Received May 6, 2021, accepted May 26, 2021, date of publication May 28, 2021, date of current version June 8, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3084738

A Priority-Based Multicast Flow Scheduling Method for a Collaborative Edge Storage Datacenter Network

WENLONG KE[1,2], **YONG WANG**[2,3], **MIAO YE**[1,4], **AND JUNQI CHEN**[2,3]
[1]School of Information and Communication, Guilin University of Electronic Technology, Guilin 541004, China
[2]Engineering Technology Research Center of Cloud Security and Cloud Service, Guilin University of Electronic Technology, Guilin 541004, China
[3]School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin 541004, China
[4]Ministry of Education Key Laboratory of Cognitive Radio and Information Processing, Guilin University of Electronic Technology, Guilin 541004, China

Corresponding author: Miao Ye (ym@mail.xidian.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61861013 and Grant 61662018, in part by the Guangxi Innovation-Driven Development Project under Grant AA18118031, in part by the Guangxi Natural Science Foundation of China under Grant 2018GXNSFAA050028, and in part by the Director Fund Project of Key Laboratory of Cognitive Radio and Information Processing of Ministry of Education under Grant CRKL190102.

ABSTRACT Edge storage, as a supplement to cloud storage, reduces latency by providing services in a timely and efficient manner near the source. In a collaborative edge storage datacenter network (CESN), not only does the edge storage datacenter (ESDC) that is closest to the user provide services, but multiple ESDCs work together to provide better services. In this collaborative work mechanism, different application session requests create large persistent multicast flows with diverse performance requirements. Existing multicast scheduling methods such as unicast shortest path (USP) and static single tree (SST) do not consider flow characteristics or performance requirements. In this paper, we first modeled the multicast flow scheduling problem in a CESN. The model is based on different types of flows with diverse network requirements. Then, we tailored a multicast flow scheduling method based on multiple-attribute decision-making and a genetic algorithm (MDGA). MDGA selects appropriate multicast routing paths for flows in a CESN by considering the requested flow types and network status. The experimental results show that the proposed MDGA method can balance network loads and reduce the average transmission delay for high-priority flows better than USP and SST.

INDEX TERMS Collaborative edge storage, datacenter network, multicast flow, multiple-attribute decision-making, genetic algorithm.

I. INTRODUCTION

With the rapid development of the Internet of Everything (IoE), the growing data storage requirement is posing a complex technical challenge for the IT community. According to the white book Data-Age-2025 [1] from the International Data Corporation (IDC), the global data scale will grow from 33 zettabytes in 2018 to 175 by 2025. At the same time, the emergence of many new applications such as unmanned driving and smart cities will lead to a greater demand for reducing data transmission delay [2]. For the traditional cloud storage architecture, which has demonstrated effective storage capacity, it is a challenge to satisfy the latency requirement, as this centralized cloud storage system is usually composed of a few large cloud storage datacenters interconnected by long-distance networks and is far from end-users. This challenge can be overcome by using an edge storage system, which employs a group of small edge storage datacenters (ESDCs) to process data in a timely and efficient manner near the source [3].

Existing works [4], [5] have shown the advantages of edge storage in improving the efficiency of delay-sensitive services. However, many applications in edge storage have become data intensive. These applications need additional comprehensive databases to support complicated data analytics. For a single ESDC with limited storage capacity, it is difficult to meet these data requirements. Although cloud storage datacenters can be used when edge servers lack the necessary data for a task, the increased processing latency is unacceptable for delay-sensitive tasks. To improve the edge

The associate editor coordinating the review of this manuscript and approving it for publication was Shaoyong Zheng.

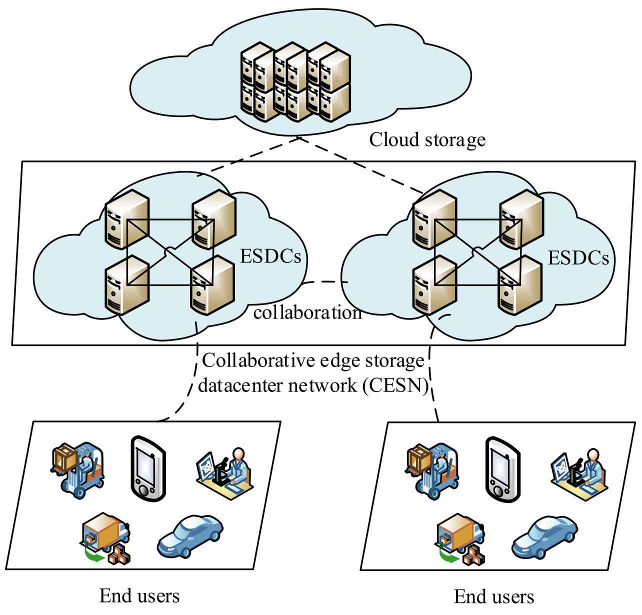


FIGURE 1. **Example of collaborative edge storage architecture.**

layer's performance, the concept of the collaborative edge network has been introduced [6]. Fig. 1 illustrates an example infrastructure architecture for collaborative edge storage.

The collaborative edge storage mechanism is a storage paradigm in which multiple ESDCs collaborate with each other through the edge network to share data and storage capacity and satisfy global goals. In this collaborative edge storage datacenter network (CESN), a variety of applications often replicate or share data among ESDCs to improve data reliability and quality of experience (QoE) for end-users [7]. This kind of data transmission in CESNs has created large persistent network flows among multiple ESDCs. Scheduling these network flows effectively has become a key factor in improving the service performance of collaborative edge storage. There are many existing works such as [6], [8]–[11], which are based on a collaborative edge network. The work of [11] shows that deploying a collaborative edge mechanism can achieve better performance in task offloading than using a single edge device. However, most of these works focus on data placement or task offloading. The work of [6] shows that most existing works solve the task scheduling problem in a collaborative edge network without considering flow scheduling, which can lead to network congestion and a long completion time. Although recent works [6], [10] have studied the network flow scheduling problem in collaborative edge networks, these works do not consider both multicast flow scheduling and QoE scheduling. In this paper, we focus on the flow scheduling problem in CESNs.

Although previous works have shown the effectiveness of deploying flow scheduling methods to improve service performance in inter-datacenter networks, it is still a challenge to tailor a specific multicast routing mechanism for CESNs to improve QoE performance for data flows from diverse applications. Compared with the flow scheduling problem in inter-datacenter networks, the routing problem in CESNs is more complicated. First, the quality of services (QoS) and QoE of flow scheduling in CESNs have more strict and diverse requirements [12]. Some online transaction processes in CESNs, such as unmanned driving and online conversation, usually have stricter delay requirements, while some big data storage processes, such as downloading or uploading high-quality movies, usually have higher requirements in terms of throughput. It is a challenge to guarantee these diverse requirements in a shared CESN environment. Second, the flow transmission mechanism in CESNs is more complicated than that in other networks. To improve the security and reliability of data storage, collaborative edge storage often uses multicopy storage mechanisms. In this scenario, the data uploaded by users is first stored in the master storage node and is then distributed from the master storage node to multiple slave storage nodes. This mechanism means that the CESN includes not only unicast transmissions but also a large number of multicast transmissions to reduce redundant traffic generated during data distribution. It is a challenge to build appropriate multicast trees for flow scheduling to achieve network load balance under the premise of improving QoE performance for diverse applications.

In this paper, we consider specific flow characteristics and propose a priority-based multicast flow scheduling method for CESNs. The main contributions of this paper can be summarized as follows:

1) A multicast flow scheduling optimization model based on service priority in CESNs is proposed. The best multicast transmission path for each service flow can be found through the model by considering the flow's specific network performance requirements and the real-time network status information. The QoE performance of a CESN can be improved by minimizing the cumulative weighted delay and balancing link utilization.
2) A multicast flow scheduling method based on multiple-attribute decision-making and a genetic algorithm MDGA) is proposed to solve the optimization model. A multicast path finding task is first decomposed into multiple unicast path finding tasks, which are processed by a multi-attribute decision-making-based method. Then, a genetic-algorithm-based method is used to select the appropriate combination of unicast paths to construct the multicast path.
3) We use Mininet to build a simulated CESN environment to verify the effectiveness of MDGA. The experimental results show that MDGA can reduce the transmission delay of high-priority flows and improve the network load balancing performance.

The rest of this paper is organized as follows: The second part summarizes the related work. The third part describes the multicast scheduling problem in CESNs and provides the optimization model. The fourth part introduces the details of our MDGA method. The implementation and performance evaluation are presented in the fifth part. The sixth part concludes the paper.

II. RELATED WORK

In this section, we look at the related work in two areas: (A) CESNs and (B) network flow scheduling.

A. CESNs

Collaborative edge storage is an effective method for improving storage efficiency and reliability in an edge computing environment. The work of [8] proposes an edge-side collaborative storage framework called ECS, which solves the data placement problem by using a graph-based iterative algorithm. By applying edge-side collaboration, ECS reduces the data-processing latency because fewer tasks are offloaded to cloud datacenters. ACMES [9] is a multiplier-based collaborative storage scheduling algorithm that aims to improve the execution efficiency of storage and optimize storage resource utilization. With ACMES, storage tasks can be adaptively distributed to individual edge nodes through comprehensive consideration of the total cost, reliability, power usage and risk of node withdrawal. However, the works of [8] and [9] solve the problem of data placement without considering network flow scheduling, which can lead to network congestion.

The work of [6] shows that most existing works solve the task scheduling problem in collaborative edge networks without considering network flow scheduling. To address this challenge, [6] proposes a multistage greedy adjustment algorithm called MSGA for data-aware task allocation. By considering both data placement and the network bandwidth consumption, MSGA achieves better performance in terms of task completion time. JPOFH [10] is an extension of MSGA. By jointly scheduling tasks and network flows, JPOFH achieves a significant improvement in the average completion time in multihop collaborative edge networks. However, [6], [10] do not consider both multicast flow scheduling and QoE scheduling, both of which are important in CESNs. Multicast flow scheduling can reduce the redundant traffic generated during data distribution. In CESNs, services, including video and audio distribution for internet protocol television (IPTV) [13], multicopy replication of data storage [14], and distribution of sensor monitoring data [15], all have a large number of multicast transmission requirements. For QoE scheduling, an effective method is to reduce the average transmission delay of high-priority flows and balance the network link utilization. Multicast flow scheduling and QoE scheduling should be jointly considered in CESNs.

B. NETWORK FLOW SCHEDULING

The flow scheduling problem in datacenter networks [16] has been well studied. For routing with optimizing flow transmission paths, ECMP [17] is the most widely used scheduling method. It uses a hash-based path selection strategy and aims to spread traffic equally across redundant paths. NetStitcher [18] aims to reduce the flow transmission delay between long-distance datacenters. The data flow is first transmitted to a low-load transfer datacenter and then forwarded to the destination datacenter. As an effective method for collecting network information and performing network management, SDN has received extensive attention for flow scheduling. B4 [19] is an SDN-based routing method proposed by Google. It uses SDN to measure network information and distributes data flows by link utilization. Jin *et al.* [20] proposed another centralized routing method based on SDN. They used SDN to collect network global information, which is used to select the transmission path and determine the transmission rate. Their method improves network utilization.

The aforementioned methods [17]–[20] have shown effectiveness in improving network performance in inter-datacenter networks. However, most of these methods aim at unicast transmission, and it is difficult for them to deal with multicast transmission scenarios. To deal with multicast scheduling scenarios, RMMR [21] uses an SDN-based multipath multicast flow scheduling mechanism. RMMR achieves a lower packet loss rate and better load balancing performance than the traditional IP multicast mechanism in video stream distribution scenarios. BCMS [22] is the first multicast flow scheduling method for fat-tree datacenter networks. It calculates the best transmission path for each multicast flow according to the flow's bandwidth demand and the residual bandwidth of each link to perform network congestion control and load balancing. MSaSDN [23] is another multicast routing mechanism for fat-tree datacenter networks. It constructs the appropriate multicast tree for flow transmission based on minimizing the link congestion overhead, and it improves the network load balancing performance. To address the lag problem of SDN-based network measurement, MSaMC [24] uses Markov chains to predict the link congestion probability of the next time slot. This prediction-based routing method improves the network throughput and reduces the average transmission delay of the data flow.

However, the aforementioned multicast flow scheduling methods [21]–[24] are mostly designed for intra-datacenter networks. It is usually difficult for these methods to deal with the routing problem in CESNs. Because the routing problem is often fairly easy to solve if the regular CLOS-based intra-datacenter network topology structure is represented in the problem formulation [25]. An efficient way to build a multicast tree in unstructured inter-datacenter networks is to use a Steiner tree [26], which can minimize the number of links in the tree required for a multicast task [27]. Iris [28] is a Steiner-tree-based multicast traffic scheduler for optimizing different completion time objectives for various requests. However, Iris is designed for bulk transfers between long-distance inter-datacenter networks, which is not appropriate for the routing requirements in CESNs, as the traffic flow in CESNs is fairly small but has greater requirements on transmission delay.

Most of the aforementioned multicast flow scheduling methods [21]–[24], [27], [28] give general scheduling

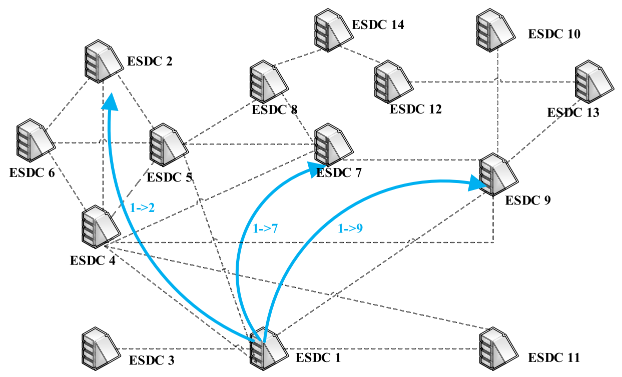


FIGURE 2. Example of multicast scheduling in a CESN. The blue lines indicate the multicast request [1, (2, 7, 9)].

strategies for all data flows. However, a CESN is often shared by multiple services for cost considerations, and the data flows generated by different services usually have different requirements for network performance. For example, in a network environment dominated by video services, network bandwidth is more important [29], while in a network environment dominated by game services, latency is the focus for users [30]. Therefore, it is difficult to achieve ideal performance for different services in CESNs by using a general scheduling strategy.

The network flow scheduling method should provide tailored solutions according to different services performance requirements. In this paper, we propose a multicast flow scheduling method called MDGA for CESN. The motivation of MDGA is to reduce the average transmission delay of high-priority flows and improve the network load balancing performance.

III. MULTICAST SCHEDULING PROBLEM IN THE CESN

In this section, we first state the multicast flow scheduling problem in CESNs. Our motivation is to find the best multicast transmission path, called the *multicast tree*, for each multicast request. Then, we give the problem formulation and propose a multicast flow scheduling optimization model based on flow priority.

A. PROBLEM STATEMENT

Fig. 2 shows an example CESN based on Equinix's datacenters [31] in New York city. Unlike inter-datacenter networks, which are usually interconnected by long-distance networks, a CESN connects a group of small ESDCs, which process data near the source to reduce latency. However, in order to increase the data reliability and QoE for end-users, not only does the ESDC closest to the user provide services, but multiple ESDCs work together to provide better services. This mechanism will generate a large number of multicast transmission requests in a CESN. Our motivation is to find the best multicast tree for each multicast transmission request.

To illustrate the multicast scheduling problem in a CESN, we show a simple multicast transmission problem in Fig. 2. The blue lines in Fig. 2 indicate a multicast transmission request, which moves from ESDC1 to ESDC2, ESDC7, and ESDC9. We formulate this request as [1, (2, 7, 9)]. As we can see in Fig. 2, the path from ESDC1 to ESDC2 may be (1, 4, 2), (1, 5, 2), (1, 4, 6, 2), etc. We call these paths sub-path set1. Similarly, sub-path set2 from ESDC1 to ESDC7 contains (1, 5, 7), (1, 4, 7), (1, 9, 7), etc. Sub-path set3 from ESDC1 to ESDC9 contains (1, 9), (1, 4, 9), (1, 5, 7, 9), etc. A multicast tree is formed by selecting one path from each sub-path set. For example, a path combination [(1, 5, 2), (1, 5, 7), (1, 5, 7, 9)] of paths selected from each sub-path set is a multicast tree for [1, (2, 7, 9)].

The aforementioned path selection method needs to consider the load conditions of each link in the CESN to improve the network load balancing performance. Additionally, different types of flows have different network performance requirements. It is also necessary to consider the performance requirements of these different service flows in path selection to improve QoE performance. In our previous work, we found that the storage system mainly contains the following three service flows [32]:

- *Heartbeat flows* are used to monitor whether the storage nodes are working properly. These flows have the highest transmission priority. They are highly sensitive to transmission delay and need little network bandwidth.
- *User data flows* are generated by upload or download tasks performed by end-users. Their completion time directly affects the user experience. These flows have the second highest transmission priority. They are sensitive to transmission delay and require a large scale of transmission bandwidth.
- *Migration flows* are generated by the load balancing mechanism of the storage system. Their completion time has little effect on the user experience. These flows have the lowest transmission priority and need a large scale of network bandwidth.

The details of the mathematical model for multicast path selection in CESNs will be introduced in section III-B.

B. PROBLEM FORMULATION

The network topology of a CESN can be modeled by a graph $G(V, E)$, in which V denotes the set of ESDCs and E indicates the set of links between adjacent ESDCs. As analyzed in section III-A, the multicast scheduling problem in CESNs is to find the best multicast path for each multicast request that minimizes the cumulative weighted transmission delay and improves the network load balancing performance. Additionally, the network hops are important for path selection in CESNs. A multicast path with fewer network hops can save more link bandwidth overhead. For easy reference, the symbols used in this section are listed in Table 1.

TABLE 1. Symbols in the problem formulation.

Symbol	Description	Symbol	Description
G	network topology	V	ESDC set
E	set of links between adjacent ESDCs	φ	multicast flow
M	number of multicast flows	e	link between two adjacent ESDCs
n_{ek}	number of nonrepeating links e in k	p	unicast path in multicast path k
$d_{\varphi p}$	delay of flow φ via path p	k	multicast path
L	number of multicast paths	b_{φ}	bandwidth cost of flow φ
b_e	bandwidth capacity of link e	$w_{d\varphi}$	delay coefficient of φ
$d_{\varphi b}$	maximum delay threshold of φ	$w_{b\varphi}$	bandwidth coefficient of φ
$x_{\varphi k}$	binary variable for path selection	R_e	residual bandwidth of link e
$w_{h\varphi}$	network hop coefficient of φ	w_{φ}	delay weight of a type of flow

1) CUMULATIVE WEIGHTED TRANSMISSION DELAY

The cumulative weighted transmission delay of a type of flow is defined as $Dcu(x_{\varphi k})$ in equation (1).

$$Dcu(x_{\varphi k}) = \sum_{\varphi=1}^{M}\sum_{k=1}^{L} w_{\varphi} \max_{p\in k}\{d_{\varphi p}\}x_{\varphi k} \tag{1}$$

where w_{φ} denotes the delay sensitivity coefficient of multicast flow φ. A flow φ that has higher priority corresponds to a larger value of w_{φ}. P denotes a unicast path in the multicast path K. $\max\limits_{p\in k}\{d_{\varphi p}\}$ is the transmission delay of multicast flow φ routing through multicast path k. $x_{\varphi k}$ denotes a binary variable: it is 1 if flow φ is routed through path k and 0 otherwise.

2) NETWORK LOAD BALANCING PERFORMANCE

We use the residual bandwidth of links to evaluate the network load balancing performance, which is defined as follows:

$$Lbp(x_{\varphi k}) = \sum_{\varphi=1}^{M}\sum_{k=1}^{L} \min_{e\in k}\{R_e\}x_{\varphi k} \tag{2}$$

where e is a link between two adjacent ESDCs. R_e denotes the residual bandwidth of link e. $\min\limits_{e\in k}\{R_e\}$ is the bottleneck of the residual bandwidth of multicast path k. A path with more residual bandwidth should be chosen for a flow with large bandwidth consumption to improve the network load balancing performance. The larger the residual bandwidth of the bottleneck link is, the better the network load balancing performance.

3) NETWORK HOPS

The network hops are defined as the number of nonrepeating links in each multicast tree. The cumulative network hops of L multicast trees are defined as follows:

$$Hop(x_{\varphi k}) = \sum_{\varphi=1}^{M}\sum_{k=1}^{L} n_{ek}x_{\varphi k} \tag{3}$$

where n_{ek} denotes the number of nonrepeating links in the k-th multicast tree. A multicast tree with fewer network hops can save more link bandwidth overhead.

4) OBJECTIVE FUNCTION

Our goal is to minimize $Dcu(x_{\varphi k})$ and $Hop(x_{\varphi k})$ and maximize $Lbp(x_{\varphi k})$. The objective function of this paper is defined as follows:

$$\min f = [Dcu(x_{\varphi k}), \frac{1}{Lbp(x_{\varphi k})}, Hop(x_{\varphi k})] \tag{4}$$

However, it is difficult to achieve the minimum values of $Dcu(x_{\varphi k})$, $1/Lbp(x_{\varphi k})$, and $Hop(x_{\varphi k})$ at the same time. For example, the multicast tree with the minimum $Hop(x_{\varphi k})$ is called the minimum-edge Steiner tree. However, it is usually difficult for the minimum-edge Steiner tree to minimize $Dcu(x_{\varphi k})$ and $1/Lbp(x_{\varphi k})$. To deal with this problem, we use the component weights to transform the objective function in equation (4) into the optimization problem in equation (5).

$$\min f = \sum_{\varphi=1}^{M}\sum_{k=1}^{L}(w_{d\varphi}w_{\varphi}\max_{p\in k}\{d_{\varphi p}\} + w_{b\varphi}\frac{1}{\min\limits_{e\in k}\{R_e\}} + w_{h\varphi}n_{ek})x_{\varphi k} \tag{5}$$

$$\text{s.t. } w_{d\varphi} + w_{b\varphi} + w_{h\varphi} = 1 \tag{6}$$

$$\sum_{k=1}^{L}\max_{p\in k}\{d_{\varphi p}\}x_{\varphi k} \le d_{\varphi b}, \quad \forall\varphi \in \{1,\ldots,M\} \tag{7}$$

$$\sum_{\varphi=1}^{M} b_{\varphi}x_{\varphi k} \le \min_{e\in k}\{b_e\}, \quad \forall k \in \{1,\ldots,L\} \tag{8}$$

$$\sum_{k=1}^{L} x_{\varphi k} = 1, \quad \forall\varphi \in \{1,\ldots,M\} \tag{9}$$

$$x_{\varphi k} \in \{0,1\}, \ \forall\varphi \in \{1,\ldots,M\}, \ \forall k \in \{1,\ldots,L\} \tag{10}$$

$w_{d\varphi}$, $w_{b\varphi}$, and $w_{h\varphi}$ in constraint (6) denote the weight coefficients of the delay, residual bandwidth, and network hops, respectively. Constraint (7) satisfies the delay requirement $d_{\varphi b}$ for flow φ. Constraint (8) ensures that the total bandwidth overhead for all multicast flows routing through path k does not exceed the bottleneck bandwidth. Constraint (9) means that only one path will be assigned to each multicast flow φ. In constraint (10), $x_{\varphi k}$ denotes a binary variable: it is 1 if flow φ is routed through path k, and 0 otherwise. The details of our flow scheduling solution are given in section IV.

IV. MDGA MULTICAST FLOW SCHEDULING METHOD

To solve the objective function in equations (5-10), MDGA is proposed. Fig. 3 presents the system architecture of our MDGA method, which includes three main modules:

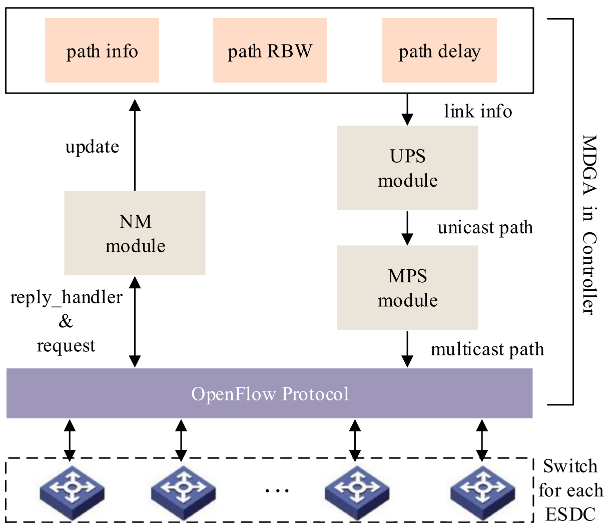


FIGURE 3. Architecture of the MDGA method. The path info, path RBW, and path delay with an orange background denote the network information in equations (11), (12), and (13), respectively.

a network measurement (NM) module, unicast path selection (UPS) module, and multicast path selection (MPS) module.

The process of MDGA finding the best multicast tree for a multicast request can be described briefly as follows: First, MDGA employs the NM module to update the real-time network information. Second, the UPS module is used to find the k-best path set for each unicast task of the multicast request, where a unicast task moves from the source node of the multicast request to one of the destination nodes of the multicast request. Last, MDGA employs the MPS module to find the appropriate combination of paths such that one path is selected from each k-best path set. The result of this combination is the multicast tree.

A. THE NM MODULE

The NM module follows the OpenFlow protocol of SDN to discover the network topology and update the real-time network information. It sends a network information request to each ESDC's switch periodically and handles the reply to calculate the network information. The network information recorded by the NM module is as follows:

$$P = \{p_1, p_2, \ldots, p_n\} \tag{11}$$

where P is the set of unicast paths between each pair of ESDCs. This path set can be found by using the Dijkstra [33] algorithm.

$$B_p = \{b_1, b_2, \ldots, b_n\} \tag{12}$$

where B_p denotes the set of residual bandwidths of each path p. The residual bandwidth of a path can be calculated by using the SDN-based network measurement method in our previous work [34].

$$D_P = \{d_1, d_2, \ldots, d_n\} \tag{13}$$

where D_P denotes the set of end-to-end delay of each path p. The delay can be obtained by using the SDN-based network measurement method [34].

$$H_P = \{h_1, h_2, \ldots, h_n\} \tag{14}$$

where H_P denotes the set of network hops of each path p. A path p is composed of multiple adjacent links. The value of the network hops of path p is the number of nonrepeating links in p.

When the proposed algorithm in equations (5-10) is implemented in the actual environment shown in Fig. 3, the necessary actual information such as the candidate paths, average path transmission delay, residual path bandwidth and path hops can be obtained by equations (11-14), respectively. Another challenge in deploying the proposed algorithm in an actual environment is to classify network flows with different priorities. In a dedicated storage network, the network port of flow can be used as the label to distinguish different types of flows.

In MDGA, the path set discovery method that uses the Dijkstra algorithm is executed only once when the program is initialized. The discovered path set can be accessed directly in the subsequent path selection processes. This mechanism reduces the computational overhead of duplicate path discovery.

B. THE UPS MODULE

When MDGA processes a point-to-multipoint multicast request, it first decomposes the multicast request into multiple point-to-point unicast tasks. Each unicast task moves from the source node of the multicast request to one of the destination nodes of the multicast request. The UPS module is used to find the k-best path set for each unicast task by using multiple-attribute decision-making. The process of the UPS module finding the k-best path set for a unicast task is described below.

1) CONSTRUCTING THE DECISION-MAKING MATRIX

$$\mathbf{M} = \begin{bmatrix} b_1 & b_2 & \ldots & b_n \\ d_1 & d_2 & \ldots & d_n \\ h_1 & h_2 & \ldots & h_n \end{bmatrix} \tag{15}$$

where $\mathbf{M}$ is the decision-making matrix for finding the k-best path set for each unicast task. Each column in $\mathbf{M}$ represents a candidate path. The symbols b, d, and h in each column represent the residual bandwidth, end-to-end delay and network hops of each path, respectively. These network attributes are obtained by the NM module.

2) NORMALIZING THE NETWORK ATTRIBUTES

To eliminate the dimension of each network attribute, the min-max normalization method is used. Equation (16) is used for the end-to-end delay attribute and the network hop attribute, which can achieve a better effect with smaller values. Equation (17) is used for the residual bandwidth attribute, which can achieve a better effect with larger values.

$$u(x) = \frac{x^{\max} - x}{x^{\max} - x^{\min}} \tag{16}$$

$$v(x) = \frac{x - x^{\min}}{x^{\max} - x^{\min}} \tag{17}$$

Then, the normalization decision-making matrix $\mathbf{M}^*$ can be obtained as follows:

$$\mathbf{M}^* = \begin{bmatrix} b_1^* & b_2^* & \dots & b_n^* \\ d_1^* & d_2^* & \dots & d_n^* \\ h_1^* & h_2^* & \dots & h_n^* \end{bmatrix} \tag{18}$$

where $b_j^* = v(b_j)$, $d_j^* = u(d_j)$, $h_j^* = u(h_j)$, and $j \in \{1, 2, \dots, n\}$. j represents the sequence number of the matrix column, corresponding to the sequence number of the candidate path.

3) CALCULATING THE WEIGHTED SUMMATION OF THE NORMALIZED ATTRIBUTES

The weighted summation of each candidate path can be calculated by equation (19).

$$Z_j = w_b b_j^* + w_d d_j^* + w_h h_j^* \tag{19}$$

where $\mathbf{W} = [w_b, w_d, w_h]$ is the vector of weighted coefficients for the residual bandwidth, end-to-end delay, and network hops, respectively. Different flow types have different vectors $\mathbf{W}$ due to their different performance requirements. Z_j represents the weighted summation of the normalized attributes of candidate path j. A path j with a larger Z_j value is a better path. In a unicast task, the top k paths with larger Z_j values are used to construct the k-best path set.

The pseudocode of the function of the UPS module is shown in Function 1.

C. THE MPS MODULE

When processing a multicast request with s destination nodes, MDGA first employs the UPS module to find sk-best path sets. Then, the MPS module is used to find the multicast tree by constructing an appropriate combination of paths such that one path is selected from each k-best path set. To improve the efficiency of searching for the best combination, the MPS module uses a genetic-algorithm-based search method. As adaptive methods based on the mechanics of natural selection, genetic algorithms are very efficient in directing the search toward relatively good prospects [35]. The structure of the search solution in the MPS module is presented in Fig. 4.

For simplicity, we suppose that a multicast request with s destination nodes has been processed by the UPS module. The results are s k-best path sets, which are denoted as set_1 $= \{p_{11}, p_{12}, \dots, p_{1k}\}$, set_2 $= \{p_{21}, p_{22}, \dots, p_{2k}\}, \dots$, and set_s $= \{p_{s1}, p_{s2}, \dots, p_{sk}\}$. The details of the search solution for this s-destination multicast request are presented below.

Function 1 Unicast Path Selection (UPS)

```
Input: candidate unicast path set $P$; path residual
       bandwidth set $B_p$; path delay set $D_P$; path hops set
       $H_P$; source node $n_{src}$ and destination node $n_{dst}$ of
       the unicast request; delay requirement $d_{\varphi b}$ of the
       unicast request; threshold value $k$; vector of
       weighted coefficients for the residual bandwidth,
       end-to-end delay, and network hops
       $\mathbf{W} = [w_b, w_d, w_h]$.
Output: the $k$-best unicast path set $kb$_set from $n_{src}$ to $n_{dst}$.
1: for path $p$ in path set $P$ do
2:    if $p$ is from $n_{src}$ to $n_{dst}$ and $d_p \le d_{\varphi b}$ do
3:       add $p$ to path set $P_n$;
4:    end if
5: end for
6: construct the decision-making matrix $\mathbf{M}$ based on $P_n$
   according to equation (15);
7: normalize $\mathbf{M}$ to $\mathbf{M}^*$ according to equations (16-18);
8: calculate the weighted summation of each candidate
   path in $\mathbf{M}^*$ according to equation (19);
9: sort $P_n$ from the path $p$ that has the largest weighted
   summation to that with the smallest one;
10: Add the first $k$ paths of the sorted $P_n$ to $kb$_set;
11: return $kb$_set.
```

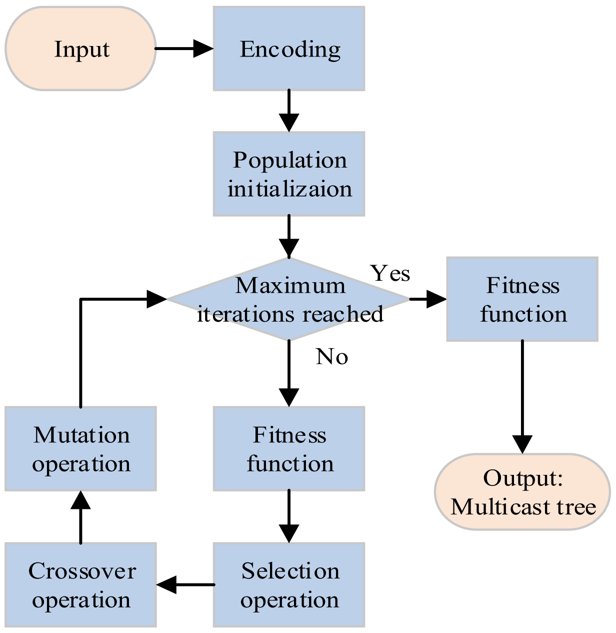


FIGURE 4. The search solution in MPS, where the modules with blue backgrounds indicate the stages of the solution.

1) ENCODING

The encoding is used to denote a multicast tree constructed from s k-best path sets. The MPS module employs one s-dimensional vector as the encoded value. This vector is shown in equation (20).

$$\mathbf{V} = [p_{1\phi}, p_{2\varpi}, \dots, p_{s\delta}] \tag{20}$$

where $p_{1\phi}$ denotes the ϕ-th path in set_1, $p_{2\varpi}$ denotes the ϖ-th path in set_2, and $p_{s\delta}$ denotes the δ-th path in set_s. ϕ, ϖ, and δ are random integers from 1 to k. The vector V represents an encoded multicast tree.

2) POPULATION INITIALIZATION

The population initialization process is used to construct an encoded vector pool for subsequent evolutionary calculations. The encoded vectors in this pool are called individuals. The pool is shown in equation (21).

$$pool = \{\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_n\} \tag{21}$$

where each individual $\mathbf{V}_i$, $i \in \{1, 2, \ldots, n\}$ is obtained randomly by using equation (20).

3) FITNESS FUNCTION

The fitness function is used to evaluate each individual $\mathbf{V}_i$ in the pool. The individual $\mathbf{V}_i$ that meets the selection criteria or the best individual during a specific number of evolutionary calculations will be selected as the multicast tree. The calculation method of the fitness function follows equations (5-10).

4) SELECTION OPERATION

The MPS module employs the selection operation to select several individuals from the pool with a certain probability. Generally, the selection method is a process of survival of the fittest based on the fitness function. The selected pool with the surviving individuals is shown in equation (22).

$$s\_pool = \{\mathbf{V}_\sigma, \mathbf{V}_\varsigma, \ldots, \mathbf{V}_\tau\}, \quad s\_pool \subseteq pool \tag{22}$$

where *s_pool* denotes the selected population pool, in which the selected individuals have the best fitness values in the unselected *pool*.

5) CROSSOVER OPERATION

After the selection operation, the number of individuals in *s_pool* is less than the number of individuals in *pool*. To increase the number of individuals in *s_pool* so that it is the same as that of *pool*, the crossover operation is used. The crossover operation is employed to create a new child individual based on two chosen parent individuals from *s_pool*. An individual with a larger fitness value has a higher probability of being chosen as the parent individual. Supposing there are two parent-individuals $\mathbf{V}_1 = [p_1, p_2, \ldots, p_s]$ and $\mathbf{V}_2 = [p_{1^*}, p_{2^*}, \ldots, p_{s^*}]$, the child-individual $\mathbf{V}_c$ is shown in equation (23).

$$\mathbf{V}_c = [p_1, \ldots, p_{\theta^*}, \ldots, p_{\upsilon^*}, \ldots, p_s] \tag{23}$$

where p_{θ^*} and p_{υ^*} are the θ^*-th and υ^*-th elements in $\mathbf{V}_2$, respectively, and $1 < \theta^* \leq \upsilon^* < s$.

6) MUTATION OPERATION

To avoid falling into a local optimum in the search process, a mutation operation is introduced. In the MPS module,

Function 2 Multicast Path Selection (MPS)

```
Input: s kb_sets for a multicast request with s destination
       nodes; the vector of weighted coefficients for the
       residual bandwidth, end-to-end delay, and network
       hops $\mathbf{W} = [w_b, w_d, w_h]$; individual number $N_{ind}$;
       iteration number $N_{ite}$; retention rate $r_{re}$; cross rate
       $r_{cr}$; mutation rate $r_{mr}$.
Output: the multicast tree $\mathbf{V}_{end}$.
1: for ($i = 0$, i++, i < $N_{ind}$) do
2:   construct $\mathbf{V}_i$ using equation (20);
3:   add $\mathbf{V}_i$ to pool;
4: end for
5: while (j < $N_{ite}$) do
6:   j = j + 1;
7:     calculate the fitness value of each $\mathbf{V}_i$ in pool using
       equation (5);
8:   select $\lfloor N^*_{ind} r_{re} \rfloor$ individuals in pool using
     equation (22) and add them to s_pool;
9:   while(len(s_pool) < $N_{ind}$) do
10:    construct a random float value $v_{f1}$ from 0 to 1;
11:    if $v_{f1} < r_{cr}$ do
12:      construct a new individual $\mathbf{V}_c$ by equation (23);
13:    else do
14:      select an individual $\mathbf{V}_c$ from pool according to its
         fitness value (an individual with a larger fitness
         value has a higher probability of being selected);
15:    end if
15:    construct a random float value $v_{f2}$ from 0 to 1;
16:    if $v_{f2} < r_{mr}$ do
17:      mutate $\mathbf{V}_c$ to $\mathbf{V}^*_c$ using equation (24);
18:      add $\mathbf{V}^*_c$ to s_pool;
19:    else do
20:      add $\mathbf{V}_c$ to s_pool;
21:    end if
22:  end while
22:  pool = s_pool;
23:  delete s_pool.
24: end while
25: calculate the fitness value of each individual in pool
    using equation (5);
26: select the individual $\mathbf{V}_{end}$ that has the largest fitness
    value.
27: return $\mathbf{V}_{end}$
```

the mutation operation is used for the child individual with a certain probability. Supposing there is a child individual $\mathbf{V}_c = [p_1, p_2, \ldots, p_s]$, the mutated child individual $\mathbf{V}^*_c$ is shown in equation (24).

$$\mathbf{V}^*_c = [p_1, \ldots, p_{\lambda^*}, \ldots, p_s] \tag{24}$$

where p_{λ^*} is a random path selected from set_λ, which contains all possible paths for the λ-th unicast task and is obtained by the UPS module. p_{λ^*} is used to replace the original p_λ in child individual $\mathbf{V}_c$ to generate the mutated child-individual $\mathbf{V}^*_c$. The parameter λ^* is such that $1 \leq \lambda^* \leq s$.

Algorithm 1 MDGA

```
Input: a multicast request from source node $n_{src}$ to
    destination node set $\{n_{dst-1}, n_{dst-2}, \ldots, n_{dst-s}\}$;
    the delay requirement $d_{\varphi b}$ of the multicast request;
    the network information $P$, $B_p$, $D_P$, and $H_P$; the
    threshold value $k$, $\mathbf{W} = [w_b, w_d, w_h]$; the
    thresholds in the GA-based algorithm: individual
    number $N_{ind}$, iteration number $N_{ite}$, retention rate
    $r_{re}$, cross rate $r_{cr}$, and mutation rate $r_{mr}$.
Output: the multicast tree $\mathbf{V}_{end}$.
1: for $n_{dst-i}$ in $n_{dst-1}, n_{dst-2}, \ldots, n_{dst-s}$ do
2:     calculate the $k$-best unicast path set i_$kb$_set from
       $n_{src}$ to $n_{dst-i}$ using the UPS Function;
3: end for
4: $s$ $kb$_sets = 1_$kb$_set + 2_ $kb$_set +...+$s$_ $kb$_set
5: calculate the multicast tree $\mathbf{V}_{end}$ by using the
   MPS Function
6: return $\mathbf{V}_{end}$
```

The pseudocode of the function of the MPS module is shown in Function 2.

With the UPS and MPS functions, we give the pseudocode of the proposed MDGA method as Algorithm 1.

The proposed MDGA algorithm is a genetic-algorithm-based online optimization method for finding the appropriate multicast tree for each multicast request. To improve the real-time performance of the algorithm, MDGA employs the UPS module to reduce the search space before performing the GA search. In the UPS module, the multiple-attribute decision-making method is used to ensure that the better solutions have a larger probability of being retained. Additionally, the crossover function in equation (23) and the mutation function in equation (24) are designed to maintain the diversity of the individuals and prevent the algorithm from falling into a local optimum, respectively. The experimental performance in section V shows the effectiveness of the proposed MDGA algorithm.

V. IMPLEMENTATION AND EVALUATION

The performance of the proposed multicast flow scheduling method is evaluated in this section. We implement our MDGA method in Ryu [36], which is an open-source SDN controller that supports the OpenFlow protocol. Mininet [37] is used to simulate a CESN topology to perform the experiments. Iperf [38] is used to generate the network flows that are introduced in section III-A to simulate a real collaborative edge storage environment. The entire experimental program is deployed on an Ubuntu 16.04 system on a Sugon A840r-G server. This server has 64*2.1 GHz AMD processors and 128 GB of memory.

In terms of the experimental topology, we use Mininet 2.3.0 to simulate a topology based on Equinix's datacenters [31] in New York city. The topology is shown in Fig. 2. The bandwidth capacity of each link in the experimental topology is set to 200 Mbps because the simulation experiment assumes limited resources. The link propagation delay is set as 5 us/km. The geographical distance between each pair of ESDCs is obtained by Google map.

TABLE 2. The statistical information of the different types of flows in the storage system.

Flow type	Priority	Speed (Mbps)	Duration (s)	Packet number
heartbeat flows	high	92.87	0.006554	54
user data flows	middle	12.93	39.31	67073
migration flows	low	4.36	654.12	340354

In terms of the flow traffic pattern, we simulate ESDCs sending flows to other ESDCs according to the step (i) pattern, which is similar to previous work [25]. The step (i) pattern means that an ESDC with index x sends flows to an ESDC with index (x+i) MOD (14), where 14 is the number of ESDCs. The statistical information of the different types of flows in the storage system is based on the measurements in our previous work [32] and is shown in Table 2.

During the simulation test, we assume that there are five storage nodes in each ESDC exchanging data with other ESDCs at one multicast request. This means that in the simulation experiment, the sending speed of each type of flow is five times that of the corresponding speed in Table 2. The speed of heartbeat flows is set to 1 Mbps to reduce the packet loss rate in the simulation environment. Additionally, limited by the test duration of the simulation experiment, the duration time of the migration flows is set to 80 seconds. To further evaluate the performance of the flow scheduling method under different network loads, three kinds of flow load scenarios are set in the experiment, as follows:

- Low-load (LL) scenario: 10 heartbeat flow requests, 10 user data flow requests and 10 migration flow requests.
- Middle-load (ML) scenario: 20 heartbeat flow requests, 20 user data flow requests and 20 migration flow requests.
- High-load (HL) scenario: 30 heartbeat flow requests, 30 user data flow requests and 30 migration flow requests.

Each flow request mentioned above is a multicast flow request that has three destination ESDCs. All multicast requests are started periodically within 180 seconds by Iperf. In the simulation experiment, we set $\mathbf{W} = [w_b, w_d, w_h]$ mentioned in equation (19) to [0, 1, 0], [0.3, 0.3, 0.4] and [0.5, 0, 0.5] for heartbeat flows, user data flows and migration flows, respectively. The parameters of the genetic algorithm used in the MPS module are shown in Table 3.

To the best of our knowledge, there has not been a specialized multicast algorithm considering diverse flow requirements in an edge storage datacenter network. Therefore, we compare MDGA with the two multicast algorithms

TABLE 3. The parameters of the genetic algorithm used in the MPS module.

Individual number	Iteration	Retention rate	Cross rate	Mutation rate
40	40	0.2	0.9	0.1

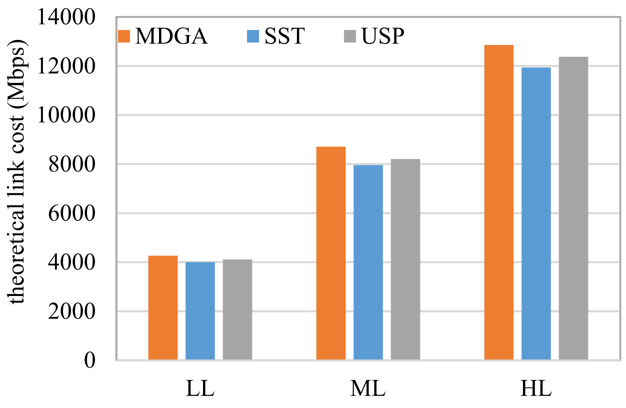


FIGURE 5. The theoretical total link overhead incurred by all the multicast flows in different load scenarios. Note that the theoretical link overhead here is not a measured value but is calculated according to the number of edges of each multicast tree.

mentioned in [28], called unicast shortest path (USP) and static single tree (SST). For USP, the multicast tree is composed of the shortest path of each unicast task. For SST, the multicast tree is the minimum-edge Steiner tree, which uses the minimum possible bandwidth.

A. NETWORK LOAD BALANCING PERFORMANCE

First, we compare the overall link overhead of each method. The theoretical total link overhead incurred by multicast flow φ can be calculated with equation (25).

$$C_\varphi = b_\varphi n_\varphi \tag{25}$$

where b_φ is the flow speed of multicast flow φ and n_φ is the number of edges of the corresponding multicast tree. Each edge of the multicast tree corresponds to a network link. According to equation (25), we give the theoretical total link overhead incurred by all the multicast flows in Fig. 5.

When SST is deployed, the theoretical total link overhead reaches the minimum possible value. This is because SST constructs a minimum-edge Steiner tree for each multicast flow as the multicast tree. Reducing the network overhead is the only objective of SST. However, as mentioned at the end of section III, the major objective of our MDGA method is to improve the flow's QoE performance and the network load balancing performance. A small amount of extra link overhead will be used to achieve these goals. The experimental results in Fig. 5 show that compared to the minimum possible link overhead, MDGA consumes 6.5%, 9.3%, and 8.5% more bandwidth in the LL scenario, ML scenario, and HL scenario, respectively. Compared with USP, MDGA consumes 3.6%,

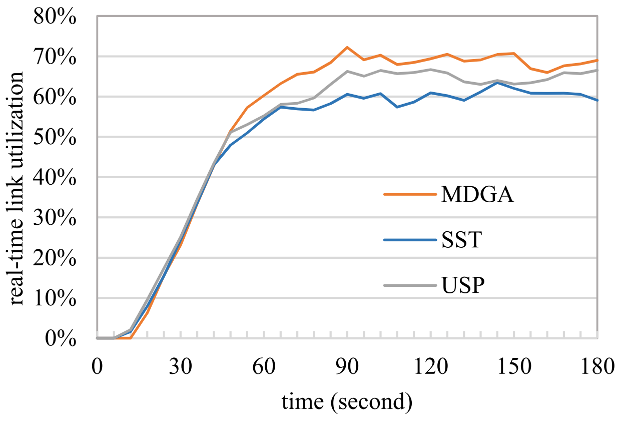


FIGURE 6. The average link utilization in the HL scenario. The link utilization here is measured in real time by the OpenFlow protocol.

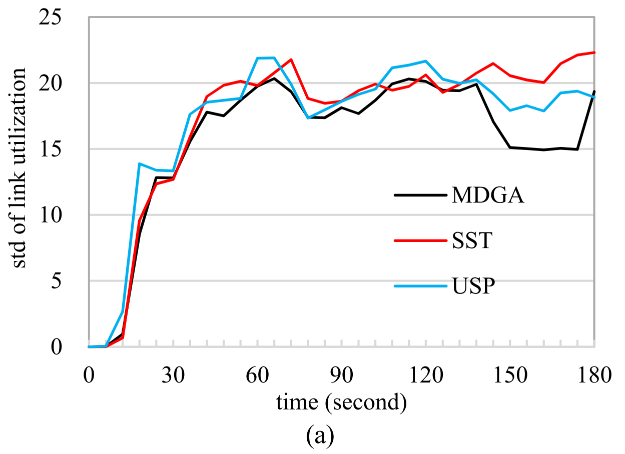
6.1%, and 4.7% more bandwidth in the LL scenario, ML scenario, and HL scenario, respectively.

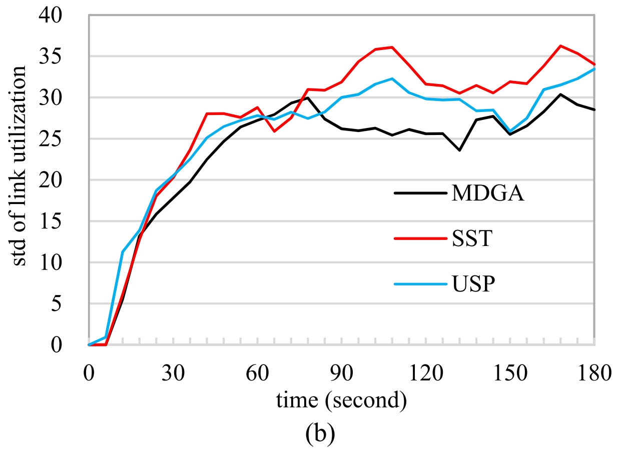
Note that the theoretical link overhead mentioned above is not a measured value but is calculated from the number of edges of each multicast tree. The actual real-time link overhead is shown in Fig. 6.

In Fig. 6, we measure the real-time average link utilization of all links in the HL scenario. The overhead of each link is measured every 6 seconds. We observe that the utilization increases during 0 to 90 seconds. This finding is because the migration flows have a duration time of 80 seconds. The first migration flow is completed in approximately 90 seconds, after which the overall load of the network is relatively stable. During the load stable time from 90 to 180 seconds, we observe that MDGA actually consumes an average of 14.3% and 6% more bandwidth than SST and USP, respectively. This actual extra bandwidth consumption is larger than the corresponding theoretical extra bandwidth consumption, which is 8.5% and 4.7%, respectively. These results indicate that there is network congestion when SST and USP are deployed in the HL scenario. Even in high-load scenarios, the extra network bandwidth used by MDGA will not cause more serious network congestion. MDGA can achieve larger network throughput due to a better load balancing performance.

To compare the load balancing performance of the candidate methods, we measure the standard deviation of link utilization, as shown in Fig. 7.

Specifically, the lower the standard deviation is, the more balanced the link loads. We observe that MDGA achieves the lowest standard deviation in all three scenarios. MDGA achieves 9.6%, 14.7%, and 25.7% lower average standard deviation than SST in the LL, ML, and HL scenarios, respectively. Compared with USP, MDGA achieves 8.4%, 9.2%, and 15.9% lower average standard deviation. These results mean that MDGA can perform better load balancing than other candidate methods. The reason for these results is that MDGA considers the residual bandwidth of paths during path selection. A path with larger residual bandwidth is





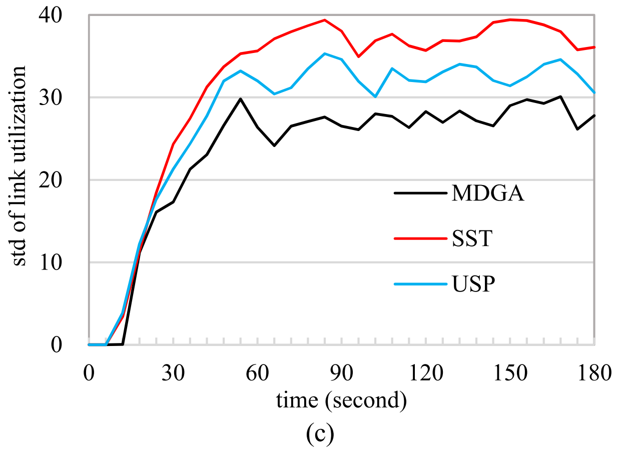
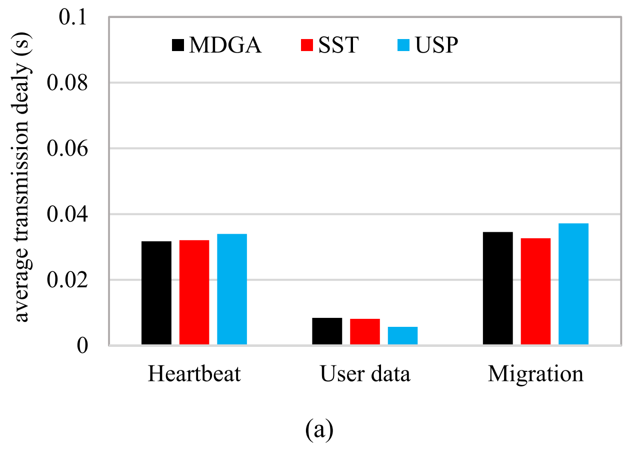
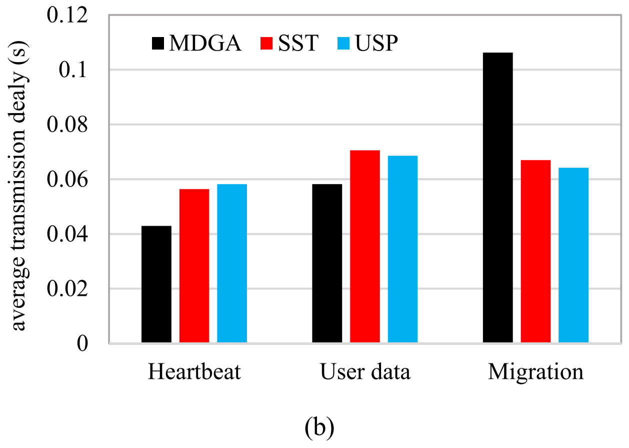


FIGURE 7. Standard deviation of link utilization: (a) measured in the LL scenario; (b) measured in the ML scenario; (c) measured in the HL scenario.





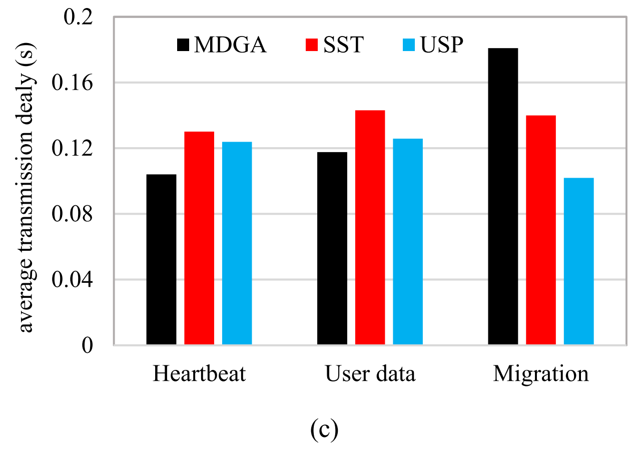


FIGURE 8. Average transmission delay of different types of flows. Heartbeat, user data, and migration in the figures represent the heartbeat flows, user data flows, and migration flows, respectively. (a) Measured in the LL scenario; (b) measured in the ML scenario; (c) measured in the HL scenario.

selected preferentially to improve the network load balancing performance.

B. AVERAGE TRANSMISSION DELAY FOR DIFFERENT TYPES OF FLOWS

The average transmission delay of different types of flows are depicted in Fig. 8. The average transmission delay here represents the average end-to-end transmission time of the packets in the flows. In the LL scenario, the three candidate methods achieve comparable average transmission delay. This is because when the network load is low, the path selection and data forwarding can be effectively handled by the SDN controller and switch, respectively. The user data flows have the lowest transmission delay in the LL scenario. This is because the limited multicast requests of user data flows are routed between closer ESDCs.

However, in the ML and HL scenarios, MDGA shows the advantages of processing high-priority flows. As mentioned in section III-A, the flows with higher priority are more sensitive to the transmission delay. These delay-sensitive flows should have shorter completion time to improve QoE performance. For the heartbeat flows, which have the highest priority, MDGA achieves 23.8% and 20% less average transmission delay than SST in the ML and HL scenarios, respectively. Compared with USP, MDGA achieves 26.1% and 15.4% less average transmission delay for heartbeat flows in the ML and HL scenarios, respectively. For the user data flows, which have the second highest priority, the average transmission delay is 17.5% and 18.1% lower when we use MDGA compared to SST in the ML and HL scenarios, respectively. Compared with USP, MDGA achieves 15.1% and 6.4% less transmission delay for user data flows in the ML and HL scenarios, respectively. These results mean that MDGA can reduce the average transmission delay for high-priority flows and improve the QoE performance of the CESN environment. This is because MDGA considers the flows' priority and network conditions during path selection. A path with lower average transmission delay is selected preferentially for high-priority flows. For migration flows, we observe that the average transmission delay is higher when MDGA is deployed. This is because of the larger computational complexity of the genetic-algorithm-based method in MDGA. However, the average transmission time of these non-delay-sensitive flows has little influence on the QoE performance of the CESN environment.

VI. CONCLUSION

In this paper, we discussed the multicast flow scheduling problem in an edge storage datacenter network. The motivation of this work is to improve the network load balancing and QoE performance. We first proposed a multicast flow scheduling optimization model based on different types of flows with diverse network requirements. Then, we presented MDGA, a multicast flow scheduling method based on multiple-attribute decision-making and a genetic algorithm, to solve the optimization model that finds the appropriate multicast tree for each multicast request. Finally, we conducted extensive experiments to show that MDGA can balance network loads and reduce the average transmission delay for high-priority flows in high-load scenarios.

Moreover, we will consider implementing MDGA in a real edge storage datacenter network testbed in the future. Another direction for future work is to consider issues related to real-time flow classification in a more complex network environment and joint consideration of cloud-edge collaboration and edge-edge collaboration.

REFERENCES

[1] L. Nielsen, R. Vestergaard, N. Yazdani, P. Talasila, D. E. Lucani, and M. Sipos, "Alexandria: A proof-of-concept implementation and evaluation of Generalised data deduplication," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2019, pp. 1–6.

[2] C. Li, Y. Wang, H. Tang, Y. Zhang, Y. Xin, and Y. Luo, "Flexible replica placement for enhancing the availability in edge computing environment," *Comput. Commun.*, vol. 146, pp. 1–14, Oct. 2019.

[3] Y. Zhao, W. Wang, Y. Li, C. C. Meixner, M. Tornatore, and J. Zhang, "Edge computing and networking: A survey on infrastructures and applications," *IEEE Access*, vol. 7, pp. 101213–101230, 2019.

[4] K. Sonbol, Ö. Özkasap, I. Al-Oqily, and M. Aloqaily, "EdgeKV: Decentralized, scalable, and consistent storage for the edge," *J. Parallel Distrib. Comput.*, vol. 144, pp. 28–40, Oct. 2020.

[5] L. Liang, H. He, J. Zhao, C. Liu, Q. Luo, and X. Chu, "An erasure-coded storage system for edge computing," *IEEE Access*, vol. 8, pp. 96271–96283, 2020.

[6] Y. Sahni, J. Cao, and L. Yang, "Data-aware task allocation for achieving low latency in collaborative edge computing," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3512–3524, Apr. 2019.

[7] E. Bocchi, I. Drago, and M. Mellia, "Personal cloud storage benchmarks and comparison," *IEEE Trans. Cloud Comput.*, vol. 5, no. 4, pp. 751–764, Oct. 2017.

[8] J. Jin, Y. Li, and J. Luo, "Cooperative storage by exploiting graph-based data placement algorithm for edge computing environment," *Concurrency Comput., Pract. Exper.*, vol. 30, no. 20, Oct. 2018, Art. no. e4914.

[9] G. Wu, J. Chen, W. Bao, X. Zhu, W. Xiao, and J. Wang, "Towards collaborative storage scheduling using alternating direction method of multipliers for mobile edge cloud," *J. Syst. Softw.*, vol. 134, pp. 29–43, Dec. 2017.

[10] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multi-hop multi-task partial computation offloading in collaborative edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1133–1145, May 2021.

[11] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 54–61, Apr. 2017.

[12] Y. Zhang, Q. Wei, C. Chen, M. Xue, X. Yuan, and C. Wang, "Dynamic scheduling with service curve for QoS guarantee of large-scale cloud storage," *IEEE Trans. Comput.*, vol. 67, no. 4, pp. 457–468, Apr. 2018.

[13] N. Vryzas, N. Tsipas, and C. Dimoulas, "Web radio automation for audio stream management in the era of big data," *Information*, vol. 11, no. 4, p. 205, Apr. 2020.

[14] S. Islam, N. Muslim, and J. W. Atwood, "A survey on multicasting in software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 355–387, 1st Quart., 2018.

[15] Z. AlSaeed, I. Ahmad, and I. Hussain, "Multicasting in software defined networks: A comprehensive survey," *J. Netw. Comput. Appl.*, vol. 104, pp. 61–77, Feb. 2018.

[16] Z. Hu, D. Li, and Z. Li, "Recent advances in datacenter flow scheduling," *J. Comput. Res. Dev.*, vol. 55, no. 9, pp. 1920–1930, 2018.

[17] M. Chiesa, G. Kindler, and M. Schapira, "Traffic engineering with equal-cost-MultiPath: An algorithmic perspective," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 779–792, Apr. 2017.

[18] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-datacenter bulk transfers with netstitcher," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 74–85, Oct. 2011.

[19] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, Aug. 2013, pp. 3–14.

[20] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford, "Optimizing bulk transfers with software-defined optical WAN," in *Proc. ACM SIGCOMM*, Florianopolis, Brazil, 2016, pp. 87–100.

[21] M.-W. Lee, Y.-S. Li, X. Huang, Y.-R. Chen, T.-F. Hou, and C.-H. Hsu, "Robust multipath multicast routing algorithms for videos in software-defined networks," in *Proc. IEEE 22nd Int. Symp. Qual. Service (IWQoS)*, May 2014, pp. 218–227.

[22] Z. Guo, J. Duan, and Y. Yang, "On-line multicast scheduling with bounded congestion in fat-tree data center networks," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 1, pp. 102–115, Jan. 2014.

[23] G. Li, S. Guo, and Y. Yang, "Multicast scheduling algorithm in software defined fat-tree data center networks," in *Proc. IEEE/ACM 25th Int. Symp. Qual. Service (IWQoS)*, Jun. 2017, pp. 127–135.

[24] G. Li, S. Guo, G. Liu, and Y. Yang, "Multicast scheduling with Markov chains in fat-tree data center networks," in *Proc. Int. Conf. Netw., Archit., Storage (NAS)*, Aug. 2017, pp. 188–194.

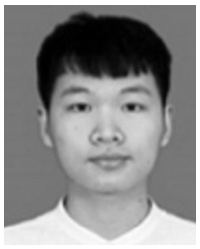
[25] W. Sehery and C. Clancy, "Flow optimization in data centers with clos networks in support of cloud applications," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 4, pp. 847–859, Dec. 2017.

[26] F. K. Hwang and D. S. Richards, "Steiner tree problems," *Networks*, vol. 22, no. 1, pp. 55–89, Jan. 1992.
[27] X. Li, H. Wang, S. Yi, and L. Zhai, "Cost-efficient disaster backup for multiple data centers using capacity-constrained multicast," *Concurrency Comput., Pract. Exper.*, vol. 31, no. 17, Sep. 2019, Art. no. e5266
[28] M. Noormohammadpour, S. Kandula, C. S. Raghavendra, and S. Rao, "Efficient inter-datacenter bulk transfers with mixed completion time objectives," *Comput. Netw.*, vol. 164, Dec. 2019, Art. no. 106903.
[29] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race, "Towards network-wide QoE fairness using openflow-assisted adaptive video streaming," in *Proc. ACM SIGCOMM Workshop Future Hum.-Centric Multimedia Netw.*, Aug. 2013, pp. 15–20.
[30] M. Amiri, H. A. Osman, S. Shirmohammadi, and M. Abdallah, "Toward delay-efficient game-aware data centers for cloud gaming," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 12, no. 5s, pp. 1–19, Dec. 2016.
[31] *New York Metro IBX Data Center Data Sheet*. Accessed: Dec. 31, 2020. [Online]. Available: https://www.equinix.com/resources/data-sheets/nyc-metro-data-sheet/
[32] W. Ke, Y. Wang, and M. Ye, "GRSA: Service-aware flow scheduling for cloud storage datacenter networks," *China Commun.*, vol. 17, no. 6, pp. 164–179, Jun. 2020.
[33] M. Barbehenn, "A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices," *IEEE Trans. Comput.*, vol. 47, no. 2, p. 263, Feb. 1998.
[34] Y. Wang, M. Ye, Q. He, Y. Huan, and W. Kang, "A new node selecting approach in Ceph storage system based on software defined network and multi-attributes decision-making model," *Chin. J. Comput.*, vol. 42, no. 2, pp. 323–338, 2019.
[35] B. Lorenzo and S. Glisic, "Optimal routing and traffic scheduling for multihop cellular networks using genetic algorithm," *IEEE Trans. Mobile Comput.*, vol. 12, no. 11, pp. 2274–2288, Nov. 2013.
[36] *Ryu*. Accessed: Dec. 31, 2020. [Online]. Available: https://github.com/faucetsdn/ryu
[37] *Mininet*. Accessed: Jan. 5, 2021. [Online]. Available: http://mininet.org/
[38] *iPerf*. Accessed: Jan. 5, 2021. [Online]. Available: https://iperf.fr/

WENLONG KE received the M.S. degree from the Guilin University of Electronic Technology, Guilin, China, in 2016, where he is currently pursuing the Ph.D. degree with the School of Information and Communication. His main research interests include cloud storage systems, datacenter networks, and network traffic classification.

YONG WANG received the Ph.D. degree from East China University of Science and Technology, Shanghai, China, in 2005. He is currently a Full Professor and a Ph.D. Supervisor with the Guilin University of Electronic Technology. His research interests include cloud computing, distributed storage systems, and information security.

MIAO YE received the Ph.D. degree from the School of Computer Science and Technology, Xidian University, Xi'an, China, in 2016. He is currently a Full Professor and a Ph.D. Supervisor with the Guilin University of Electronic Technology. His research interests include wireless sensor networks, distributed storage systems, deep learning, and optimization methods and their application in engineering.



JUNQI CHEN received the bachelor's degree from the Changshu Institute of Technology, Suzhou, China, in 2019. He is currently pursuing the master's degree with the School of Computer Science and Information Security, Guilin University of Electronic Technology, Guilin, China. His main research interests include distributed storage systems and software defined networking.