Stateless and Proactive Routing for Dynamic Multicast with Deep Reinforcement Learning

Qing Li, *Senior Member, IEEE*, Lie Lu, Dan Zhao, Zeyu Luan, Yuan Yang,
Yong Jiang, *Member, IEEE*, Jingpu Duan, Ruobin Zheng, Shaoteng Liu, Dingding Chen

***Abstract*—Stateful multicast protocols manage multicast group memberships by maintaining state information about active groups and their members. They have seen limited adoption in the modern internet due to lack of scalability, simplicity, and flexibility. Although stateless multicast protocols, like BIER, eliminate extensive state management, they still face complex tree computation and limited scalability for concurrent requests. In this paper, we propose Hawkeye, a stateless multicast mechanism with deep reinforcement learning (DRL) for real-time responses to dynamic multicast requests with near-optimal multicast TE performance. This mechanism is suited for Software-Defined Networking (SDN) environment where the controller has a global view of the network and supports flexible configuration of network resources for traffic engineering. 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 are able to build multicast trees proactively for upcoming requests. We develop a novel source aggregation mechanism to facilitate the convergence of the DRL agent under high volume of multicast requests. Moreover, to improve the practicality and robustness of Hawkeye, we design incremental deployment and single failure handling mechanisms, which take advantages of source aggregation and fit well with multicast routing. Evaluation with real-world topologies and multicast requests demonstrates that Hawkeye responds effectively to dynamic multicast requests. It offers rapid routing decisions, e.g., making routing decisions in under 5ms on a tested topology, and reduces path latency variation by up to 89.5%, with less than a 10% increase in bandwidth consumption compared to the offline theoretical minimum.**

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

I. Introduction

In recent years, multimedia traffic has experienced rapid growth, projected to exceed 70% of all internet traffic by 2023 [1]. The surge in bandwidth-consuming traffic places a substantial strain on network infrastructures. Multicast can potentially ease this burden by efficiently distributing data from a single sender to multiple recipients, thereby conserving bandwidth through the elimination of redundant data transmission. Unfortunately, traditional stateful network-layer multicast [2] 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 [3]. Recently, IETF introduced Bit Index Explicit Replication (BIER) [4], a stateless source routing paradigm. Instead of preserving per-flow states in intermediary nodes, BIER uses explicit bit-based forwarding instructions for more efficient and scalable multicast packet delivery.

Traffic engineering have been proved an effective network utilization optimization measure in many prior works [5]–[8]. Tree Engineering for BIER (BIER-TE) [9] inherits the advantages of BIER and further enables flexible construction of multicast trees. BIER-TE incorporates "packets BitString" to indicate the edges of the multicast tree, allowing fine-grained path control over multicast traffic. Nonetheless, the challenge of efficient multicast tree construction persists, particularly in the context of today's highly dynamic environments. Unlike unicast-based TE that can be formulated as a multicommodity flow problem or similar variants, multicast TE is more complicated. The current multicast standard, IETF PIM-SM [2], generates routes by shortest-path tree (SPT), which always uses the shortest path from the source to each destination and result in sub-optimal bandwidth consumption. Steiner Tree (ST) provides the optimal solution to minimize bandwidth consumption. ST is NP-hard [10] and only works for static trees. [11]–[13] investigated multicast TE with static multicast requests. However, they still suffer from high computation complexity even under fixed group membership assumptions. Moreover, they only make shortsighted decisions for current multicast requests, without considering upcoming demands. Consequently, they struggle to achieve sustained long-term performance. [14], [15] considered multicast TE with dynamic requests. However, these studies rely on responsive heuristics rather than capturing the underlying patterns of how multicast demands change over time, which limits their ability to achieve optimal long-term performance.

To tackle these challenges, Deep Reinforcement Learning (DRL) emerges as a promising approach due to its capacity for real-time decision-making and optimization of long-term rewards. It has found application in addressing various unicast TE problems. Geng et al. [16] adopts DRL to solve complex inter-domain TE problems, achieving less congestion and bet-

This work is supported by the Major Key Project of PCL under grant No. PCL2023A06 and Shenzhen R&D Program under grant NO. KJZD20230923114059020. (*Corresponding author: Dan Zhao*)

Qing Li, Dan Zhao, Zeyu Luan and Jingpu Duan are with Peng Cheng Laboratory, Shenzhen, Guangdong 518066, China (e-mail: liq@pcl.ac.cn; zhaod01@pcl.ac.cn; luanzy@pcl.ac.cn; duanjp@pcl.ac.cn).

Lie Lu is with Alibaba Cloud, Alibaba Group, Hangzhou, Zhejiang 311121, China (e-mail: lul16@foxmail.com).

Yong Jiang is with Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen 518055, China (e-mail: jiangy@sz.tsinghua.edu.cn).

Yuan Yang is with Tsinghua University, Beijing 100084, China (e-mail: yuanyang@tsinghua.edu.cn).

Ruobin Zheng, Shaoteng Liu and Dingding Chen are with the 2012 Lab, Huawei Technologies Co. Ltd., Shenzhen, Guangdong 518129, China (e-mail: zhengruobin@huawei.com; liushaoteng@huawei.com; chendingding5@huawei.com).

ter scalability than traditional methods. Liu et al. [17], [18] exploit DRL's adaptability to dynamic environments for efficient online routing, thus providing near-optimal TE performance under changing network statistics with multiple constraints. Nevertheless, none of the existing research endeavors have investigated the use of DRL within multicast TE scenarios.

In this paper, we propose Hawkeye, a DRL-based multicast mechanism built upon BIER-TE protocol for real-time responses to dynamic multicast requests. This mechanism is particularly suited for network environments where administrators have the capability to manage all network devices and monitor traffic, such as in a Software-Defined Networking (SDN) environment. We first formulate multicast TE as an optimization problem to minimize the long-term total network cost under the path stability constraints. Then, we leverage DRL to learn the multicast traffic pattern from historical requirements and make routing decisions proactively, providing near-optimal multicast TE performance.

Designing 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. Consequently, the solution space of the DRL algorithm increases drastically with dynamic group membership, making it difficult to converge, especially in the presence of numerous requests. Second, the hidden temporal relationship among historical multicast requirements should be effectively mined to facilitate the decision making of the DRL agent.

To address convergence issues, we propose a source aggregation mechanism to reduce the solution space of dynamic multicast. Recognizing that groups stemming from the same source can share portions of the same multicast tree with minimal performance impact, this method consolidates multicast requirements originating from the same root source into a single aggregated requirement capturing the dominant traffic patterns. Then, the DRL agent handles only the aggregated requirements, rather than the massive original requirements, ensuring quicker convergence and response. Based on source aggregation, we also 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 can learn the temporal relationship of consecutive multicast trees. Combining source aggregation with DRL, Hawkeye outputs link weights, based on which proactive routing decisions are made. It implicitly considers the prospect of future major traffic, ensuring real-time response and long-term TE performance.

To further enhance Hawkeye's practicality, we design incremental deployment and failure handling. Incremental deployment allows Hawkeye to function in hybrid networks with nodes lacking BIER-TE support. Specifically, we propose an overlay network construction method for multicast routing that endeavours to save bandwidth consumption. The failure handling mechanism leverages BIER-TE and source aggregation to rapidly identify single failures at source routers, only using feedback from destinations, and enabling local recovery through pre-installed backup rules.

We evaluate Hawkeye with simulations on real-world topologies and multicast requests. The results show source aggregation can effectively accelerate both convergence speed and decision making speed. In our tests, Hawkeye is able to make routing decisions within 5ms on a tested topology. The DRL-based TE solution outperforms prior multicast methods, reducing path latency variation by up to 89.5% with only less than 12% additional bandwidth consumption compared with the optimal solution. The efficiency of the incremental deployment and failure handling mechanisms is also validated.

Our contributions can be summarized as follows:

- A multicast source routing framework based on BIER-TE with source aggregation. It improves the efficiency of the DRL algorithm and reduces the storage overhead at the source routers.
- A DRL-based multicast routing algorithm. It considers the static and dynamic multicast performance jointly and provides responsive and far-sighted decisions.
- An overlay network construction method for incremental deployment. It ensures the tree diversity in abstracted overlay topologies, and thus prevents severe performance degradation.
- A source-driven single failure detection mechanism. It simplifies the failure handling process and enables fine-grained recovery methods to be deployed locally.

While previously presented in part in [19], this extended version integrates significant enhancements, including incremental deployment, failure handling mechanism, accompanying experimental evaluations, and theoretical foundation for DRL reward design.

The remainder of this paper is organized as follows. Section II introduces the background and motivation. Section III presents the overall design of Hawkeye with routing problem formulation and incremental deployment consideration. The proposed source aggregation and DRL framework are described in depth in Sections IV and V. The failure handling framework is explained in Section VI. Section VII shows the evaluation results. Finally, we review the related work in Section VIII and conclude this paper in Section IX.

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).

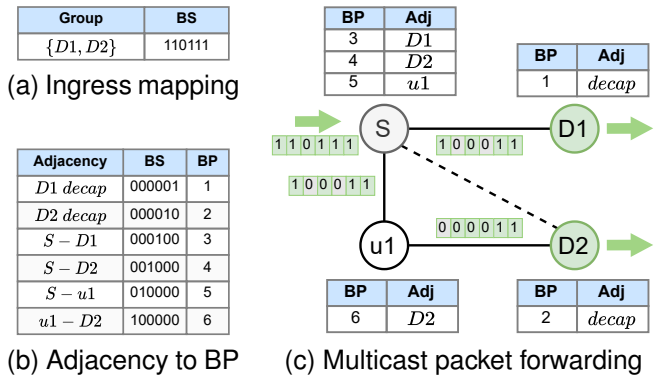


Fig. 1. An example of BIER-TE. (a) Ingress mapping of the BFIR (node S). (b) Mapping from adjacencies to BPs. (c) Forwarding process.

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 and encapsulates a BIER-TE header into 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: DRL combines neural networks with reinforcement learning to enable agents to learn optimal behaviors in dynamic environments through trial and error, showing tremendous potential for solving sequential decision-making problems in networking scenarios with high-dimensional state spaces [16], [18], [20], [21]. For example, [20] proposes a DRL-based approach to make online virtual network function deployment decisions according to network resources and network function characteristics states. In the context of traffic engineering tasks, DRL presents distinct advantages over traditional TE approaches under dynamic traffic demands, making it particularly well-suited for our problem. DRL can respond to network dynamics quickly, providing fast and efficient routing decisions [22]. This rapid response capability is particularly beneficial for latency-sensitive applications and services. Moreover, as an experience-driven, model-free approach, DRL learns hidden patterns and causal relationships rather than relying on explicit mathematical models. This enables it to efficiently handle complex problems involving multiple interactive factors, such as topology structures and traffic demands in TE tasks while optimizing for accumulated rewards over the long term. For example, DRL has been used to decide link weights [23], traffic splitting ratios [16], and the next hops [18]. As discussed in [24], 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.

B. Motivation

1) Why BIER-TE: Compared with stateful multicast protocols, BIER-TE not only fits large-scale multicast better but also provides flexible control of multicast trees for TE. It 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, and consume less network resources than the traditional stateful protocols. Moreover, 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 due to three main reasons.

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 [25] and heuristics, are mainly designed for myopic offline optimization. They may provide local optima that is 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 locations of users in a multicast group may follow some probabilistic models, such as the Poisson distribution [26], [27]. In dynamic multicast, however, the traffic pattern changes over time, making 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. Network states and performance can be easily measured. For example, multicast requests and join/leave events are naturally collected by protocols (e.g., IGMP [28]), and multicast trees generated by the agent can be evaluated using network topology information. 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

In this section, first, we formulate the multicast routing with dynamic membership as an optimization problem. Then, we present the overall workflow of Hawkeye, from DRL-based multicast tree configuration to packet forwarding.

A. Problem Formulation

We formulate a multicast tree construction optimization problem, aiming to minimize long-term total bandwidth consumption under path length variation constraints in response to dynamic multicast requests. 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 the static and dynamic costs of multicast trees, respectively. Specifically, ω_e^{st} denotes the cost per unit bandwidth and the static cost measures the 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:

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

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

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

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

Objective function Eq.(1). The objective is to minimize the total static cost, i.e., the costs of trees of all groups at all time slots, where 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 toward 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}.

The above optimization problem can be reduced to a set of Steiner tree problems by setting the weight ω_e^{dy} in the dynamic cost constraint (4) to 0. Since the Steiner tree problem is proven to be NP-hard, this problem is also NP-hard [10]. The optimal solution can be obtained offline using ILP solvers (e.g., Gurobi [29]), 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.

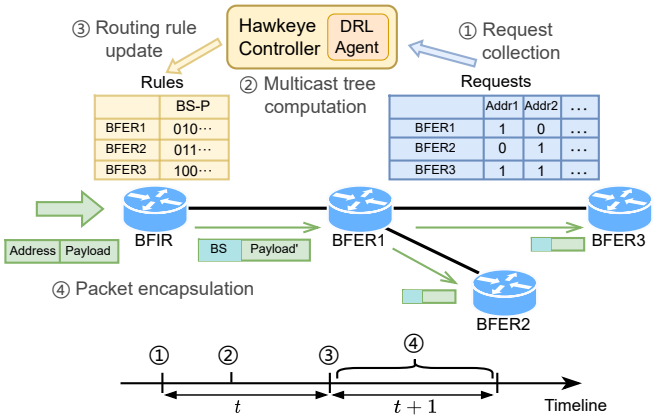


Fig. 2. System overview.

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 of 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 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.

C. Incremental Deployment

Since there may have devices that cannot support BIER-TE (e.g., an IP/BIER-TE hybrid network where most routers can only perform packet forwarding and the multicast requests are mainly processed by several BIER-TE routers), incremental deployment of Hawkeye is often desired for robust iterative performance improvement. Different from incremental deployment in unicast [30]–[32], where the main objective is to reserve as many paths as possible in the overlay network, path reuse efficiency becomes the top priority in multicast, making the incremental deployment non-trivial. Multicast relies on packet replication for bandwidth saving, so the paths to all destinations should overlap as much as possible to facilitate path reuse and better bandwidth utilization. The construction of the overlay network should strive to adhere to this principle to avoid bandwidth waste.

In this paper, we propose an overlay network construction method for incremental deployment of Hawkeye. Assume we have already selected k upgraded nodes. We name the upgraded nodes as *core nodes*, and other nodes as *marginal nodes*. Because the marginal nodes only support basic packet forwarding but not BIER-TE, we can only use the shortest paths to connect overlay nodes. Thus, each link in the overlay network is actually a shortest path in the underlay network, and the link attribute is calculated based on that of the original path. For example, the delay of the overlay link is the sum of delays of all links in the corresponding path.

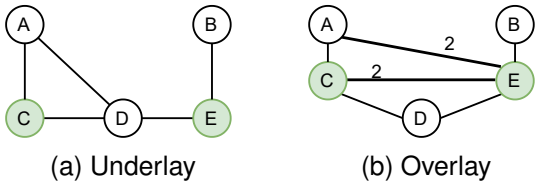


Fig. 3. An example of overlay network construction.

To avoid bandwidth waste caused by path duplication, we define *good path*, which is a path without core nodes, except the first and the last nodes. In a good path, only the starting point and the terminal point can be core nodes. We also define a tree diversity factor b, which facilitates path reusing for further bandwidth saving. With good path and diversity factor b, the construction method can be described as follows:

(1) Connecting core nodes: For every core node, compute the shortest paths to other core nodes, and connect them with the path if it is a good path.
(2) Connecting marginal nodes: For every marginal node, compute the shortest paths to all core nodes, reserve good paths among them, and choose the b shortest of them as overlay links. If the number of usable paths is less than b, use all of them.

Intuitively, instead of connecting marginal node to only one core node, we connect marginal node to several core nodes to furnish more feasible multicast trees, thus increasing the chance of finding a tree with lower bandwidth consumption.

Fig. 3 shows an example of our method. The underlay network consists of two core nodes, C and E, and three marginal nodes, A, D, and B. The cost of each link in the underlay network is assumed to be 1 unit for simplicity. To construct the overlay network, we first connect core nodes C and E with path $\{C$–D–$E\}$, so an overlay link C–E with cost 2 is obtained, as indicated in Fig.3(b). Next, we need to connect each marginal node to core nodes C and E. Here, we set $b=2$, i.e., a marginal node can be linked to at most 2 core nodes. For node A, the paths to nodes C and E, i.e., $\{A$–$C\}$ and $\{A$–D–$E\}$, are both good paths, so we get two overlay links A–C and A–E, as shown in Fig. 3(b). Similarly, node D connects to core nodes C and E with links D–C and D–E, respectively, which are qualified as overlay links as well. For node B, the path to C cannot be used as an overlay link since it passes through core node E, while the link B–E is a good path and therefore counted as an overlay link. Finally, we get the overlay network as Fig. 3(b). This example also shows the feature of overlay network routing, that is, the routing costs in the overlay network may be higher than those in the underlay network. For example, the minimal cost of the path including A, D, and E is 3 units ($\{A$–E–$D\}$) instead of 2 units ($\{A$–

D–E}) in the underlay network.

Next, we briefly discuss the core node selection and forwarding mechanism of the incremental deployment. For core node selection, we use *betweenness centrality* as metric, which reflects the number of shortest paths that pass through a node, and is frequently used in hybrid communication network [33], [34]. The nodes with high betweenness centrality are taken in priority to be upgraded. For overlay network forwarding, BIER-TE provides a mechanism of *forward_routed* adjacency, which uses encapsulation such as MPLS or IP tunnel to forward packets between overlay neighbors. Regardless of the encapsulation technique the underlay devices use, in overlay networks, the *forward_routed* adjacency can be regarded as a single link represented by a BP. Then, the forwarding process is the same as the conventional BIER-TE network, except that the *link* connecting two core nodes is actually a shortest path between them in the underlay network.

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. 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., $\vec{f_b} = [f_{b,v}]_{v \in V}$.

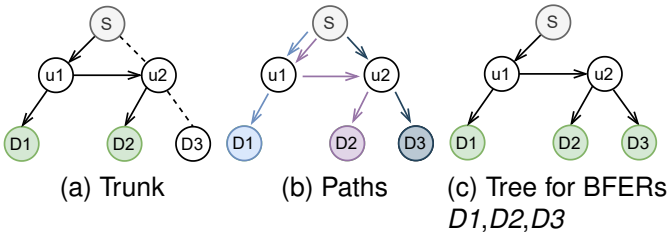


Fig. 4. An example of multicast tree building based on source aggregation. (a) The trunk. (b) Paths to each destination. (c) The tree spanning *D1*, *D2* and *D3*.

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 by our DRL agent more easily. We define aggregation ratio λ, 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 with 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 the 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.

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

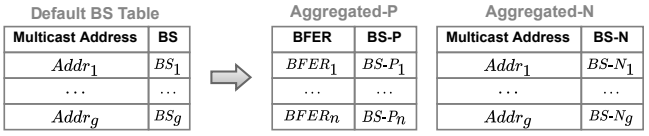


Fig. 5. The BFIR table before and after aggregation.

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. 4 to illustrate the process of constructing forwarding paths given the trunk. In Fig. 4(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. 4(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. 4(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. 5). 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 $\mathcal{O}(|\mathbf{g}_b| \cdot |E|)$.

The above routing table arrangement solution has significantly reduced routing table storage overhead. To further mitigate the table size bottleneck, we suggest two possible strategies to be considered when necessary. First, in BIER-TE, only the BFIRs of multicast groups need to store the multicast routing tables. Therefore, when selecting BFIRs, switches with more abundant table storage, i.e., ternary content addressable memory (TCAM), should be preferred. Second, TCAM-based rule caching systems [35], [36] can be adopted, which use TCAM for high-speed caching of heavy-hitting rules and RAM for the complete ruleset. These systems combine the fast lookup characteristics of TCAM and the large storage capacity (in the scale of gigabytes) of RAM.

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,

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

Different from the original dynamic cost defined in (4), which considers the destinations in the intersection of D_g^t and D_g^{t+1}, Eq. (7) considers the destinations in the union of D_g^t and D_g^{t+1}. Such a design establishes the relation of the multicast trees before and after source aggregation. As will be proved in Theorem 1, Eq. (7) actually defines the upper bound of original dynamic cost, which can be used to ensure the feasibility of DRL solutions. Specifically, if this upper bound exceeds the predefined threshold, the original dynamic cost constraint is at the risk of violation. In this case, the training process should be cut off to discourage the agent to make such decisions.

Theorem 1: Let r and r_A be the original and aggregated requirements, respectively. The destination sets of r for two consecutive time slots t and $t+1$ are denoted as D^t and D^{t+1}. Similarly, those of r_A are denoted as D_A^t and D_A^{t+1}. Let Δdc_d denote the dynamic cost variation of destination d for simplicity. Then, we have

$$\sum_{d \in D^t \cap D^{t+1}} \Delta\text{dc}_d \le \sum_{d \in D_A^t \cup D_A^{t+1}} \Delta\text{dc}_d. \tag{8}$$

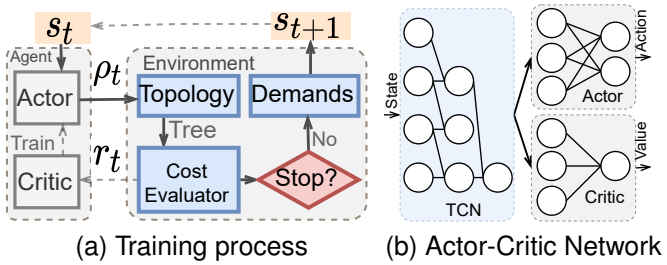


Fig. 6. DRL design of Hawkeye.

Proof: Assume there are n destinations in $D^t \cap D^{t+1}$, $\{d_1, d_2, \ldots, d_n\}$. Without loss of generality, we can divide $D^t \cap D^{t+1}$, $\{d_1, d_2, \ldots, d_n\}$, into four subsets, D_1, D_2, D_3 and D_4 as follows.

D_1 contains destinations belong to both D_A^t and D_A^{t+1}, i.e., $d \in D_A^t$ and $d \in D_A^{t+1}$, $\forall d \in D_1$. All the paths of destinations in D_1 are extracted from the aggregated multicast tree, so we have $\sum_{d \in D_1} \Delta\text{dc}_d \le \sum_{d \in D_A^t \cap D_A^{t+1}} \Delta\text{dc}_d$.

D_2 contains destinations that appear in D_A^{t+1} but not in D_A^t, that is, $d \notin D_A^t$ and $d \in D_A^{t+1}$, $\forall d \in D_2$. From t to $t+1$, the destinations in D_2 switch from the shortest paths to those generated from the RL agent. The dynamic cost of D_2 is controlled by $\sum_{d \in D_2} \Delta\text{dc}_d \le \sum_{d \in D_A^{t+1} \setminus D_A^t} \Delta\text{dc}_d$.

Similarly, D_3 contains destinations that appear in D_A^t but not in D_A^{t+1}, that is, $d \in D_A^t$ and $d \notin D_A^{t+1}$, $\forall d \in D_3$. We have $\sum_{d \in D_3} \Delta\text{dc}_d \le \sum_{d \in D_A^t \setminus D_A^{t+1}} \Delta\text{dc}_d$.

D_4 contains destinations belongs to neither D_A^t nor D_A^{t+1}, that is, $d \notin D_A^t$ and $d \notin D_A^{t+1}$, $\forall d \in D_4$. Destinations in D_4 use the shortest paths in both time slots t and $t+1$, so $\sum_{d \in D_4} \Delta\text{dc}_d = 0$.

Let $X = D^t \cap D^{t+1}$, $Y = D_A^t \cup D_A^{t+1}$, $Z = D_1 \cup D_2 \cup D_3 \cup D_4$, and $W = (D_A^t \cap D_A^{t+1}) \cup (D_A^{t+1} \setminus D_A^t) \cup (D_A^t \setminus D_A^{t+1})$. Finally, we can sum up all the dynamic costs as

$$\sum_{d \in X} \Delta\text{dc}_d = \sum_{d \in Z} \Delta\text{dc}_d \le \sum_{d \in W} \Delta\text{dc}_d = \sum_{d \in Y} \Delta\text{dc}_d. \tag{9}$$

■

B. Training

We leverage Proximal Policy Optimization (PPO) [37], which is a policy-based algorithm designed for continuous 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. 6a shows the training process. 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

$$L = \min\left(\omega_\theta A(s,\rho),\ \text{clip}\left(\omega_\theta, 1-\epsilon, 1+\epsilon\right) A(s,\rho)\right), \quad (10)$$

$$\omega_\theta = \frac{\pi_\theta(\rho \mid s)}{\pi_{\theta_{old}}(\rho \mid s)}. \quad (11)$$

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 [38].

C. Neural Network Architecture

We use a TCN [39] 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 [40]. 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

$$(S \star f)(t) = \sum_{i=0}^{K-1} f(i) S(t - d \times i), \quad (12)$$

where d is the dilation factor controlling the convolution interval. As illustrated in Fig. 6b, 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. Failure Handling

Failure handling is a crucial component of network operations, ensuring robust performance when failures occur, such as node disruptions caused by power outages or hardware failures. It can also step in when a node or a link is strained beyond its bandwidth capacity. In this paper, we discuss the failure detection and recovery mechanisms in Hawkeye. Some failure recovery methods for multicast [41], [42] use redundant trees for reliable multicast, where packets are transmitted along several disjoint paths towards each destination. Such solutions induce excessive bandwidth consumption, i.e., at least two times of the original multicast tree. Another kind of solutions use backup paths for failure recovery [43], [44]. However, they either rely on other protocols to report failure or borrow unicast fast reroute mechanism to handle failure, causing high configuration complexity.

Designing an efficient failure recovery mechanism in the context of BIER-TE faces several challenges. First, because multicast trees are computed by the controller, generating a new path to bypass the found failure requires communication between the source router and the controller, resulting in high response latency. Second, even if we pre-install backup path rules to protect against failures, under the demand of TE, simple methods like disjoint trees are difficult to obtain. For example in Fig. 7, assume node S is the source and C is the destination. If we use the shortest path S–B–C as the default path from S to C, the backup disjoint path could be S–A–C, as shown in Fig. 7(b). However, if we use S–B–A–C as the default path for TE purpose, there is no disjoint path for failure protection, as shown in Fig. 7(c). Third, due to the routing information in BSs, local recovery with low-level unicast-based mechanism may cause redundant transmissions. Assume we use unicast reroute for failure recovery in Fig. 8. When the link S–B fails, the backup path from S to B is S–A–C–B. However, after node B receives the tunneling packet from C, it will still transmit a packet copy to node A along B–C–A according to the BS encapsulated in the header, resulting in redundant transmission at B–C and C–A.

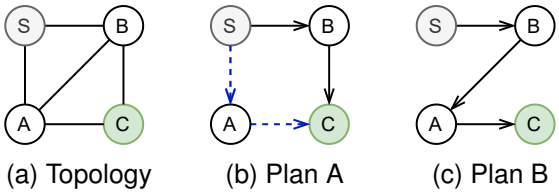


Fig. 7. An example of no backup path with TE.

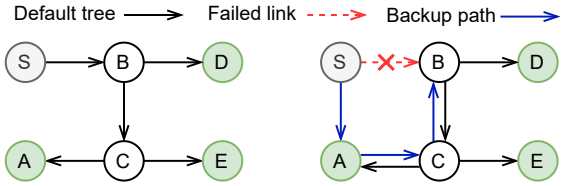


Fig. 8. An example of redundant transmission with unicast reroute.

In Hawkeye, the source routers maintain the multicast tree information of all multicast groups, and can collect feedbacks from failed destinations (e.g., NACK). Inspired by this, we propose a source-driven failure handling mechanism, which detects and recovers single failure at the source router using only the feedback from BFERs and backup rules at the BFIR. The proposed mechanism detects single failure using only feedback of destinations, thus releases dependence on other detection mechanisms, and conserves the simplicity and reliability of the system. Besides, it recovers from failure solely by the BFIR and BFRs, without involving the controller, and therefore can ensure low latency.

A. Failure Detection

1) Single Link Failure Detection: Algorithm 1 shows the proposed single link failure detection. To locate the failed link, we monitor the feedback of some requirements at the source router. We choose the monitored requirements R randomly. Other selection methods can also be used. We initialize the accused link set BP_F to include all links in the topology (Line

Algorithm 1 Source-driven Failure Detection

Input:
Monitored requirements R, Network link set E.
Output:
Accused links BP_F.

1: Initialize accused link set $BP_F = E$
2: **for** r in R **do**
3: Get destinations of r as D_r
4: Get failed destinations D_f of r
5: Get normal destinations $D_n = D_r \setminus D_f$
6: Initialize candidate failed link set $BP_f = E$
7: **for** d in D_f **do**
8: Get link set of the path to d as BSP_d
9: $BP_f = BP_f \cap BSP_d$
10: **end for**
11: Initialize normal link set $BP_n = \emptyset$
12: **for** d in D_n **do**
13: Get link set of the path to d as BSP_d
14: $BP_n = BP_n \cup BSP_d$
15: **end for**
16: $BP_F = BP_F \cap (BP_f \setminus BP_n)$
17: **end for**
18: return BP_F

1). Then, we examine the feedback of each requirement in the monitored set R (Line 2-15). For requirement r, we get its destinations as D_r. According to the feedback of r, we identify its failed destinations D_f and the normal ones D_n (Line 3–5). Since the paths to destinations in D_f are disconnected, the failed link must be one of their common links. Therefore, we obtain a candidate failed link set BP_f, which must contain the failed link, by calculating the intersection of their link sets (Line 6–10). For destinations in D_n, all paths are normally connected, indicating that the failed link is not in any of these paths. Therefore, we obtain a set of normal links BP_n by calculating the union of the links in these paths (Line 11–15). Finally, we update the accused link set BP_F by the difference between BP_f and BP_n, i.e., the set of all accused links according to r's feedback with evidenced normal links excluded (Line 16). By repeating the above process for all monitored requirements, we get the final accused link set BP_F. Given this set, we can process the accused links for failure recovery. Under single link failure assumption, the smaller the set BP_F is, the more precise the result is. If there is only one link in BP_F, it must be the failed one.

Fig. 9 shows an example of the detection algorithm. The topology in Fig. 9(a) consists of nodes $A \sim G$ and the source node S. There are 2 requirements in R, r_1 with members D and F, and r_2 with members E and G, whose multicast trees are shown in Fig. 9(b) and Fig. 9(c), respectively. Assume a link failure affects both requirements. For r_1, based on feedback, S finds that node F is a failed destination and D is a normal destination. We have $BP_f = \{S-A, A-B, B-F\}$ and $BP_n = \{S-A, A-D\}$ and the BP_F is set to $\{A-B, B-F\}$. Similarly, for r_2, links $\{S-A, A-B, B-E\}$ are suspected because they are on the path of the disconnected node E. By taking the intersection of the results of r_1 and

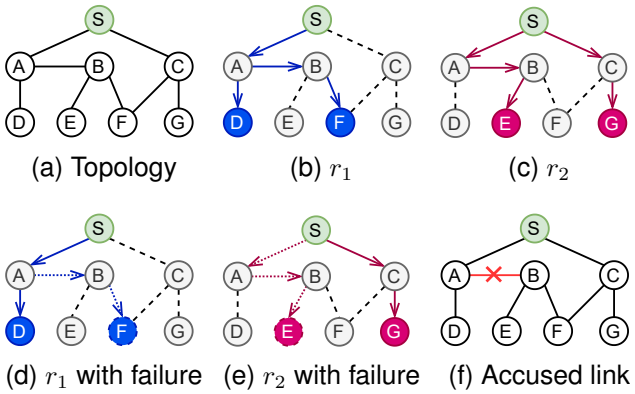


Fig. 9. An example of failure detection algorithm.

r_2, the final accused link set BP_F is obtained as $\{A-B\}$, indicating the failed link is $A-B$.

Since links are represented by bits in BSs, we can simplify Algorithm 1 by replacing the set operations by bit operations. For example, *intersection* and *union* operations can be replaced by *bitwise AND* and *bitwise OR*, respectively. The time complexity of Algorithm 1 is $\mathcal{O}(|R||N|)$, where $|N|$ is the number of nodes in the topology. In Hawkeye, the source router already maintains paths to all destinations in BSs, and has enough information to locate single link failure using Algorithm 1 without the assistance of controller. Moreover, this algorithm introduce no storage overhead at the source. The only costs arises from collecting feedback for monitored multicast requirements and running the detection algorithm.

The performance of Algorithm 1 depends on the topology, the selected requirements R, and their member distribution. We can adjust the performance by changing the number of requirements and the selection method of R. As will be shown in Section VII-E, the number of accused links in BP_F decreases dramatically as the number of monitored requirements increases, and only a small number of monitored requirements is enough to accurately locate the failed link even under large topology and random selection.

2) Single Node Failure: Single node failure detection can be achieved by a simple modification to Algorithm 1: we use the possible failure set BP_F in Algorithm 1 to record the successor nodes of the accused links, instead of the links themselves. The reason why the above modification realizes single node failure detection is as follows. A multicast tree can be formulated as a directed tree, where each node's in-degree equals 1, except the source node. Thus, the representation of a single node failure can be considered as the incoming link failure. In turn, under single node failure, the failed node must appear in the successor nodes of the accused links using the above algorithm. Because the source is unaware of which two nodes a link connects by default, we need to store this information at the source router to map links to their successor nodes, requiring a storage overhead of $\mathcal{O}(|E|)$.

B. Failure Recovery

Classic multicast failure recovery mechanisms, including tree protection, path protection, and link protection, offer

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		

protection granularities spanning from low to high. Tree protection calculates backup trees in advance against possible failures. The backup tree can be link- or node-disjoint to improve reliability. Upon failure, the source node can transmit multicast packets to both the default and backup trees or switch to the backup tree. This method suffers from the highest bandwidth consumption, especially with node disjoint backup tree. Sometimes there may have no usable backup tree due to the topology. Path protection backups paths for failure recovery. The backup path can also be link- or node-disjoint. In multicast, even a single link failure may cause many disconnected destinations. Thus, the bandwidth consumption performance of path protection can be as bad as tree protection under the worst case. Link protection bypasses failures by slightly adjusting the default paths, for example, finding another path to the next-hop for link failure protection, and to the next-next-hop for node failure protection. Theoretically, this method yields least modification to the multicast tree. However, it needs the precise failure location and plenty of backup rules for all possible failures.

As will be elaborated in the evaluation in Section VII-F, link protection methods work the best for Hawkeye among the three. First, it works well in different topologies, as new paths only bypassing the failed links are relatively easy to find. In fact, tree protection could barely work in the tested topologies, with a success rate of less than 2%, and path protection also fails more frequently than link protection. Second, link protection achieves the best bandwidth performance among the three. Especially, link protection incurs less bandwidth consumption than path protection in large topologies. Third, link protection is relatively easy to implement in Hawkeye by distributing the backup rules at routers locally. The precursor node of the failed link can rewrite the BS of packets with BIER-TE headers to switch them to the backup path. In contrary, we must maintain at least a backup BS for every possible destination at the source router for path protection, even with source aggregation. As a result, the storage overheads of path protection can be 4 times or even 10 times higher than those of link protection in tested topologies.

VII. EVALUATION

We implement the PPO algorithm based on the SpinningUp [45] 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 VII-C. ILP models are solved using the Gurobi Optimizer [29] with

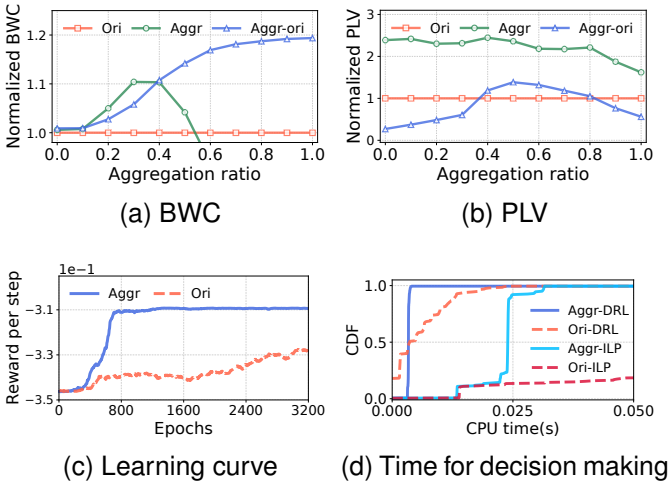


Fig. 10. Performance of source aggregation.

a 16-core 2.3 GHz CPU. We use four real-world topologies from SNDlib [46], namely Abilene, Geant, Germany50 and GtsCe. Abilene is a small topology with 11 nodes and 14 bidirectional links. Geant has 23 nodes and 37 bidirectional links, Germany50 has 50 nodes and 88 bidirectional links, and GtsCe has 149 nodes and 193 bidirectional links.

The multicast requests are generated based on real-world data. For Abilene, we use a dataset derived from Facebook [47]. 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, Germany50 and GtsCe, we use a typical multicast traffic model to simulate the dynamic membership [26], where the requirements changes with locations and time, proportional to the total traffic obtained from SNDlib.

In this section, we first evaluate the source aggregation mechanism and DRL-based routing algorithm of Hawkeye, and the effect of its hyperparameters. Then, we test the incremental deployment and failure handling mechanisms.

A. 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. 10, *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. 10a 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. 10b, 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. 10c, 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. 10d. *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 5ms in Abilene.

B. Performance of Multicast Trees

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].

(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 [48]. 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.

We compare the cumulated BWC and PLV of each method in four topologies, shown in Fig. 11, 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 all four topologies.

[1]Some multi-tree static and dynamic multicast routing algorithms, as discussed in Section VIII, are not included in this study because the path length variation is not considered in those approaches and OPT provides the optimal solution for the considered problem.

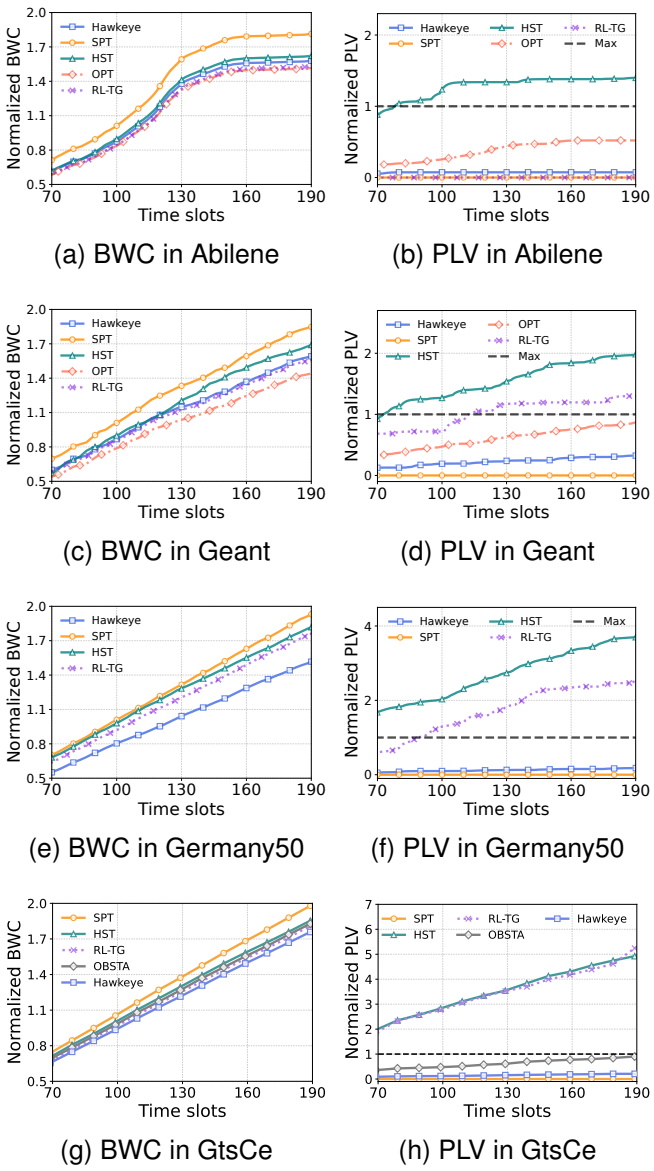


Fig. 11. Performance of multicast trees in four topologies.

In Abilene, a small network, dynamic membership has a limited impact, resulting in similar BWC for methods other than SPT, as shown in Fig. 11(a). However, the PLV of OPT is only 37% of HST, indicating more stable routing can be achieved with a neglectable BWC increase. Hawkeye's BWC and PLV performances closely match OPT, with a difference of less than 5%, showcasing its effectiveness in small-scale topologies. The BWC in Geant are similar, but the PLV of RL-TG exceeds the limit while that of Hawkeye remains low. For SPT, the BWC grows with the topology scales, which indicates that SPT is not efficient in large network.

In topologies with more nodes, e.g., Germany50 and GtsCe, the ILP model of OPT cannot be solved in a reasonable time, so its results are omitted. Hawkeye achieves the lowest BWC while maintaining a low PLV in both Germany50 and GtsCe. Due to the large search space, RL-TG fails to find the optimal trees, and consequently results in BWCs and

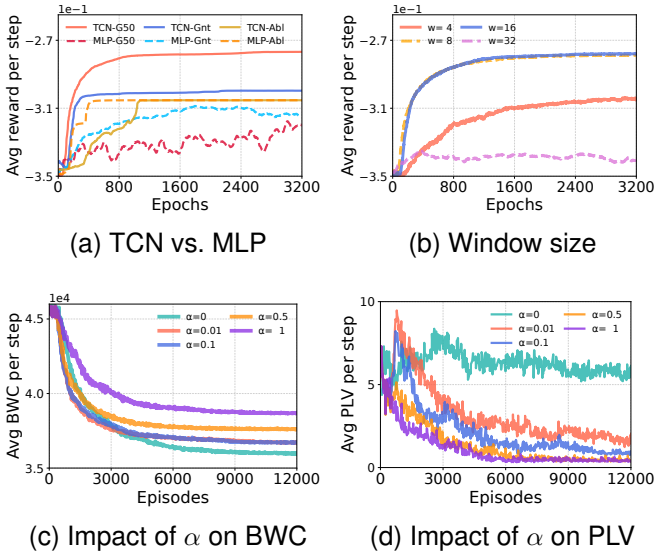


Fig. 12. Impact of Hawkeye hyperparameters.

PLVs that degrades dramatically in these two topologies. SPT is once again the most bandwidth-inefficient one among all methods, although it maintains a relatively low PLV. HST performs incompetitively in both Germany50 and GtsCe for BWC and PLV, as it cannot effectively capture the temporal patterns of multicast requests and fails to account for changes in path lengths during decision-making. We also compare the performance of Hawkeye with OBSTA [14] in GtsCe topology. OBSTA is designed to reduce tree costs while maintaining route stability. It computes nodes' stability indices and prioritizes routes to more stable nodes when establishing multicast trees. As shown in Fig. 11(g) and Fig. 11(h), Hawkeye demonstrates notable advantages over OBSTA in both BWC and PLV. This is because Hawkeye employs a more rigorous and comprehensive method to learn and extract patterns from multicast request dynamics than OBSTA, making it more effective in optimizing long-term performance.

These experiments demonstrate that Hawkeye is able to adjust multicast trees to satisfy stability requirements while keeping low bandwidth cost in topologies of different sizes.

Hawkeye is designed to avoid the complexity explosion problems seen in traditional approaches by incorporating BIER-TE, source aggregation and DRL. In practical scenarios, for very large topologies, if necessary, we can also divide the network into smaller, more manageable subdomains. This division aligns with the fact that BIER-TE support multiple subdomains. For scenarios involving a large number of multicast groups, source aggregation inherently enhances scalability by combining multiple groups into aggregated traffic. If necessary, multicast groups rooted from the same source can be further divided into smaller batches, with source aggregation applied to each batch.

C. Hyperparameters

1) TCN vs. MLP: Fig. 12(a) shows the learning curves with TCN/MLP of three topologies abbreviated as Abl, Gnt, and G50. 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 slower 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 performs stably. To summarize, MLP is too simple to model the complicated temporal relationship in large networks.

2) State Window Size w: 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. 12(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.

3) PLV Weight α: The PLV weight α in (6) controls the relative importance of PLV versus BWC. Fig. 12 shows the average BWC and PLV during training in Germany50. When $\alpha = 0$, the agent only focuses on BWC, and the BWC slowly converges to the lowest at the expense of a high PLV. As α increases, the agent pays more attention to PLV constraints, PLV decreases more rapidly, but the performance of BWC degrades. This parameter serves as a knob for exploration-exploitation trade-off. A smaller α encourages more exploration, potentially leading to a lower BWC.

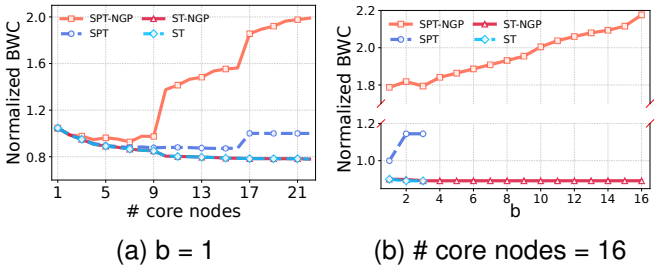


Fig. 13. Bandwidth consumption of different multicast tree computation and overlay network construction methods w/wo good path in Geant.

D. Incremental Deployment

First, we evaluate the BWCs obtained by different multicast tree computation and overlay network construction methods, with or without *good path*, as shown in Fig. 13. SPT and ST represent shortest path trees and Steiner trees, respectively. The NGP suffix indicates the "good path" concept is not used when constructing the overlay topology, i.e., we use the path as long as it exists, regardless of whether or not it is a good path. The results are normalized by BWC of the shortest path trees in the underlay network. In Fig. 13(a), we set $b = 1$, i.e., every marginal node is only connected to the nearest core node. As the number of core nodes increases, the BWCs of all methods decrease in the beginning because of richer topology information. However, the BWCs of SPT-based methods increase when the number of core nodes becomes larger. This is because this heuristic routing method

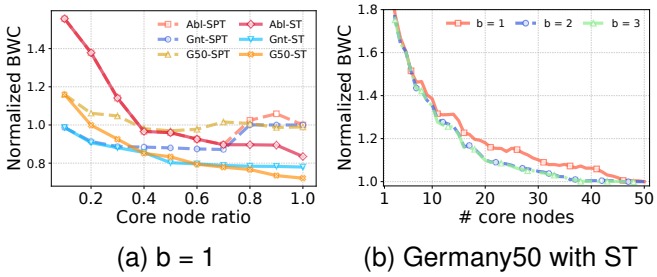


Fig. 14. Routing performance with different overlay construction settings.

TABLE II
NORMALIZED BWC OF DIFFERENT RECOVERY METHODS

	Abilene	Geant	Germany50
SPT	1.000	1.000	1.000
SPT-LP	1.031 (8%)	0.998	1.003
SPT-PP	1.026 (11%)	1.003	0.995
MST	0.909	0.807	1.050
MST-LP	0.955 (8%)	0.813	1.050
MST-PP	0.951 (37%)	0.877 (28%)	1.040
RL	0.956	0.917	0.884
RL-LP	0.991 (8%)	0.909	0.890
RL-PP	0.989 (27%)	0.918	0.894

focus on optimizing the path length to each destination but not the global bandwidth consumption. With less core nodes, there are less usable paths between nodes, and the shortest paths are forced to overlap with each other, resulting in less packet duplication and hence less BWC. In contrast, ST always searches for the global optimal solution in terms of BWC in the network, so its BWC keeps improving as the number of core nodes increases. Besides, the BWCs of ST and ST-NGP are similar, which means the use of good path has little effects on the bandwidth consumption for fully optimized methods.

Fig. 13(b) shows BWC under different b, with 16 core nodes. The tests are terminated if the results keep unchanged with increasing b. As can be seen, using good path makes the BWC converge faster for both SPT and ST. Conversely, without good path, there are always new solutions as b increases, which may provide similar or even worse performance. In addition, the BWCs of SPT-based methods show an increasing trend, which corroborates that more redundant topology information tends to degrade the performance of SPT.

Second, we evaluate the impacts of the core node ratio, i.e., the ratio of core nodes among all nodes in the topology, and the diversity factor b when constructing overlay networks. As illustrated in Fig. 14(a), for all three tested topologies, BWCs of ST decrease as the ratio increases while those of SPT fluctuate descend first and then ascend. Overall, when $b = 1$, a larger ratio leads to better routing performance.

Fig. 14(b) shows the impact of tree diversity factor b on BWC of ST in Germany50. As can be seen, b affects how fast BWC decreases with increasing number of core nodes. Particularly, the BWC descends faster when $b = 2$ than $b = 1$. When $b = 1$, more than 45 core nodes (90%) are needed to achieve the optimal performance while when $b = 2$, only around 36 are needed (72%). However, the difference between $b = 2$ and $b = 3$ is negligible, which suggests the core node ratio no longer bring more benefits to the BWC after 2.

E. Failure Detection

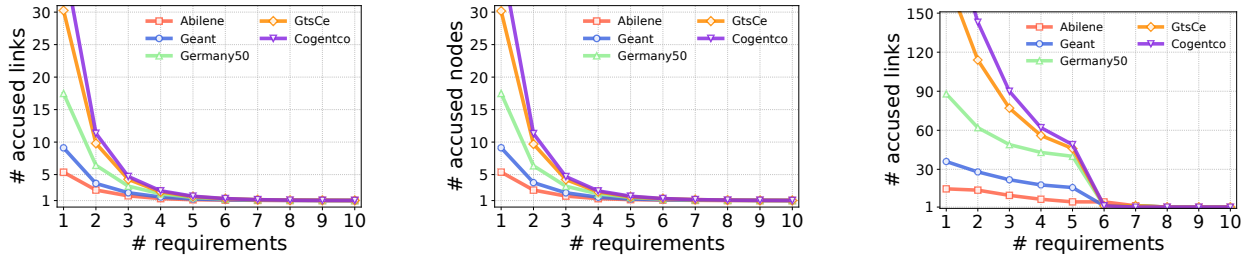
We evaluate the failure detection accuracy of Algorithm 1 in 5 topologies, including extra 2 large ones, GtsCe (149 nodes, 193 links) and Cogentco (197 nodes, 243 links). We simulate uniform random failure in each topology for 1000 times, with randomly chosen group size and destination nodes. Fig. 15(a) and Fig. 15(b) show the average numbers of accused links and nodes. Regardless of the topology, the average numbers of both accused links and nodes decrease rapidly as the number of monitored requirements increases, and become 1 when the number of monitored requirements exceeds 6, indicating the failure is located precisely. Fig.15(c) shows the 99th percentile number of accused link, i.e., extreme case performance for link failure. The 99th percentile also decreases significantly as the number of monitored requirements increases, and converges to 1 when the number of monitored requirements exceeds 6, regardless of the scale of topology. The result for node failure is similar, and is omitted here due to limited space.

Fig. 16 shows the minimal number of requirements needed to locate all single link failures accurately, i.e., the maximal number of accused links in all 1000 times tests equals 1, under different *Destination Number Ratio* (DNR). DNR is the ratio of the destination number of the minimal group in R to the number of nodes in the topology, e.g., DNR $= 0.5$ means that the minimal group in R has $\lfloor 0.5N \rfloor$ members, where N is the number of nodes in the topology. As DNR increases, the monitored groups become larger, and there are more destinations in a group to provide information for failure detection, so less requirements are needed. Note that even if the DNR is 0.1, the required size of R is less than 20. As real networks usually have hundreds or more requirements, we could say our algorithm is efficient for single failure location even with random requirement selection; a well-designed selection method could further enhance its performance.

F. Failure Recovery

Table II presents the average normalized BWCs of different failure recovery methods under single link failure. We adopt 3 basic multicast tree computation methods, shortest path tree (SPT), minimum spanning tree (MST), and reinforcement learning (RL), and 2 recovery methods, link protection (LP) and path protection (PP). While the cost of the backup path is usually larger than that of the original ones in unicast, the use of backup path in multicast may result in less replication and bandwidth waste. The BWCs achieved by PP and LP are similar in most cases. BWC of LP is slightly better in Geant with MST and RL. This is because MST and RL focus more on bandwidth utilization optimization while SPT does not. So the use of backup path in SPT may promote better overall bandwidth performance. Overall, considering both the performance of bandwidth consumption and stability, LP is still a better choice than PP.

Note that the recovery process may sometimes fail due to the characteristics of topology and multicast tree. The recovery



(a) Mean for link failure (b) Mean for node failure (c) P99 for link failure

Fig. 15. Performance of failure detection algorithm.

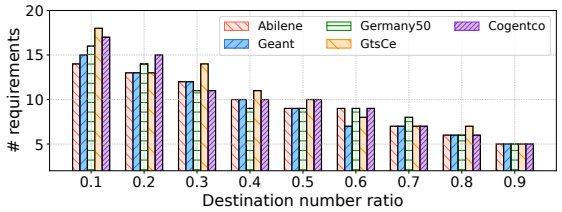


Fig. 16. Failure detection performance under varying group size.

failure rate (if any) is indicated by a percentage in parentheses. Recovery failure is more likely to occur in smaller topologies because there are less possible backup paths. Abilene cannot always guarantee successful recovery, and the failure rate of PP is higher compared with LP because it is more difficult to find all disjoint paths than bypassing a single link. PP also experiences a significant failure rate with MST in Geant, while LP always succeeds in Geant and Germany50.

VIII. Related Work

Stateful multicast protocols require frequent distributed state updates and SDN is a promised mechanism to solve the problem. Mohammadi et al. [11] use a nature-inspired optimization algorithm to compute the minimum cost tree regarding the end-to-end delay. Huang et al. [12] designed multicast traffic engineering for multiple trees in SDN, which targets at minimizing bandwidth consumption under node and link capacity constraints. However, these methods are designed for static multicast and are not suitable for intertemporal choice problems, where earlier routing decisions can impact later available routing choices and long-term performance. Chiang et al. [14] designed a traffic engineering solution to reduce bandwidth consumption and rerouting costs for single-tree scenarios with dynamic requests. They introduced the notions of budget and deposit to account for the temporal correlation of trees, determining how aggressively to construct a more bandwidth-efficient tree within each time slot. Additionally, they utilized a Reference Tree to stabilize routing decisions for frequently used paths. Chiang et al. [15] jointly optimize bandwidth cost and rerouting overhead in dynamic multicast with multiple trees by accommodating minor changes through partial rerouting, and only performing a complete rerouting of the tree when significant deterioration happens. However, these studies rely on simple heuristics in a reactive manner, not adequately capturing the underlying patterns of dynamic requests that manifest over a sufficient time scope, and thus, long-term performance may not be fully optimized.

Explicit Multicast [49] 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. [50] 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. [51] 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. [52] leverage multi-objective DRL to generate optimal policies for all possible routing preferences. Geng et al. [16] use multi-agent DRL to achieve distributed TE for global objective optimization. Liu et al. [18] 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 [53], capacity limitation [54] or energy consumption [17], [55]. 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 [56].

IX. 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. Incremental deployment and failure handling are also designed to improve Hawkeye's compatibility and robustness. 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. In future research, we aim to further enhance the performance

of our multicast framework in resource-constrained scenarios by exploring dynamic rate adjustment and congestion control strategies.

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] 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.

[3] 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.

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

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

[6] 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.

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

[8] 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.

[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] X. Jia and L. Wang, "A group multicast routing algorithm by using multiple minimum steiner trees," *Computer Communications*, vol. 20, no. 9, pp. 750–758, 1997.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] L. Lu, Q. Li, D. Zhao, Y. Yang, Z. Luan, J. Zhou, Y. Jiang, and M. Xu, "Hawkeye: A dynamic and stateless multicast mechanism with deep reinforcement learning," in *IEEE INFOCOM*, 2023, pp. 1–10.

[20] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, "NFVdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*, 2019, pp. 1–10.

[21] N. He, S. Yang, F. Li, S. Trajanovski, F. A. Kuipers, and X. Fu, "A-DDPG: Attention mechanism-based deep reinforcement learning for NFV," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, 2021, pp. 1–10.

[22] N. Geng, M. Xu, Y. Yang, C. Liu, J. Yang, Q. Li, and S. Zhang, "Distributed and adaptive traffic engineering with deep reinforcement learning," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, 2021, pp. 1–10.

[23] 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.

[24] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to Route," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, New York, NY, USA, Nov. 2017, pp. 185–191.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

[30] Y. Guo, W. Wang, H. Zhang, W. Guo, Z. Wang, Y. Tian, X. Yin, and J. Wu, "Traffic Engineering in Hybrid Software Defined Network via Reinforcement Learning," *Journal of Network and Computer Applications*, vol. 189, p. 103116, Sep. 2021.

[31] Z. Luan, L. Lu, Q. Li, and Y. Jiang, "EPC-TE: Explicit Path Control in Traffic Engineering with Deep Reinforcement Learning," in *GLOBECOM*, Dec. 2021, pp. 1–6.

[32] B. Ren, D. Guo, Y. Yuan, G. Tang, W. Wang, and X. Fu, "Optimal Deployment of SRv6 to Enable Network Interconnection Service," *IEEE/ACM Transactions on Networking*, vol. 30, no. 1, pp. 120–133, Feb. 2022.

[33] S. Chen, J. Zhang, and Q. Gao, "An efficient hybrid routing based on contact history in delay tolerant networks," in *2010 Seventh International Conference on Wireless and Optical Communications Networks - (WOCN)*, Sep. 2010, pp. 1–6.

[34] Z. Zhang, S. Ding, X. Wang, N. An, Z. Luo, W. Zhang, S. Yin, and S. Huang, "A Hybrid Switching Strategy Based on Betweenness Centrality in SDM Optical Transport Network," in *2022 IEEE 8th International Conference on Computer and Communications (ICCC)*, Dec. 2022, pp. 215–219.

[35] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "Cacheflow: Dependency-aware rule-caching for software-defined networks," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '16. New York, NY, USA: Association for Computing Machinery, 2016.

[36] Z. Luan, Q. Li, Z. Zhang, Y. Jiang, M. Chen, Y. Wang, and K. Li, "Awesome-cache: Dependency-free rule-caching for arbitrary wildcard patterns in tcam," in *2023 IEEE 31st International Conference on Network Protocols (ICNP)*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2023, pp. 1–12.

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

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

[39] 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.

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

[41] A. Fei, J. Cui, M. Gerla, and D. Cavendish, "A "dual-tree" scheme for fault-tolerant multicast," in *ICC 2001. IEEE International Conference on Communications. Conference Record (Cat. No.01CH37240)*, vol. 3, Jun. 2001, pp. 690–694 vol.3.

[42] D. Kotani, K. Suzuki, and H. Shimonishi, "A Design and Implementation of OpenFlow Controller Handling IP Multicast with Fast Tree Switching," in *2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet*, Jul. 2012, pp. 60–67.

[43] D. Li, M. Xu, Y. Liu, X. Xie, Y. Cui, J. Wang, and G. Chen, "Reliable Multicast in Data Center Networks," *IEEE Transactions on Computers*, vol. 63, no. 8, pp. 2011–2024, Aug. 2014.

[44] J. Chen, F. Yan, D. Li, S. Chen, and X. Qiu, "Recovery and Reconstruction of Multicast Tree in Software-Defined Network: High Speed and Low Cost," *IEEE Access*, vol. 8, pp. 27 188–27 201, 2020.

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

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

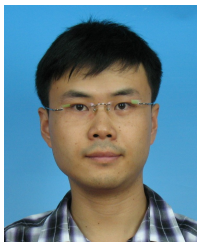
[47] E. Baccour, A. Erbad, M. Guizani, and M. Hamdi, "FacebookVideo-Live18: A live video streaming dataset for streams metadata and online viewers locations," in *ICIoT*, 2020, pp. 476–483.
[48] H.-J. Heo, N. Kim, and B.-D. Lee, "Multicast Tree Generation Technique Using Reinforcement Learning in SDN Environments," in *SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI*, Oct. 2018, pp. 77–81.
[49] R. Boivie, Y. Imai, W. Livens, and D. Ooms, "Explicit Multicast (Xcast) concepts and options," RFC Editor, Tech. Rep. RFC5058, 2007.
[50] 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.
[51] K. Diab and M. Hefeeda, "Yeti: Stateless and generalized multicast forwarding," in *NSDI*, 2022, pp. 1093–1114.
[52] 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.
[53] 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.
[54] 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.
[55] 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.
[56] 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.

Qing Li received the B.S. degree (2008) from Dalian University of Technology, Dalian, China, the Ph.D. degree (2013) from Tsinghua University, Beijing, China; both in computer science and technology. He is currently a full professor at Peng Cheng Laboratory, Shenzhen, China. His research interests include reliable and scalable routing of the Internet, software defined networking, network function virtualization, in-network caching/computing, edge computing, traffic scheduling, transmission control, video delivery, etc.

Lie Lu received the B.S. degree (2020) and M.S. degree (2023) from Tsinghua University, China. He is currently with Alibaba Cloud, Alibaba Group, Hangzhou, China. His research interests include network routing and the application of Artificial Intelligence in routing optimization.

Dan Zhao is currently an assistant researcher with Peng Cheng Laboratory, Shenzhen, China. She was a Postdoctoral Researcher with School of Electronic Engineering, Dublin City University. Dr Dan Zhao obtained her bachelor's degree from Beijing University of Posts and Telecommunications in 2011, and Ph.D. degree from Queen Mary University of London in 2015. Her research interests include network routing, in-network intelligence and network security.

Zeyu Luan received the B.S. and M.S. degrees from Tianjin University in 2015 and 2018, respectively. He obtained the Ph.D. degree in computer science at Tsinghua Shenzhen International Graduate School, Tsinghua University, in 2024. He is currently a Postdoctoral Researcher in Pengcheng Laboratory, Shenzhen, China. His research interests include software-defined networking, traffic engineering, and deep reinforcement learning.



Yuan Yang received the B.Sc., M.Sc., and Ph.D. degrees from Tsinghua University. He was a Visiting Ph.D. Student with The Hong Kong Polytechnic University. He is currently an Associate Researcher with the Department of Computer Science and Technology, Tsinghua University. His major research interests include computer network architecture and routing protocols.

Yong Jiang received the B.S. degree (1998) and the Ph.D. degree (2002) from Tsinghua University, Beijing, China, both in computer science and technology. He is currently a full professor at the Graduate school at Shenzhen, Tsinghua University. His research interests include the future network architecture, the Internet QoS, software defined networks, network function virtualization, etc.

Jingpu Duan received his B.E. degree from Huazhong University of Science and Technology in 2013, and his Ph.D. degree from The University of Hong Kong in 2018. He is currently an assistant researcher in the Department of Broadband Communiation, Pengcheng Laboratory. His research interests include designing and implementing high-performance networking systems.

Ruobin Zheng received the B.Eng degree (1996) in Electronic Engineering and the M.Sc degree (1999) in Radio Physics all from Xiamen University, China and the second M.Sc degree (2003) in Wireless Communication from Univ. of Waterloo, Canada. He has been with Huawei Technologies for more than twenties years and is currently a technical expert at the Huawei's 2012 Labs. He is the inventor of over two hundred patents which cover both wired and wireless communication areas. His research interests include broadband access, compute networking and service routing.

Shaoteng Liu received the B.S. and M.S. degrees in Microelectronics from Fudan University, Shanghai, China and the Ph.D. degree in electronic and computer system from the KTH Royal Institute of Technology, Sweden. Currently, he is a technical expert and scientist in Huawei. His research interests include electronic systems, networks-on-chip, network coding, network topology, machine learning and etc.

Dingding Chen received his B.Sc. degree in Computer Science from Faculty of Information Engineering and Automation of Kunming University of Science and Technology, Kunming, China, in 2015, and the Ph.D. degree in Computer Science at Chongqing University, Chongqing, China, in 2022. He is currently a Senior Engineer with the Network Technology Lab, Central Research Institute, Huawei 2012 Labs. His research interests include constraint programming, coding theory and machine learning.