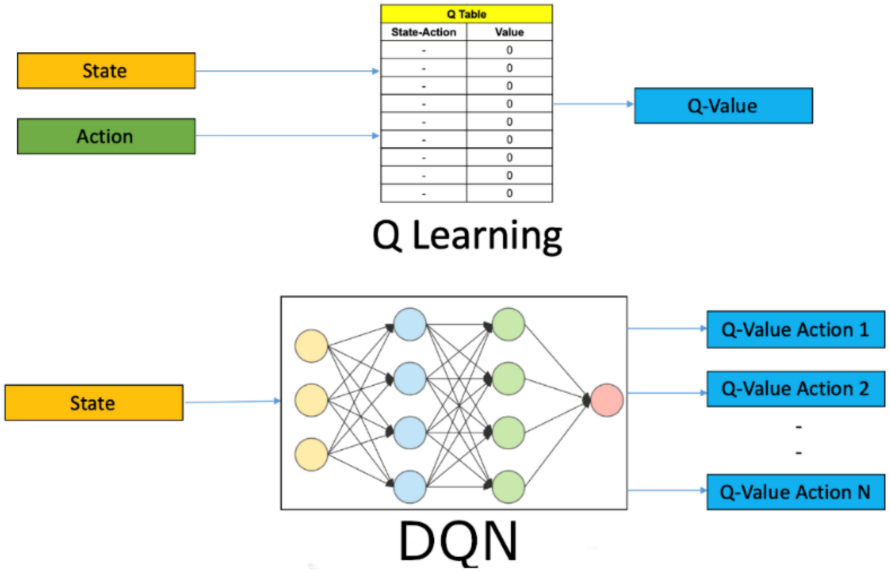

Figure 1. Q-learning vs. DQN.

4. Framework

The TCP-DQN framework is shown in Figure 2. The core of reinforcement learning is to abstract the environment and agent. We take the virtual data space network as the environment, and the agent adjusts the congestion window by observing the state information in the environment to optimize the performance of the virtual data space network. After making the action and interacting with the environment, the agent gains the reward from the environment. The agent updates the artificial neural network parameters to maximize the Q value.

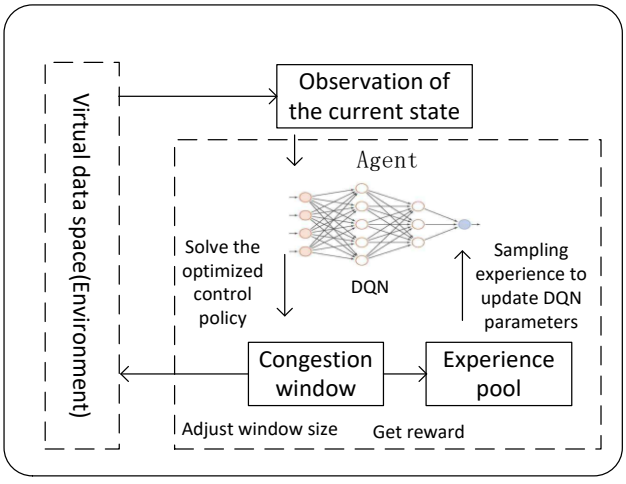

Figure 2. TCP–DQN framework.

4.1. Problem Formulation

The TCP congestion control process can be abstracted as a partially observable Markov decision process, which is defined as a quintuple {S, A, R, P, γ}. Where S is the set of all states. $s_t \in$ S is the state observed at time t, and the initial state is s_0; A is the set of actions, and $a_t \in$ A is the action taken at time t; R is the reward function defined as

$$\mathrm{R}(s_t, a_t) = \mathrm{E}(\mathrm{R}_{t+1}|s_t, a_t) \tag{2}$$

According to Equation (2), if the agent is in state st at time-step t, and selects the action at, then the reward R_{t+1} is received from the environment at time-step $t + 1$; P is the transition probability matrix, defined as

$$P^{a}_{ss'} = P(s_{t+1} = s' | s_t = s, a_t = a) \tag{3}$$

Equation (3) represents the probability from the state s' to the state s if the action a is chosen at time t. $\gamma \in [0, 1]$ is the discounting factor, which reflects the decay rate of the future rewards. The discount factor reflects the design idea of the reinforcement learning algorithm, which means the reward that can be obtained immediately is preferred. Whereas the reward obtained in the future will be decayed by a certain proportion.

The reinforcement learning algorithm starts from an initial state s_0. According to the observed state s_t, the action a_t is selected by a policy function $\pi(a_t \mid s_t)$. According to the state transition matrix $P(s_{t+1} \mid s_t, a_t)$, the reinforcement learning algorithm gets the reward R_{t+1} from the environment. The goal of reinforcement learning is to optimize the strategy function to maximize the expected reward, which is defined as

$$\begin{aligned} G_t &= R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} \ldots + \gamma^T R_{t+2} \\ &= \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \end{aligned} \tag{4}$$

In Equation (4), T is the time-step at which the episode terminates.

4.2. State Space

Designing a reasonable state space s_t is the key to achieve an efficient reinforcement learning goal. Only by observing enough information, the reinforcement learning algorithm can choose good actions. However, too much state information will increase the computation cost and slow down the learning speed. Therefore, we design the state space s_t including the following parameters:

(1) The relative time t. The time that has been elapsed since the TCP connection was established. In cubic algorithm, the window size is designed as a cubic function of time t. Therefore, we consider t as an important parameter to determine the congestion window.

(2) Congestion window size. Congestion control algorithm needs to adjust the new window size based on the current one. If the current congestion window is small, the agent will increase the window faster; else if the window is large, the agent will stop increasing the window size or slowly increasing the window size.

(3) Number of unacknowledged bytes. The number of bytes sent but not confirmed by the receiver. If the network link is compared to a water pipe, the number of unacknowledged bytes can be understood as the water stored in the pipeline. If there is enough water in the pipeline, the water injected into the pipeline should be stopped or reduced, else the water should be injected more. Therefore, the water injection rate (congestion window size) should be determined according to the water volume (number of unacknowledged bytes) in the pipeline.

(4) Number of ACK packets. This parameter can indirectly reflect the congestion situation. If the number of received ACK packets is normal, it indicates that the network is in a good condition and congestion has not occurred. The size of the congestion window can be increased. Otherwise, it indicates that the network is congested and the congestion window should be maintained or reduced.

(5) Average RTT (round trip time) value. The average RTT value in the observation period. RTT refers to the total time spent from sending to receiving ACK for a data packet. The RTT value is closely related to the network congestion. If the network congestion is serious, the RTT value will increase significantly. Therefore, RTT value can reflect the network congestion. The congestion control algorithm should adjust the congestion window according to RTT value.

(6) Throughput. The throughput during the observation period. Throughput is defined as the number of data bytes confirmed by the receiver per second. This parameter directly reflects the network status. High throughput indicates that enough packets have been sent into the network link. Otherwise, it indicates that the current network bandwidth is more redundant, and more packets can be sent on the link.
(7) Number of lost packets. The number of lost packets indicates the congestion situation of the current network. If the number of lost packets is small, the agent should increase the congestion window. Otherwise, the agent should decrease the congestion window.

4.3. Action Space

a_t is defined as the adjusting action of the congestion window at time T. We define the action as the increase in n segments to the congestion window, as shown below

$$cwnd = cwnd + n * segment \tag{5}$$

A segment is the unit of end-to-end transmission in the TCP protocol. A segment consists of a TCP header followed by application data. A segment is transmitted by encapsulation inside an IP datagram.

The action is defined as the choice of n, which is a one-dimensional discrete vector. Its value range is set to $(-50, 50)$.

The design idea of Equation (5) is to provide a generalized equation to determine the growth rate of congestion window according to the observed state information. In different network scenarios, the agent should choose reasonable strategies to effectively utilize the network bandwidth. In a high bandwidth environment, the congestion window grows exponentially by adjusting $n > 1$, Otherwise, in a low bandwidth environment, the congestion window grows linearly by adjusting $n = 1$. In case of network congestion, the agent adjusts $n <= 0$ to keep or reduce the congestion window and relieve the congestion pressure.

4.4. Reward Function

r_t is defined as the reward received from the environment at time t. The reward function is shown in Equation (6).

$$r_t = \alpha \left(\frac{\mathrm{T} - \mathrm{T}_{\min}}{\mathrm{T}_{\max} - \mathrm{T}_{\min}} \right) - (1 - \alpha) \left(\frac{rtt - rtt_{\min}}{rtt_{\max} - rtt_{\min}} \right) \tag{6}$$

T represents the current observed throughput. $\mathrm{T}_{\max}$ represents the historically observed maximum throughput. rtt represents the average round trip time during the observation period. $rtt_{\min}$ represents the minimum round-trip time observed during the observation period. α is the weight factor, which is a super parameter, reflecting the weight ratio of throughput and RTT to the reward value. α determines that the optimization goal of congestion control algorithm is more focused on throughput or RTT. In the experiment, $\alpha = 0.5$ is chosen to balance throughput and RTT.

5. Results and Discussion

5.1. Experimental Environment

A high-performance server is used to measure and compare the performance of TCP-DQN

(1) CPU: Intel (R) Xeon (R) silver 4110 CPU @ 2.10 GHz processor;
(2) Memory: 32 g DDR4 memory;
(3) GPU: NVIDIA Titan V;
(4) Operating system: Red Hat 4.8.5-28.

We implemented TCP-DQN method and tested its performance in virtual data space. The TCP-DQN is compared with the representative TCP congestion control algorithms such as cubic, NewReno, and HighSpeed. NewReno is a classic congestion control algorithm; HighSpeed is a congestion control algorithm designed for the high-speed network environment; cubic is the default TCP congestion control algorithm since Linux kernel version 2.6.19. According to our real test, the average network bandwidth of virtual data space is close to 67.8 Mbps, the delay is close to 56 ms, and the packet loss rate is close to 0.01%. The time step of TCP-DQN is set to 0.1 s. After training with 60,000 steps, the reward obtained has become stable, indicating that the algorithm has converged.

As shown in Figure 3, we use a simple Muti-Layer Perception to build our DQN network architecture. There are two hidden layers and each layer has 64 nodes. The input layer has seven nodes, which are used to observe the current state. The output layer has 101 nodes, each represents the Q value of the corresponding action. The first node represents the action -50, the second node represents the action $-49, \ldots$, and the last node represents the action 50. If exploration is not triggered in the current time-step, then the action with max Q value will be selected. For example, if the first output node has the max Q value, then the action -50 is selected. According to Equation (5), the cwnd should be reduced by 50 segments.

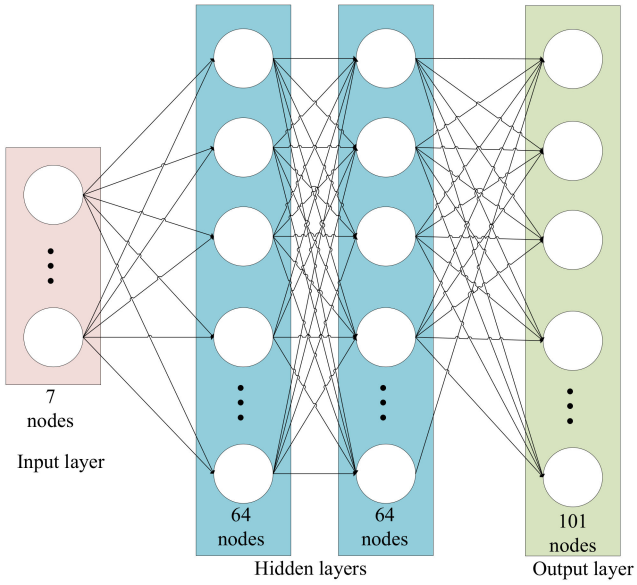

Figure 3. DQN architecture.

We set the core parameters of DQN as follows: gamma = 0.99, learning_rate = 0.0005, buffer_size = 50000, exploration_fraction = 0.1, exploration_final_eps = 0.02, exploration_initial_eps = 1.0. The exploration_fraction indicates the fraction of entire training period over which the exploration rate is annealed; exploration_final_eps indicates the final value of random action probability; exploration_initial_eps indicates the initial value of random action probability. At the first time-step, the agent will definitely explore the environment because the initial value of random action probability is set to 1. Then, the exploration rate will be annealed. When the exploration terminates (the exploration fraction reaches up to 10%), the exploration rate is annealed to 0.02.

5.2. Throughput Comparison

We define the throughput as the size of the bytes acknowledged by the sender per second. As shown in Figure 4, the network throughput of TCP-DQN is about twice than that of HighSpeed and cubic methods, and is about three times than that of NewReno method. In addition, the throughput of TCP-DQN is quite stable compared with the other three methods. Figure 5 shows the cumulative probability density function curve

of throughput to compare the four congestion control methods. The TCP-DQN performs much better than the other three methods. About 80% of the sampling points of TCP-DQN have a throughput of more than 6 MB/s. The result suggests that the traditional congestion control algorithms, such as NewReno, cannot adapt to the WAN environment of virtual data space, and is not suitable for data migration in virtual data space. Cubic and HighSpeed have significant performance improvements over NewReno, but they still cannot make full use of the available bandwidth to realize high-speed transmission. While TCP-DQN has the best performance; after a certain amount of learning, it can make full use of network bandwidth to realize efficient data migration in virtual data space.

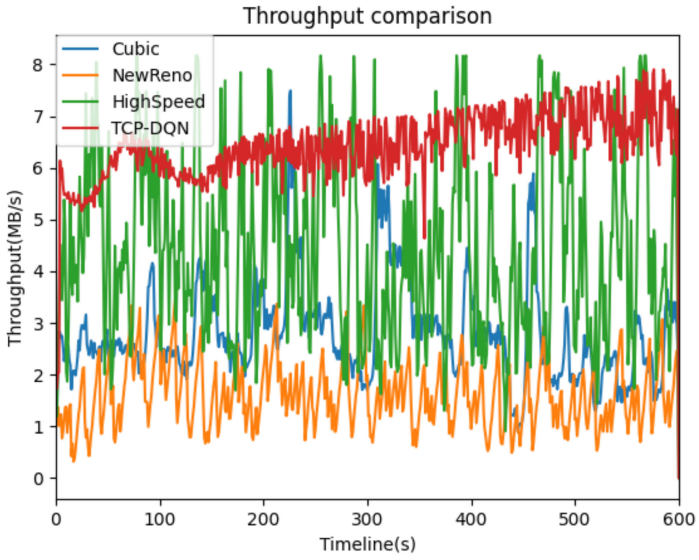

Figure 4. Throughput performance comparison.

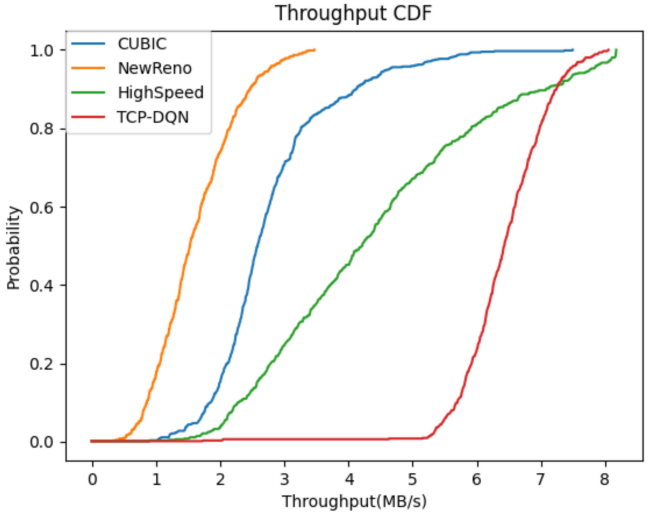

Figure 5. Comparison of cumulative probability density distribution of throughput.

5.3. RTT Comparison

RTT represents the amount of time it takes for a packet to be sent plus the amount of time it takes for an acknowledgement of that packet to be received, which reflects the network delay. As shown in Figure 6, in general, the RTT values of TCP-DQN and HighSpeed method are lower than that of the other two methods. This is because the TCP- DQN tries to balance the throughput and RTT. Figure 7 shows the comparison of cumulative probability density functions of RTT values. TCP-DQN and HighSpeed have better network latency. In total, 90% RTT of TCP-DQN and HighSpeed are less than

58 ms, whereas only 82% RTT of cubic are less than 58 ms; 76% RTT of NewReno are less than 58 ms. The result suggests that the traditional NewReno algorithm not only has poor throughput, but also has poor time latency. The time latency of cubic is better than NewReno, but still can be improved. HighSpeed works well on WAN. TCP-DQN has a same time latency compared with Highspeed.

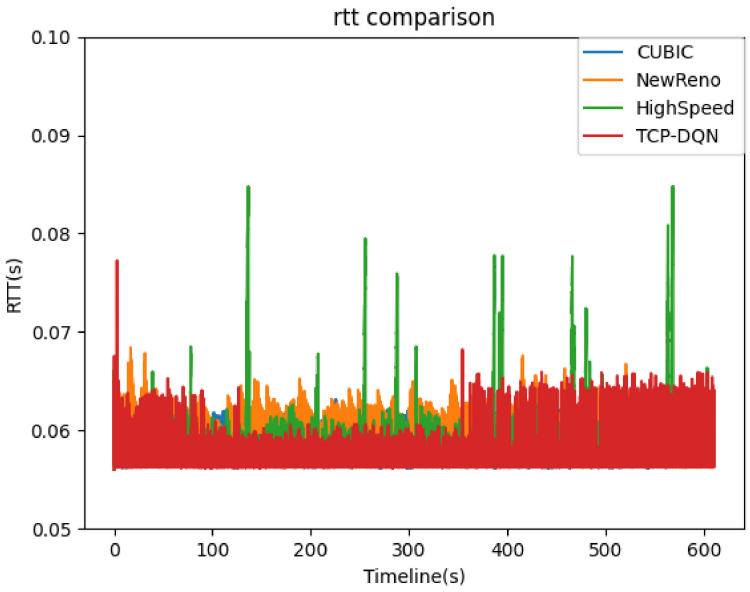

Figure 6. RTT comparison.

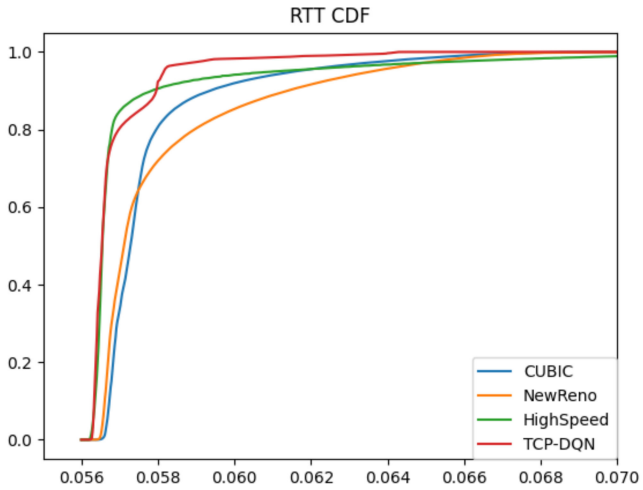

Figure 7. Throughput performance comparison.

5.4. Packet Loss Rate Comparison

We compared the packet loss rate of the four algorithms. As shown in Figure 8, the packet loss rate of the four algorithms is close to 0.01%, which is consistent with the packet loss rate of the WAN. The packet loss rate of TCP–DQN is 0.0124%, which is slightly higher than that of the other three algorithms. This is because TCP–DQN sends the most packets, and some packets are discarded because the router cache is full, resulting in packet loss.

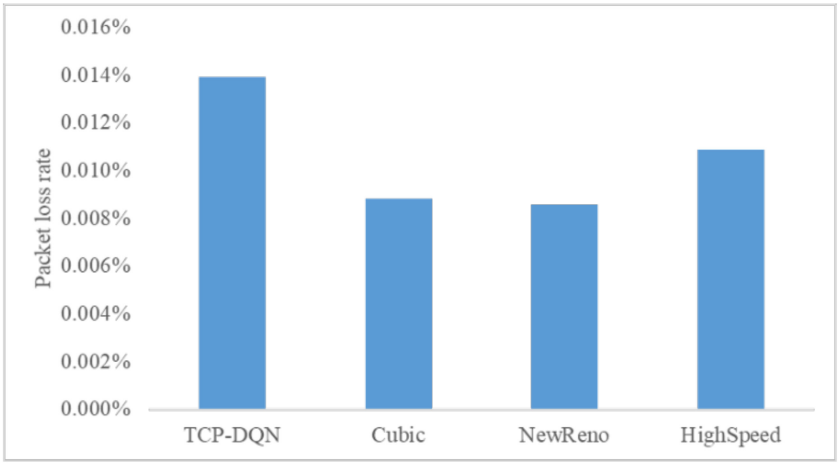

Figure 8. Comparison of packet loss rate.

6. Conclusions

Virtual data space is of great significance for aggregating national high-performance computing resources. Efficient and reliable data transmission is the core technology of building virtual data space. We propose a TCP congestion control algorithm based on near-end policy optimization algorithm to achieve efficient and reliable data migration in virtual data space.

The main conclusions are as follows:

(1) This paper proposes a TCP congestion control algorithm based on a DQN algorithm, which abstracts the TCP congestion control process based on reinforcement learning into a partially observable Markov decision process. By referring to the mainstream algorithm, the state space and action space are reasonably designed, and the reasonable reward function is designed.

(2) The throughput of DQN is up to 7 MB/s on average when the bandwidth is 10 MB/s. Compared with highspeed, cubic and NewReno algorithms, the throughput of TCP-DQN can reach more than 2–3 times. The RTT of DQN is 58 ms on average when the minimal RTT of the physical link is 56 ms.

In the future, we will test the performance of TCP-DQN method to obtain more results, and further propose optimization methods according to the test results to better serve the national high-performance computing environment.

Author Contributions: Conceptualization, Y.W.; methodology, L.W.; software, L.W.; validation, X.D.; writing—original draft preparation, Y.W.; writing—review and editing, L.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not Applicable, the study does not report any data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Floyd, S.; Henderson, T. RFC2582: The NewReno Modification to TCP's Fast Recovery Algorithm. 1999. Available online: https://dl.acm.org/doi/10.17487/RFC2582 (accessed on 29 September 2021).
2. Brakmo, L.S.; Peterson, L.L. Tcp Vegas—End-to-End Congestion Avoidance on a Global Internet. *IEEE J. Sel. Areas Commun.* **1995**, *13*, 1465–1480. [CrossRef]
3. Floyd, S. HighSpeed TCP for Large Congestion Windows. Rfc: 2003. Available online: https://www.hjp.at/doc/rfc/rfc3649.html (accessed on 29 September 2021).
4. Ha, S.; Rhee, I.; Xu, L. CUBIC: A new TCP-friendly high-speed TCP variant. *ACM Sigops Oper. Syst. Rev.* **2008**, *42*, 64–74. [CrossRef]
5. Xiao, L.; Lu, X.; Xu, D.; Tang, Y.; Wang, L.; Zhuang, W. UAV Relay in VANETs Against Smart Jamming with Reinforcement Learning. *IEEE Trans. Veh. Technol.* **2018**, *67*, 4087–4097. [CrossRef]

6. Niroui, F.; Zhang, K.; Kashino, Z.; Nejat, G. Deep Reinforcement Learning Robot for Search and Rescue Applications: Exploration in Unknown Cluttered Environments. *IEEE Robot. Autom. Lett.* **2019**, *4*, 610–617. [CrossRef]
7. Huang, S.; Lv, B.; Wang, R.; Huang, K. Scheduling for Mobile Edge Computing with Random User Arrivals: An Approximate MDP and Reinforcement Learning Approach. *IEEE Trans. Veh. Technol.* **2020**, *69*, 7735–7750. [CrossRef]
8. Cao, Z.; Lin, C.; Zhou, M.; Huang, R. Scheduling Semiconductor Testing Facility by Using Cuckoo Search Algorithm with Reinforcement Learning and Surrogate Modeling. *IEEE Trans. Autom. ENCE Eng.* **2019**, *16*, 825–837. [CrossRef]
9. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
10. Li, W.; Zhou, F.; Chowdhury, K.R.; Meleis, W.M. QTCP: Adaptive Congestion Control with Reinforcement Learning. *IEEE Trans. Netw. Sci. Eng.* **2018**, *6*, 445–458. [CrossRef]
11. Watkins, C.J.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
12. Sun, G.; Zhou, R.; Sun, J.; Yu, H.; Vasilakos, A.V. Energy-efficient provisioning for service function chains to support delay-sensitive applications in network function virtualization. *IEEE Internet Things J.* **2020**, *7*, 6116–6131. [CrossRef]
13. Sun, G.; Li, Y.; Yu, H.; Vasilakos, A.V.; Du, X.; Guizani, M. Energy-efficient and traffic-aware service function chaining orchestration in multi-domain networks. *Future Gener. Comput. Syst.* **2019**, *91*, 347–360. [CrossRef]
14. Brakmo, L.S.; O'Malley, S.W.; Peterson, L.L. TCP Vegas: New techniques for congestion detection and avoidance. In Proceedings of the Conference on Communications Architectures, Protocols and Applications, London, UK, 31 August–2 September 1994; pp. 24–35.
15. Gerla, M.; Sanadidi, M.; Wang, R.; Zanella, A.; Casetti, C.; Mascolo, S. TCP Westwood: Congestion window control using bandwidth estimation. In *Global Telecommunications Conference, 2001. GLOBECOM '01*; IEEE: New York, NY, USA, 2001; Volume 3, pp. 1698–1702.
16. Tan, K.; Song, J.; Zhang, Q.; Sridharan, M. A compound TCP approach for high-speed and long distance networks. In Proceedings of the IEEE INFOCOM 2006, Barcelona, Spain, 23–29 April 2006.
17. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR: Congestion-based congestion control. *Queue* **2016**, *14*, 20–53. [CrossRef]
18. Busch, C.; Kannan, R.; Vasilakos, A.V. Approximating Congestion+ Dilation in Networks via "Quality of Routing" Games. *IEEE Trans. Comput.* **2011**, *61*, 1270–1283. [CrossRef]
19. Liu, L.; Song, Y.; Zhang, H.; Ma, H.; Vasilakos, A.V. Physarum optimization: A biology-inspired algorithm for the steiner tree problem in networks. *IEEE Trans. Comput.* **2013**, *64*, 818–831.
20. Dvir, A.; Vasilakos, A.V. Backpressure-based routing protocol for DTNs. In Proceedings of the ACM SIGCOMM 2010 Conference, New Delhi, India, 30 August–3 September 2010; pp. 405–406.
21. Ji, Y.; Wang, J.; Xu, J.; Fang, X.; Zhang, H. Real-time energy management of a microgrid using deep reinforcement learning. *Energies* **2019**, *12*, 2291. [CrossRef]
22. Fang, Y.; Huang, C.; Xu, Y.; Li, Y. RLXSS: Optimizing XSS detection model to defend against adversarial attacks based on reinforcement learning. *Future Internet* **2019**, *11*, 177. [CrossRef]
23. Yi, J.-H.; Xing, L.-N.; Wang, G.-G.; Dong, J.; Vasilakos, A.V.; Alavi, A.H.; Wang, L. Behavior of crossover operators in NSGA-III for large-scale optimization problems. *Inf. Sci.* **2020**, *509*, 470–487. [CrossRef]
24. Le, T.; Szepesvári, C.; Zheng, R. Sequential learning for multi-channel wireless network monitoring with channel switching costs. *IEEE Trans. Signal Process.* **2014**, *62*, 5919–5929. [CrossRef]
25. Liu, N.; Li, Z.; Xu, J.; Xu, Z.; Lin, S.; Qiu, Q.; Tang, J.; Wang, Y. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, USA, 5–8 June 2017; pp. 372–382.
26. Xu, Z.; Wang, Y.; Tang, J.; Wang, J.; Gursoy, M.C. A deep reinforcement learning based framework for power-efficient resource allocation in cloud RANs. In Proceedings of the 2017 IEEE International Conference on Communications (ICC), Paris, France, 21–25 May 2017; pp. 1–6.
27. Lu, T.; Chen, X.; McElroy, M.B.; Nielsen, C.P.; Wu, Q.; Ai, Q. A Reinforcement Learning-Based Decision System for Electricity Pricing Plan Selection by Smart Grid End Users. *IEEE Trans. Smart Grid* **2021**, *12*, 2176–2187. [CrossRef]
28. Sp, A.; Paa, B.; Jmma, B. Energy-conscious optimization of Edge Computing through Deep Reinforcement Learning and two-phase immersion cooling. *Future Gener. Comput. Syst.* **2021**, *125*, 891–907.
29. Jung, C.; Shim, D.H. Incorporating Multi-Context into the Traversability Map for Urban Autonomous Driving Using Deep Inverse Reinforcement Learning. *IEEE Robot. Autom. Lett.* **2021**, *6*, 1662–1669. [CrossRef]
30. Deltetto, D.; Coraci, D.; Pinto, G.; Piscitelli, M.; Capozzoli, A. Exploring the Potentialities of Deep Reinforcement Learning for Incentive-Based Demand Response in a Cluster of Small Commercial Buildings. *Energies* **2021**, *14*, 2933. [CrossRef]
31. Fischer, F.; Bachinski, M.; Klar, M.; Fleig, A.; Müller, J. Reinforcement learning control of a biomechanical model of the upper extremity. *Sci. Rep.* **2021**, *11*, 14445. [CrossRef] [PubMed]
32. Habachi, O.; Shiang, H.-P.; Van Der Schaar, M.; Hayel, Y. Online learning based congestion control for adaptive multimedia transmission. *IEEE Trans. Signal Process.* **2013**, *61*, 1460–1469. [CrossRef]

33. Hemmati, M.; Yassine, A.; Shirmohammadi, S. An online learning approach to QoE-fair distributed rate allocation in multi-user video streaming. In Proceedings of the 2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS), Gold Coast, Australia, 15–17 December 2014; pp. 1–6.
34. Van Der Hooft, J.; Petrangeli, S.; Claeys, M.; Famaey, J.; De Turck, F. A learning-based algorithm for improved bandwidth-awareness of adaptive streaming clients. In Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, Canada, 11–15 May 2015; pp. 131–138.
35. Cui, L.; Yuan, Z.; Ming, Z.; Yang, S. Improving the Congestion Control Performance for Mobile Networks in High-Speed Railway via Deep Reinforcement Learning. *IEEE Trans. Veh. Technol.* **2020**, *69*, 5864–5875. [CrossRef]
36. Xiao, K.; Mao, S.; Tugnait, J.K. TCP-Drinc: Smart congestion control based on deep reinforcement learning. *IEEE Access* **2019**, *7*, 11892–11904. [CrossRef]
37. Bachl, M.; Zseby, T.; Fabini, J. Rax: Deep reinforcement learning for congestion control. In Proceedings of the ICC 2019–2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6.
38. Wang, S.; Bi, J.; Wu, J.; Vasilakos, A.V.; Fan, Q. VNE-TD: A virtual network embedding algorithm based on temporal-difference learning. *Comput. Netw.* **2019**, *161*, 251–263. [CrossRef]