# Multicast Tree Generation using Meta Reinforcement Learning in SDN-based Smart Network Platforms

**Jihun Chae and Namgi Kim***
School of Computer Sciences and Engineering, Kyonggi University, Suwon, South Korea
[e-mail: ngkim@kgu.ac.kr]
*Corresponding author: Namgi Kim

*Received April 4, 2021; revised June 23, 2021; accepted July 8, 2021; published September 30, 2021*

## Abstract

Multimedia services on the Internet are continuously increasing. Accordingly, the demand for a technology for efficiently delivering multimedia traffic is also constantly increasing. The multicast technique, that delivers the same content to several destinations, is constantly being developed. This technique delivers a content from a source to all destinations through the multicast tree. The multicast tree with low cost increases the utilization of network resources. However, the finding of the optimal multicast tree that has the minimum link costs is very difficult and its calculation complexity is the same as the complexity of the Steiner tree calculation which is NP-complete. Therefore, we need an effective way to obtain a multicast tree with low cost and less calculation time on SDN-based smart network platforms. In this paper, we propose a new multicast tree generation algorithm which produces a multicast tree using an agent trained by model-based meta reinforcement learning. Experiments verified that the proposed algorithm generated multicast trees in less time compared with existing approximation algorithms. It produced multicast trees with low cost in a dynamic network environment compared with the previous DQN-based algorithm.

***Keywords:*** Multicast tree, Meta reinforcement learning, Multimedia routing, SDN, Deep learning

This work was supported by a Kyonggi University Research Grant in 2020.

## 1. Introduction

Multimedia streaming services, such as YouTube [1], Netflix [2], Amazon Prime Video [3], and Hulu [4], have significantly increased in popularity on the Internet [5]. According to this trend, the multicast technique that copies and forwards the same multimedia data from a source node to several destinations has been actively investigated. The multicast increases the utilization of network resources, and efficiently provides multimedia services to numerous users [6]. However, on traditional network platforms, the network-level multicast has not been widely implemented. This is because the traditional routers that are produced by different vendors and operated by different Internet service providers are heterogeneous, and independently controlled.

The complexity and closeness of networks mean that it is hard in the traditional network platform to develop a new service or update existing services. To overcome these problems, the open and flexible smart network platform based on Software-Defined Networking (SDN) has been proposed [7]. In the SDN-based smart network platform, switches responsible for data forwarding, like routers, are managed by a centralized controller. The centralized controller obtains flow-related information from switches, and globally generates flow tables for switches. This mechanism allows a multicast technique to be implemented naturally at the network level.

A multicast technique requires an appropriate multicast tree when a multicast service user changes. A multicast tree comprises switches and links that are used to deliver multimedia data from a source node to all destination nodes. The controller globally generates flow tables based on the multicast tree to copy and forward multimedia packets to the following links. As the sum of link costs forming the multicast tree decreases, the utilization of network resources increases. However, the complexity of calculating the multicast tree with the minimum sum of link costs is the same as the calculation of the Steiner tree that exhibits NP-complete [8]. As the size of the network topology increases, it becomes more difficult to generate an effective multicast tree. Various approximation algorithms have been developed to solve this problem. However, they provide insufficient performance to satisfy the quality of multimedia services, or require a significant amount of time to construct a tree. Therefore, in this paper, we propose a novel algorithm that produces a multicast tree that has a low sum of link costs within a short time on SDN-based smart network platforms. The proposed algorithm uses the meta reinforcement learning to train an agent to generate a multicast tree.

This paper is organized as follows: Section 2 reviews SDN structures and existing multicast tree generation algorithms. Section 3 introduces the proposed algorithm, which efficiently generates multicast trees on SDN-based smart network platforms. Section 4 provides an evaluation of the performance of the multicast tree generation algorithms. Finally, Section 5 concludes the paper.

## 2. Background and Related Works

In the traditional network structure, a router locally manages network information, and independently generates a forwarding table. Diverse vendors produce heterogeneous routers, and adopt independent and non-interoperable router control mechanisms. As a result, new network services are difficult to deploy and manage in this structure. SDN technology is developed to solve the rigidity and closeness of the traditional network structure. **Fig. 1** shows that in the SDN architecture, data and control planes are separated. Distributed switches perform data forwarding in the data plane. A centralized controller globally generates and manages flow tables of switches in the control plane. A network administrator transfers a policy on a service to the controller through an application server. This process enables new services to be flexibly and easily deployed and managed.

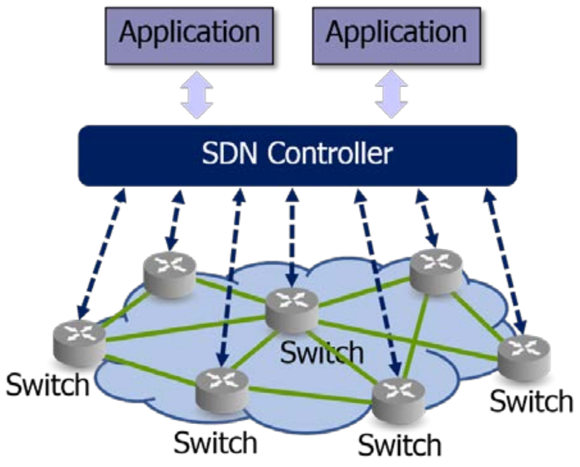


**Fig. 1.** Architecture of the SDN.

In the SDN, the centralized controller globally obtains network topology information, so that it can effectively generate a multicast tree. The SDN controller generates flow tables based on the multicast tree, and sends them to target switches for the multimedia service. Hence, multicasts can be easily and effectively implemented in the SDN-based smart network platform [9-12].

To efficiently implement multicasts by increasing the utilization of network resources, the controller should generate a multicast tree with a low sum of link costs. The Steiner tree is a multicast tree that has a minimum sum of link costs. The calculation for deriving the Steiner tree exhibits NP-complete complexity. In this regard, numerous approximation algorithms have been proposed to heuristically derive low-cost multicast trees [13–17]. However, the approximation algorithms require a lot of computation time, and as the network size increases, demonstrate deteriorated performance.

Recently, a machine learning technology that utilizes a considerable amount of data has achieved outstanding performance. It has attracted significant attention in various fields, and network researchers have also attempted to solve complex and difficult network problems by the technology. In various machine learning techniques, the multicast tree generation problem can be solved using reinforcement learning. The reinforcement learning is performed based on interactions with the surrounding environment, and rewards are provided through such interactions. It evaluates and updates the current policy to obtain a solution based on the rewards provided.

Studies using reinforcement learning in the network routing field have been conducted at the initial level. S.-C. Lin *et. al* [18] designed a reward function in consideration of delay, loss, and bandwidth in a hierarchical SDN environment, and proposed an adaptive routing method based on Q-learning. T. Hendriks *et. al.* [19] developed an adaptive Q routing method to provide Quality of Service (QoS) in ad hoc wireless networks, and to optimize the overhead and quality of discovered paths. Q-learning is a table-based reinforcement learning. It has limitations in solving practical problems that require massive action space, owing to issues such as table space and searching cost. Instead of Q-learning, a Deep Q-Network (DQN) using a neural network was proposed [20]. H.-J. Heo *et. al.* [21] developed a multicast routing tree generation method based on a DQN agent in a static SDN environment; however, as the study was conducted at an early stage, it was evaluated only in a fixed network environment.

In some studies, the reinforcement learning was applied to identify a unicast routing path, instead of a multicast routing path. C. Yu *et. al.* [22] presented a method for searching a routing path with the minimum delay using a Deep Deterministic Policy Gradient algorithm. Z. Yuan *et. al.* [23] proposed a method to enhance the network routing performance using a State–Action–Reward–State–Action algorithm. Y.-R. Chen *et. al.* [24] introduced a method for solving traffic engineering issues using a Dueling Double DQN to improve the throughput and delay of a network. However, these methods were evaluated using small-sized network topologies that included 20 or fewer nodes and 40 or fewer links. In addition, most of them were developed in a static network environment with a fixed network topology. But in the real world, the network environment and topology dynamically change at any time. Therefore, a multicast tree generation algorithm that has good performance in dynamically changing network environments needs to be developed.

## 3. The Proposed Method

**Fig. 2** shows the process of the proposed algorithm for generating a multicast tree and establishing flow tables on SDN-based network platforms. The SDN controller collects network topology information from switches, and transfers it to the agent. The controller also delivers the information of source and destination nodes for a multicast service to the agent. The agent generates a multicast tree connecting a source node and destination nodes, and informs the controller of it. Based on the tree, the controller forms flow tables for a multicast service, and sends them to the target switches. As a result, data for a multicast service from the source node are transferred to all destination nodes via the flow tables at switches that exist on the paths of the multicast tree.

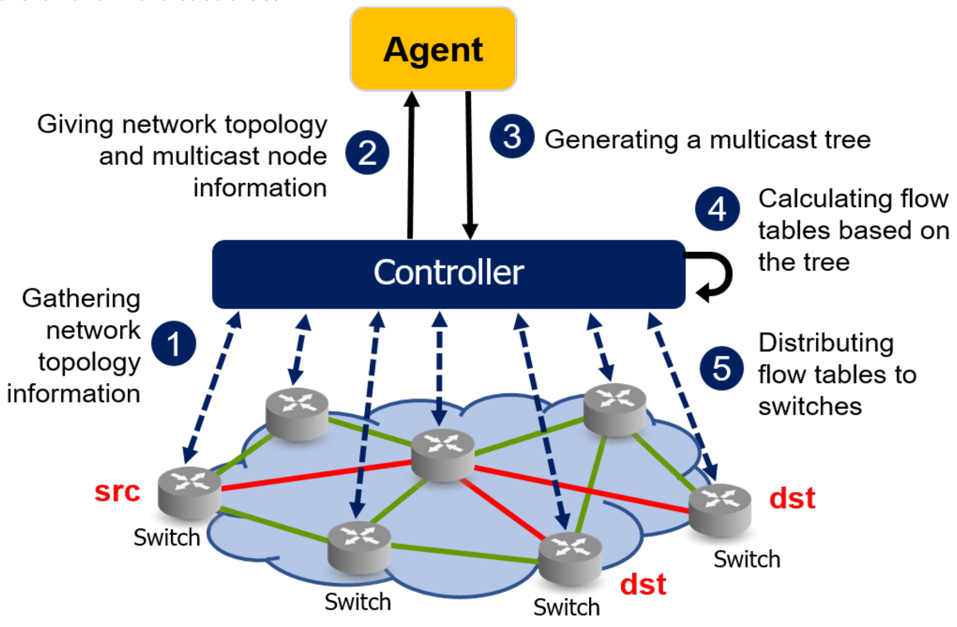

Agent
Giving network topology and multicast node information
2
3
Generating a multicast tree
4
Calculating flow tables based on the tree
Controller
Gathering network topology information
1
5
Distributing flow tables to switches
src
dst
Switch
Switch
Switch
dst
Switch

**Fig. 2.** Procedure of Constructing Flow Tables for Multicast.

In this study, a meta reinforcement learning [25] was utilized to train an agent. The meta reinforcement learning learns both training data and learning method to exhibit adaptively good performance, even in a new environment. This allows the agent to be trained to generally exhibit satisfactory performance on the network topology and multicast node sets that the agent did not experience during training.

Meta reinforcement learning can be classified into model- and optimization-based learnings. In this study, we adopt the model-based meta reinforcement learning. In model-based learning, the black box architecture is applied to identify the implicit and complex meaning of internal tasks and objectives that are similar to each other. The black box architecture additionally uses a recurrent neural network with an internal memory to memorize experiences, and is well matched to the reinforcement learning. Model-based meta reinforcement learning comprises outer-loop training for learning a new environment, and inner-loop training for achieving a goal in a selected environment. During inner-loop training, various types of existing reinforcement learning networks can be applied. Among them, the asynchronous advantage actor–critic (A3C) network, which has reasonably good performance, was adopted in this study.

A multicast tree construction problem is related to the determination of the order of links to be passed through to generate a low-cost tree from a source node to destination nodes. This problem can be defined as a sequential action decision problem, and expressed through the Markov Decision Process (MDP). To train an agent to construct low-cost multicast trees, diverse network topologies and multicast node sets should be sampled. A network topology and a multicast node set derived through the sampling process can be defined as MDP $M_i$, which is extracted from an environmental distribution $D$. **Table 1** lists the components that $M_i$ contains. State $S$ refers to the information obtained by the controller through observing the state of networks. It includes network topology and node set information that are sent to the agent. The node set information includes the source node and destination nodes for multicast services. Action $A$ is a selected link in a set of links that are connected to the current tree, but not included in that tree. The agent chooses a link that can be connected to new nodes by $A$ in the given $S$. The reward $R$ refers to the value of a reward obtained when the current tree being generated reaches one of the destination nodes. The discount factor $\gamma$ refers to a discount rate that determines the present value of future rewards. It has a value in the range (0–1).

**Table 1.** Components for the Markov Decision Process.

| **Component** | **Means** | **Description** |
|---|---|---|
| S | State | Network topology information, multicast set information, currently constructed multicast tree. |
| A | Action | A selected link among links that are adjacent to nodes in the current tree, but not included in that tree. |
| R | Reward | Awarded reward when the currently constructing tree reaches a destination. |
| $\gamma$ | Discount factor | A value determining how much rewards in the future are cared about. |

**Fig. 3** shows the network architecture for the agent. Numbers in the figure indicate the numbers of outputs. FC means a fully connected layer, while LSTM means a long short-term memory layer. In the architecture, actor and critic networks are shared. The action–critic network gets information of the network topology, multicast node sets, currently constructed multicast tree, and a link selected by the last action. It outputs a policy and a state value that determine an action. Since the number of links that can be connected varies with each action, the discrete action space of the actor output layer is variable, and cannot be fixed. To solve this problem, we make the actor layer output the mean distribution and variation of each action variable, based on the assumption that action variables follow mutually independent Gaussian distributions. The critic layer for the state evaluation outputs the value function of the present state.

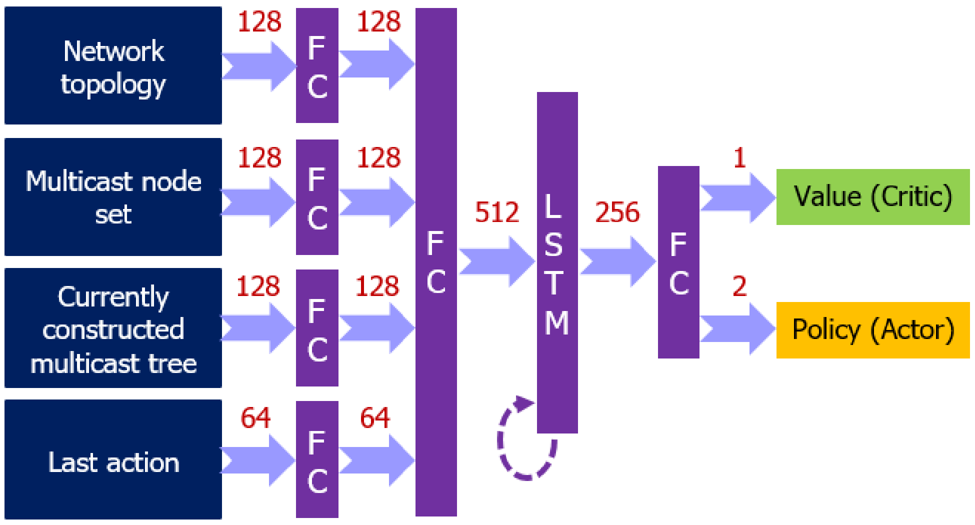


**Fig. 3.** Agent Architecture to Construct a Multicast Tree.

**Fig. 4** shows the learning procedure of the agent to generate a multicast tree. This procedure comprises five steps. When the fifth step is completed, the cycle starts from the first step again to ensure that the policy approaches optimal. In the first step, a new network topology containing links and nodes is sampled in the network topology distribution. A node set including the source and destination nodes is also sampled in the multicast node set distribution. In the second step, information regarding the state at the current time $t$ ($S_t$) and the state at the next time $t+1$ ($S_{t+1}$) in the network environment and the reward awarded by the last action ($R_t$, if any) are delivered to the agent. In the third step, the agent calculates the policy and the current state value through the actor–critic network, and determines an action for the current state ($A_t$) based on the policy. In the fourth step, transaction data ($S_t$, $A_t$, $S_{t+1}$, $R_t$, $A_{t-1}$) experienced during the action determination is stored in an episode buffer. A cycle beginning from the second step to the fourth step is repeated until the currently constructing tree is completely generated to contain all destination nodes. The fifth step begins after generating a complete multicast tree. In this step, transaction data stored in an episode buffer are loaded and train the actor–critic network of the agent. Once training on an episode is completed, a cycle beginning from the first step is repeated to train a new network environment.

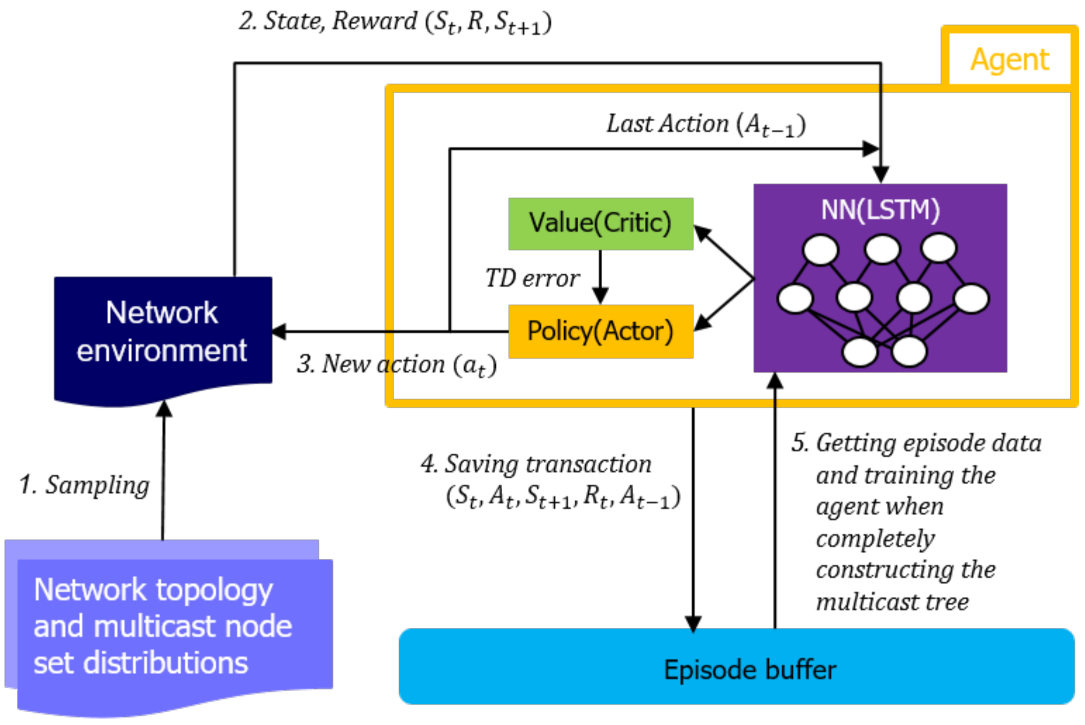

2. State, Reward $(S_t, R, S_{t+1})$
Agent
Last Action $(A_{t-1})$
Value(Critic)
NN(LSTM)
TD error
Network environment
Policy(Actor)
3. New action $(a_t)$
1. Sampling
4. Saving transaction $(S_t, A_t, S_{t+1}, R_t, A_{t-1})$
5. Getting episode data and training the agent when completely constructing the multicast tree
Network topology and multicast node set distributions
Episode buffer

**Fig. 4.** Agent Learning Procedure.

## 4. Experiments

Experiments were performed to evaluate the proposed algorithm. For a comparative evaluation, the performances of the TM algorithm [13], as the representative of approximation algorithms, were assessed. The multicast tree construction algorithms [21] applying DQN-based reinforcement learning were also evaluated. **Table 2** shows the details of the environment established for the experiments. A3C reinforcement learning of the proposed algorithm was implemented with eight parallel threads.

**Table 2.** Experimental Environments.

| Name | Value |
|---|---|
| OS | Ubuntu 18.04 |
| Framework | Tensorflow 1.15 |
| GPU | GTX 2080 Ti |
| Number of threads for A3C learning | 8 |

**Table 3** shows the parameters used in the experiments. The numbers of nodes included in the network topology were (20, 40, and 80), respectively. A multicast node set comprised one source node and more than two destination nodes. As the number of nodes in the network topology increased, the amount of state space to be trained increased as well. Hence, as the number of nodes increased, we significantly increased the number of training episodes. For the

test, we randomly selected 2,000 network topologies and multicast node sets that were not used.

**Table 3.** Experimental Parameters.

| Parameter | Values | | |
|---|---|---|---|
| Number of nodes | 20 | 40 | 80 |
| Number of source nodes | 1 | 1 | 1 |
| Number of destination nodes | 2 – 19 | 2 – 39 | 2 – 79 |
| Number of training episodes | 80,000 | 100,000 | 275,000 |
| Number of testing episodes | 20,000 | 20,000 | 20,000 |

**Fig. 5** shows the average cost of multicast trees constructed by the three algorithms. Since the cost of a link was set as 1 in the experiments, the cost of a multicast tree corresponded to the sum of the number of links included in the tree. The experimental result indicated that the average cost of the constructed multicast trees increased with the number of network nodes. In particular, the average cost of the tree constructed by the DQN-based reinforcement learning algorithm increased significantly with the number of network nodes. This occurred because in an environment in which the network topology and multicast service sets change dynamically, as the size of the network topology increases, DQN-based algorithms exhibit lower training efficiency. Meanwhile, compared with the other algorithms, the proposed algorithm applying meta reinforcement learning generated multicast trees with the lowest cost. Hence, it can be inferred that even in a dynamically changing network environment, the proposed algorithm affords prompt adaptive training performance.

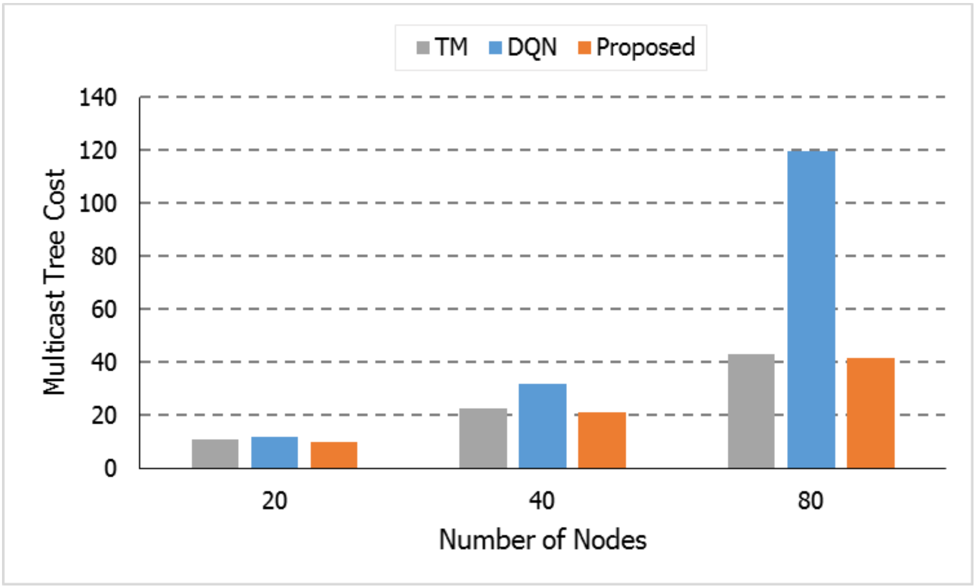


**Fig. 5.** Average Costs of Generated Multicast Trees.

**Fig. 6** shows the average time required for multicast tree generation. It was discovered that the time to construct a multicast tree increased with the number of network nodes in the entire algorithm. In particular, compared with the other learning-based algorithms, the TM algorithm required a significant amount of time to generate a multicast tree. This is because an approximation algorithm, such as the TM algorithm, begins calculation to generate a multicast tree when information regarding a network environment is received. Meanwhile, the algorithm using a trained agent, once it receives information regarding the network environment, instantly generates and returns a multicast tree. So, the learning-based tree construction algorithm using an agent requires less time for multicast tree generation.

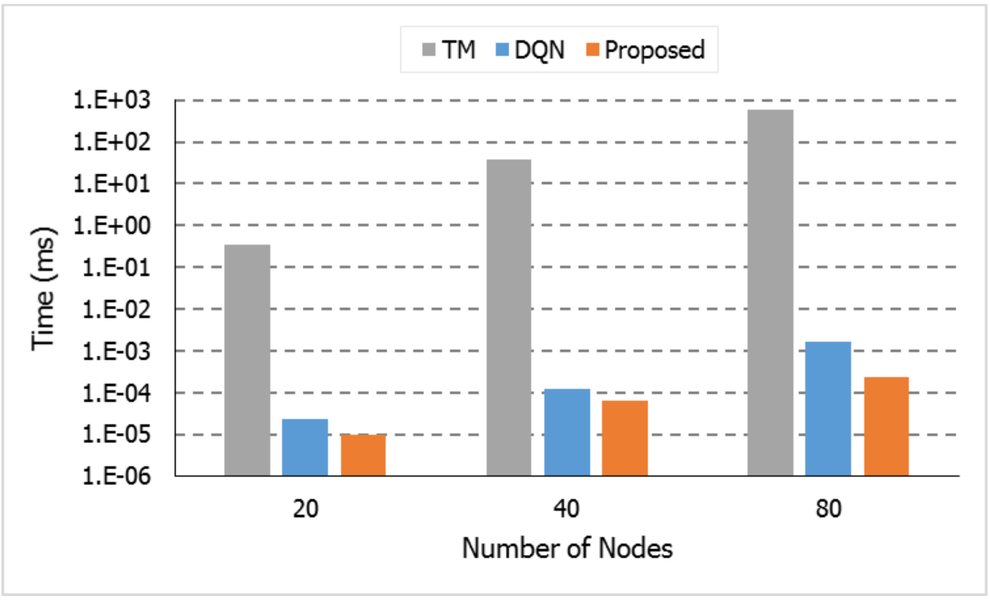


**Fig. 6.** Average Time to Construct Multicast Trees.

**Fig.s 7, 8, & 9** show the average costs of generated multicast trees with different link densities when the network nodes are (20, 40, and 80), respectively. In each network node environment, experiments were performed for (25, 50, and 75) % link densities. A 50 % link density means that in comparison to a fully connected network topology, 50 % of links are connected to nodes. The results indicate that the average cost of multicast trees was higher on lower link density. In particular, when the link density was low, the average cost of the DQN-based algorithm was very high. If many links exist on a network topology, there are many ways to construct a low-cost multicast tree. However, low link density reduces the chances of generating a low-cost multicast tree. This can be especially serious if the agent does not cope well with a new network topology. Therefore, the DQN-based algorithm, which does not work well in the dynamic network environment, has the highest average cost of multicast trees on a low link density. On the other hand, the average costs of multicast trees of the proposed and TM algorithms do not go high, even on a low link density.

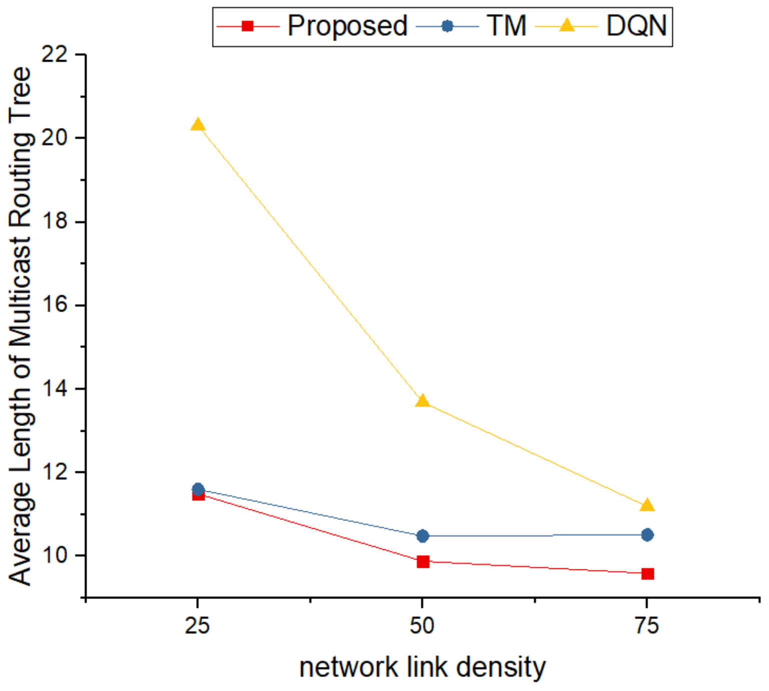


**Fig. 7.** Average Multicast Tree Costs with 20 nodes.

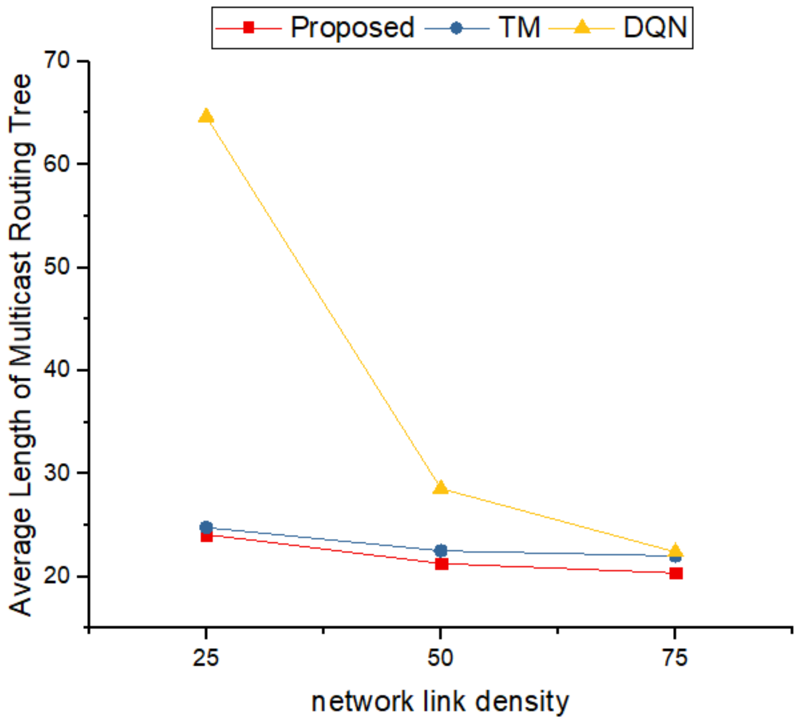


**Fig. 8.** Average Multicast Tree Costs with 40 nodes.

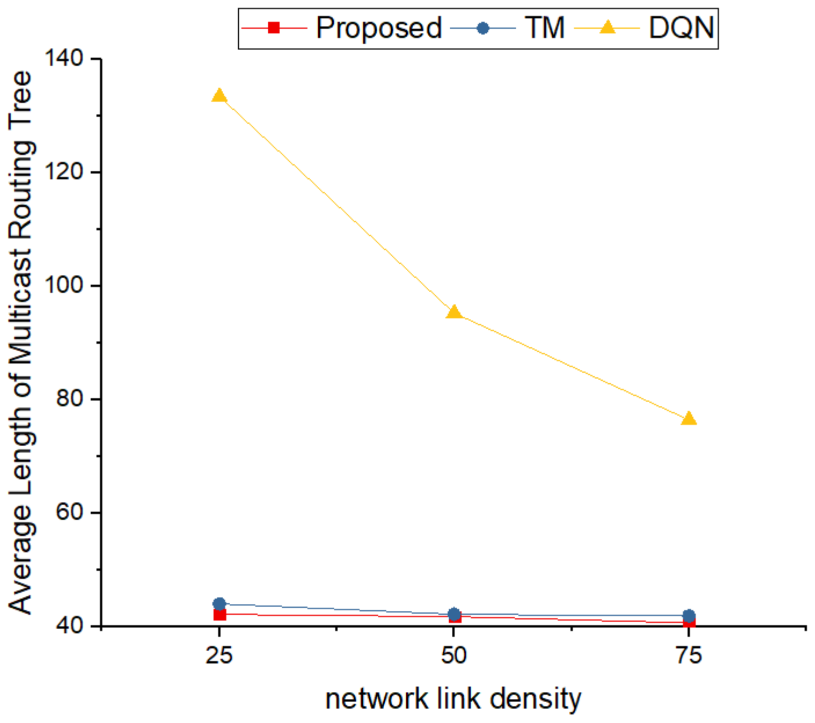


**Fig. 9.** Average Multicast Tree Costs with 80 nodes.

## 5. Conclusion

In this paper, an algorithm that can effectively generate a multicast tree by applying meta reinforcement learning on SDN-based smart network platforms is proposed. Generating an optimal multicast tree requires NP-complete complexity. To solve this problem, several studies have developed approximation algorithms. However, owing to the complex mathematical calculations involved, they require a significant amount of time for multicast tree construction. As the size of networks increases, the amount of calculation of approximation algorithms increases exponentially.

A recently developed algorithm applying DQN-based reinforcement learning uses a trained agent to generate a multicast tree. However, as the amount of state space to be searched increases, it cannot learn effectively. Hence, we propose an algorithm applying model-based meta reinforcement learning; the algorithm facilitates prompt adaptive learning, even in a dynamically changing network environment. The experimental results indicate that compared with the TM approximation algorithm, the proposed algorithm generates a multicast tree within a significantly shorter time. In a dynamically changing network environment, it exhibits better performance than the DQN-based reinforcement learning algorithm.

This study was conducted under the assumption that the changes of the network environments, such as network topologies and multicast node sets, were not correlated. However, in the real world, network topologies and multicast node sets do not change drastically. In the practical network environment, they tend to change gradually over time. Therefore, further studies will be performed to develop a method that enables an agent to simultaneously effectively generate and learn a multicast tree in a gradually changing network environment.

## References

[1] YouTube, http://www.youtube.com/

[2] Netflix, http://www.netflix.com/

[3] Amazon Prime Video, http://www.primevideo.com/

[4] Hulu, http://www.hulu.com

[5] N. Kim and B.-D. Lee, "Analysis and Improvement of MPEG-DASH-based Internet Live Broadcasting Services in Real-world Environments," *KSII Transactions on Internet and Information Systems*, Vol. 13, No. 5, pp. 2544-2557, May 2019. Article (CrossRef Link)

[6] P. Pragyansmita and S. V. Raghavan, "Survey of Multicast Routing Algorithms and Protocols," in *Proc. of the international conference on computer communication*, Vol. 15, No. 3, p. 902. 2002. Article (CrossRef Link)

[7] Jim Esch, "Software-Defined Networking: A Comprehensive Survey," *Proc. of the IEEE*, Vol. 103, No. 1, pp. 10-13, Jan. 2015. Article (CrossRef Link)

[8] F. K. Hwang and D. S. Richards, "Steiner tree problems," *Networks*, Vol. 22, No. 1, pp. 55-89, 1992.

[9] C. Ren, S. Wang, J. Ren and X. Wang, "Traffic Engineering and Manageability for Multicast Traffic in Hybrid SDN," *KSII Transactions on Internet and Information Systems*, Vol. 12, No. 6, pp. 2492-2512, June 29, 2018. Article (CrossRef Link)

[10] X. Chen, J. Wu and T. Wu, "The Top-K QoS-aware Paths Discovery for Source Routing in SDN," *KSII Transactions on Internet and Information Systems*, Vol. 12, No. 6, pp. 2534-2553, June 29, 2018. Article (CrossRef Link)

[11] Z. Zhao, X. Meng, S. Lu and Y. Su, "Different QoS Constraint Virtual SDN Embedding under Multiple Controllers," *KSII Transactions on Internet and Information Systems*, Vol. 12, No. 9, pp. 4144-4165, September 29, 2018. Article (CrossRef Link)

[12] T. Lee, L. Chang, W. Cheng, "Design and Implementation of SDN-based 6LBR with QoS Mechanism over Heterogeneous WSN and Internet," *KSII Transactions on Internet and Information Systems*, Vol. 11, No. 2, pp. 1070-1088, February 27, 2017. Article (CrossRef Link)

[13] H. Takahashi and A. Matsuyama, "An Approximate Solution for the Steiner Problem in Graphs," *Math. Japonica*, Vol. 24, No. 6, pp. 573-577, 1980. Article (CrossRef Link)

[14] A. Z. Zelikovsky, "An 11/6-Approximation Algorithm for the Network Steiner Problem," *Algorithmica*, Vol. 9, No. 5, pp. 463-470, 1993. Article (CrossRef Link)

[15] P. Berman and V. Ramajyer, "Improved Approximations for the Steiner Tree Problem," *Journal of Algorithms*, Vol. 17, No. 3, pp. 381-408, 1994. Article (CrossRef Link)

[16] H. J. Prömel and A. Steger, "RNC-Approximation Algorithms for the Steiner Problem," in *Proc. of Annual Symposium on Theoretical Aspects of Computer Science*, Springer, Berlin, Heidelberg, pp. 559-570, 1997. Article (CrossRef Link)

[17] M. Karpinski and A. Zelikovsky, "New Approximation Algorithms for the Steiner Tree Problems," *Journal of Combinatorial Optimization*, Vol. 1, No. 1, pp. 47-65, 1997. Article (CrossRef Link)

[18] S.-C. Lin, I. F. Akyildiz, P. Wang and M. Luo, "QoS-Aware Adaptive Routing in Multi-Layer Hierarchical Software Defined Networks: A Reinforcement Learning Approach," in *Proc. of IEEE International Conference on Services Computing (SCC)*, 2016. Article (CrossRef Link)

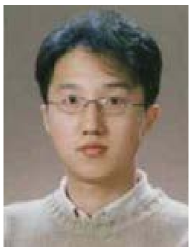
[19] T. Hendriks, M. Camelo and S. Latré, "$Q^2$-Routing: A QoS-Aware Q-Routing Algorithm for Wireless Ad Hoc Networks," in *Proc. of IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 108-115, 2018. Article (CrossRef Link)

[20] S. Sendra, A. Rego, J. Lloret, J. M. Jimenez and O. Romero, "Including Artificial Intelligence in a Routing Protocol using Software Defined Networks," in *Proc. of IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 670-674, 2017. Article (CrossRef Link)

[21] H.-J. Heo, N. Kim and B.-D. Lee, "Multicast Tree Generation Technique using Reinforcement Learning in SDN Environments," *IEEE SmartWorld*, 2018. Article (CrossRef Link)

[22] C. Yu, J. Lan, Z. Guo and Y. Hu, "DROM: Optimizing the Routing in Software-Defined Networks with Deep Reinforcement Learning," *IEEE Access*, Vol. 6, pp. 64533-64539, 2018. Article (CrossRef Link)
[23] Z. Yuan, P. Zhou, S. Wang and X. Zhang, "Research on Routing Optimization of SDN Network using Reinforcement Learning Method," in *Proc. of IEEE International Conference on Safety Produce Information (IICSPI)*, pp. 443-445, 2019. Article (CrossRef Link)
[24] Y.-R. Chen, A. Rezapour, W. G. Tzeng and S. C. Tsai, "RL-Routing: An SDN Routing Algorithm based on Deep Reinforcement Learning," *IEEE Transactions on Network Science and Engineering*, Vol. 7, No. 4, pp. 3185-3199, 2020. Article (CrossRef Link)
[25] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran and M. Botvinick, "Learning to Reinforcement Learn," *arXiv preprint arXiv:1611.05763*, 2016. Article (CrossRef Link)

**Jihun Chae** received the B.S. and the M.S. degree in Computer Science from Kyonggi University, Korea, in 2019 and 2021, respectively. His research interests include deep learning, network routing, multicast, meta-reinforcement learning.



**Namgi Kim** received the B.S. degree in Computer Science from Sogang University, Korea, in 1997, and the M.S. degree and the Ph.D. degree in Computer Science from KAIST in 2000 and 2005, respectively. From 2005 to 2007, he was a research member of the Samsung Electronics. Since 2007, he has been a faculty of the Kyonggi University. His research interests include wireless system, cloud computing, SDN, and mobile platform, image processing, deep learning.