applied sciences

MDPI

Article

Deep Reinforcement Learning-Based Network Routing Technology for Data Recovery in Exa-Scale Cloud Distributed Clustering Systems

Dong-Jin Shin [1] and Jeong-Joon Kim [2,*]

[1] Department of Computer Engineering, Anyang University, Anyang-si 14058, Korea; djshin@gs.anyang.ac.kr
[2] Department of ICT Convergence Engineering, Anyang University, Anyang-si 14058, Korea
* Correspondence: jjkim@anyang.ac.kr; Tel.: +82-31-467-0700

Abstract: Research has been conducted to efficiently transfer blocks and reduce network costs when decoding and recovering data from an erasure coding-based distributed file system. Technologies using software-defined network (SDN) controllers can collect and more efficiently manage network data. However, the bandwidth depends dynamically on the number of data transmitted on the network, and the data transfer time is inefficient owing to the longer latency of existing routing paths when nodes and switches fail. We propose deep Q-network erasure coding (DQN-EC) to solve routing problems by converging erasure coding with DQN to learn dynamically changing network elements. Using the SDN controller, DQN-EC collects the status, number, and block size of nodes possessing stored blocks during erasure coding. The fat-tree network topology used for experimental evaluation collects elements of typical network packets, the bandwidth of the nodes and switches, and other information. The data collected undergo deep reinforcement learning to avoid node and switch failures and provide optimized routing paths by selecting switches that efficiently conduct block transfers. DQN-EC achieves a 2.5-times-faster block transmission time and 0.4-times-higher network throughput than open shortest path first (OSPF) routing algorithms. The bottleneck bandwidth and transmission link cost can be reduced, improving the recovery time approximately twofold.

Keywords: deep reinforcement learning; network routing; deep q-network; erasure coding; distributed clustering systems

Citation: Shin, D.-J.; Kim, J.-J. Deep Reinforcement Learning-Based Network Routing Technology for Data Recovery in Exa-Scale Cloud Distributed Clustering Systems. *Appl. Sci.* **2021**, *11*, 8727. https://doi.org/10.3390/app11188727

Academic Editors: Seongsoo Cho and Bhanu Shrestha

Received: 12 August 2021
Accepted: 17 September 2021
Published: 18 September 2021

1. Introduction

Owing to the recent development of technologies such as smartphones, IoT, artificial intelligence, and big data, large-capacity big data are being generated and utilized. We previously used a replication technology-based distributed file system to store such data; however, with the spread of cloud computing, we have recently begun storing big data more efficiently using an erasure coding-based distributed file system [1–3]. Replication techniques divide the original data into multiple data blocks and store them on each distributed server through n-duplex replication. Because such techniques replicate n data, the efficiency of disk storage is extremely low owing to the disadvantage of data availability n times. To improve this, a distributed file system based on the erasure coding technique is used. The erasure coding technique divides the original data into data blocks and generates parity blocks through encoding operations. Decoding is a method for recovering original data through an operation that combines distributed and stored data blocks with parity blocks [4]. In such erasure coding-based distributed file systems, although space efficiency problems have been overcome, a large disk overhead occurs during encoding and decoding. Furthermore, block transfers are not efficiently achieved owing to the occurrence of countless variables through the routing path of the network. When decoding, it is important to efficiently send data and parity blocks to the destination node without

a network delay. To address this problem, continuous research has been conducted to optimize the network routing paths. Typically, we establish a server that can aggregate the data and reduce the number of transmission links required for transferring blocks while addressing network bottlenecks [5,6]. However, such studies have not focused on detecting dynamic bandwidths that occasionally change in actual network topologies. In a real-world network topology, the bandwidth is transmitted by multiple packets to manage the health of each server, as well as the block transfers related to erasure coding.

SDN is a new network technology that separates the control of switches and routers within a network from the data area [7]. SDN controllers allow for the central control of the entire network and network management. In addition, SDN controllers have a global view of the entire network, allowing comprehensive network information to be collected. However, although data can be collected through an SDN for efficient network management, efficient packet delivery remains a challenge in terms of network throughput and latency. Therefore, optimizing network resources requires deploying and resolving routing paths that are optimized for the network. The optimized routing path problem is how to efficiently forward data traffic from the source node to all reachable destination nodes and switches, and to find routing paths to destination nodes that conduct decoding operations. Traditionally used routing path algorithms in networks include OSPF [8]. However, OSPF cannot predict network changes because it does not consider dynamic changes in bandwidth based on the shortest paths. In other words, the optimal routing path cannot be found in a network with a dynamically changing bandwidth and traffic distribution.

In summary, in an erasure coding-based distributed file system, the network routing methodology has difficulty identifying real-time changes in the network bandwidth and failed nodes because it does not take real data into account. In addition, efficient network routing paths are difficult to create because supervised learning-based network routing methodologies do not consider data from real networks. Although reinforcement-based network routing can generate efficient routing paths compared to supervised learning, it is difficult to apply the model's suitability and network environment.

We therefore implement DQN-EC, which leverages erasure coding to provide optimized routing paths based on deep reinforcement learning to address the throughput, bandwidth, and transmission link cost issues that arise when transferring blocks. Instead of building an accurate mathematical model for the underlying network, DQN-EC solves this problem through the experience of deep reinforcement learning. DQN-EC applies reward values designed for the collection of comprehensive network information for the state, configuring one-to-many networks to select nodes and switches for the next link, and optimizing the network throughput, bandwidth, and transmission costs. As the learning progresses, the agent learns a policy that predicts the future behavior of the underlying network and finally proposes improvements to the routing path between the source and destination nodes.

Section 2 describes the basic principles behind the erasure coding-based distributed file system as well as DQN and deep reinforcement learning. Section 3 discusses research examples related to the methodology applied to solve the problem of machine learning (supervised and reinforcement learning)-based network routing paths and the methodology for network optimization of erasure coding. Section 4 discusses the data items used in the proposed DQN-EC learning, the design of neural network models, and the configuration of the hyperparameters [9]. In Section 5, we describe evaluation results showing the improvements when applying DQN-EC to the erasure coding network topology. Finally, some concluding remarks are provided.

2. Background Principle

This section introduces the principles of erasure coding and the fundamentals of reinforcement learning, which consists of a network environment for an experimental evaluation.

2.1. Principle of Erasure Coding

The typical method of data storage in a distributed file system is to divide the original data into blocks of data of a certain size using replication techniques, and to replicate the divided blocks of n and store them separately on multiple nodes. However, traditional replication technique-based distributed file systems store replicated blocks of data, resulting in an n-fold increase in spatial efficiency. Therefore, to improve this, a recent erasure coding technology-based distributed file system that has been extended from existing replication technique-based distributed file systems is being utilized. Erasure coding is represented in Figure 1 as a technique for storing data through the encoding and recovery of the decoding data, unlike how data blocks are divided into constant sizes, replicated, and stored [10–12].

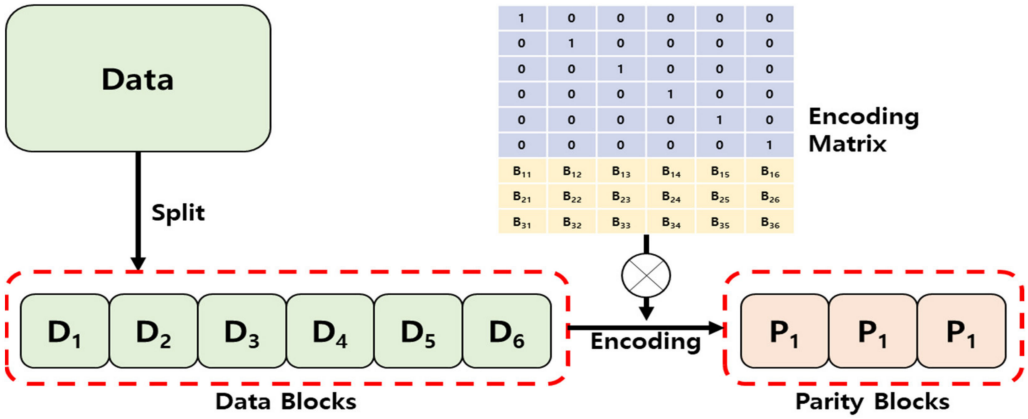


Figure 1. Encoding process of erasure coding.

Figure 1 shows a structure in which a Reed-Solomon (RS) code (6, 3) is used to construct six data blocks and three parity blocks generated through encoding. That is, when the original data are entered, they are divided by the RS erasure coding (6, 3) using as many blocks of data. In addition, each data block generates parity blocks through an encoding matrix and calculations, and thus all blocks (data blocks and parity blocks) are distributed and stored for each disk or network environment they use, as shown in Equation (1).

$$P_i = \sum_{j=1}^{k} \alpha_{i,j} D_j \tag{1}$$

The parity block P_i is generated by multiplying the data block D_j by the encoding matrix coefficient $\alpha_{i,j} (i = row,\ j = column)$ in Figure 1; although the decoding operations have been expressed through multiplication, they are actually computed through various algorithms, such as the RS erasure coding, liberation, AVX:CRS, and XOR algorithms.

Figure 2 shows the behavioral process of the decoding operations in which nodes possessing data block 2 and parity block 1 fail owing to factors such as a network connectivity delay or a natural disaster and do not respond. However, a failure is recoverable by decoding the parity block and the original data block generated through encoding. The number of blocks that can be decoded and recovered reaches only up to three, which is the number of parity blocks, when configured in an RS code (6, 3). In other words, three out of nine blocks can be recovered during a failure, as shown in Equation (2).

$$d^* = (\beta_1, \beta_2, \beta_3, \ldots, \beta_i) \times \begin{pmatrix} P_1 \\ P_2 \\ P_3 \\ \ldots \\ P_4 \end{pmatrix} = \sum_{i=1}^{k} \beta_i P_i \tag{2}$$

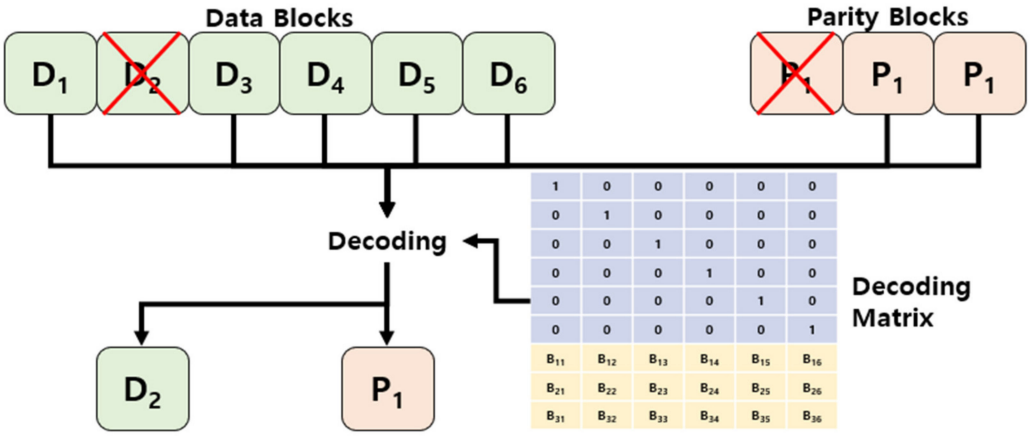


Figure 2. Decoding process of erasure coding.

Blocks d* that require recovery are regenerated into new nodes by decoding the decoding matrix coefficients $(\beta_1, \beta_2, \beta_3, \ldots, \beta_i)$ and the generated parity blocks $(P_1, P_2, P_3, \ldots, P_i)$.

2.2. Principle of Deep Q-Network

Q-learning, the reinforcement learning underlying a DQN, teaches agents that interact with the environment. In the early stages of learning, the agent learns slowly over time, taking random actions in various situations called 'states'. Policies are configured by updating all values to the table-type Q-table, depending on the status operation, and then referring to them and selecting the action. This is expressed in Figure 3 and Equation (3) as a way to update the Q-table to maximize the final cumulative reward while sequentially progressing the episode [13].

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a') \tag{3}$$

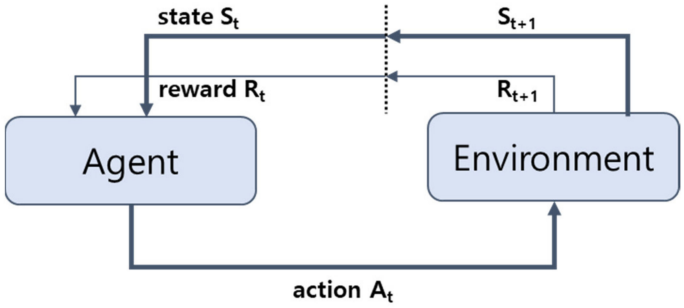


Figure 3. Q-learning process.

The sum $Q(s,a)$ of all rewards that can be received when taking action a in the current state s can be calculated as the sum of the immediate rewards that can be taken and received in the future. On the right side of the equation, $r(s,a)$ represents the immediate reward value to be received when action a is taken in the current state s. In addition, s' is the very next state that is reached by taking action a in the current state s, and $\max_a Q(s',a')$ is the maximum value of the reward that can be received in the next state s'. The goal of the agent is to select an action that maximizes this value. In addition, γ is a value called a discount, which controls the importance of a future value. The larger the value of γ is, the greater the value of the future reward, and the smaller the value is, the greater the importance of the immediate reward.

Because Q-learning must have all values for a state-action corresponding to the Q-table, there is a limit to how many states can be stored. To improve this, there was a way to learn Q-learning using neural network models; however, when applying high correlation

problems between learning data and gradient descent methods, the target value, Target, is moving. Therefore, using neural network models instead of a Q-table, DQN combines Q-learning with deep learning, enabling neural network models to approximate the Q values, and constructs the target neural networks separate from a replay of the experience. This is expressed in Figure 4 and Equation (4) [14].

$$Cost = \left[\left(r(s,a) + \gamma \max_{a} Q(s',a';\theta') - Q(s,a;\theta)\right)\right]^2 \quad (4)$$

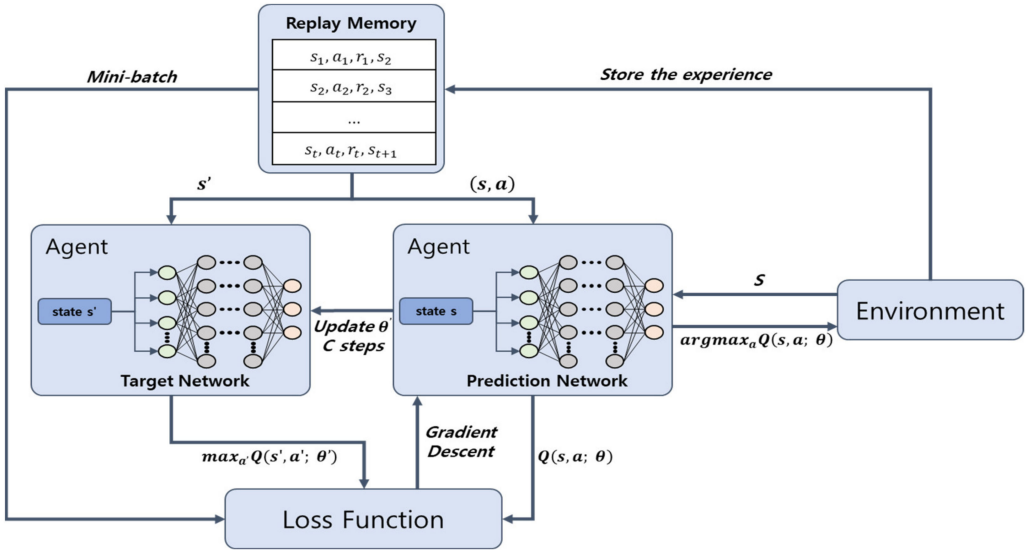


Figure 4. Deep Q-network process.

In the existing Q-learning in Equation (3), the final goal is to converge and reach the true value when the left and right sides are the same. The DQN shown in Equation (4) also has the same goal, i.e., to learn the neural network models in the direction of minimizing the difference between the two values through the loss function of the neural network models. In addition, in Equation (4), $Q(s',a';\theta')$ is the Q-Target and refers to the state, action, and parameters used in a neural network applied in deep-learning-based reinforcement learning. The target network optimizes θ of the real network, holding the weight of the value of the Q-Target θ', even when minimizing the loss function, similar to how it holds the target when conducting supervised learning. In other words, when learning a predictive network representing $Q(s,a;\theta)$, as shown in Figure 4, a target network representing $Q(s',a';\theta')$ is created separately to fix the weights. It then passes the weights of the forecasting network to the target network at every step. This learning is not affected by the weights the target is learning when narrowing the difference between the values of the target network and the predictive network. Therefore, more stable learning is possible than with Q-learning.

As shown in Figure 4, the DQN takes actions that are considered optimal at every moment through the learned neural network model and stores data on the status, behavior, and rewards in the playback memory until the end of the episode. In other words, if the batch size required for learning is n, the agent takes n actions and stores n data sequentially in the replay memory. As a difference from Q-learning, it does not learn from the recently stored data in replay memory, and instead randomly extracts mini-batch-sized samples from n data stored thus far to learn the neural network. This is because, if the recently stored data in the replay memory are used as is, the correlation between the data will be too large, and learning is therefore not properly applied.

Therefore, DQN has two improvements. First, because the network is separated using two deep learning neural network models, the learning instability of the target network

can be improved. Second, the correlation among the data can be resolved because they are trained by randomly extracting mini-batch sized samples using replay memory.

3. Related Studies and Motivation

This section first introduces algorithms and methodologies related to erasure coding, which will consist of a network environment for an experimental evaluation. Second, we introduce network routing algorithms and methodologies that utilize supervised learning and reinforcement learning applied in machine learning. Finally, the need for this study is explained.

3.1. Studies Related to Erasure Coding

Erasure coding has a significantly increased space efficiency, taking up less disk space than traditional replication-technique-based distributed file systems. However, because various cost problems, such as disk overhead and network bottlenecks, occur in data encoding and decoding, various algorithms and methodologies have been proposed to resolve such issues. A distributed erasure-coding-based file system can divide the data encoding/decoding process into five stages: client overhead, master process, parity calculation, data distribution, and slave process. A cost analysis that occurs when the network bandwidth is sequentially increased to 1 G, 10 G, 40 G, and 100 G suggests that client overhead and data distribution are the costliest [15,16]. Disk contention availability is applied to efficiently distribute multiple recovery requests using a parallel recovery to reduce client overhead costs, random chunk allocation is used to increase the load balancing efficiency, and asynchronous recovery is utilized to increase the concurrency [17]. To reduce the costs incurred during data distribution, a study has significantly reduced the recovery bandwidth associated with the decoding performance by appropriately adjusting the recovery speed [18]. A study on a top-down data transfer technique also used distributed data calculations for updating the parity nodes [19]. In another previous study, the recovery of failed data was achieved at a small scale throughout the storage node, reducing the recovery time to approximately equal the normal read time for the same amount of data [20]. Other studies [3,4] also identified and avoided the traffic transmission paths of nodes and associated nodes where bottlenecks occur, thereby reconfiguring the routing paths and eliminating bottlenecks across the entire system to reduce the traffic transmission costs.

In erasure-coding-related studies, network routing methodologies and algorithms improve the routing paths to the destination nodes where the data and parity blocks are transmitted when decoding the data, reducing the costs of bottlenecks and transmission links. However, because aggregated hardware-based switches are used to reduce the bottlenecks and traffic transmission costs, additional hardware costs are required; in addition, bandwidth is always evaluated at a fixed value and the block transmission rates are inefficient when the bandwidth is suddenly lowered. Therefore, because DQN-EC collects dynamically changing data in a network environment using erasure coding and identifies failed switches, DQN neural network model learning can efficiently yield the optimal routing paths and transfer blocks.

3.2. Supervised Learning-Based Network Routing

Supervised learning is a method of learning the correct answer data to predict values and categories for new data [21]. In [22], the authors proposed graph-aware deep-learning-based intelligent routing to solve such problems as slow fusion and degradation that occur when constructing complex dynamic networking conditions. In addition, in [23], the authors proposed an ML-assisted left-loaded algorithm, which is a combination of the Bayes classifier and the existing least-loaded algorithm, configured to solve the connection failure problem in the following situations of a circuit-switched network. Moreover, the authors of [24,25] proposed a RouteNet algorithm to help optimize routing schemes for mean delays by leveraging the ability of graph neural networks (GNNs) to learn and model graph-based information. In [26], the authors proposed a deep learning routing algorithm that

configures a deep-learning-based neural network, with the input of the model consisting of a set of numbers indicating the number of packets forwarded through each node of the network, and the output using network traffic information in the form of an interface for forwarding packets. In addition, in [27], the authors studied the input of the deep learning model as a routing strategy based on the deep learning architecture. The deep believe architecture (DBA) was defined between the number of inbound packets observed on each router and the time interval, and an algorithm was proposed to construct a restricted Boltzmann machine on the detailed hidden layer of the DBA. In [28], the authors also proposed an automated protocol design method for controlling and managing a network by applying two important aspects of the GNN model—i.e., distributing the topology information between different nodes and calculating the topology and link-weighted paths. In addition, in [29], the authors also proposed MLProp by leveraging a variety of input parameters—such as the buffer capacity and node access rates, respectively—to indicate whether packets can be delivered successfully along these links, improving the network link performance and probabilistic routing approaches to intermittent opportunistic networking. Finally, in [30], the authors developed a flexible supervised learning framework through a training model that includes learning paths and all path pairs of mixed integral linear programming and proposed a routing methodology that utilizes deep neural network methods to minimize the identity of network systems instead of always selecting the shortest paths.

Related studies on supervised-learning-based network routing methodologies predict the routing paths by learning the machine learning models, such as naïve Bayes classification, decision tree, and deep learning-based neural network models. However, to conduct supervised learning, a dataset must be prepared in advance, and the performance varies greatly depending on the type of model used. In addition, although the correct labels for the datasets used must exist, it is difficult to produce an efficient predictive path because the correct labels can always vary depending on the bandwidth and traffic intensity generated by the actual network. Therefore, DQN-EC can yield efficient routing paths compared to studies related to supervised learning-based networks because the approach collects both static and dynamic data through SDN controllers and utilizes deep reinforcement learning, which does not select specific models.

3.3. Reinforcement-Learning-Based Network Routing

Reinforcement learning is a method of learning how to behave in each environment, and proceeds by receiving feedback as positive or negative rewards for the actions conducted by the agent under various situations [31]. In [32], the authors proposed a QAR algorithm for achieving time-efficient and adaptive QoS provisioning packet delivery through reinforcement learning and QoS-aware compensation capabilities. Furthermore, the simulation results demonstrate that QAR outperforms existing learning solutions and provides fast convergence with QoS provisioning to facilitate practical implementation in large software-service-defined networks. In [33], the authors proposed RL4Net, which is a type of deep deterministic policy gradient-based Q-learning. The state is encapsulated into the amount of traffic flowing between each pair of routers (total size), which corresponds to a weight update determining how the router selects the interface to forward, and the reward is calculated based on the delay. In addition, in [34], the authors proposed applying Q-learning directly to packet routing, and proposed an efficient routing policy in dynamically changing networks without needing to know the network topology or traffic patterns in advance. In [35], the authors proposed CQ-routing, through which the learned routing policy under a high load shows more than twice the performance of Q-routing in terms of the average packet delivery time. It has also been indicated that CQ-routing can sustain higher load levels than both Q-routing and shortest path routing. Moreover, in [36], the authors proposed NNQ-routing using a neural network approximation to improve the scalability of Q-routing. The Q function was replaced by a three-layer-conceptron model, and the successful packet delivery of Q-learning adopted positive compensation, delay,

and loss packets as negative compensation. In [37], the authors proposed two adaptive reinforcement-learning-based spectral recognition routing protocols, with Q-learning and dual-reinforcement learning, respectively. Spectrum-aware DRQ routing learns the optimal routing policies 1.5-times faster than spectrum-aware Q-routing at low and middle network loads. Under high network loads, the routing policies learned are seven times better than those of spectrum-aware Q-routing. Moreover, in [38], the authors proposed critical flow rerouting-reinforcement learning (CFR-RL), a reinforcement learning-based scheme that learns policies to automatically select critical flows for traffic matrices. The state uses a traffic matrix for time t, and the action generates multiple actions for time step t at each state. A reward is used to reroute the critical flows to balance the link utilization and set up the flows to reflect the network performance. In [39], the authors also proposed positive and negative compensation values using network throughput and delay to optimize the reward for network throughput. After proper learning, the agent proposes dueling double-deep Q-learning-based RL-routing, which predicts the future behavior of the underlying network and suggests better routing paths between switches. Moreover, in [40], the researchers improved the problem of a poor response ability of neural networks when generating multicast routing trees by always learning the DQN for the same environment. In other words, we propose a multicast routing tree methodology that learns and responds to a new environment each time it learns.

Related studies on reinforcement-learning-based network routing methodologies yield routing paths by learning in the direction in which the routing nodes become agents and reduce the latency based on Q-learning without specific models. In particular, deep reinforcement learning is efficient because agents yield optimal routing paths through the learning of neural network models based on data collected from the network. However, related studies have not considered switches that fail in the routing paths, and studies on applying DQN when transferring requested data blocks, and parity blocks during the decoding of erasure coding remain insufficient. Therefore, DQN-EC leverages DQN, a type of deep-reinforcement-learning-based Q-Learning, to collect network data and yield efficient routing transmission paths.

3.4. Motivation

Figure 5 shows when a node and switch fail in an existing network routing path and the network routing transmission path is disconnected. In the event of a failure, it is difficult to find an efficient path to which links should be used to transmit data to the destination node. In other words, an optimized routing path is needed to measure the network bandwidth that changes in real time to avoid the link with the best link bandwidth and when nodes fail due to natural geological and other factors. In addition, when selecting links, reducing the link transfer cost as much as possible should also be considered, as mentioned in the erasure coding-based improved study.

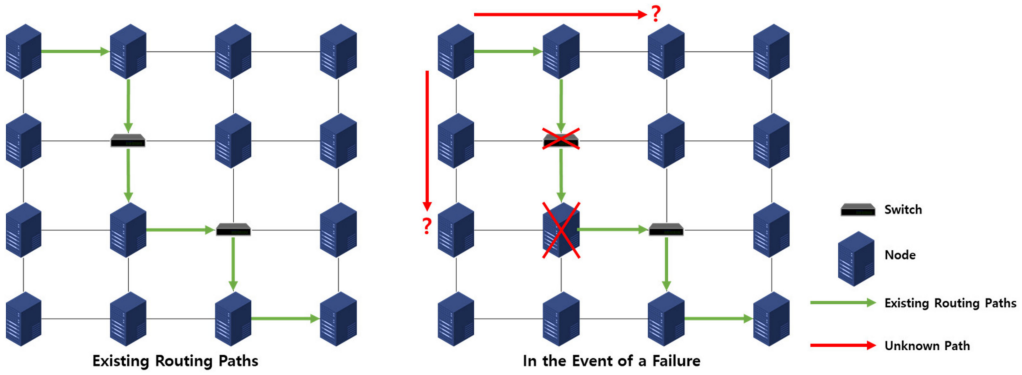


Figure 5. Example of a failure in a network routing path.

Therefore, in this study, we apply DQN-based deep reinforcement learning to a network environment configured with an erasure-coding-based distributed file system. In addition, we propose a methodology that adds not only the dataset related to network routing as input values to the neural network model applied, but also the input parameter values required for decoding in erasure coding, and modifies the layers used in the neural network model.

4. System Design

This section discusses the methodology used for applying DQN-based network routing algorithms to the erasure coding network topology.

4.1. Overview

Figure 6 shows the overall structure of applying the DQN-based network routing algorithm to the erasure coding network topology environment proposed in this paper. The algorithm utilizes SDN controllers, a network architecture approach that can intelligently and centrally control or program networks using software applications. Therefore, it can collect relevant data from the erasure coding network topology environment.

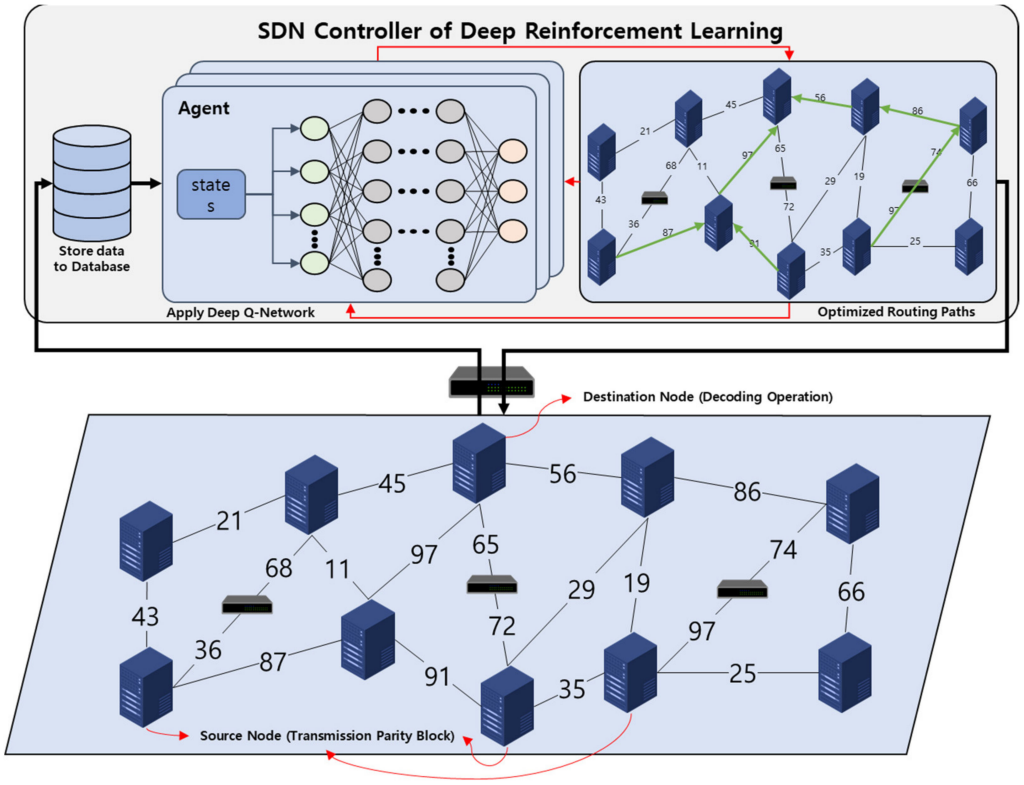


Figure 6. Structure of the DQN-based EC network routing architecture.

Data required when configuring network routing paths using dynamically changed network bandwidth and failed node and switch information are collected and stored in the database at regular time intervals. The stored data correspond to the input layer parameters of the neural network model in a DQN-based network routing algorithm.

4.2. Database Storage

The parameters entered into the DQN neural network model were learned from data collected over a period of time in the database. We collected general data from the network and data from the erasure coding network topology. The main data collected from the networks are the link bandwidth, link-available bandwidth, and hop count, and the main data collected from erasure coding are the failed nodes and switches, block sizes to be

transmitted, source nodes, and destination nodes. The list of data stored in the database and the parameters entered into the DQN neural network model are shown in Table 1.

Table 1. Input parameters of DQN neural network.

Notation	Symbol	Definition
$Time$	x_1	Time when network data was collected
$G(V, E)$	x_2	Location of node or switch (V consists of network nodes, E consists of all links between nodes)
RTT	x_3	Round tip time (Ms)
MSS	x_4	Maximum segment size (Byte)
AB_{sw}	x_5	Available bandwidth currently available for transmission
$PreBw_{sw}$	x_6	Previous link bandwidth (Gbps)
Now_{sw}	x_7	Location of the current node or switch
$NextBw_{sw}$	x_8	Next link bandwidth (Gbps)
$Prehob_{sw}$	x_9	Previous hop node or switch
$Nexthob_{sw}$	x_{10}	Next hop node or switch
$TNode_{count}$	x_{11}	The total number of node or switch
$PNode_{count}$	x_{12}	The number of parity nodes
$DNode_{count}$	x_{13}	The number of data nodes
$Node_{live}$	x_{14}	Current living nodes
$Node_{dead}$	x_{15}	Current dead nodes
$Block_{src}$	x_{16}	Source node that transmits blocks
$Block_{des}$	x_{17}	Destination node that be received blocks
$Block_{size}$	x_{18}	Transmitted size of blocks

The parameters entered into the DQN neural network model correspond to symbols x_1 through x_{18}, which corresponds to the period of time in which x_1 is collected. When transferring data and parity blocks, elements that can be collected and stored in a typical network topology correspond to x_2 through x_9, and elements that can be collected and stored in an erasure coding network topology correspond to x_{10} through x_{18}. In particular, among the data that can be collected through the SDN controller, dynamic data are stored through a calculation process conducted once before the data collection. The RTT sends the immediately returned packets to the connected switch and measures them divided by two, when considering the bidirection received through the response. Here, AB_{sw} is measured by calculating the difference between the maximum bandwidth of the link and the data rate of the next connected switch. The data rate is the difference between the number of bytes transmitted at time t and the number of bytes transmitted at time t+1.

The x_2 element, $G(V, E)$, represents the location and orientation in a two-dimensional planar form, where nodes and switches can be located in a network topology. The x_3 and x_4 elements, RTT and MSS, are the elements of network packets needed to transmit parity blocks, x_5 represents the current available bandwidth in the link, and x_6, x_7, and x_8 are the bandwidths of nodes and switches present before and after the present time. In addition, x_9 and x_{10} represent the hop count of the nodes and switches that exist before and after the current location, and x_{11}, x_{12}, and x_{13} represent the total number of nodes with data blocks and parity blocks in the erasure coding network topology, and the number of data nodes and parity nodes for each block. Moreover, x_{14} and x_{15} are currently active and inactive nodes used to determine the number of failed nodes, and x_{16}, x_{17}, and x_{18} are the data blocks, the source nodes to which the parity blocks are to be transmitted, and the size of the destination nodes and blocks.

4.3. Performance of Deep Q-Network

In Section 4.2, the state is described as the input parameters required by the agent to derive actions from the DQN neural network model. All elements were used as inputs to the neural network model, along with the amount of change before each element. For example, the bandwidth of the following nodes and switches is shown in Equation (5)

through the current time t, which is the time information, and the previous time t-1, which is the amount of change.

$$\Delta NextBw_{sw} = NextBw_{sw}(t) - NextBw_{sw} \quad (5)$$

The action is passed on to the erasure coding network topology according to the policy configured by the agent based on any state value. In addition, when a proper action is applied according to the state value, the agent can obtain a positive reward value for nodes that need to transfer blocks in the erasure coding network topology, and a negative reward value for moving to the wrong node and switch. Thus, the entire *Action_set* group performs k different actions for each time period t, as expressed through Equation (6).

$$Action\_set_t = \{a_1, a_2, \ldots, a_k\} \quad (6)$$

A reward is utilized to update the weights in the DQN neural network model, and unlike using labels for correct answers in supervised learning, each weight is updated through backpropagation using action-driven rewards. Therefore, the learning outcomes of neural network models may vary depending on how the reward is defined. In this study, the $Reward_{total}$ values were configured in three forms. $Reward_1$ has a positive value when data are sent using the correct link, $Reward_2$ has a negative value when data are sent using the wrong link, and $Reward_3$ has an extremely small negative value because it enters into a loop unless it chooses a switch link with another node. The overall reward value is shown in Equation (7).

$$Reward_{total} = Reward_1 + Reward_2 + Reward_3 \quad (7)$$

$Reward_1$ is shown in Equation (8) by setting it as a positive integer value when the current node and switch location (Now_{sw}), the node and switch location ($Prehob_{sw}$) of the previous link, and the next node and switch location ($Nexthob_{sw}$) are all moved throughout the entire path.

$$Reward_1 = Now_{sw} + Prehob_{sw} + Nexthob_{sw} (0 < Reward_1 \leq 1) \quad (8)$$

$Reward_2$ is shown in Equation (9) by setting the value to a negative integer when the link is moved from the current node and switch position (Now_{sw}) to the failed node and switch ($Node_{dead}$).

$$Reward_2 = Now_{sw} - Node_{dead} \ (-1 \leq Reward_2 < 0) \quad (9)$$

$Reward_3$ is a negative value for exiting the loop state as a constant cycle is repeated. If a large negative value is set, one can exit the loop and select another link; however, because there is an error that allows one to select the wrong link, an extremely small negative value can be set, as shown in Equation (10).

$$Reward_3 = -0.02 \leq Reward_3 < 0 \quad (10)$$

Replay memory is the memory space that stores the entire history that the agent applies during repetitive learning. The DQN neural network model proceeds with learning over time from the data contained in the constant cycle time collected. At this time, the learning is unstable owing to the high correlation because the data were collected sequentially. Therefore, when learning from replay memory is carried out, the correlation is reduced as much as possible through mini-batch learning using randomly extracted data, as previously mentioned in the description of the DQN principle.

The input layer, hidden layer, and output layer design of the DQN neural network model are listed in Table 2.

Table 2. Layer design of DQN neural network.

Layers	Size	Activation Function
Input Layer	18	None
Hidden Layer 1	32	Leaky ReLU
Hidden Layer 2	64	Leaky ReLU
Hidden Layer 3	128	Leaky ReLU
Output Layer	16	None

The input layer was designed with 18 elements corresponding to the state, and three hidden layers were used. The first hidden layer was set to 32, the second hidden layer to 64, and the last hidden layer to 128. In addition, the activation function used by the hidden layer was specified as a leaky ReLU [41]. The output layer is set to 16 to select the highest Q-value of the links between the following nodes and switches on the source node holding the data and parity blocks. Finally, the learning rate was tested by varying the values from 0.001 to 0.1, where 0.01 was the best value obtained, and the discount rate was designated as 0.95.

4.4. Optimized Routing Paths

When the optimal routing path is calculated through a DQN, the routing path is transmitted through OpenFlow to the erasure coding network topology. OpenFlow has a flow table that contains information regarding the path and method of packet forwarding used to the transmit blocks. When a packet occurs, it first ensures that the flow table contains information about the packet. If information about a packet exists, the packet is processed accordingly, and if information does not exist, the OpenFlow controller requests control information about that packet. Upon receiving control information from the switch, the OpenFlow controller checks the packet control information that exists inside and forwards the result to the OpenFlow switch. Packet control information within the OpenFlow controller can be entered through an API in an external program. The OpenFlow switch stores the control information received from the controller in the flow table and then uses the information in the flow table to forward the packet when the same packet occurs.

In the erasure coding network topology, the source nodes that transmit the data blocks and parity blocks required for decoding to the destination node are transmitted using optimized network routing paths. The process of calculating the optimized routing paths is shown in Algorithm 1.

First, the input parameters stored in the database, the hyperparameters used to design the DQN neural network model, and the discount factors (step 1) are specified, and the elements that are entered into the DQN neural network model are initialized. An element is a replay memory used to solve the correlation problem, a Q-value of the action from the state, a Q-value of the target network to calculate the next action, a P list that stores routes from each source node to the destination node, and a 2d LP list that stores routes from all source nodes to destination nodes (step 2).

The detailed algorithm works through a double loop (step 3) because the block transfer required for the decoding operation must consider all routing paths from each source node to the destination node (step 4). The following indicates the same e-greedy methodology as Q-Learning and selects an action on which link to select from the source node to the next node and acquires a reward value to move to the next state (step 5). To reduce the correlation problems described, previous records are stored in the replay memory, and mini-batch learning is applied (step 6). The action is determined, the value is calculated to estimate the Q-value of the target network, and the weight is updated to optimize the learning model by applying gradient detection (step 7).

Algorithm 1 DQN-based network routing in Erasure Coding network topology

1 **Input:** hyper parameters of neural network, discount factor γ
2 Initialize experience replay memory RD
Initialize action-value function Q with random weights θ
Initialize target action-value function $\widetilde{Q}$ with weights $\widetilde{\theta} = \theta$
Initialize network route list P
Initialize complete network route 2d list LP
3 **For** 1: i do ($(i = \ number\ of\ source\ nodes)$
4 **For** episodes = 1:100 do
5 With the probability select random action a_t, otherwise select $a_t = argmax_a Q(s_t, a;\ \theta)$
Execution action a_t, get reward r_t and next state $s_t^{'}$ then update the network state
6 Store the experience $(s_t,\ a_t, r_t, s_t^{'})$ to RD
Sample random mini batch perform $(s_j, a_j, r_j, s_j^{'})$ from in RD
7 **If** episode terminates at step, $j + 1$ then
Set target $y_j = \ r_j$
Else
Set target $y_j = \ r_j + \gamma max_{a'} \widetilde{Q}\left(s_j^{'},\ a';\ \widetilde{\theta}\right)$
End If
Perform a gradient descent step on learning rate and $\left(y_j - \ \widetilde{Q}\left(s', a';\ \widetilde{\theta}\right)\right)^2$ with θ
Every C steps reset $\widetilde{Q} = Q$
End For
8 Append the last route path to list P
List P append to list LP
9 Initialize all variable except LP $(RD,\ Q,\ \widetilde{Q},\ P_i, a_t,\ r_t,\ y_t,\ \ldots)$
End For
10 **Output:** Complete network route path LP from all source node to destination node

When a routing route from one source node to the destination node is completed, it is stored in a P-list as well as in a 2d LP list composed of dual lists (step 8). In addition, because another route from the source node to the destination node must be produced, the remaining variables are initialized except for the LP list that stores the final optimized route (step 9). At the end of every iteration, the LP list stores the entire routing path from each source node to the destination node, and thus in the erasure coding network topology, each node transfers the blocks required for decoding to the destination node according to the path (step 10).

5. Evaluation

This section describes the results measured in the DQN simulation environment and the erasure coding network topology used to identify and analyze the experimental evaluation results of the proposed method.

5.1. Simulation Environment

This section describes the specifications and simulation environments of deep-learning workstations to evaluate the performance of DQN-based erasure coding network topologies. The deep learning simulation was conducted on a workstation running the Ubuntu 20.04.4 operating system, with a XEON 4110 (8 core $\times$ 2) CPU, 128 GB of DDR4 memory, and four RTX 2080 graphics cards.

OpenvSwitch was used to set up an environment containing virtual nodes and a set of links connected to the switch, and the SDN network controllers used open-source-based Ryu as an OpenFlow network controller. For network bandwidth fluctuations, the link bandwidth of each node and switch reached up to 1 Gbps and was configured to be randomly changed within the range of 300 to 600 Mbps. We randomly specified a 15% probability of switch failures. The traffic intensity was set to 30% to 50% because it would be necessary to assume that some data were being transmitted. The key variables and the set values used in the experiments are listed in Table 3.

Table 3. Parameters of network topology simulation.

Parameters	Configure Value
Training steps	100
DQN episodes	100
Link maximum bandwidth	1 Gbps
Link real-time bandwidth	300–600 Mbps
Traffic intensity	30–50%
Probability of link failure	0–15%

The network topology used to experiment with the DQN-EC performance was given a fat tree [42]. The fat-tree network topology is configured using parameter α and consists of $\alpha^3/4$ servers and $5 \times \alpha^2/4$ switches. In the experimental evaluation, parameter α was specified as 8, resulting in a network topology with 128 servers and 80 switches. For 128 servers, RS (6, 3) erasure coding is applied to randomly generate six data blocks and three parity blocks, which are distributed and stored on nodes, and the size of the blocks stored is 128 MB. For various experimental evaluations, the test was conducted by increasing the probability of a switch failure from 0% to 15% when requesting a decoding.

Figure 7 shows a topology example consisting of 16 servers and 20 switches by specifying parameter α as 4 to help understand the fat-tree network topology with erasure coding. The topology configuration shows that green-shaded nodes are source nodes for transferring blocks in a decoding operation, and blue-shaded nodes are destination nodes for receiving blocks and performing decoding operations. In the above simulation environment and EC network topology, the DQN-EC-based network routing method was labeled DQN-EC and compared with the underlying network routing algorithm OSPF.

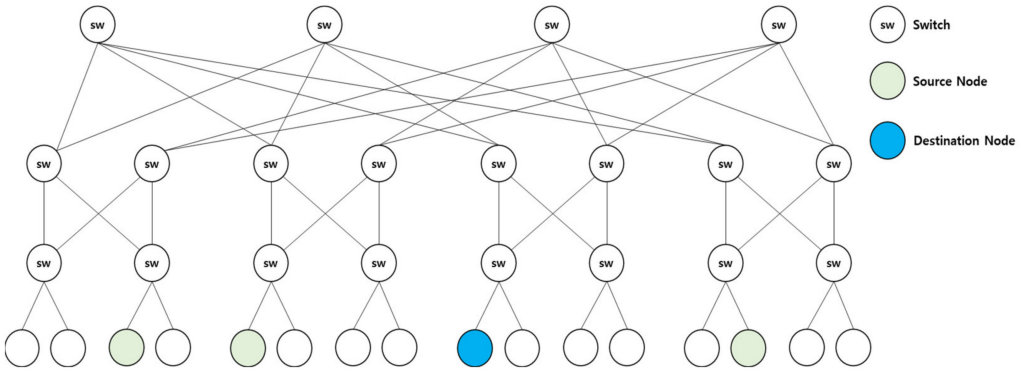


Figure 7. Example of applying EC to fat-tree network topology.

5.2. Evaluation Result of DQN-EC

This section introduces the results of the rate of change in compensation per episode associated with the designed DQN neural network model. It also introduces the time and network throughput results of transferring blocks from the erasure coding network topology to the target nodes, depending on the probability of a switch failure. Finally, we introduce the recovery time results based on the number of recovery requests.

Figure 8 shows the cumulative change in reward values according to the learning process of the agent when using DQN-EC and OSPF.

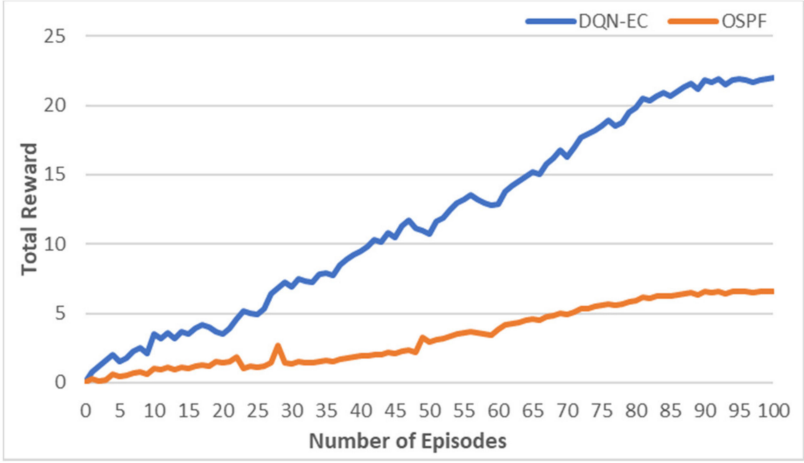


Figure 8. Total reward based on number of episodes.

Figure 8 shows the results of measuring the change in the cumulative reward value, with the *x*-axis representing the number of episodes and the *y*-axis representing the sum of the rewards according to the number of episodes. On average, 95 different routing paths were generated, some quite similar, and some completely different. However, as the number of episodes was repeated, a similar route was created in the last 80–100 paths. Using the e-greedy method, the agent initially does not have sufficient knowledge of the current network topology. Thus, most of agents explored the environment and obtained a lower reward value; however, as the number of episodes increased, the reward value increased and converged to the maximum value, indicating that optimized network routing paths were created during the last episode.

Figure 9 shows the block transfer time according to the probability of failure of 0% to 15% on a switch corresponding to the simulation parameters when using DQN-EC and OSPF.

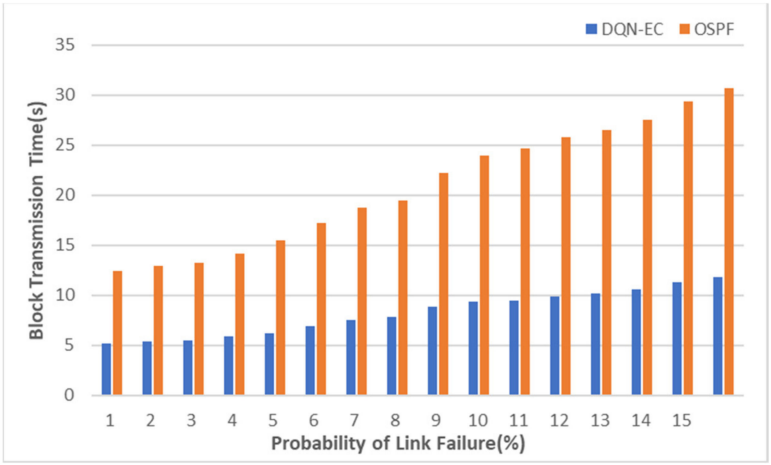


Figure 9. Block transmission time based on the probability of a link failure.

Figure 9 shows the measurement of the block transfer time from the source node to the destination node according to the probability of a switch failure, in which the decoding request was conducted only once. When a decoding request is made, nine blocks with a size of 128 MB are sent to the destination node. Compared to the existing OSPF routing algorithms, the block transfer time is reduced. For up to 15% of the maximum probability of a switch failure, DQN-EC has a slight increase in the block transfer time from 5.2 to 11.8 s. However, OSPF increases significantly from 12.48 to 30.68 s as it reaches 15%. DQN-EC measured a block transfer time of 8.25 s on average, and OSPF averaged 20.91 s, showing a

difference in the block transfer rate of approximately 2.5. The DQN-EC neural network model provides a faster block transfer time than the Dijkstra algorithm-based shortest-path OSPF because the agent avoids failed switches and selects links corresponding to optimized routing paths when considering the bandwidth.

Figure 10 shows the network throughput according to the probability of a failure of 0% to 15% on a switch when using DQN-EC and OSPF.

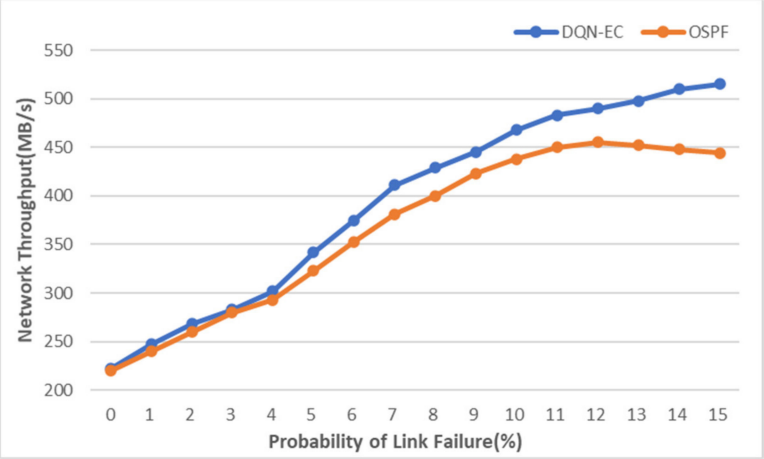


Figure 10. Network throughput based on the probability of a link failure.

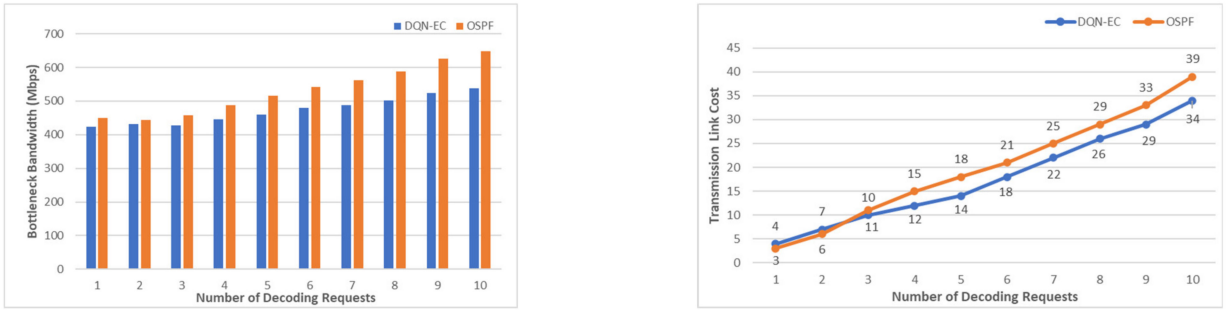
Figure 10 shows the improvement in network throughput compared to traditional OSPF routing algorithms because of the measurement of the network throughput. The performance of OSPF is measured similar to that of DQN-EC under a probability of failure of 0–4%. However, from 5% of the probability of a switch failure, the network throughput of DQN-EC and OSPF becomes increasingly different owing to increased network traffic and congestion probability in the network. In OSPF, when the probability of a switch failure is 12%, the measured network throughput of 455 MB/s decreases slightly as the probability of a switch failure increases to 15%. Conversely, DQN-EC can maintain the congestion probability as low as possible throughout learning, and thus the network throughput is not reduced by up to 15%, which is the maximum probability of a switch failure, and can continue to improve, reaching up to 515 MB/s.

The performance of the recovery times that occur during decoding using optimized routing paths is assessed. The recovery time is the sum of the network bottleneck costs and the link count costs and is compared and evaluated using OSPF routing network algorithms. The relevant content is shown in Equation (11). Because the measured recovery time varies depending on the probability of failure of the switch, the probability of a switch failure was fixed and measured at 5% to maintain a constant block transfer time.

$$\begin{aligned} RT &= \textstyle\sum_{r=1}^{n}(BB_r + TL_r), \\ BB_r &= argmin(\textstyle\sum_{k=1}^{n} NB_k), \\ TL_r &= \frac{\sum_{k=1}^{n} NB_k}{\sum_{k=1}^{n} L_k}, \end{aligned} \tag{11}$$

where BB_r indicates the bottleneck bandwidth, TL_r indicates the transmission link, NB_k indicates the network bandwidth and L_k is the number of links.

Above, RT is the recovery time, which is the sum of the BB and TL costs, where the former indicates the bottleneck bandwidth in routing paths, and the latter indicates the transmission link costs in the routes. The lower the measured value, the better, because the network bottleneck bandwidth indicates the bandwidth at the point where network traffic occurs at a particular point while configuring the network routing path. The transmission link cost is the cost of summing the measured network bandwidth in the optimized routing path and dividing the total sum of the link counts of the links. The results related to network bottlenecks and transmission link costs are shown in Figure 11.



(**a**) (**b**)

Figure 11. Graphs of BB_r, TL_r by equation 11 as: (**a**) Bottleneck bandwidth based on number of decoding requests; (**b**) Transmission link cost based on number of decoding requests.

Figure 11a shows that the network bottleneck bandwidth of DQN-EC averaged 493 Mbps, which is slightly lower than that of OSPF at 633 Mbps. The network bottlenecks were reduced compared to OSPF because under intense traffic decoding operations using optimized routing paths that avoid high transport links are applied as much as possible. Figure 11b shows that the transmission link costs measured are high in OSPF, which can simply maintain short transmission links up to two recovery requests. However, the more recovery requests that exist, the more network routes that occur, and the greater the complexity, the lower the DQN-EC, which considers the bandwidth and transport links. In addition, the bottleneck bandwidth is likely to be due to the processing limitations of the internal bus bandwidth of the deep learning workstation. Therefore, checking the GPU, memory, and GPU entries in the monitoring log recorded during this experiment, it was shown that a pull-load phenomenon did not occur, and thus was less relevant to the internal bus bandwidth.

Figure 12 shows the combined recovery time for network bottlenecks and link count costs.

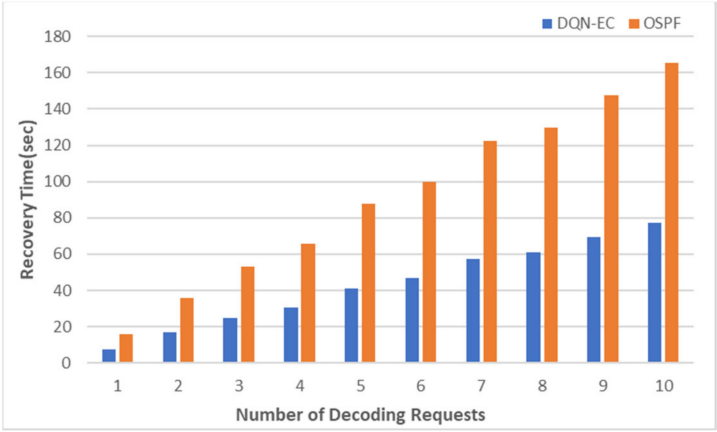


Figure 12. Recovery time based on number of decoding requests.

The measurement of the recovery time of DQN-EC and OSPF gradually varies as the number of recovery requests increases. DQN-EC takes 77 s to recover when the maximum number of recovery requests is 10, whereas OSPF requires 165 s for the same number of maximum requests. The optimized routing path of DQN-EC avoids bottlenecks as much as possible and chooses links in which the transmission link costs are low. Therefore, even if the bandwidth of the transfer link is low, the recovery time is shorter than that of the OSPF used to reduce the cost of the simple transfer link.

5.3. Comparison and Summary of DQN-EC and Experiment Results

Similar to DQN-EC, RouteNet [24,25] improves the delay and jitter to the source and destination by up to 1.4-times compared to OSPF. In [26], the authors compared between the deep learning system and OSPF; in this case, the overall network throughput was improved by approximately 2.3-fold, and the average hop delay was measured to be approximately 12-times lower in the proposed deep learning system. In addition, the authors of [27] indicated that the results of a methodology experiment showed approximately a 3-fold improvement in network throughput and average delay per hop compared to OSPF. In [34], when the network load level is between 2 and 2.5, the average delivery time OSPF converges to the maximum, whereas Q-routing records an average delivery time similar to that of OSPF when the network load level is 4, which is an improvement of approximately 1.8-fold. In [35], CQ-routing recorded the same average packet delivery time as OSPF until the route had 1500 learning times. However, as the learning count reached 3000, OSPF continued to increase, recording an average packet delivery time of 70 s; however, CQ-routing converges at close to zero. In [37], DRQ-routing learns the optimal routing policies 1.5-times faster than spectrum-aware Q-routing at low and intermediate network loads, and the average packet delivery time differs by up to 30 s compared to Q-routing. In addition, in [39], RL-routing improved the average file transfer rate by approximately 3-fold and the average utilization rate by approximately 2-fold compared to those of OSPF.

The DQN-EC proposed in this paper constructs an experimental environment by using a DQN, deep reinforcement learning, the addition of elements necessary for decoding in erasure coding, and the selection of OSPF, the subject of the comparison of the experiment results. The block transmission time, which has a similar meaning as the average packet delivery time, improved 2.5-fold when the probability of a link failure was 15%. Network throughput improved 0.4-times, and bottleneck bandwidth was measured up to 100 Mbps lower than OSPF, reducing bottleneck bandwidth. The data may be recovered twice as fast as when OSPF is used and the number of decoding requests is 10.

Compared with the above related studies, the purpose of calculating the optimal routing path, which is the final goal of DQN-EC, is the same. However, DQN-EC incorporates a large-capacity distributed clustering system using erasure coding into the network topology. In addition, it is possible to quickly recover the data because a switch that has failed in the network is avoided, and the transmission link is not simply reduced, but instead the bandwidth of the transmission link is detected and the block is transmitted using an efficient routing path when applying the decoding.

6. Conclusions

In this study, various hyperparameters and erasure coding elements were applied to the fat-tree network topology for a deep-reinforcement learning-based DQN algorithm applied to neural network models. We proposed a DQN-EC in which blocks are efficiently transmitted when a decoding operation is requested by designing a suitable reward value and applying it to an erasure coding network topology. DQN-EC identifies switches that fail in a simulated environment in which the bandwidth of each node and switch is not fixed and dynamically changes, and thus avoids them by providing optimized routing paths. Compared to the OSPF algorithm, the block transfer time was measured as shorter as the probability of a switch experiencing a failure and the network throughput both increased. Even when the number of decoding requests gradually increased, the recovery time could be reduced by 15–20% overall compared to that of the OSPF algorithms. The DQN-EC proposed in this paper showed a more efficient appearance than the underlying OSPF algorithm in a simulation environment using RS (6, 3) erase codes, which are commonly applied; however, the results may vary depending on the parameters that make up the erasure coding. Therefore, we analyzed other network topologies—B-Cube, NSFNet, and APRANet—and classified and extracted elements that can apply DQN-EC. Finally, we will further study an optimized neural network model design methodology suitable for each

network topology and demonstrate the efficiency of DQN-EC through an application and experimental evaluation.

Author Contributions: Conceptualization, D.-J.S. and J.-J.K.; Methodology, D.-J.S. and J.-J.K.; Software, D.-J.S.; Validation, J.-J.K.; Formal analysis, D.-J.S. and J.-J.K.; Investigation, D.-J.S.; Resources, J.-J.K.; Data curation, D.-J.S.; Writing—original draft preparation, D.-J.S.; Writing—review and editing, J.-J.K.; Visualization, D.-J.S.; Supervision, J.-J.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Jiang, Y.; Liu, Q.; Dannah, W.; Jin, D.; Liu, X.; Sun, M. An Optimized Resource Scheduling Strategy for Hadoop Speculative Execution Based on Non-cooperative Game Schemes. *Comput. Mater. Contin.* **2020**, *62*, 713–729. [CrossRef]
2. Astyrakakis, N.; Nikoloudakis, Y.; Kefaloukos, I.; Skianis, C.; Pallis, E.; Markakis, E.K. Cloud-Native Application Validation & Stress Testing through a Framework for Auto-Cluster Deployment. In Proceedings of the 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Limassol, Cyprus, 11–13 September 2019; pp. 1–5. [CrossRef]
3. Saadoon, M.; Hamid, S.H.A.; Sofian, H.; Altarturi, H.H.; Azizul, Z.H.; Nasuha, N. Fault tolerance in big data storage and processing systems: A review on challenges and solutions. *ASEJ* **2021**, in press. [CrossRef]
4. Shin, D.J. A Study on Efficient Store and Recovery Technology to Reduce the Disk Loading of Distributed File System in Big Data Environment. Master's Thesis, Korea Polytechnic University, Siheung-si, Korea, 2020.
5. Xia, J.; Guo, D.; Xie, J. Efficient in-network aggregation mechanism for data block repairing in data centers. *Future Gener. Comput. Syst.* **2020**, *105*, 33–43. [CrossRef]
6. Wang, F.; Tang, Y.; Xie, Y.; Tang, X. XORInc: Optimizing data repair and update for erasure-coded systems with XOR-based in-network computation. In Proceedings of the 35th Symposium on Mass Storage Systems and Technologies (MSST), Santa Clara, CA, USA, 20–24 May 2019; pp. 244–256. [CrossRef]
7. Kreutz, D.; Ramos, F.M.V.; Verissimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-defined networking: A comprehensive survey. *Proc. IEEE* **2014**, *103*, 14–76. [CrossRef]
8. Traffic Engineering (TE) Extensions to OSPF Version 2. Available online: https://www.hjp.at/doc/rfc/rfc3630.html (accessed on 5 June 2021).
9. Pano-Azucena, A.D.; Tlelo-Cuautle, E.; Ovilla-Martinez, B.; Fraga, L.G.D.L.; Li, R. Pipeline FPGA-based Implementations of ANNs for the Prediction of up to 600-steps-ahead of Chaotic Time Series. *J. Circuits Syst. Comput.* **2021**, *30*, 2150164. [CrossRef]
10. Shin, D.J.; Kim, J.J. Research on improving disk throughput in EC-based distributed file system. *Psychol. Educ.* **2021**, *58*, 9664–9671. [CrossRef]
11. Plank, J.S. Erasure Codes for Storage Systems: A brief primer. *Usenix Mag.* **2013**, *38*, 44–50.
12. Deng, M.; Liu, F.; Zhao, M.; Chen, Z.; Xiao, N. GFCache: A Greedy Failure Cache Considering Failure Recency and Failure Frequency for an Erasure-Coded Storage System. *Comput. Mater. Contin.* **2019**, *58*, 153–167. [CrossRef]
13. Kim, H.W.; Lee, W.C. Real-Time Path Planning for Mobile Robots Using Q-Learning. *Inst. Korean Electr. Electron. Eng.* **2020**, *24*, 991–997. [CrossRef]
14. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
15. Cook, J.D.; Primmer, R.; de Kwant, A. Compare cost and performance of replication and erasure coding. *Hitachi Rev.* **2014**, *63*, 304–310.
16. Kim, D.O.; Kim, H.Y.; Kim, Y.K.; Kim, J.J. Cost analysis of erasure coding for exa-scale storage. *Supercomputing* **2019**, *75*, 4638–4656. [CrossRef]
17. Kim, D.O.; Kim, H.Y.; Kim, Y.K.; Kim, J.J. Efficient techniques of parallel recovery for erasure-coding-based distributed file systems. *Computing* **2019**, *101*, 1861–1884. [CrossRef]
18. Silberstein, M.; Ganesh, L.; Wang, Y.; Alvisi, L.; Dahlin, M. Lazy means smart: Reducing repair bandwidth costs in erasure-coded distributed storage. In Proceedings of the SYSTOR (International Conference on Systems and Storage), Haifa, Israel, 30 June–2 July 2014; pp. 1–7. [CrossRef]

19. Pei, X.; Wang, Y.; Ma, X.; Xu, F. T-update: A tree-structured update scheme with top-down transmission in erasure-coded systems. In Proceedings of the IEEE INFOCOM (International Conference on Computer Communications), San Francisco, CA, USA, 10–14 April 2016; pp. 1–9. [CrossRef]
20. Li, R.; Li, X.; Lee, P.P.; Huang, Q. Repair Pipelining for Erasure-Coded storage. In Proceedings of the USENI ATC (Annual Technical Conference), Santa Clara, CA, USA, 12–14 July 2017; pp. 567–579.
21. Nasteski, V. An overview of the supervised machine learning methods. *Horizons* **2017**, *4*, 51–62. [CrossRef]
22. Zhuang, Z.; Wang, J.; Qi, Q.; Sun, H.; Liao, J. Graph-Aware Deep Learning Based Intelligent Routing Strategy. In Proceedings of the IEEE LCN (Conference on Local Computer Networks), Chicago, IL, USA, 1–4 October 2018; pp. 441–444. [CrossRef]
23. Li, L.; Zhang, Y.; Chen, W.; Bose, S.K.; Zukerman, M.; Shen, G. Naïve Bayes classifier-assisted least loaded routing for circuit-switched networks. *IEEE Access* **2019**, *7*, 11854–11867. [CrossRef]
24. Rusek, K.; Suárez-Varela, J.; Mestres, A.; Barlet-Ros, P.; Cabellos-Aparicio, A. Unveiling the potential of Graph Neural Networks for network modeling and optimization in SDN. In Proceedings of the ACM Symposium on SDN Research, San Jose, CA, USA, 3–4 April 2019; pp. 140–151. [CrossRef]
25. Rusek, K.; Suárez-Varela, J.; Almasan, P.; Barlet-Ros, P.; Cabellos-Aparicio, A. RouteNet: Leveraging Graph Neural Networks for network modeling and optimization in SDN. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 2260–2270. [CrossRef]
26. Kato, N.; Fadlullah, Z.M.; Mao, B.; Tang, F.; Akashi, O.; Inoue, T.; Mizutani, K. The deep learning vision for heterogeneous network traffic control: Proposal, challenges, and future perspective. *IEEE Wirel. Commun.* **2017**, *24*, 146–153. [CrossRef]
27. Mao, B.; Fadlullah, Z.M.; Tang, F.; Kato, N.; Akashi, O.; Inoue, T.; Mizutani, K. Routing or computing? The paradigm shift towards intelligent computer network packet transmission based on deep learning. *IEEE Trans. Comput.* **2017**, *66*, 1946–1960. [CrossRef]
28. Geyer, F.; Carle, G. Learning and generating distributed routing protocols using graph-based deep learning. In Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks, Budapest, Hungary, 20 August 2018; pp. 40–45. [CrossRef]
29. Sharma, D.K.; Dhurandher, S.K.; Woungang, I.; Srivastava, R.K.; Mohananey, A.; Rodrigues, J.J. A machine learning-based protocol for efficient routing in opportunistic networks. *IEEE Syst.* **2016**, *12*, 2207–2213. [CrossRef]
30. Reis, J.; Rocha, M.; Phan, T.K.; Griffin, D.; Le, F.; Rio, M. Deep neural networks for network routing. In Proceedings of the IJCNN (International Joint Conference on Neural Networks), Budapest, Hungary, 14–19 July 2019; pp. 1–8. [CrossRef]
31. Szepesvári, C. Algorithms for reinforcement learning. *Synth. Lect. Artif. Intell. Mach. Learn.* **2010**, *4*, 1–103. [CrossRef]
32. Lin, S.C.; Akyildiz, I.F.; Wang, P.; Luo, M. QoS-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach. In Proceedings of the SSC (International Conference on Services Computing), San Francisco, CA, USA, 27 June–2 July 2016; pp. 25–33. [CrossRef]
33. Wu, J.; Li, J.; Xiao, Y.; Liu, J. Towards Cognitive Routing Based on Deep Reinforcement Learning. *arXiv*. 2020. Available online: https://arxiv.org/abs/2003.12439 (accessed on 20 June 2021).
34. Boyan, J.A.; Littman, M.L. Packet routing in dynamically changing networks: A reinforcement learning approach. In Proceedings of the 6th NIPS (Neural Information Processing Systems), San Francisco, CA, USA, 29 November–2 December 1993; pp. 671–678.
35. Kumar, S.; Miikkulainen, R. Confidence-based q-routing: An online adaptive network routing algorithm. In Proceedings of the Artificial Neural Networks in Engineering, St Louis, MO, USA, 1–4 November 1998; pp. 758–763.
36. Newton, W. A Neural Network Algorithm for Internetwork Routing. Bachelor's Thesis, University of Sheffield, Sheffield, UK, 2002.
37. Xia, B.; Wahab, M.H.; Yang, Y.; Fan, Z.; Sooriyabandara, M. Reinforcement learning based spectrum-aware routing in multi-hop cognitive radio networks. In Proceedings of the CROWNCOM (Cognitive Radio Oriented Wireless Networks and Communications), Hanover, Germany, 22–24 June 2009; pp. 1–5. [CrossRef]
38. Zhang, J.; Ye, M.; Guo, Z.; Yen, C.Y.; Chao, H.J. CFR-RL: Traffic engineering with reinforcement learning in SDN. *IEEE J. Sel. Areas Commun.* **2020**, *38*, 2249–2259. [CrossRef]
39. Chen, Y.R.; Rezapour, A.; Tzeng, W.G.; Tsai, S.C. RL-routing: An SDN routing algorithm based on deep reinforcement learning. *IEEE Trans. Netw. Sci. Eng.* **2020**, *7*, 3185–3199. [CrossRef]
40. Chae, J.H.; Lee, B.D.; Kim, N.K. An Efficient Multicast Routing Tree Construction Method with Reinforcement Learning in SDN. *J. Korean Inst. Inf. Technol.* **2020**, *18*, 1–8. [CrossRef]
41. Zhang, X.; Zou, Y.; Shi, W. Dilated convolution neural network with LeakyReLU for environmental sound classification. In Proceedings of the DSP (International Conference on Digital Signal Processing), London, UK, 23–25 August 2017; pp. 1–5. [CrossRef]
42. Al-Fares, M.; Loukissas, A.; Vahdat, A. A scalable, commodity data center network architecture. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 63–74. [CrossRef]