Hawkeye: A Dynamic and Stateless Multicast Mechanism with Deep Reinforcement Learning

Lie Lu[*§], Qing Li[§], Dan Zhao[§], Yuan Yang[‡], Zeyu Luan[*§], Jianer Zhou[§†], Yong Jiang[*§], and Mingwei Xu[‡]

[*]Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen, China
[§]Department of Mathematics and Theories, Peng Cheng Laboratory (PCL), Shenzhen, China
[‡]Department of Computer Science and Technology, Tsinghua University, Beijing, China
[†]Institute of Future Networks, Southern University of Science and Technology, Shenzhen, China

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10228869

***Abstract*—Multicast traffic is growing rapidly due to the development of multimedia streaming. Lately, stateless multicast protocols, such as BIER, have been proposed to solve the excessive routing states problem of traditional multicast protocols. However, the high complexity of multicast tree computation and the limited scalability for concurrent requests still pose daunting challenges, especially under dynamic group membership. In this paper, we propose Hawkeye, a dynamic and stateless multicast mechanism with deep reinforcement learning (DRL) approach. For real-time responses to multicast requests, we leverage DRL enhanced by a temporal convolutional network (TCN) to model the sequential feature of dynamic group membership and thus is able to build multicast trees proactively for upcoming requests. Moreover, an innovative source aggregation mechanism is designed to help the DRL agent converge when faced with a large amount of multicast requests, and relieve ingress routers from excessive routing states. Evaluation with real-world topologies and multicast requests demonstrates that Hawkeye adapts well to dynamic multicast: it reduces the variation of path latency by up to 89.5% with less than 12% additional bandwidth consumption compared with the theoretical optimum.**

***Index Terms*—multicast routing, DRL, BIER-TE**

I. Introduction

Multimedia traffic produced by IPTV, online conferencing, and live streaming grows rapidly and dominates the Internet traffic in recent years. Cisco predicts that video traffic will account for more than 70% of Internet traffic by 2023 [1]. Such momentum continues due to the development of ubiquitous digital devices and emerging applications like 4K/8K videos and AR/VR. Consequently, a great challenge is imposed to traffic engineering (TE) to efficiently use network resources and accommodate user traffic. A number of studies have been proposed to address this issue [2]–[5].

Multicast is promising for multimedia traffic in terms of efficient usage of network resources because a large amount of redundant transmission can be avoided. Unfortunately, traditional network-layer multicast [6] incurs prohibitive control overheads due to per-flow state maintenance in routers. Application-layer multicast reduces traffic pressure on servers, but the efficiency of network resources is not sufficiently optimized [7]. Recently, IETF proposed a source routing paradigm, Bit Index Explicit Replication (BIER) [8], to forward multicast traffic without the requirement of per-flow states in intermediate nodes. On the basis of BIER, Tree Engineering for BIER (BIER-TE) [9] is proposed as a multicast tree construction mechanism, which inherits the advantages of BIER and further enables flexible construction of multicast trees. Although BIER-TE is promising for fine-grained path control over multicast traffic, the problem of efficient multicast tree computation still remains unsolved.

Unlike unicast-based TE that can be formulated as a multi-commodity flow problem or other variants, multicast TE is more complicated. Typically, multicast-based TE is formulated as the Steiner Tree problem, which is NP-hard [10]. Worse still, multicast-based TE with dynamic group membership is even harder. Prior studies usually suffer from high computation complexity even under the assumption of fixed group membership [11], [12]. Moreover, as these methods cannot learn from the past, they only make shortsighted decision for each incoming multicast request, without considering upcoming demands. As such, existing approaches fail to achieve good performance in the long term [13], [14].

We leverage deep reinforcement learning (DRL) to tackle the above challenge. Owing to the ability of online decision making and long-term reward optimization, DRL has been introduced to solve TE related problems in various *unicast* scenarios. Geng et al. [15] leverages the inference ability of DRL to solve complex inter-domain TE problems, achieving less congestion and better scalability than traditional methods. Liu et al. [16], [17] take advantage of DRL's adaptability to dynamic environments for efficient online routing, thus providing near-optimal TE performance under changing network statistics with multiple constraints.

We propose Hawkeye, a DRL-based multicast mechanism built upon BIER-TE protocol to achieve real-time responses to dynamic multicast requests. We first formulate multicast TE as an optimization problem to minimize total network cost under the constraint of path stability. Then, we leverage DRL to learn the multicast traffic pattern from historical requirements and make routing decisions proactively, pro-

This work is supported by the National Key Research and Development Program of China under grant No. 2020YFB1804704, the National Natural Science Foundation of China under grant No. 61972189, the Major Key Project of PCL under grant No. PCL2021A03-1, Shenzhen Science and Technology Innovation Commission: Research Center for Computer Network (Shenzhen) Ministry of Education, and the Shenzhen Key Lab of Software Defined Networking under grant No. ZDSYS20140509172959989.

Corresponding author: Qing Li. Email: liq@pcl.ac.cn

viding near-optimal multicast TE performance. The design of such a DRL-based dynamic multicast mechanism faces two main challenges. First, unlike unicast with only $\mathcal{O}(n^2)$ possible source-destination pairs for routing, in multicast, the combinations of $\mathcal{O}(2^n)$ multicast trees significantly exacerbate the complexity of the problem. As such, the solution space of DRL algorithm increases drastically with dynamic group membership, making it difficult to converge, especially given a large number of requests. Second, the hidden temporal relationship among historical multicast requirements should be effectively mined to facilitate the decision making of the DRL agent. In this paper, we address the above challenges with the following key ideas.

Targeting the convergence problem, we propose a source aggregation mechanism to reduce the solution space of dynamic multicast. Inspired by the observation that groups originating from the same source can share a part of a same multicast tree with little performance degradation, we design a source aggregation method. It merges all multicast requirements rooted from the same source into an aggregated requirement to capture the dominant traffic patterns. Then, the DRL agent only needs to deal with the sequence of aggregated requirements instead of the massive original requirements, thereby ensuring faster convergence and response. Based on source aggregation, we propose an efficient means to build trees, and a storage-efficient method for routing table arrangement.

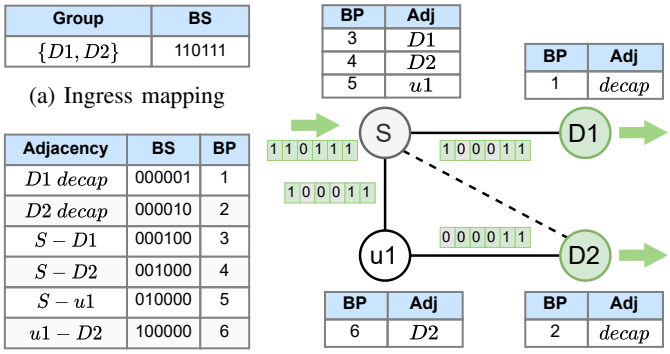
To capture the temporal relationship of multicast requirements, we design a temporal convolutional network (TCN)-based DRL approach for multicast tree generation. It regards the agent's output as a sub-policy to build a multicast tree at each step of a training episode. As such, the agent is able to learn the temporal relationship of consecutive multicast trees. Combining source aggregation with DRL, Hawkeye outputs link weights, based on which proactive routing decisions can be made. It implicitly takes the prospect of future major traffic into consideration, thus ensuring not only real-time response but long-term TE performance.

We evaluate Hawkeye with comprehensive simulations on real-world topologies and multicast requests. The results show that source aggregation can effectively accelerate convergence speed and the DRL-based TE solution outperforms prior multicast methods in terms of bandwidth consumption and path stability. In particular, Hawkeye reduces path latency variation by up to 89.5% with less than 12% additional bandwidth consumption compared with the optimal solution. Compared with ILP-based heuristics, Hawkeye achieves not only faster response but also better online performance.

II. Background and Motivation

A. Background

1) BIER-TE: BIER-TE is a stateless path control mechanism for multicast. It encodes a multicast tree as a Bit String (BS) and encapsulates it in packet headers. Each Bit Position (BP) in BS indicates an unambiguous adjacency of a router in the network, which means an entity adjacent to the router. A router receiving a multicast packet checks the BS in the header, and then forwards (and also replicates if necessary) the packet to an adjacency if the corresponding BP is set to 1. The router is called a Bit Forwarding Router (BFR). In particular, an ingress router is called a Bit Forwarding Ingress Router (BFIR) and an egress router is called a Bit Forwarding Egress Router (BFER).



(a) Ingress mapping

(b) Adjacency to BP/BS

(c) Multicast packet forwarding

Fig. 1: An example of BIER-TE. (a) Ingress mapping of the BFIR (node S). (b) Mapping from adjacencies to BPs. (c) Forwarding process.

Fig. 1 shows an example of BIER-TE. There are two kinds of BPs in Fig. 1(b), i.e., node-BP and link-BP. A node-BP (e.g., *D1 decap*) is set if the node should decapsulate the packet out of the BIER-TE domain. A link-BP (e.g., *S-D1*) is set if the link is part of the multicast tree, i.e., the packet should traverse this link. Assume a multicast packet enters the BIER-TE domain from BFIR *S* towards BFERs *D1* and *D2*, and the multicast tree is {*S–u1*, *u1–D2*, *S–D1*}. First, the BFIR receives the packet encapsulates a BIER-TE header into the it, in which the BS is $\langle 110111 \rangle$ according to the ingress mapping. Then, *S* updates the BS in the BIER-TE header as $\langle 100011 \rangle$ by ANDing the original BS $\langle 110111 \rangle$ and a mask $\langle 100011 \rangle$, which effectively rules out all adjacencies of *S* from the BS to avoid loops. Since BPs 3 and 5 are set in the original BS, *S* sends a copy of the packet along the adjacencies *S-D1* and *S-u1*, respectively. When *D1* receives the packet, it finds BP 1 in the BS, which indicates node-BP *D1* decap, so it decapsulates this packet and passes its payload for higher layer processing. Similarly, the other copy of the packet sent to *u1* is forwarded to *D2* and *D2* decapsulates and processes the packet locally.

2) DRL: Recently, deep neural networks have been widely used in RL for value estimation and policy improvement [18]. This paradigm, called DRL, shows tremendous potential for sequential decision-making problems, making it appropriate for online routing optimization in the context of TE. For example, it has been used to decide link weights [19], traffic splitting ratios [15], and the next hops [17]. As discussed in [20], using link-level presentation as the agent action provides high training efficiency and flexibility for routing policy generation. This action form also benefits the construction of multicast trees. In Hawkeye, we design the agent to observe past multicast requirements and output link-level priorities for

future multicast trees generation.

B. Motivation

1) Why BIER-TE: Compared with traditional stateful multicast protocols, BIER-TE not only fits large-scale multicast better but also provides flexible control of multicast trees for TE. BIER-TE collects multicast requests with a centralized controller and updates routing rules directly at the source router. It can thus respond to multicast requests much faster, consuming less network resources, than the traditional stateful protocols. Moreover, by the specific designs of BS and BP, it enables representations of arbitrary multicast trees with node-BPs and link-BPs to control multicast traffic.

2) DRL for Multicast: In dynamic multicast, the multicast tree is expected to change with dynamic group memberships. Building efficient multicast trees following the real-time group membership is a sequential decision-making problem, which fits the logic of DRL. We summarize three major reasons why DRL is appropriate for dynamic multicast routing problem.

Sequential decisions and delayed rewards. Dynamic multicast routing requires a sequence of decisions that optimize the long-term performance. Traditional methods, such as Integer Linear Programming (ILP) models [21] and heuristics are mainly designed for myopic offline optimization. As a result, they provide local optima which may be far inferior to the global optimum from a long-term perspective. In contrast, the DRL agent evaluates the long-term effect of each decision with the objective of improving the total return containing not only current but also future rewards.

Predictable requirements. The number and location of users in a multicast group have been found to follow some probabilistic models, such as the Poisson distribution [22], [23]. In dynamic multicast, however, the traffic pattern changes over time, which makes it difficult for traditional statistical models to fit the trend timely and accurately. Learning from the past experience, DRL can take full advantage of the inherent characteristics of requirements, and capture the trend of multicast group membership, thus enabling proactive routing.

Readily available training data. The data needed for training is relatively easy to obtain in multicast networks. On the one hand, the network states and performance can be measured easily. For example, multicast requests and join/leave events are naturally collected by protocols (e.g., IGMP [24]), and multicast trees generated by the agent can be evaluated using network topology information. On the other hand, unlike traditional methods that require exact information of the incoming requirements to make correct routing decisions, DRL learns to route based on historical request data which can be accumulated at any time.

III. Design Overview

We present the high-level ideas of Hawkeye design. First, we formulate the multicast routing with dynamic membership as an optimization problem and explain why it cannot be handled efficiently using ILP solvers. Then, we present the overall workflow of Hawkeye, from DRL-based multicast tree configuration to packet forwarding.

A. Problem Formulation

We model the network as a graph $G = (V, E)$, where V and E denote node and edge sets, respectively. Multicast requests arrive at discrete time slots $t = 1, 2, \ldots, T$, with different group memberships. Let F denote the set of multicast flows, and each flow is bound to a group g whose bandwidth demand is q_g. At a specific time slot t, the set of members (destinations) in group g is denoted as D_g^t. There are two binary variables $\psi_{g,e,t}$ and $\phi_{g,d,e,t}$, which indicate the state of edge e in group g at time slot t. The former is set if e is in the multicast tree, and the latter denotes whether e is in the path to node d. We use two kinds of edge weights to calculate static and dynamic costs of multicast trees, respectively. Specifically, ω_e^{st} denotes the cost per unit bandwidth and the static cost measures total bandwidth consumption. ω_e^{dy} denotes the edge latency and the dynamic cost measures the stability of path latencies.

The problem of multicast tree construction with dynamic membership can be formulated as follows:

$$\min \quad \sum_{g \in F} \sum_{e \in E} \sum_{t=1}^{T} q_g \cdot \psi_{g,e,t} \cdot \omega_e^{\text{st}} \tag{1}$$

$$\text{s.t.} \quad \sum_{e \in E: s_e = u} \phi_{g,d,e,t} - \sum_{e \in E: d_e = u} \phi_{g,d,e,t} = \Delta(g,u), \\ t = 1, 2, \ldots, T, \forall g \in F, u \in V, d \in D_g^t \tag{2}$$

$$\phi_{g,d,e,t} \leq \psi_{g,e,t}, \\ t = 1, 2, \ldots, T, \forall g \in F, d \in D_g^t, e \in E \tag{3}$$

$$\sum_{t=1}^{T-1} \sum_{d \in D_g^t \cap D_g^{t+1}} |\sum_{e \in E} (\phi_{g,d,e,t+1} - \phi_{g,d,e,t}) \cdot \omega_e^{\text{dy}}| \leq C_g^{\text{dy}}, \\ \forall g \in F. \tag{4}$$

Objective function Eq.(1). The objective is to minimize the total static cost, including the trees of all groups at all time slots, and the cost of a tree is the weighted sum of all its link costs. The bandwidth consumption is a significant and commonly-used metric for multicast routing.

Flow conservation constraint Eq.(2). Flows entering a node u should also leave the node, i.e., the left traffic equals zero at intermediate nodes, except that u is the source or a destination. In this constraint, s_e and d_e denote the start-point and end-point of a specific edge e. $\Delta(g, u)$ equals 1 if u is the source of the group g, -1 if u is a destination, and 0 otherwise.

Tree constraint Eq.(3). It represents the relationship between two decision variables $\phi_{g,d,e,t}$ and $\psi_{g,e,t}$. Since $\phi_{g,d,e,t}$ and $\psi_{g,e,t}$ are binary variables, $\psi_{g,e,t}$ equals 1 for group g if and only if the edge e appears on the tree of g at any time slot t.

Dynamic cost constraint Eq.(4). It constrains the overheads incurred by tree modifications. Multicast trees may change with dynamic group membership, resulting in changed latencies towards the destinations. Similar to unicast jitter, we accumulate the variations of path latencies of the common destinations between consecutive time slots as a metric, which is limited by a predefined capacity C_g^{dy}.

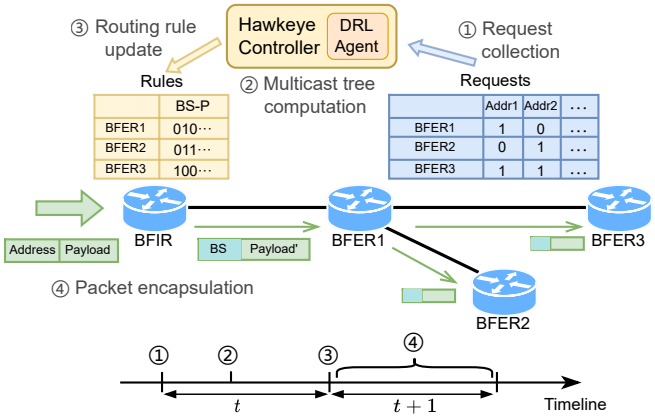


Fig. 2: System overview.

This problem is NP-hard[1]. The optimal solution can be obtained offline using ILP solvers (e.g., Gurobi [25]), with the prior knowledge of destinations D_g^t of each group g at each time. Nevertheless, future multicast requests are inaccessible (or cannot be accurately predicted) at each time slot, making ILP solvers impractical. Even worse, solving this NP-hard problem is time-consuming especially in large topologies.

B. System Overview

Hawkeye adopts DRL for proactive dynamic multicast routing, which not only provides near-optimal decisions quickly but also learns the inherent property of group membership changes. However, the large number of concurrent multicast groups prevents the DRL agent from efficiently learning. To accelerate the learning process, we propose a *source aggregation* mechanism, which abstracts the major traffic patterns by aggregating all multicast requirements rooted at the same source. The DRL agent only deals with the aggregated traffics rather than the original ones.

In the following, we present the workflow of Hawkeye that contains 4 phases spanning two consecutive time slots, as illustrated in Fig.2. The Hawkeye controller performs phases 1–3 to generate multicast trees, and the final phase is to process the multicast packet in the BFIR.

Phase 1: multicast request collection. At the beginning of time slot t, the Hawkeye controller collects multicast requests from all destinations for a specific source. We denote multicast demands as a matrix. Each column in the matrix represents a multicast group identified by a multicast address, and each element equals 1 if there is a multicast request from a specific BFER. Once the Hawkeye controller obtains all requests from BFERs, it performs source aggregation to obtain an aggregated requirement (detailed in Section IV-A.). Then, the controller submits the aggregated requirement to the DRL agent.

Phase 2: multicast tree computation. The agent is designed to make proactive routing decisions for upcoming requirements, i.e., it plans routes for time slot $t+1$ at t, so that the ingress packets can be delivered immediately instead of experiencing time-consuming computation at $t+1$. However, the multicast requirements of $t+1$ are unavailable at time t, and hence the destinations are unknown. So we compute in advance a spanning tree connecting all the BFERs in the network for $t+1$. In particular, the agent takes the *aggregated requirements* in the past time slots as input and outputs link priorities, based on which a spanning tree is generated. During this process, the major multicast traffic patterns are implicitly taken into consideration by training on historical requests.

Phase 3: routing rule update. At the beginning of the second time slot $t+1$, the requests of $t+1$ are collected and then aggregated as an aggregated requirement. The controller generates the routing table for these requests as follows. First, the controller directly extracts paths for BFERs of the aggregated requirement from the spanning tree generated in Phase 2 to form a *trunk* that carries the majority traffic. Then, the controller finds paths for other BFERs, i.e., BFERs not involved in the aggregated requirement, using the weighted shortest path algorithm (e.g., Dijkstra's Algorithm). Given these paths, multicast trees can be generated by merging the paths of all involved BFERs. Particularly, paths on the trunk always take precedence when conflicts of path segments happen (detailed in Section IV-B). Finally, the paths are translated to routing rules, sent to and installed in the BFIR.

Phase 4: packet forwarding. When a multicast packet encounters the first BFIR in the the domain, the BFIR looks up the involved BFERs of this packet according to its multicast address and sets corresponding BPs to generate the BS. Specifically, each BS consists of both node-BPs of the destinations and link-BPs of the links on the path to these destinations. Then, the BFIR encapsulates a BIER-TE header encoded with BS into the packet for multicast forwarding in this domain.

IV. Source Aggregation

To optimize the routing policy in dynamic multicast scenarios, the DRL agent needs to explore the solution space of all possible multicast requirements. However, the DRL agent struggles to converge to the optimal solution due to numerous multicast requests and dynamic group memberships. Therefore, we design a pre-processing technique named source aggregation to aggregate original multicast requirements, thus reducing the solution space to be explored by the DRL agent. Besides, source aggregation relieves the BFIR from excessive storage overheads and improves the scalability of BIER-TE.

The proposed mechanism aggregates concurrent multicast requirements rooted from the same source based on their bandwidth requirements and group membership. In particular, source aggregation is used in both training and inference of the agent. During training, the historical requirements are aggregated using the aggregation method described in Section IV-A, and the aggregated requirements are taken as inputs instead of the original ones to ensure fast convergence. During inference, to keep consistency with the optimization objective used in training, Hawkeye first aggregates the real-time requirements and builds a multicast tree for the aggregated requirement.

[1]The Steiner tree problem can be reduced to this problem by setting the weight ω_e^{dy} as 0, and the former is proven to be NP-hard [10].

Then, the multicast trees for the original real-time requests are generated following the steps outlined in Section IV-B.

A. Aggregating original requirements

Recall that a BFIR is an ingress router of a multicast domain, and the BFIR accommodates source nodes of different multicast groups. We aggregate the bandwidth requirements of these groups and create an aggregated bandwidth requirement for this BFIR. Note that members of a multicast group may differ from those of other groups, so we need to consider possible members (i.e. BFERs) all at once.

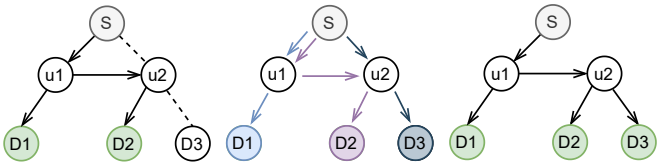
Formally, let $b \in V$ denote a BFIR, and $\mathbf{g}_b$ be the set of multicast groups whose source nodes are located at b. For each $v \in V$ and $g \in \mathbf{g}_b$, let $\delta_{v,g}$ be a binary variable, which equals 1 if v is a member of group g, and vice versa. Recall that q_g is the bandwidth requirement of group g. Let $f_{b,v}$ denote the total bandwidth requirement that BFIR b needs to deliver to BFER v. Then we have $f_{b,v} = \sum_{g \in \mathbf{g}_b} \delta_{v,g} q_g$. We organize the total bandwidth requirements from all BFERs as a vector of $|V|$ elements, i.e. $\vec{f_b} = [f_{b,v}]_{v \in V}$.

Note that the total bandwidth requirements from different BFER v may be different from each other, which is not in the form of a single multicast group requirement. We process $\vec{f_b}$ in such a way that the aggregated requirement follows the form of a single multicast group requirement and can be dealt with our DRL agent more easily. We define aggregation ratio λ, which is a fractional constant between 0 and 1. Let $\eta = \lambda \max_{v \in V} f_{b,v}$ be a threshold. For each $v \in V$, we replace $f_{b,v}$ in $\vec{f_b}$ by 0 if $f_{b,v} < \eta$, and obtain vector $\vec{f_b'}$. Then, we compute the average of the non-zero elements in $\vec{f_b'}$, and replace each non-zero element in $\vec{f_b'}$ by the average. The resulting vector is denoted by $\vec{f_b^A}$ which represents the aggregated requirement.

We illustrate the above process by an example. Consider a network with one BFIR b and three BFERs v_1, v_2, v_3, and there are two multicast groups with the BFIR as source node. The first group involves 4 units of bandwidth requirement, and the members are v_2 and v_3. The second group involves 6 units of bandwidth requirement, and the members are v_1 and v_3. The total bandwidth requirement vector is then $\vec{f_b} = [0+6, 4+0, 4+6] = [6, 4, 10]$. Given the aggregation ratio $\lambda = 0.5$, the threshold is then $10 \times 0.5 = 5$. Thus, we obtain $\vec{f_b'} = [6, 0, 10]$ by replacing the second element which is less than the threshold by 0. The average of non-zero elements of $\vec{f_b'}$ is 8, so finally, we obtain the aggregated requirement $\vec{f_b^A} = [8, 0, 8]$. We can see that this aggregated requirement preserves some traffic patterns of the original requirements.

B. Building Trees for original requirements

Recall that Hawkeye uses DRL to build a multicast tree, i.e., the trunk, for the aggregated requirement in Phases 2 and 3. Thus, an incoming user request arriving at a BFER that is already in the trunk can be satisfied directly. However, the trunk may not cover all possible BFERs, and we need to generate routing rules for incoming requests arriving at the uncovered BFERs.



(a) Trunk (b) Paths (c) Multicast tree for BFERs *D1*,*D2*,*D3*

Fig. 3: An example of multicast tree building based on source aggregation. (a) The trunk. (b) Paths to each destination. (c) The tree spanning *D1*, *D2* and *D3*.

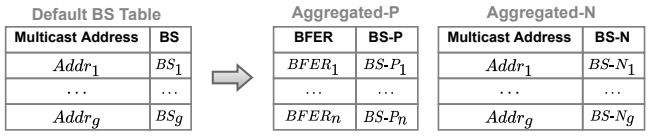


Fig. 4: The BFIR table before and after aggregation.

We observe that with source aggregation and the constructed trunk, the majority of multicast traffic can be delivered efficiently with little impact on the objective. Thus, it is unnecessary to rebuild the whole multicast tree with the DRL agent, and we take a simple heuristic to route the incoming request. Following the notations used above, let $b \in V$ be the BFIR, $o \in V$ be the BFER where a new request arrives, T be the trunk rooted at b, and $SP_{i,j}$ denote the shortest path from i to j. Then, we use $SP_{b,o}$ directly as the forwarding path of the new request if $T \cap SP_{b,o} = \phi$, or else, we let u be the node on $T \cap SP_{b,o}$ that is nearest to o, and concatenate $SP_{u,o}$ to trunk T to obtain the forwarding path. This heuristic enables a short latency from BFIR b to BFER o.

We use an example shown in Fig. 3 to illustrate the process of constructing forwarding paths given the trunk. In Fig. 3(a), the solid arrows indicate the trunk, which is rooted at BFIR S and covers BFERs *D1* and *D2*. Thus, the paths from S to $D1$ and $D2$ can be extracted directly. In particular, P_1 is $\langle S\text{-}u1\text{-}D1 \rangle$ and P_2 is $\langle S\text{-}u1\text{-}u2\text{-}D2 \rangle$. Then, we compute the forwarding path for BFER $D3$. We first compute the shortest paths from S to $D3$, and obtain $P_3 = \langle S\text{-}u2\text{-}D3 \rangle$, as shown in Fig. 3(b). We see that P_3 intersects with the trunk at node *u2*, and if we use P_3 directly, there will be duplicated packets when $D2$ and $D3$ are in the same group, which degrades the performance. To address the issue, we concatenate sub-path $u2\text{-}D3$ to T and obtain $P_3' = \langle S\text{-}u1\text{-}u2\text{-}D3 \rangle$. As a result, a multicast tree containing any arbitrary subset of {*D1*, *D2*, *D3*} can be generated. For example, the multicast tree for a group with BFERs *D1*, *D2*, *D3* is shown in Fig. 3(c).

C. Storage-efficient Routing Table Arrangement

Although we aggregate the majority of requirements and construct a trunk to deliver traffic, routing rules should be maintained for each group separately, because different multicast groups may have different members. We propose storage-efficient routing table arrangement to reduce the storage overhead on BFIRs.

Under the default setting of BIER-TE, a BFIR maintains a BS for each multicast group (see *Default BS Table* in Fig. 4). Each BS contains node-BPs and link-BPs indicating the specific multicast tree. Based on source aggregation, there is only one path to each destination, regardless of which group the destination belongs to. We split the default BS table into two tables. In particular, an *Aggregated-P* table maintains only link-BPs (BS-P), which indicate forwarding paths to BFERs. An *Aggregated-N* table records the members of each group using node-BPs (BS-N). Upon receiving a multicast packet, the BFIR first looks up the *Aggregated-N* table with the multicast address, and obtains BS-N indicating the destinations (BFERs). The BFERs are used as inputs to look up the *Aggregated-P* table, and we obtain the BS-Ps. Then, we perform logical OR operations on these BS-Ps to merge the forwarding paths, and concatenate the result to the BS-N to obtain the complete BS. Finally, the BS is encapsulated into the BIER-TE header of the packet.

We analyse the storage conservation on a BFIR. Recall that there are $|\mathbf{g}_b|$ groups whose sources are located at BFIR b. Let l denote the multicast address length. The length of a single BS-P is $|E|$, and the length of a single BS-N is $|V|$. The storage cost before source aggregation is $\mathcal{O}(|\mathbf{g}_b| \cdot (l + |E| + |V|))$, and the storage cost after source aggregation is $\mathcal{O}(|\mathbf{g}_b| \cdot (l + |V|) + |V| \cdot (|V| + |E|)) = \mathcal{O}(|\mathbf{g}_b| \cdot (l + |V|))$, a reduction of $\mathcal{O}(|\mathbf{g}_b| \cdot |E|)$.

V. Multicast Tree Computation

Though source aggregation improves the learning efficiency of the DRL agent, it is still non-trivial for the DRL algorithm to achieve good TE performance due to the complicated spatial-temporal correlations. In Hawkeye, we sophistically design a DRL approach to facilitate multicast tree computation.

A. State, Action, and Reward

State. The state is represented as a *sliding window* of historical requirements. In particular, a requirement $req_t \in R^n$ represents the bandwidth demand of each node at time slot t. The state $S_t \in R^{n \times w}$ concatenates requirements in the past w time slots in reverse chronological order, i.e., $S_t = (req_{t-1}, req_{t-2}, \ldots, req_{t-w})$. The window size w determines the scope of historical information taken by the RL agent. A larger window may facilitate the DRL agent to make better decision at the expense of degraded training efficiency.

Action. To learn the relationship among consecutive requirements, we prefer generating a multicast tree directly within a single step, which poses two challenges to action design. First, it is time-consuming for the DRL agent to converge in the solution space for all possible multicast trees in the topology. Second, the output dimension of valid multicast trees varies with groups, which violates the requirement of the fixed-dimension action space in DRL. To cope with the two challenges, we adopt *policy-based tree generation*, which uses the output action of the agent as sub-policies to assist the generation of multicast trees. We define the output action of the DRL agent at time step t as follows:

$$\rho(e_i \mid s_t) = p_t^i, \quad i = 1, 2, \ldots, m, \tag{5}$$

where s_t is the state, e_i is an edge, and p_t^i indicates the priority of this edge. Having computed the priorities of all edges, the multicast tree can be constructed as follows:

(1) Initialize a subgraph with all terminal nodes (i.e., the source and destinations) according to the requirement, use the source node as the starting node, and add all its neighboring edges to the candidate edge set E_c.
(2) Choose an edge from E_c with the highest priority, add this edge as well as its endpoint to the subgraph, and update E_c by deleting this edge and adding neighboring edges of the just added end node.
(3) If all the terminal nodes are connected, merge the shortest paths from the source to destinations to generate a multicast tree. Otherwise, back to step (2).

This procedure enables the agent to compute a multicast tree given the destinations of a multicast requirement. To proactively make a routing decision for upcoming multicast request, we regard all nodes as potential terminal nodes in step (1), and make the agent to generate a spanning tree in advance. Then, routing rules generated from the spanning tree can be installed in BFIRs at the beginning of the next time slot, thus allowing fast response for arrived multicast requests.

Reward. The main reward is the bandwidth cost of the current multicast tree generated at each step. Meanwhile, the dynamic cost is accumulated from the start of an episode. If the dynamic cost exceeds the pre-defined threshold, this episode is terminated, and returns a negative reward (-10) as a penalty. Moreover, the dynamic cost is also added to the reward with a small weight α. This leads the agent to reduce the dynamic cost to satisfy the capacity constraint, thereby speeding up the training process. The reward is represented as

$$r(s_t, \rho) = -(\text{cost}_t^{\text{st}} + \alpha \cdot \text{cost}_t^{\text{dy}}), \tag{6}$$

where $\text{cost}_t^{\text{st}}$ denotes the bandwidth cost of the tree, $\text{cost}_t^{\text{dy}}$ denotes the dynamic cost with respect to step t.

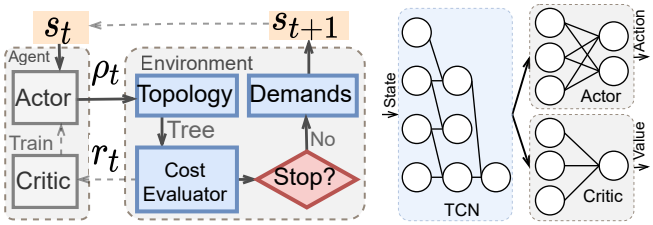
The dynamic cost of the aggregated requirement may be inconsistent with that of the original requirements, which violates the dynamic capacity constraint (4). To address this problem, we use a stricter calculation for dynamic cost,

$$\text{cost}_t^{\text{dy}} = \begin{cases} 0, & \text{for } t = 1, \\ \sum_{d \in D_g^t \cup D_g^{t+1}} |\text{delay}_t^d - \text{delay}_{t-1}^d|, & \text{otherwise.} \end{cases} \tag{7}$$

Unlike (4), which uses the intersection of D_g^t and D_g^{t+1}, the dynamic cost here depends on the destinations in either D_g^t or D_g^{t+1}. Using this upper bound of original dynamic cost, we ensure the feasibility of DRL solutions.

B. Training

We leverage Proximal Policy Optimization (PPO) [26], which is a policy-based algorithm designed for continuous



(a) Training process (b) Actor-Critic Network

Fig. 5: DRL design of Hawkeye.

TABLE I: Hawkeye Hyperparameters

Parameters	**Value**	**Parameters**	**Value**
Max steps per epoch	1000	Max epochs to train	3200
PPO-clip ratio ϵ	0.2	Discount factor γ	0.99
Actor learning rate	0.0003	GAE lambda λ	0.97
Critic learning rate	0.001	MLP hidden layers	16x16
State window size w	4,8,16,32	Input network type	TCN/MLP
PLV weight α	0, 0.01, 0.1, 0.5, 1		

control. PPO strikes a good balance between learning efficiency and structure simplicity, thereby satisfying the requirements of fast response and high robustness for multicast routing. It consists of two neural networks, actor $\pi_{\theta_\pi}(\rho \mid s)$ and critic $V_{\theta_v}(s)$. Fig. 5(a) shows the training process. At time step t, a sequence of historical requirements are gathered as s_t. At time step t, a sequence of historical requirements are gathered as s_t. The agent takes s_t as input and outputs a sub-policy ρ_t, which is used to generate a multicast tree. The environment integrates the topology information and multicast requests to evaluate the static and dynamic cost of the tree. Then, it returns a reward used by the critic to update the neural networks through backpropagation. Before transferring to the next state s_{t+1}, it decides whether to terminate the episode at this step by comparing the cumulative dynamic cost. If this is the case, the episode is cut off and a new one is initiated.

We use the clip version of PPO, where the loss of actor is

$$L = \min\left(\omega_\theta A(s,\rho),\ \text{clip}\left(\omega_\theta, 1-\epsilon, 1+\epsilon\right) A(s,\rho)\right), \quad (8)$$

$$\omega_\theta = \frac{\pi_\theta(\rho \mid s)}{\pi_{\theta_{old}}(\rho \mid s)}. \quad (9)$$

ω_θ represents the difference between the current policy under updating and the old policy used for action sampling. The clip operation prevents the policy from changing dramatically, where the clip ratio ϵ sets the upper bound of the loss. $A(s,\rho)$ represents the advantage of ρ, calculated using GAE-λ [27].

C. Neural Network Architecture

We use a TCN [28] to encode the series of requirements as a state embedding in the actor-critic network. TCN is essentially a convolutional network with specific designs for temporal sequence modeling, which has been reported to achieve better performance than RNN [29]. Through 1D-convolution, it extracts the temporal features of sequences and model the internal relationship across different time slots. Owing to dilation, its receptive field increases exponentially with depth, enabling efficient long sequence process. Mathematically, given the state with w multicast requirements $s_t = S^{t-1:t-w} = (req_{t-1}, \ldots, req_{t-w})$ and a filter f with size K, the output of the TCN is represented as

$$(S \star f)(t) = \sum_{i=0}^{K-1} f(i) S(t - d \times i), \quad (10)$$

where d is the dilation factor controlling the convolution interval. As illustrated in Fig. 5(b), the state is firstly processed by a 2-layer TCN with filter size $K = 2$, and then forwarded to the actor and critic. The actor and critic networks are both Multi-Layer Perceptrons (MLPs) which extract the spatial relationship among nodes in the topology.

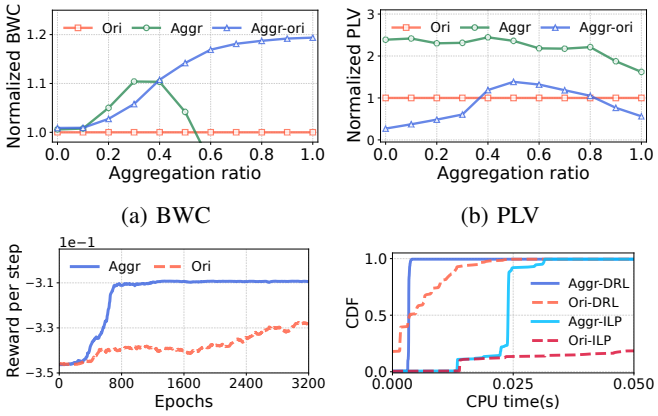
VI. Evaluation

We implement the PPO algorithm based on the SpinningUp [30] framework. We find the algorithm insensitive to most hyperparameters, so we use the default setting of SpinningUP as listed in Table I, and others are evaluated in Section VI-C. ILP models are solved using the Gurobi Optimizer [25] with a 16-core 2.3 GHz CPU. We use three real-world topologies from SNDlib [31], namely Abilene, Geant, and Germany50. Abilene is a small topology with 11 nodes and 14 bidirectional links. Geant has 23 nodes and 37 bidirectional links, and Germany50 has 50 nodes and 88 bidirectional links.

The multicast requests are generated based on real-world data. For Abilene, we use a dataset derived from Facebook [32]. It provides two-week public live video requests, including the locations of online streamers and the viewers. We treat each video as a multicast group, allocate its users to each node by their locations, and update its status every 5 minutes. There are about 500 groups for each source with dynamic membership. For Geant and Germany50, we use a typical multicast traffic model to simulate the dynamic membership [22], where the requirements changes with locations and time, proportional to the total traffic obtained from SNDlib.

To measure the efficiency and stability of multicast routing, we use the bandwidth cost (BWC) as the static cost, and the path latency variation (PLV) as the dynamic cost. We choose four multicast routing algorithms for comparison.

(1) **SPT**. The shortest path tree connects the source and each destination with the shortest path.
(2) **OPT**. The optimal solution connects the source and destinations with the minimum BWC under PLV constraints.
(3) **RL-TG**. The tree generated by a DRL-based algorithm [33]. It simply builds a tree for every instant requirement without the consideration of path stability.
(4) **HST**. A heuristic computes multicast trees proactively where the upcoming requests are estimated with the average traffic demand of the past hour. Routing decisions are made by the ILP model with source aggregation.



(a) BWC (b) PLV

(c) Learning curve (d) Time for decision making

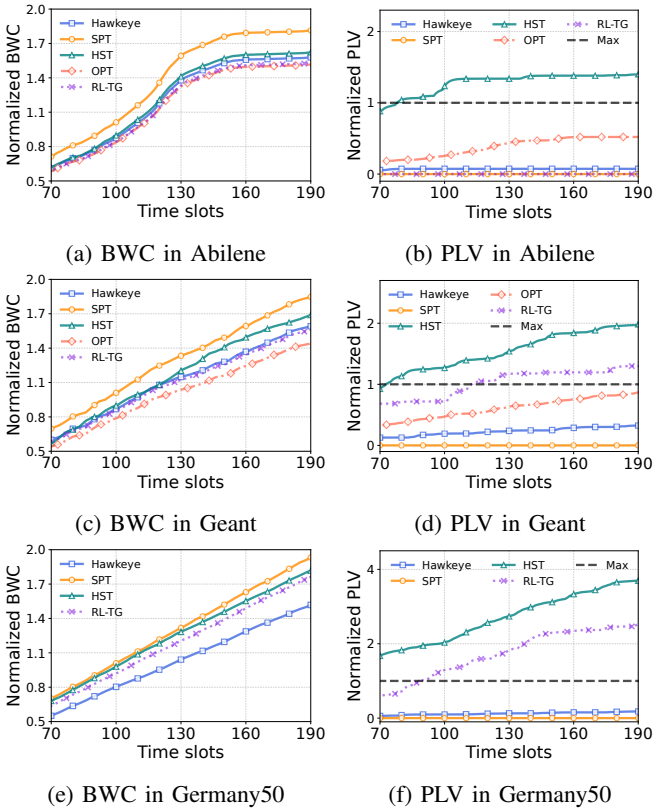
Fig. 6: Performance of source aggregation.

A. Performance of Source Aggregation

We compare the performance of OPT before and after source aggregation under different aggregation ratios. The ratio controls the number of group members after aggregation. If the ratio is 0, all the members appearing in the original requirements would be taken into the aggregated requirement. As the ratio increases, the nodes with lower bandwidth demands are removed. Finally, only nodes with the maximum bandwidth demands are taken when it equals 1.

In Fig. 6, *Aggr* means the optimal multicast tree of the aggregated requirement, and *Ori* means that of original requirements without aggregation. *Aggr-ori* represents the performance of Hawkeye, which builds trees for original requirements based on the optimal tree of the aggregated requirement. Fig. 6(a) indicates the extra BWC induced by source aggregation is less than 10% comparing *Aggr-ori* with *Ori* when the aggregation ratio is less than 0.4, with the minimum BWC gap less than 1%. As the aggregation ratio increases, the tree of *Aggr-ori* gradually regresses into the shortest path tree so its BWC converges to that of SPT. Besides, the PLV of *Aggr-ori* is always less than that of *Aggr* as shown in Fig. 6(b), which is consistent with the design of (7) that preserves the feasibility of source aggregation.

Next, we measure how the agent perform with and without source aggregation. In Fig. 6(c), the *Aggr* agent learns faster and better, while *Ori* cannot converge within the same training iterations. We also measure the CPU time usage for 200 time slots comparing ILP solver with DRL (GPU disabled) in Fig. 6(d). *Ori* has to make decisions for every group and *Aggr* only focuses on the aggregated requirement and then applies the solution to original requirements. At the start of a time slot, *Aggr* need to aggregate requirements, make a decision for the aggregated requirement, and transform the solution to original ones. The operation of aggregation and transformation takes nearly fixed time, and the overall time usage remains low compared with *Ori*. This suggests that source aggregation can accelerate the response speed significantly especially for the DRL agent, which takes no more than 5 ms in Abilene.



(a) BWC in Abilene (b) PLV in Abilene

(c) BWC in Geant (d) PLV in Geant

(e) BWC in Germany50 (f) PLV in Germany50

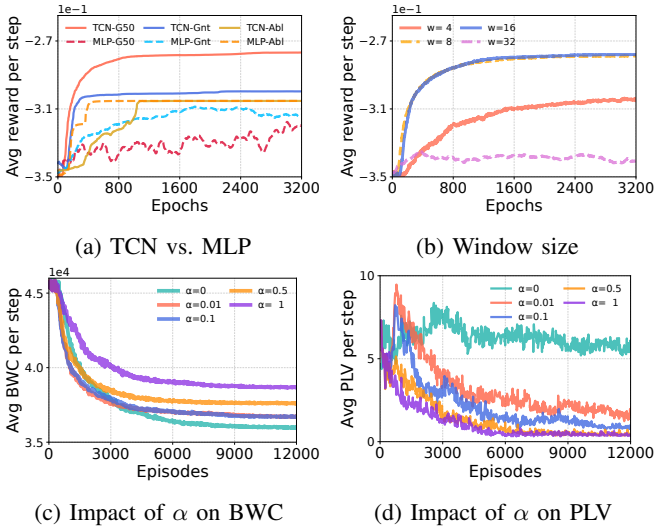
Fig. 7: Performance of multicast trees in three topologies.

B. Performance of Multicast Trees

We compare the cumulated BWC and PLV of each method in three topologies, shown in Fig. 7, where the dashed line *Max* represents the maximum PLV allowed in the network. For all topologies, the BWC of SPT is the highest among all the methods, while PLV shows an opposite trend. This is because SPT always uses the shortest paths to build each multicast tree, which remain unchanged with dynamic membership, resulting in high bandwidth consumption. In contrast, OPT optimizes the global BWC under PLV constraints, which presumes an precise future vision towards the upcoming requests, providing the best but impractical performance. Meanwhile, without the help of DRL, HST fails to make full use of historical experience, resulting in sub-optimal BWC and inferior PLV.

In Abilene, a small-scale network, the impact of dynamic membership is limited, so methods except SPT have similar BWC, as shown in Fig. 7(a). However, the PLV of OPT is only 37% of HST, indicating more stable routing can be achieved with a neglectable BWC increase. The performances of Hawkeye on both BWC and PLV are close to OPT, where the differences are less than 5%, which demonstrates that Hawkeye performs well in small-scale topology. The BWC results of Geant are similar, but the PLV of RL-TG exceeds the limit while that of Hawkeye remains low due to its sepcific design. For SPT, the BWC grows with the topology scales, which indicates that SPT is not efficient in large network.

In Germany50, the ILP model of OPT cannot be solved in



(a) TCN vs. MLP

(b) Window size

(c) Impact of α on BWC

(d) Impact of α on PLV

Fig. 8: Impact of Hawkeye hyperparameters.

a reasonable time, so its results are omitted. Due to the large search space, RL-TG fails to find the optimal trees whose BWC and PLV degrades dramatically. Hawkeye also falls into a local optimum where the PLV is only 17.2% of *Max*, which means it could trade more PLV for less BWC. However, it makes proactive routing decisions while ensuring expected PLV, with excess BWC less than 12% of OPT in Abilene and Geant, which still outperforms SPT and RL-TG. Different from HST, Hawkeye adjusts multicast trees to satisfy stability requirements while keeping low bandwidth cost.

C. Hyperparameters

TCN vs. MLP. Fig. 8(a) shows the learning curves with TCN/MLP of three topologies abbreviated as Abl, Gnt, and G50, respectively. For Abilene, TCN and MLP converge to the same reward, suggesting the DRL algorithm is able to solve such problems in simple networks. TCN converges lightly slowly due to more complex structure. In Geant, the reward improves slowly with MLP and finally stops at a less optimal level. In Germany50, MLP cannot converge while TCN still performs stably. To summarize, MLP is too simple to model the complicated temporal relationship in large networks.

State Window Size. The window size w determines how far the agent looks back, thereby affecting the learning process. We measure the rewards during training under different w in Germany50, as shown in Fig. 8(b). If the window is too short (e.g. $w = 4$), the reward remains low due to a lack of information, while an overly large window may mislead the agent since the input state becomes too complicated. The overlapping results of $w = 8$ and 16 suggest the agent can achieve good performance within a reasonable range of w.

PLV Weight. The PLV weight α in (6) controls the relative importance of PLV versus BWC. A larger α guides the agent to pay more attention to PLV constraints. Fig. 8 shows how the average BWC and PLV change during training in Germany50. When $\alpha = 0$, i.e., the agent only focuses on BWC, the BWC slowly converges to the lowest at the expense of a high PLV. As α increases, the PLV decreases more and more rapid, but the performance of BWC degrades. To some extent, this parameter serves as a knob for exploration-exploitation trade-off. With a smaller α, the agent tends to explore more thus having the potential to achieve a lower BWC.

VII. Related Work

Stateful multicast protocols require frequent distributed state updates and SDN is a promised mechanism to solve the problem. Huang et al. [12] build multicast trees in SDN under constraints of node and link capacity. Mohammadi et al. [11] use a nature-inspired optimization algorithm to compute the minimum cost tree regarding the end-to-end delay. However, these methods are designed for static multicast and do not work well with dynamic membership. Chiang et al. [14] optimize the bandwidth cost and rerouting overhead jointly in dynamic multicast. These studies are all short-sighted heuristics and lack consideration of long-term performance.

Explicit Multicast [34] is a traditional *stateless* protocol where the packet header carries IP addresses of all destinations, which inherently cannot work with many concurrent group members. Cheng et al. [35] propose a source routing method based on bloom filter, which also uses a BS to mark destinations but may cause unnecessary bandwidth waste due to false positives. Khaled et al. [36] design a label-based system to support multicast forwarding with general graphs, which still poses high overhead at ingress routers. Hawkeye adopts efficient source aggregation to provide both flexible traffic control and better scalability at source routers.

DRL has been widely used for the *unicast* traffic management. Li et al. [37] leverage multi-objective DRL to generate optimal policies for all possible routing preferences. Geng et al. [15] use multi-agent DRL to achieve distributed TE for global objective optimization. Liu et al. [17] design an online routing algorithm for multiple QoS requirements. DRL applied in *multicast* is mostly limited in wireless networks with objectives about resource allocation, such as interference mitigation [38], capacity limitation [39] or energy consumption [16], [40]. These methods are mainly designed for ad hoc networks, which is difficult to generalize to arbitrary topologies. In addition, existing DRL-based research for multicast *routing* focuses mainly on simple objectives without constraints [41].

VIII. Conclusion

We present Hawkeye, a dynamic multicast system based on BIER-TE. Hawkeye adopts source aggregation for efficient training and source router storage saving, and designs a TCN-based DRL framework for high-performance multicast tree computation. Evaluation results show that Hawkeye produces near-optimal solutions for multicast routing with dynamic membership, provideing more stable multicast trees with little additional bandwidth consumption. Furthermore, it makes proactive routing decisions based on the traces of historical requirements to ensure real-time responses, which overcomes the slow convergence issue of traditional multicast methods.

REFERENCES

[1] "Cisco annual internet report - (2018–2023) white paper," https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[2] J. Zhang, X. Zhang, and M. Sun, "Two-level decomposition for multi-commodity multicast traffic engineering," in *2017 IPCCC*, Dec. 2017, pp. 1–2.

[3] S. Yang, C. Xu, L. Zhong, J. Shen, and G.-M. Muntean, "A QoE-Driven Multicast Strategy With Segment Routing—A Novel Multimedia Traffic Engineering Paradigm," *IEEE Trans. Broadcast.*, vol. 66, no. 1, pp. 34–46, Mar. 2020.

[4] R. Singh, S. Agarwal, M. Calder, and P. Bahl, "Cost-effective Cloud Edge Traffic Engineering with Cascara," in *NSDI 21*, 2021, pp. 201–216.

[5] G. Bernárdez, J. Suárez-Varela, A. López, B. Wu, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Is Machine Learning Ready for Traffic Engineering Optimization?" in *2021 ICNP*, 2021, pp. 1–11.

[6] B. Fenner, M. J. Handley, H. Holbrook, and L. Zheng, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)," Internet Engineering Task Force, Request for Comments RFC 7761, 2016.

[7] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Georganas, "A survey of application-layer multicast protocols," *IEEE Commun. Surv. Tut.*, vol. 9, no. 3, pp. 58–74, 2007.

[8] I. Wijnands, E. Rosen, A. Dolganow, T. Przygienda, and S. Aldrin, "Multicast Using Bit Index Explicit Replication (BIER)," RFC Editor, Tech. Rep. RFC8279, 2017.

[9] T. Eckert, M. Menth, and G. Cauchie, "Tree Engineering for Bit Index Explicit Replication (BIER-TE)," Internet Engineering Task Force, Internet Draft draft-ietf-bier-te-arch-13, 2022.

[10] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. Springer US, 1972, pp. 85–103.

[11] R. Mohammadi, R. Javidan, M. Keshtgari, and R. Akbari, "A novel multicast traffic engineering technique in SDN using TLBO algorithm," *Telecommun. Syst.*, vol. 68, no. 3, pp. 583–592, 2018.

[12] L.-H. Huang, H.-C. Hsu, S.-H. Shen, D.-N. Yang, and W.-T. Chen, "Multicast traffic engineering for software-defined networks," in *IEEE INFOCOM*, 2016, pp. 1–9.

[13] S.-H. Chiang, J.-J. Kuo, S.-H. Shen, D.-N. Yang, and W.-T. Chen, "Online multicast traffic engineering for software-defined networks," in *IEEE INFOCOM*, 2018, pp. 414–422.

[14] J.-J. Kuo, S.-H. Chiang, S.-H. Shen, D.-N. Yang, and W.-T. Chen, "Dynamic multicast traffic engineering with efficient rerouting for software-defined networks," in *IEEE INFOCOM*, 2019, pp. 793–801.

[15] N. Geng, T. Lan, V. Aggarwal, Y. Yang, and M. Xu, "A multi-agent reinforcement learning perspective on distributed traffic engineering," in *IEEE ICNP*, 2020, pp. 1–11.

[16] R. Raghu, M. Panju, V. Aggarwal, and V. Sharma, "Scheduling and power control for wireless multicast systems via deep reinforcement learning," *Entropy*, vol. 23, no. 12, p. 1555, 2021.

[17] C. Liu, M. Xu, Y. Yang, and N. Geng, "DRL-OR: Deep reinforcement learning-based online routing for multi-type service requirements," in *IEEE INFOCOM*, 2021, pp. 1–10.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv:1312.5602 [cs]*, 2013.

[19] 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. 64 533–64 539, 2018.

[20] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to Route," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XVI. New York, NY, USA: Association for Computing Machinery, Nov. 2017, pp. 185–191.

[21] N. Vesselinova, R. Steinert, D. F. Perez-Ramirez, and M. Boman, "Learning combinatorial optimization on graphs: A survey with applications to networking," *IEEE Access*, vol. 8, pp. 120 388–120 416, 2020.

[22] K. Almeroth and M. Ammar, "Collecting and modeling the join/leave behavior of multicast group members in the MBone," in *Proc. 5th IEEE Int. Symp. High Perform. Distrib. Comput.*, 1996, pp. 209–216.

[23] J. Tadrous, A. Eryilmaz, and H. E. Gamal, "Proactive resource allocation: Harnessing the diversity and multicast gains," *IEEE Trans. Inf. Theory*, vol. 59, no. 8, pp. 4833–4854, 2013.

[24] B. Cain, S. E. Deering, B. Fenner, I. Kouvelas, and A. Thyagarajan, "Internet Group Management Protocol, Version 3," Internet Engineering Task Force, Request for Comments RFC 3376, 2002.

[25] "Gurobi - the fastest solver - Gurobi," https://www.gurobi.com/.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347 [cs]*, 2017.

[27] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv:1506.02438 [cs]*, 2018.

[28] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal Convolutional Networks for Action Segmentation and Detection," in *2017 CVPR*, 2017, pp. 156–165.

[29] S. Bai, J. Z. Kolter, and V. Koltun, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," Apr. 2018.

[30] "Welcome to Spinning Up in Deep RL! — Spinning Up documentation," https://spinningup.openai.com/en/latest/.

[31] "SNDlib," http://sndlib.zib.de/home.action.

[32] E. Baccour, A. Erbad, M. Guizani, and M. Hamdi, "FacebookVideo-Live18: A live video streaming dataset for streams metadata and online viewers locations," in *ICIoT*, 2020, pp. 476–483.

[33] H.-J. Heo, N. Kim, and B.-D. Lee, "Multicast Tree Generation Technique Using Reinforcement Learning in SDN Environments," in *SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI*, Oct. 2018, pp. 77–81.

[34] R. Boivie, Y. Imai, W. Livens, and D. Ooms, "Explicit Multicast (Xcast) concepts and options," RFC Editor, Tech. Rep. RFC5058, 2007.

[35] G. Cheng, D. Guo, L. Luo, and Y. Qin, "Optimization of multicast source-routing based on Bloom Filter," *IEEE Commun. Lett.*, vol. 22, no. 4, pp. 700–703, 2018.

[36] K. Diab and M. Hefeeda, "Yeti: Stateless and generalized multicast forwarding," in *NSDI 22*, 2022, pp. 1093–1114.

[37] X. Li, F. Tang, L. T. Yang, and L. Chen, "AUTO: Adaptive congestion control based on multi-objective reinforcement learning for the satellite-ground integrated network," in *USENIX ATC 21*, 2021, pp. 611–624.

[38] P. Gong, C. Wang, J. Sheu, and D. Yang, "Distributed DRL-based Resource Allocation for Multicast D2D Communications," in *2021 GLOBECOM*, 2021, pp. 01–06.

[39] S. O. Somuyiwa, A. György, and D. Gündüz, "Multicast-aware proactive caching in wireless networks with deep reinforcement learning," in *IEEE SPAWC*, 2019, pp. 1–5.

[40] X. Zhang, P. Yu, L. Feng, F. Zhou, and W. Li, "A DRL-based resource allocation framework for multimedia multicast in 5G cellular networks," in *2019 BMSB*, Jun. 2019, pp. 1–5.

[41] J. Chae and N. Kim, "Multicast tree generation using meta reinforcement learning in sdn-based smart network platforms," *KSII Trans. Internet Inf. Syst.*, vol. 15, no. 9, pp. 3138–3150, 2021.