# Cost-Effective App Data Distribution in Edge Computing

Xiaoyu Xia, Feifei Chen, Qiang He, *Member, IEEE*, John C. Grundy, *Senior Member, IEEE*, Mohamed Abdelrazek, and Hai Jin, *Fellow, IEEE*

**Abstract**—Edge computing, as an extension of cloud computing, distributes computing and storage resources from centralized cloud to distributed edge servers, to power a variety of applications demanding low latency, e.g., IoT services, virtual reality, real-time navigation, etc. From an app vendor's perspective, app data needs to be transferred from the cloud to specific edge servers in an area to serve the app users in the area. However, according to the pay-as-you-go business model, distributing a large amount of data from the cloud to edge servers can be expensive. The optimal data distribution strategy must minimize the cost incurred, which includes two major components, the cost of data transmission between the cloud to edge servers and the cost of data transmission between edge servers. In the meantime, the delay constraint must be fulfilled - the data distribution must not take too long. In this article, we make the first attempt to formulate this Edge Data Distribution (EDD) problem as a constrained optimization problem from the app vendor's perspective and prove its $\mathcal{NP}$-hardness. We propose an optimal approach named EDD-IP to solve this problem exactly with the Integer Programming technique. Then, we propose an $O(k)$-approximation algorithm named EDD-A for finding approximate solutions to large-scale EDD problems efficiently. EDD-IP and EDD-A are evaluated on a real-world dataset and the results demonstrate that they significantly outperform three representative approaches.

**Index Terms**—Edge computing, optimization, data distribution, cost-effectiveness, edge server network

## 1 Introduction

The world has witnessed exponentially growing mobile data traffic in this decade promoted by a huge increase in mobile devices and Internet of Things (IoT) connected devices [1]. This explosion of mobile data traffic has led to a wealth of research aiming to relieve the enormous data transmission loads on networks. Conventional network paradigms facilitated by cloud computing, including content delivery networks, content-centric networks and information centric networks, cannot handle the increases in network latency and network congestion caused by the rapidly increasing mobile traffic. In recent years, edge computing has emerged as a new computing paradigm that push computing and storage resources to the edge of the cloud [2]. These edge servers, each powered by one or many physical machines, are deployed at base stations or access points that are geographically close to devices [3]. Vendors of mobile and IoT applications (referred to as *app vendors* together hereafter) can hire computing and storage resources on edge servers for hosting their apps to serve their own app users with low latency [4]. Offloading computation tasks from app users' end-devices to nearby edge servers can ease the computation burden and energy consumption on those resource-limited end-devices [5], [6], [7], [8]. This is also a key technology of the 5G mobile network [9].

As edge servers become the entry points to the Internet for a larger number of mobile and IoT devices, a much larger proportion of the rapidly increasing mobile traffic data will be transmitted through those edge servers from the cloud. From an app vendor's perspective, caching app data on edge servers can considerably reduce the latency for their own users' data retrieval. In addition, it will largely reduce the volume of their app data transmitted between the cloud and its app users. This in turn will decrease the corresponding data transmission costs [10]. The new challenges raised by data caching in the edge computing environment have attracted many researchers' attention in recent years [11], [12], [13], [14], [15].

However, existing research efforts have focused on how to cache data across edge servers to achieve different optimization objectives, e.g., to minimize caching cost [14], to minimize retrieval latency [15], to guarantee the quality of transmissions [12], etc. The fact that data transmission from within the cloud to distributed edge servers may incur excessive costs is largely ignored. For example, Amazon Web Services charges up to US$0.09 + US$0.02 to transfer 1 GB data out of its S3 data storage facilities to the internet.[1] It


- *Xiaoyu Xia, Feifei Chen, and Mohamed Abdelrazek are with the School of Information Technology, Deakin University, Geelong, Victoria 3217, Australia. E-mail: {xiaoyu.xia, feifei.chen, mohamed.abdelrazek}@deakin.edu.au.*
- *Qiang He is with the School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Victoria 3122, Australia. E-mail: qhe@swin.edu.au.*
- *John Grundy is with the Faculty of Information Technology, Monash University, Melbourne, Victoria 3800, Australia. E-mail: john.grundy@monash.edu.*
- *Hai Jin is with Services Computing Technology and System Lab, Big Data Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technolgoy, HuaZhong University of Science and Technology, Wuhan 430074, China. E-mail: hjin@hust.edu.cn.*


*Manuscript received 28 Jan. 2020; revised 22 June 2020; accepted 16 July 2020. Date of publication 21 July 2020; date of current version 31 July 2020. (Corresponding author: Qiang He.) Recommended for acceptance by D. Medhi. Digital Object Identifier no. 10.1109/TPDS.2020.3010521*

1. https://aws.amazon.com/s3/pricing/

is a significant component in the cost structure for app vendors to consider in the edge computing environment, similar to a large body of work on cloud computing [16], [17]. Unlike data transmission in cloud computing and wireless sensor networks [18], [19], [20], [21], app data distribution in the edge computing environment consists of two major components: 1) data transmission from the cloud to edge servers; 2) and data transmission between edge servers. Both components must be considered in a systematic manner to formulate cost-effective data distribution strategies for app vendors. We refer to this problem as the edge data distribution (EDD) problem in this paper.

The scale of an EDD scenario can be very large and finding a solution is not trivial. To help app vendors formulate cost-effective EDD strategies that minimize the EDD cost while fulfilling the app vendor's EDD time constraint, this paper makes the first attempt to study the EDD problem from the app vendor's perspective. The key contributions of this paper are as follows:

- We formulate and model the EDD problem as a constrained optimization problem (COP) from the app vendor's perspective and prove that it is $\mathcal{NP}$-hard.
- We develop an optimal approach, namely EDD-IP, for finding optimal solutions to EDD problems with the Integer Programming technique.
- We develop an approximation approach named EDD-A for finding approximate solutions to large-scale EDD problems rapidly.
- We conduct extensive experiments on a widely-used real-world dataset to evaluate the proposed approaches against three representative approaches.

The rest of this paper is organized as follows. Section 2 motivates this research with an example. Section 3 formulates the EDD problem and proves its $\mathcal{NP}$-hardness. Section 4 presents and analyzes our optimal approach and approximation approach for solving the EDD problem. Section 5 evaluates the proposed approaches experimentally. Section 6 reviews the related work. Section 7 concludes this paper and points out the future work.

## 2 Motivating Example

Facebook Horizon[2] is a representative application that can significantly benefit from caching their data on edge servers. Facebook users wearing Oculus headsets can access VR videos and VR games on Facebook Horizon. Caching popular VR videos on edge servers will allow Facebook users covered by those edge servers to access the videos with minimum latency, which is critical because VR users are highly latency-sensitive. It will also reduce the data traffic between the Facebook Horizon server in the cloud and Facebook Horizon users. However, assuming a similar price for data transmission asked by Amazon, cost-ineffective data distribution strategies may cost Facebook Horizon significantly just to distribute VR videos to edge servers. As the number of Facebook Horizon users continues to grow, such extra expense will increase rapidly.

2. https://www.oculus.com/facebookhorizon

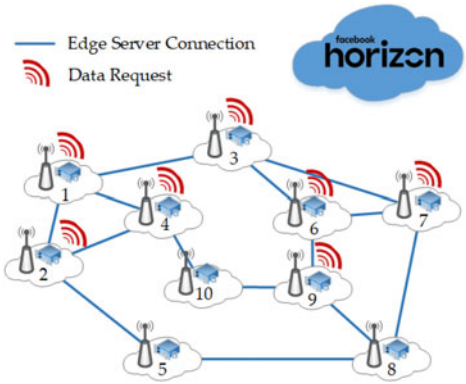



Fig. 1. An example EDD scenario.

Fig. 1 presents an example EDD scenario with 10 edge servers in a specific geographic area. Those edge servers are connected via high-speed links to facilitate data transmissions between them [7], [22]. Let us assume that a Facebook Horizon VR video is to be cached on 7 of those edge servers.[3] There are many possible strategies for distributing the VR video onto those 7 edge servers. A straightforward EDD strategy is to transmit this VR video from the cloud to each individual edge server directly. We refer to such data transmissions as cloud to edge server (C2E) transmissions hereafter. Alternatively, the VR video can first be transmitted from the cloud onto one of the edge servers, which then transmits the VR video onto other edge servers via the high-speed links between them. The data transmissions between edge servers are referred to as edge server to edge server (E2E) transmissions hereafter. A third possible EDD strategy is similar to the second one, but the VR video is first transmitted from the cloud to several of the edge servers instead of one. Given the same amount of data to transmit, E2E transmissions cost less than C2E transmissions because of the much shorter distance between adjacent edge servers and the zero burden caused by E2E transmissions on the internet backbone [23]. Therefore, different EDD strategies incur different costs.

From Facebook's perspective, it is critical to formulate a cost-effective data distribution strategy that minimizes the data distribution cost. While pursing low data distribution cost, the time taken to distribute the VR video onto all the 7 edge servers must also be considered. As discussed above, low latency is one of the major objectives of edge computing [24]. Thus, an EDD strategy must also fulfill Facebook Horizon's time constraint.[4]

## 3 Problem Formulation

In this section, we first formulate the EDD problem as a constrained optimization problem, then prove the $\mathcal{NP}$-hardness of this problem based on the Steiner Tree problem. The notations used in this paper are summarized in Table 1.

3. When different VR videos are to be cached, their distribution processes are not correlated in terms of transmission cost and transmission delay. Thus, their corresponding EDD strategies are formulated individually.

4. Please note that the time constraint for EDD varies from application to application.

TABLE 1
Summary of Notations

| Notation | Description |
|---|---|
| $c$ | cloud node |
| $d_{limit}$ | delay limit defined by app vendor |
| $d_v$ | delay that edge server $v$ obtain data after data arrives the edge server network |
| $\mathcal{D}_v$ | depth of $v$, equal to $d_v + 1$ |
| $\mathcal{D}_{limit}$ | depth limit, equal to $d_{limit} + 1$ |
| $E$ | set of links between edge servers |
| $e_{u,v}$ | edge/link between edge server $u$ and $v$ |
| $G$ | graph presenting a particular area |
| $\mathcal{H}$ | latency limit defined by app vendor |
| $\mathcal{R}$ | set of destination edge servers |
| $\gamma$ | the ratio of data transmission cost of $C2E$ and a 1-hop transmission cost of $E2E$ |
| $S$ | set of binary variables indicating the selection of the initial transit edge servers |
| $s_v$ | binary variable indicating whether $v$ is an initial transit edge server |
| $\mathcal{T}$ | set of binary variables indicating the data distribution path |
| $T_{ce}$ | tree with root $c$ with edges $e_{c,v}, \forall v \in \{T_{ms} - c\}$ |
| $T_{edd-a}$ | tree returned by Algorithm 2 |
| $T_{ms}$ | tree returned by Algorithm 1 |
| $\mathcal{T}_{u,v}$ | binary variable indicating data transmitted from $u$ to $v$ |
| $V$ | set of edge servers |
| $v$ | edge server $v$ |
| $u$ | edge server $u$ |

### 3.1 Problem Statement

Edge computing is significantly different from cloud computing which facilitates content-centric network and content delivery network. In an edge computing environment, adjacent edge servers deployed at different base stations and access points can communicate with their neighbor edge servers and share their storage resources via high-speed links [7], [22]. Thus, the edge servers in a particular area constitute an *edge server network*, which can be modeled as a graph where a node represents an edge server and an edge represents the link between two edge servers.

In this research, the $n$ edge servers in a particular area are modeled as a graph $G$. For each edge server $v$, graph $G$ has a corresponding node. For each pair of linked edge servers ($u$, $v$), graph $G$ has a corresponding edge $e_{u,v}$. We use $G(V, E)$ to represent the graph, where $V$ is the set of nodes in $G$ and $E$ is the set of edges in $G$. *In the remainder of this paper, we will speak inter-changeably of an edge server and its corresponding node in graph $G$, denoted as $v, \forall v \in V$.* Let $\mathcal{R}$ denote the set of *destination edge servers* in graph $G$, i.e., the edge servers to which the data are to be transmitted from the cloud.

**Example 1.** As discussed above, the edge server network in Fig. 1 can be modeled as the graph $G$ presented in Fig. 2, while the destination edge servers are presented as black nodes and others are white nodes. In graph $G$, there are 10 edge servers with 14 edges in total, where $\mathcal{R} = \{1, 2, 3, 4, 6, 7, 9\}$.

The fees for data transmissions charged by different cloud service providers, e.g., Amazon and Google, are different. Amazon even has different pricing models in its different regions. Similarly, edge infrastructure providers usually have different pricing models for their E2C transmissions and E2E transmissions. Thus, we use a ratio $\gamma$ to indicate the difference between the C2E transmission cost and the E2E transmission cost generically. For example, $\gamma = 20$ indicates that a C2E transmission costs 20 times as much to transmit a data than an E2E transmission - C2E transmissions are usually more expensive than E2E transmissions as discussed in Section 2. In addition, the data transmission latency between two edge servers is measured by the number of hops between them in graph $G$. Thus, the C2E transmission cost can be converted to $\gamma$ times of the 1-hop E2E transmission cost. This way, the optimization objective and the corresponding constraints in the EDD problem can be modeled in a more generic manner. Specific pricing models and network latency models can be easily integrated to calculate the actual EDD cost and EDD time consumption, i.e., the total cost and time of transmitting the data from the cloud to the destination edge servers, respectively.

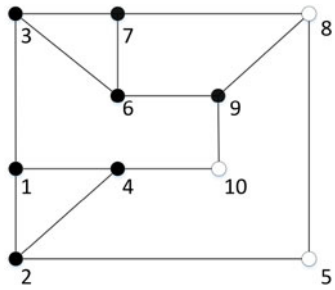


Fig. 2. Graph structure of EDD scenario in Fig. 1.

There are two possible phases of EDD. 1) The $C2E$ transmission, where the data is transmitted from the cloud to one or many of the edge servers in the area, which are referred to as the "initial transit edge servers" hereafter. 2) The $E2E$ transmission, where the data is transmitted from the initial transit edge servers to the destination edge servers via other transit edge servers. Please note that an initial transit edge server is not necessarily a destination edge server. A destination edge server may also be a transit edge server because it may transmit the data further to other destination edge servers. Accordingly, an EDD strategy consists of two parts, a C2E strategy and an E2E strategy. A C2E strategy specifies the initial transit edge servers. It is denoted as a vector $S = < s_1, \ldots, s_n >$, where $s_v$ $(1 \leq v \leq n)$ indicates whether edge server $v$ is selected as an initial transit edge server to receive the data from the cloud directly

$$s_v = \begin{cases} 0 & \text{if } v \text{ is selected as an initial transit edge server} \\ 1 & \text{if } v \text{ is NOT selected as an initial transit edge server} \end{cases}. \tag{1}$$

An E2E strategy is also represented by a vector $\mathcal{T}_{E2E} = < \mathcal{T}_{1,1}, \mathcal{T}_{1,2}, \ldots, \mathcal{T}_{n,n} >$, where $\mathcal{T}_{u,v}(u, v \in V)$ denotes whether the data is transmitted through edge $e_{u,v}$ in $G$

$$\mathcal{T}_{u,v} = \begin{cases} 1 & \text{if data is transmitted through edge } e_{u,v} \\ 0 & \text{if data is NOT transmitted through edge } e_{u,v} \end{cases}. \tag{2}$$

Since a valid EDD strategy must connect each destination edge server $v \in R$ to an initial edge server in $S$ through $\mathcal{T}_{E2E}$, constraint (3) must be fulfilled

$$isConnected(v, S, \mathcal{T}_{E2E}) = true, \forall v \in R. \tag{3}$$

The details to fulfil constraint (3) will be discussed later in Theorem 3.

As discussed above, the EDD time constraint is determined by the app vendor - i.e., application-specific. As discussed in Section 1, we formulate the EDD problem in a generic manner. The data transmission latency between two edge servers is measured by the number of hops between them in $G$. Let $d_{limit}$ denote the app vendor's EDD time constraint. Please note that the EDD time constraint here does not include the C2E latency because it is ensured by the edge infrastructure provider and does not impact the formulation of the EDD strategy - it can never be avoided or reduced by an EDD strategy. Thus, each destination edge server $v$'s *E2E latency*, i.e., the data transmission latency between $v$ and its connected initial edge server in $S$ must not exceed this constraint

$$0 \leq d_v \leq d_{limit}, d_v \in Z^+, \forall v \in \mathcal{R}. \tag{4}$$

**Example 2.** Take Fig. 2 as an example. Let us assume that the app vendor's EDD time constraint is $d_{limit} = 2$. This means that it must not take more than two hops for a destination edge server to receive the data from an initial transit edge server. In Fig. 2, if node 3 is the only edge server selected as the initial transit edge server, one possible E2E strategy is to select edges $\{e_{3,1}, e_{3,6}, e_{3,7}, e_{1,2}, e_{1,4}, e_{6,9}\}$. This means that in $\mathcal{T}_{E2E}$ there is $\mathcal{T}_{3,1} = \mathcal{T}_{3,6} = \mathcal{T}_{3,7} = \mathcal{T}_{1,2} = \mathcal{T}_{1,4} = \mathcal{T}_{6,9} = 1$. Accordingly, we can obtain the 7 destination edge servers' E2E latency: $d_3 = 0$, $d_1 = d_6 = d_7 = 1$, $d_2 = d_4 = d_9 = 2$.

Given an EDD time constraint $d_{limit}$, the app vendor's optimization objective is to minimize the EDD cost, which consists of the part incurred by the C2E transmission(s) and the E2E transmissions

$$\text{minimize } Cost_{C2E}(S) + Cost_{E2E}(\mathcal{T}_{E2E}), \tag{5}$$

while fulfilling the EDD time constraint (4).

### 3.2 Problem Hardness

Now we prove that the EDD problem is $\mathcal{NP}$-hard by proving Theorems 1 and 2.

**Theorem 1.** *The EDD problem is $\mathcal{NP}$.*

**Proof.** As there are no more than $(|V| + |E| + 2|R|)$ constraints in total, any solution to this EDD problem can be validated in polynomial time by checking whether the solution satisfies the constraints (1), (2), (3) and (4). Thus, the EDD problem is $\mathcal{NP}$. □

**Theorem 2.** *The EDD problem is $\mathcal{NP}$-hard.*

**Proof.** To prove the $\mathcal{NP}$-hardness of the EDD problem, we first introduce the classic Rooted Minimum Steiner Tree (RMST) problem. The RMST problem is well-known to be $\mathcal{NP}$-hard [25], [26], and can be defined as follows. Given a graph $G = (V, E)$, a set of destination nodes $\mathcal{N}$ in $G$, and a root node $root$ of the Steiner tree $ST$. For each edge $e \in E$, there is a variable $\mathcal{Y}_e$ to indicate whether it is in $ST$ ($\mathcal{Y}_e = 1$) or not ($\mathcal{Y}_e = 0$). Moreover, each edge $e$ has its own weight $\mathcal{W}_e$. Function $path(n, root, ST)$ is used to obtain the possible path from $root$ to the node $n$ through the edges in $E$. If node $n$ is the root of $ST$ or not in $ST$, there should not exist a path from root to $n$, which means $path(n, root, ST) = null$. The formulation is presented below

$$object: \min \sum \mathcal{Y}_e \cdot \mathcal{W}_e \tag{6a}$$

$$s.t.: \ \mathcal{Y}_e \in \{0, 1\} \tag{6b}$$

$$path(n, root, ST) \neq null, \forall n \in \mathcal{N}. \tag{6c}$$

Now we prove that the RMST problem can be reduced to an instance of the EDD problem. The reduction can be done as follows: 1) add the cloud server as a node $r$ into $G$ to obtain a new graph $G'$; 2) add the edges from node $r$ to every other node in $G'$; 3) relax the EDD time constraint $d_{limit}$ to $|V|$. Given any instance $RMST(G, root, R, \mathcal{W})$, we can correspondingly construct $EDD(G', cloud, R, Cost_{C2E}, Cost_{E2E})$ with the reduction above in polynomial time where $|G| = |G'|$, while $Cost_{C2E}$ and $Cost_{E2E}$ can be treated as the weights of the edges. By the reduction, the constraint (4) can be relaxed properly. As the constraint (6b) in the RMST problem only considers the edge variables, we can convert constraint (1) to $S_v = \mathcal{T}_{c,v} \in \{0, 1\}$. By combining this with constraint (2), constraint (6b) is fulfilled. Additionally, both constraints (3) and (6c) ensure that all the nodes in $R$ are connected to $root$. Moreover, objective (6a) of RMST is to obtain the minimum total weight of $ST$, which can be projected to (5) by mapping the $Cost_{C2E}$ and $Cost_{E2E}$ to the weights of edges.

In conclusion, any solution $\mathcal{Y}$ always satisfies the RMST problem if $\mathcal{Y}$ satisfies the reduced EDD problem. Therefore, the EDD problem is reducible from the RMST problem and it is thus $\mathcal{NP}$-hard. □

## 4 Edge Data Distribution Strategy Formulation

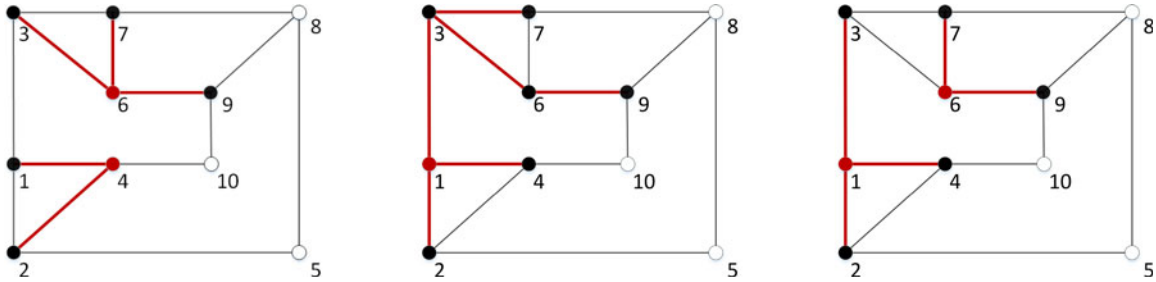
We first present an optimization approach, namely EDD-IP, to exactly solve the EDD problem formulated in the previous section. Then, we introduce an $O(k)$-approximate approach named EDD-A to solve large-scale EDD problems approximately, followed by a theoretical analysis of its performance.

### 4.1 Optimization Approach

The solution to the EDD problem must minimize the EDD cost while fulfilling the app vendor's EDD time constraint $\mathcal{D}_{limit}$. Thus, the EDD problem can be modeled as a constrained optimization problem.

Following the methodology of the proof of Theorem 3, we use similar techniques to convert the EDD problem to an integer program model that can be solved by integer programming solvers, such as IBM CPLEX Optimizer[5] and

5. https://www.ibm.com/analytics/cplex-optimizer




(a) EDD-IP solution (b) CMST solution (c) EDD-A solution

Fig. 3. EDD solutions.

Gurobi.[6] Accordingly, this optimal approach is named EDD-IP.

First, we add the cloud $c$ into $V$, then add the edges from $c$ to each edge server $v \in V \setminus c$ in graph $G$. Then, the C2E strategy $S =< s_1, \ldots, s_n >$ can be formulated by selecting edges in graph $G$, i.e., $\mathcal{T}_c =< \mathcal{T}_{c,1}, \mathcal{T}_{c,2}, \ldots, \mathcal{T}_{c,n} >$, where $\mathcal{T}_{c,v}$ $(v \in V \setminus c)$ denotes whether the data is transmitted from the cloud $c$ to $v$ through the new edge $e_{c,v}$. Here, we combine the C2E strategy $S$ and the E2E strategy $\mathcal{T}_{E2E}$ as the data distribution strategy $\mathcal{T} = < \mathcal{T}_{c,1}, \ldots, \mathcal{T}_{c,n}, \mathcal{T}_{1,1}, \ldots, \mathcal{T}_{n,n} >$, where variable $\mathcal{T}_{u,v} \in \{0,1\}$ $(u \in \{c\} \cup [1,n], v \in [1,n])$ indicates whether edge $e_{u,v}$ is included in $\mathcal{T}$. Second, we implement the definition of depth in the Steiner tree to represent the order of transmitting the data. Denote $\mathcal{D}_v$ as the depth of edge server $v$, where $\mathcal{D}_v = d_v + 1$ and $\mathcal{D}_c = 0$. Then, the EDD time constraint $d_{limit}$ can be defined as a depth limit $\mathcal{D}_{limit} = d_{limit} + 1$.

After this, we define the variable for each edge server $v$

$$I_v = \begin{cases} 0 & \text{if } v \text{ is visited during the EDD process} \\ 1 & \text{if } v \text{ is NOT visited during the EDD process} \end{cases}. \tag{7}$$

Now, the COP model for the EDD problem is formally expressed as follows:

$$\min \gamma \sum_{v \in V \setminus c} \mathcal{T}_{c,v} + \sum\sum_{v,u \in V \setminus c} \mathcal{T}_{u,v} \tag{8}$$

$$I_v = 1, \forall v \in R \tag{9}$$

$$\sum_{u \in V} \mathcal{T}_{u,v} = I_v, \forall u, v \in V \setminus c \tag{10}$$

$$\mathcal{T}_{u,v} \leq I_u \cdot I_v, \forall u, v \in V \tag{11}$$

$$I_v, \mathcal{T}_{u,v} \in \{0,1\}, \forall u, v \in V \tag{12}$$

$$\mathcal{D}_c = 0 \tag{13}$$

$$\mathcal{D}_c < \mathcal{D}_v \leq \mathcal{D}_{limit}, \forall v \in V \setminus c \tag{14}$$

$$\mathcal{D}_v - \mathcal{D}_u = 1, \forall u, v \in V, \mathcal{T}_{u,v} = 1, \tag{15}$$

where $\gamma$ is the ratio that generically indicates the ratio of the C2E transmission unit cost over the E2E transmission unit cost, as introduced in Section 3.1.

**Example 3.** Fig. 3a shows the EDD strategy formulated by EDD-IP with EDD time constraint $d_{limit} = 1$. The C2E strategy specifies that nodes 6 and 4 are selected as the initial transit edge server. They receive the data from the cloud and then transmit it to all other destination edge servers. The EDD strategy selects edges $\{e_{c,4}, e_{c,6}, e_{6,3}, e_{6,7}, e_{6,9}, e_{4,1}, e_{4,2}\}$ for data transmissions. The total cost is $2\gamma + 6$ times of a 1-hop E2E transmission cost.

**Theorem 3.** *EDD-IP computes an optimal solution to the EDD problem.*

**Proof.** Let $\mathcal{C}_{C2E}$ and $\mathcal{C}_{E2E}$ denote the unit costs of C2E data transmission and E2E data transmission, respectively. This way, objective (5) can be converted to $\mathcal{C}_{C2E} \cdot \sum_{v \in V \setminus c} \mathcal{T}_{c,v} + \mathcal{C}_{E2E} \cdot \sum\sum_{u,v \in V \setminus c} \mathcal{T}_{u,v}$, with $Cost_{C2E}(S)$ calculated by $\mathcal{C}_{C2E} \cdot \sum_{v \in V \setminus c} \mathcal{T}_{c,v}$ and $Cost_{E2E}(\mathcal{T}_{E2E})$ calculated by $\mathcal{C}_{E2E} \cdot \sum\sum_{u,v \in V \setminus c} \mathcal{T}_{u,v}$. Denote $\gamma = \frac{\mathcal{C}_{C2E}}{\mathcal{C}_{E2E}}$, objective (5) can be presented as $\mathcal{C}_{E2E} \cdot (\gamma \sum_{v \in V \setminus c} \mathcal{T}_{c,v} + \sum\sum_{u,v \in V \setminus c} \mathcal{T}_{u,v})$. Since $\mathcal{C}_{E2E}$ is a constant, objective (5) can be transformed to objective (8) by removing $\mathcal{C}_{E2E}$.

There are two situations in Eq. (10): 1) if $v$ is not visited, any edge pointing to $v$ will not be selected in the C2E strategy; 2) if $v$ is visited, there must be exactly selected one edge that points to $v$. Constraints (9) and (10) ensure that all the destination edge servers in $R$ are visited during the EDD process. Constraint (11) ensures that, if edge $e_{u,v}$ is selected, both edge servers $u$ and $v$ must be selected. Otherwise, $e_{u,v}$ can never be included into the E2E strategy.

Thus, constraints (9), (10), (11) and (12) collectively ensure that every destination edge server $v \in R$ is connected to the cloud server in $G$, fulfilling (3) in Section 3.1.

Constraint (13) makes sure that the EDD process always starts from the cloud. In addition, constraint (14) ensures that the EDD time constraint is fulfilled, while constraint (15) guarantee the the depth of $u$ is always one less than that of $v$, if edge $\{u, v\}$ exists.

Thus, EDD-IP computes an optimal solution to the EDD problem. □

As discussed in Section 3.1, specific cost models and latency models can be easily integrated to calculate the actual EDD costs and EDD time consumption in real-world

6. http://www.gurobi.com/

EDD scenarios. For example, given a cost function $cost(u, v)$ that represents the transmission cost between $u$ and $v$, the total EDD cost can be calculated by $\sum_{v \in V \setminus c} \mathcal{T}_{c,v} \cdot cost(c, v) + \sum \sum_{u,v \in V \setminus c} \mathcal{T}_{u,v} \cdot cost(u, v)$. Given a specific latency constraint $\mathcal{L}_{limit}$ and a latency function $\mathcal{L}(u, v)$ that represents the latency between $u$ and $v$ via $e_{u,v}$, constraints (13) (14) and (15) can be replaced by (16), (17) and (18) respectively

$$\mathcal{L}_{\mathcal{T}}(c, c) = 0 \tag{16}$$

$$\mathcal{L}(c, v) \leq \mathcal{L}_{\mathcal{T}}(c, v) \leq \mathcal{L}_{limit}, \forall v \in V \setminus c \tag{17}$$

$$\mathcal{L}_{\mathcal{T}}(c, v) - \mathcal{L}_{\mathcal{T}}(c, u) = \mathcal{L}(u, v), \forall u, v \in V, \mathcal{T}_{u,v} = 1, \tag{18}$$

where $\mathcal{L}_{\mathcal{T}}(c, v)$ represents the total transmission latency between $c$ and $v$ via the path from $c$ to $v$ indicated by $\mathcal{T}$.

### 4.2 Approximation Algorithm

As proven in Section 3.2, the EDD problem is $\mathcal{NP}$-hard. Finding the optimal solution is intractable in large-scale EDD scenarios. To address this issue, this section presents an approximation approach, named EDD-A, for finding approximate solutions to large-scale EDD problems efficiently. The approximation ratio of EDD-A is $O(k)$, which means that the ratio of the EDD cost incurred by EDD-A and that incurred by the optimal solution is $O(k)$ in the worst case, where $k$ is a constant.

Similar to the techniques used in Section 4.1, EDD-A is an approach designed based on the concept of Steiner tree by adding the cloud server $c$ and the corresponding edges into $G$. There are two parts in EDD-A: 1) calculating an approximate minimum Steiner tree $T_{ms}$ based on a simple but fast algorithm presented in Algorithm 6 (CMST); 2) splicing and pruning $T_{ms}$ with the latency constraint $\mathcal{H} = \mathcal{D}_{limit}$, based on Algorithm 2.

We first introduce our Connectivity-oriented Minimum Steiner Tree (CMST) algorithm for calculating the approximation to the minimum Steiner tree. CMST is based on the algorithm proposed in [27] which is simple but effective. In our CMST algorithm, it first collects the nodes closest (with the lowest cost) to $T_{ms}$ as a set, then selects the one with the highest connectivity (Lines 3-4 in Algorithm 1).

**Algorithm 1.** CMST Algorithm

1: **Input:** $G(V, E)$, $R$, $c$
2: **Output:** A minimum Steiner Tree $T_{ms}$
3: initialize a tree $T_{ms}$ based on $G \bigcup \{cloud\}$, only consisting of the node $\{cloud\}$;
4: return $T_{ms}$ if all the destination edge servers in $R$ have been added into $T_{ms}$, else go to step ~~3~~; [5]
5: find a set of edge servers $C$ that are closest to $T_{ms}$, while $C \bigcap T_{ms} = \varnothing$
6: find edge server $v$ with the highest connectivity, then add $v$ to $T_{ms}$;
7: go to step ~~2~~. [4]

度最大

**Example 4.** Take Fig. 2 as an example. By applying CMST, edges $\{e_{c,1}, e_{1,2}, e_{1,3}, e_{1,4}, e_{3,6}, e_{3,7}, e_{6,9}\}$ are added into $T_{ms}$, as shown in Fig. 3b. The total cost of $T_{ms}$ is $\gamma + 6$. Let us assume that all E2E transmissions must be finished within one hop ($d_{limit} = 1$) to facilitate the following discussion. This means $\mathcal{H} = 2$. Thus, Nodes 6, 7 and 9 violate this limit. Thus, the EDD-A algorithm needs to fix such violations.

**Algorithm 2.** EDD-A Algorithm

1: **Input:** $G(V, E)$, $R$, $c$, $\mathcal{H}$, $T_{ms}$
2: **Output:** A low-cost Steiner Tree $T_{edd-a}$ within $\mathcal{H}$
3: $T_{edd-a} \leftarrow T_{ms}$, $parents[c] \leftarrow null$
4: For each edge server $v \in G$, set the parent of $v$ in $T_{edd-a}$ as $parents[v]$, the data retrieval latency of edge server $v$ in $T_{edd-a}$ as $d[v]$ where $d[c] \leftarrow 0$, and the cost from $v$ to $c$ in $T_{edd-a}$ as $costs[v]$
5: **for** each edge server $v$ in $T_{edd-a}$ in DFS order **do**
6: **if** $v \notin R$ **or** $d[v] \leq \mathcal{H}$ **then**
7: continue
8: **end if**
9: find the edge server $s \in \{T_{edd-a} - c\}$ that minimizes the cost of path $[c - s - v]$.
10: **if** $\Delta d[s, v] + d[s] \leq \mathcal{H}$ **then**
11: update $parents[]$ and $costs[]$ for edge servers on path $[c - s - v]$, add path $[s - v]$ into $T_{edd-a}$ and update $d[]$
12: **for** each edge server $u \in R$ **do**
13: **if** $costs[u] > costs[v] + cost(v, u)$ **then**
14: $costs[u] \leftarrow costs[v] + cost(v, u)$
15: add path $(v, u)$ into $T_{edd-a}$ and update $d[]$
16: **end if**
17: **end for**
18: **else**
19: $parents[v] \leftarrow c$
20: $costs[v] \leftarrow cost(c, v)$
21: update $d[]$
22: **end if**
23: **for** each child $u$ of $v \in T_{edd-a}$ **do**
24: **if** $costs[u] > costs[v] + cost(v, u)$ **then**
25: $parents[u] \leftarrow v$
26: $costs[u] \leftarrow costs[v] + cost(v, u)$
27: **end if**
28: **if** $costs[v] > costs[u] + cost(u, v)$ **then**
29: $parents[v] \leftarrow u$
30: $costs[v] \leftarrow costs[u] + cost(u, v)$
31: **end if**
32: update $d[]$
33: **end for**
34: **end for**
35: prune unused edges in $T_{edd-a}$ based on $parents[]$
36: **return** $T_{edd-a}$

Algorithm 2 presents the pseudo code of EDD-A. It takes the minimum Steiner tree $T_{ms}$ returned by Algorithm 1 as input. First, it initializes $T_{edd-a}$ by $T_{ms}$ and sets the parent of cloud server $c$ as null (Line 3). Then, it initializes $parents[]$ for recording the parent of each node in $T_{edd-a}$, $d[]$ for recording the transmission latency between $c$ and each node in $T_{edd-a}$, and $costs[]$ for recording the transmission costs between $c$ and each node in $T_{edd-a}$. After that, the algorithm visits each node $v$ that violates the latency limit $\mathcal{H}$ and finds the minimum-cost path $[c - s - v]$. If path $[c - s - v]$ helps $v$ eliminate the violation, EDD-A adds path $[c - s - v]$ into $T_{edd-a}$ and updates $parents[]$, $d[]$ and $costs[]$ accordingly (Lines 11-17). If the latency limit is still violated, i.e., the sum of latency $\Delta d[s, v]$ and edge server $s$'s latency $d[s]$

这里是怎么定义节点的 双亲节点和孩子节点的？？

exceeds the latency limit $\mathcal{H}$, $v$ will be connected to $c$ directly (Lines 19-21). Next, EDD-A visits each child $u$ of $v$ to update the shortest paths and the parents for both $u$ and $v$ (Lines 23-33). Finally, the algorithm prunes the unused edges in $T_{edd-a}$ (Line 35), and returns $T_{edd-a}$ as the final result.

**Example 5.** Continuing with Example 2, EDD-A selects node 6 to connect with the cloud node $c$ directly by adding edge $e_{c,6}$. Now, node 9, i.e., the child of node 6 in $T_{edd-a}$, can obtain the data via 1 hop. After this, EDD-A visits node 7 and finds the shortest path $[c-6-7]$. Thus, EDD-A adds edge $e_{6,7}$ into $T_{edd-a}$. After running EDD-A, the result is presented in Fig. 3c. The total cost now is $2\gamma+5$, and the EDD time constraint, $\mathcal{H}=1$, is fulfilled. In this case, the EDD strategy formulated by by EDD-A has the same cost as EDD-IP.

In the EDD-A algorithm, the computational overhead of finding the minimum Steiner tree $T_{ms}$ is $O(|R|^2)$. It takes at most $O(|V|^2)$ to read each edge server in $T_{edd-a}$ in DFS order and obtain the shortest path by the Dijkstra algorithm. Thus, the total computational overhead of EDD-A is $O(|R|^2+|V|^3)=O(|V|^3)=O(n^3)$.

In the rest of this section, we prove that EDD-A is an $O(k)$-approximation algorithm based on the following theorems and lemmas, where $k$ is a constant.

**Theorem 4.** *CMST is a 2-approximation algorithm to calculating the Steiner minimum tree.*

**Proof.** The original idea of CMST is the same as the algorithm proposed in [27]. The difference between them is the comparison in the connectivity between the nodes that have the same cost. This difference does not impact CMST's approximation ratio. Thus, CMST is a 2-approximation algorithm, the same as the approximation algorithm presented in [27]. □

Let us assume a tree $T_{ce}$ that consists of all the edge servers in $T_{ms}$ and the root $c$ with the edges $[c,v], \forall v \in \{T_{ms}-c\}$. Denote $cost(T_{ce})$, $cost(T_{ms})$ and $cost(T_{edd-a})$ as the total cost of $T_{ce}$, $T_{ms}$ and $T_{edd-a}$ accordingly.

**Theorem 5.** *For tree $T_{edd-a}$ produced by EDD-A, the cost of $T_{edd-a}$ is $\frac{2\gamma}{\mathcal{H}}+1$ times of the cost of $T_{ms}$ at most.*

**Proof.** Let $v_0=c$, and $v_i$ be the $i$th edge server, where $i \in 1,\dots m$, that introduces additional paths during the DFS traversal. Once the shortest path $[c-v_i]$, which is edge $[c,v_i]$, is added into $T_{edd-a}$, the cost is exactly $cost_{T_{ce}}(c,v_i)=\gamma$. Moreover, if the path from $c$ to edge server $v_i$ contains edge $[c,v_{i-1}]$ after updating the tree $T_{edd-a}$, we can obtain $cost_{T_{edd-a}}(c,v_i) \le cost_{T_{ce}}(c,v_{i-1}) + cost_{T_{ms}}(v_{i-1},v_i)$. However, if the direct path from $c$ to $v_i$ is added into $T_{edd-a}$, it means that $d[v_i]$ exceeds $\mathcal{H}$, and there is $cost_{T_{edd-a}}(c,v_i) \ge \gamma+\mathcal{H} = (1+\frac{\mathcal{H}}{\gamma})\cdot cost_{T_{ce}}(c,v_i)$. Thus, we can obtain the following equation:

$$\begin{aligned}\left(1+\frac{\mathcal{H}}{\gamma}\right)\cdot cost_{T_{ce}}(c,v_i) &\le cost_{T_{edd-a}}(c,v_i)\\ &\le cost_{T_{ce}}(c,v_{i-1}) + cost_{T_{ms}}(v_{i-1},v_i).\end{aligned} \tag{19}$$

Summing (19) for all the $m$ edge servers, we can obtain

$$\begin{aligned}\left(1+\frac{\mathcal{H}}{\gamma}\right)\sum_{i=1}^{m} cost_{T_{ce}}(c,v_i)\\ \le \sum_{i=1}^{m}(cost_{T_{ce}}(c,v_{i-1}) + cost_{T_{ms}}(v_{i-1},v_i,T_{ms})).\end{aligned} \tag{20}$$

Because of $\sum_{i=1}^{m} cost_{T_{ce}}(c,v_i) \ge \sum_{i=1}^{m-1} cost_{T_{ce}}(c,v_i)$, there is

$$\frac{\mathcal{H}}{\gamma} cost(T_{ce}) \le \sum_{i=1}^{m} cost_{T_{ms}}(v_{i-1},v_i), \tag{21}$$

where $cost(T_{ce})=\sum_{i=1}^{m} cost_{T_{ce}}(c,v_i)$.

For each edge server $v$ changing the path in $T_{edd-a}$ but not adding path $[c,v]$, the update cost, $cost_{update}(v)$, must be less than $cost_{T_{ce}}(c,v)$. Thus, the total cost after constructing $T_{ms}$ cannot be more than $\sum_{i=1}^{m} cost_{T_{ce}}(c,v_i)$.

Based on the DFS traversal, each edge is visited twice. Thus, the total cost of $cost_{T_{ms}}(v_{i-1},v_i)$ is no more than twice of $cost(T_{ms})$

$$\sum_{i=1}^{m} cost_{T_{ms}}(v_{i-1},v_i) \le 2\cdot cost(T_{ms}). \tag{22}$$

Thus, the total cost of $T_{edd-a}$ should be bounded by

$$cost(T_{edd-a}) \le cost(T_{ce}) + cost(T_{ms}) \le \left(\frac{2\gamma}{\mathcal{H}}+1\right)\cdot cost(T_{ms}). \tag{23}$$

□

**Theorem 6.** *EDD-A is an $O(k)$-approximation algorithm.*

**Proof.** As discussed in Theorem 4, the cost of $T_{ms}$ is at most twice the cost of the minimum Steiner tree $T$. However, the cost of the optimal solution of the EDD problem, $OPT$, cannot be less than that of the minimum Steiner tree. Thus, we can obtain

$$\frac{cost(T_{edd-a})}{cost(T_{OPT})} \le \frac{cost(T_{edd-a})}{cost(T)} \le 2\left(\frac{2\gamma}{\mathcal{H}}+1\right) = \frac{4\gamma}{\mathcal{H}}+2. \tag{24}$$

From (24), the approximation ratio of EDD-A is $2+\frac{4\gamma}{\mathcal{H}}$. Let $k=2+\frac{4\gamma}{\mathcal{H}}$. As both $\gamma$ and $\mathcal{H}$ are constant inputs, $k$ is a constant. Thus, EDD-A is an $O(k)$-approximation algorithm where $k$ is a constant. □

Similar to EDD-IP, specific cost and latency models can also be easily integrated into the EDD-A algorithm. Let $\{\gamma_1,\gamma_2,\dots,\gamma_n\}$ denote the C2E costs, $\{\alpha_{1,1},\alpha_{1,2},\dots,\alpha_{n,n}\}$ as the E2E costs, and $\mathcal{L}(u,v)$ as the latency model. Cost functions $cost(c,v)$ and $cost(u,v)$ can be replaced by $\gamma_v$ and $\alpha_{u,v}$, respectively, in Algorithm 2, and the latency function $\Delta d[s,v]$ can be calculated by $\mathcal{L}(u,v)$. Now we prove that EDD-A is still an $O(k)$-approximation algorithm in real-world EDD scenarios with specific cost and latency models.

**Theorem 7.** *EDD-A is an $O(k)$-approximation algorithm with specific cost and latency models.*

**Proof.** We denote $\gamma_{max}=\{\gamma_1,\gamma_2,\dots,\gamma_n\}$ as the maximum C2E cost, $\alpha_{min}=\{\alpha_{1,1},\alpha_{1,2},\dots,\alpha_{n,n}\}$ as the minimum E2E

cost, $\mathcal{L}_{E2E}^{max}$ as the maximum E2E latency and $\mathcal{L}_{C2E}^{min}$ as the minimum C2E latency. In this case, a specific latency limit $\mathcal{L}_{limit}$ is given by the app vendor to replace $\mathcal{H}$. Let $v_0 = c$, and $v_i$ ($i \in \{1, \dots m\}$) be the $i$th edge server that introduced additional paths during the DFS traversal in Algorithm 2. Once edge $[c, v_i]$, is added into $T_{edd-a}$, the cost is exactly $cost_{T_{ce}}(c, v_i) = \gamma_i$. If the path from $c$ to edge server $v_i$ contains edge $[c, v_{i-1}]$ after updating tree $T_{edd-a}$, we can obtain

$$cost_{T_{edd-a}}(c, v_i) \leq cost_{T_{ce}}(c, v_{i-1}) + cost_{T_{edd-a}}(v_{i-1}, v_i). \tag{25}$$

However, if the direct path from $c$ to $v_i$ is in $T_{edd-a}$, it means $d[i] > \mathcal{L}_{limit}$. Given $\alpha_{min}$ and $\mathcal{L}_{E2E}^{max}$, the ratio of cost over latency for any E2E edge is more than or equals to $\frac{\alpha_{min}}{\mathcal{L}_{E2E}^{max}}$. Denote $\mathcal{H}' = \frac{\alpha_{min}}{\mathcal{L}_{E2E}^{max}}(\mathcal{L}_{limit} - \mathcal{L}_{C2E}^{min})$, we can obtain

$$cost_{T_{edd-a}}(c, v_i) \geq \gamma_i + \mathcal{H}' = \left(1 + \frac{\mathcal{H}'}{\gamma_i}\right) \cdot cost_{T_{ce}}(c, v_i). \tag{26}$$

Combing (26) and (25), (27) stands

$$\left(1 + \frac{\mathcal{H}'}{\gamma_i}\right) \cdot cost_{T_{ce}}(c, v_i) \leq cost_{T_{ce}}(c, v_{i-1}) + cost_{T_{ms}}(v_{i-1}, v_i). \tag{27}$$

Summing (27) for all the $m$ edge servers, we can obtain

$$\sum_{i=1}^{m}\left(1 + \frac{\mathcal{H}'}{\gamma_i}\right) cost_{T_{ce}}(c, v_i) \leq \sum_{i=1}^{m}(cost_{T_{ce}}(c, v_{i-1}) + cost_{T_{ms}}(v_{i-1}, v_i)). \tag{28}$$

Since $\gamma_i$ is always lower than or equals to $\gamma_{max}$, we can obtain

$$\left(1 + \frac{\mathcal{H}'}{\gamma_{max}}\right) \sum_{i=1}^{m} cost_{T_{ce}}(c, v_i) \leq \sum_{i=1}^{m}\left(1 + \frac{\mathcal{H}'}{\gamma_i}\right) cost_{T_{ce}}(c, v_i). \tag{29}$$

Because of $\sum_{i=1}^{m} cost_{T_{ce}}(c, v_i) \geq \sum_{i=1}^{m-1} cost_{T_{ce}}(c, v_i)$ and (29), there is

$$\frac{\mathcal{H}'}{\gamma_{max}} cost(T_{ce}) \leq \sum_{i=1}^{m} cost_{T_{ce}}(v_{i-1}, v_i), \tag{30}$$

where $cost(T_{ce}) = \sum_{i=1}^{m} cost_{T_{ce}}(c, v_i)$.

Based on Theorem 4, we can obtain $k = 2 + \frac{4\gamma_{max}}{\mathcal{H}'}$. The corresponding process is omitted here because it is the same as in Theorems 5 and 6. Thus, Theorem 7 holds. □

## 5 Experimental Evaluation

We have experimentally evaluated the performance of EDD-IP and EDD-A. All experiments were conducted on a Windows-10 machine equipped with Intel Core i7-8665U processor (4 CPUs, 1.90 GHz) and 8 GB RAM.

### 5.1 Simulation Settings

#### 5.1.1 Approaches in Comparison

In our experiments, we evaluate the performance of EDD-IP and EDD-A against three representative approaches:

- *EDD-IP*: This approach finds the optimal EDD solutions by solving the COP defined in Section 4.1 with IBM's CPLEX CP Optimizer. Specifically, IloConstraint[7] is used to implement the constraints in EDD-IP, including constraint (11) with the product terms of binary variables.
- *EDD-A*: This approach finds near-optimal EDD solutions with Algorithm 2 described in Section 4.2.
- *Minimum-cost Multi-cast Routing (MMR)* [28]: This approach deals with the data routing problem in communication networks. It is also based on Steiner tree and presented as Algorithm 1 in [28].
- *Greedy Connectivity (GC)*: In this approach, we define the connectivity of edge server as the number of edge servers in $R$ that have yet to receive the data. This approach always selects the edge servers with the highest connectivity to receive data from the cloud, which will then transmit the data to other destination edge servers in $R$, until all the destination edge servers in $R$ can receive the data within the EDD time constraint $d_{limit}$.
- *Random*: This approach randomly selects edge servers to receive the data from the cloud, which then transmit the data to other destination edge servers in $R$ , one after another, until all the destination edge servers in $R$ receive the data within the EDD time constraint $d_{limit}$.

#### 5.1.2 Experiment Data

Two sets of experiments are conducted on a widely-used EUA dataset[8][29]. This dataset contains the geographical locations of 1,464 real-world base stations in Melbourne, Australia. As discussed in Section 3.1, $\gamma$ is dependent on specific edge infrastructure providers. In the experiments, we set $\gamma = 20$. The links between edge servers are randomly generated and to ensure a connected graph, based on the widely-used Erdös Rényi random graph model [30].

#### 5.1.3 Experiment Parameters

To simulate different EDD scenarios, four parameters that impact the performance of EDD-IP and EDD-A are varied in the experiments.

- *The number of edge servers ($n$) in $G$*. This parameter impacts the size of graph $G$.
- *Edge density ($density$)*. We define the edge density with $density = |E|/n$. This parameter impacts the density of graph $G$.
- *Ratio of destination edge servers ($ratio$)*. This ratio is calculated by $ratio = |R|/n$. It determines the number of destination edge servers in graph $G$.

7. https://www.ibm.com/support/knowledgecenter/SSSA5P_12.8.0/ilog.odms.studio.help/pdf/usrcpoptimizer.pdf
8. https://github.com/swinedge/eua-dataset

TABLE 2
Parameter Settings

| | $n$ | $d$ | $ratio$ | $limit$ | $T_{limit}$ | $\|variables\|$ in EDD-IP | $\|constraints\|$ in EDD-IP |
|---|---|---|---|---|---|---|---|
| **Set #1.1** | 10, 15, ..., 35 | 1.0 | 0.6 | 2 | $\infty$ | 42, 62, ..., 142 | 67, 100, ..., 232 |
| **Set #1.2** | 20 | 1.0, 1.2..., 2.0 | 0.6 | 2 | $\infty$ | 82, 86, ..., 102 | 133, 141, ..., 173 |
| **Set #1.3** | 20 | 1.0 | 0.2, 0.4 ..., 1.0 | 2 | $\infty$ | 82 | 125, 129, ..., 141 |
| **Set #1.4** | 20 | 1.0 | 0.6 | 1, 2, ..., 5 | $\infty$ | 82 | 133 |
| **Set #2.1** | 200, 300, ..., 700 | 2.0 | 0.6 | 2 | 10 | 1002, 1502, ..., 3502 | 1721, 2581, ..., 6021 |
| **Set #2.2** | 200 | 2.0, 2.4, ..., 4.0 | 0.6 | 2 | 10 | 1002, 1082, ..., 1402 | 1721, 1881, ..., 2521 |
| **Set #2.3** | 200 | 2.0 | 0.2, 0.4, ..., 1.0 | 2 | 10 | 1002 | 1641, 1681, ..., 1801 |
| **Set #2.4** | 200 | 2.0 | 0.6 | 1, 2, ..., 5 | 10 | 1002 | 1721 |
| **Set #2.5** | 200 | 2.0 | 0.6 | 2 | 2, 4, ..., 10 | 1002 | 1721 |

- *EDD time constraint (limit)*. Measured by the number of hops, this parameter indicates the app vendor's preference for the time taken by the EDD process.

As mentioned in Section 4.2, finding the optimal solutions is intractable in large-scale EDD scenarios. Thus, we limit the maximum running time (in seconds) of EDD-IP in Set #2.

- *Maximum running time ($T_{limit}$)*. EDD-IP will stop and return the optimal solution among all the solutions that have been found within $T_{limit}$.

Table 2 summarizes the parameter settings. There are two main sets of experiments, Set #1 of small-scale experiments and Set #2 of large-scale experiments. Every time the value of a parameter varies, the experiment is repeated for 100 times and the results are averaged. The last two columns in Table 2 are the number of variables ($3n + dn + 2$), consisting of $I$ $(n + 1)$, $\mathcal{D}$ $(n + 1)$ and $\mathcal{T}$ $(dn + n)$, and the number of constraints $(4n + 2dn + ratio \cdot n + 1)$ based on (9), (10), (11), (13), (14) and (15) in the EDD-IP model presented in Section 4.1.

### 5.1.4 Performance Metrics

The objective of the EDD problem is to minimize the EDD cost. Thus, this *cost* is a significant metric for evaluating the effectiveness of EDD-IP and EDD-A. In addition, as discussed in Section 1, the applications in the edge computing environment are latency-sensitive. It must not take too much time to find an EDD solution. Thus, the computational overhead is selected to evaluate the efficiency of EDD-IP and EDD-A.

- *EDD cost (cost)*, calculated by (5), the lower the better.
- *Computational overhead (time)*, measured by the time taken by an approach to find the solution, the lower the better.

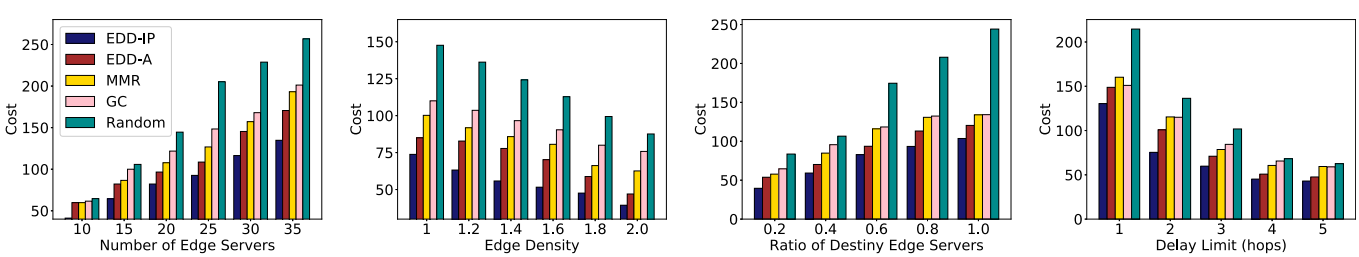
## 5.2 Experimental Results

### 5.2.1 Experiment Set #1

Through comparison with MMR, GC and random, Fig. 4 demonstrates the effectiveness of EDD-IP and EDD-A in experiment Set #1 and the impacts of the four parameters. It can be seen that *the EDD strategies formulated by EDD-IP can distribute the data at the lowest cost*, followed by EDD-A. Across the four subsets of experiments, the average advantages of EDD-IP are 18.22 percent over EDD-A, 28.01 percent over MMR, 32.54 percent over GC and 48.34 percent over Random. The average advantages of EDD-A are 11.96 percent over MMR, 17.62 percent over GC and 36.87 percent over Random.

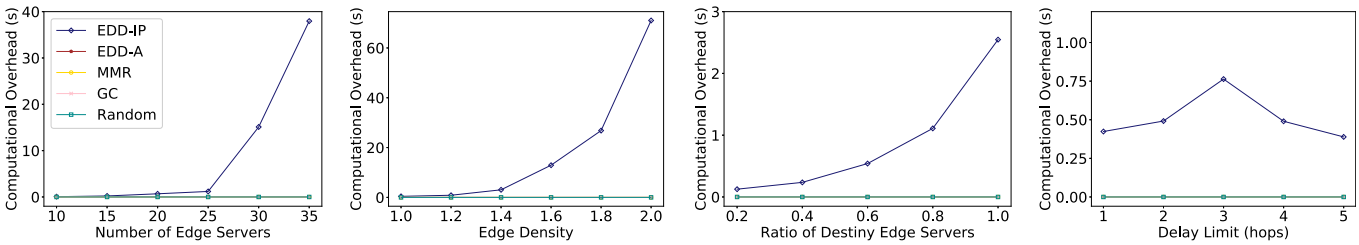
*Effectiveness*. Fig. 4a shows the effectiveness results of experiment Set #1.1. When the number of edge servers $n$ increases from 10 to 35, the costs of the EDD strategies formulated by all five approaches increase, from 41.2 to 134.8 by 227.18 percent for EDD-IP, from 59.8 to 170.6 by 185.28 percent for EDD-A, from 59.8 to 193.2 by 223.08 percent for MMR, from 61.6 to 201.2 by 226.62 percent by GC, and from 64.6 to 256.8 by 297.52 percent for Random. *Of all the five approaches, EDD-A has the lowest overall increase*.

Fig. 4b shows the results of experiment Set #1.2. Again, *EDD-IP and EDD-A outperform other approaches* with significant margins. The average advantages of EDD-IP are 21.38 percent over EDD-A, 31.97 percent over MMR, 40.43 percent over GC and 53.17 percent over Random, while the numbers for EDD-A are 13.46 percent over MMR, 24.23 percent over GC and 40.44 percent over Random. As the edge density $d$ increases from 1.0 to 2.0, the costs decrease for all the approaches. This is because, with the number of edge servers $n$ fixed, a higher edge density $d$ gives destination edge servers higher chances to receive the data within the EDD time constraint. This reduces the cost incurred by C2E




(a) *cost* vs. $n$ (Set #1.1) (b) *cost* vs. $d$ (Set #1.2) (c) *cost* vs. *ratio* (Set #1.3) (d) *cost* vs. *limit* (Set #1.4)

Fig. 4. Effectiveness comparison in Set #1.




(a) $time$ vs. $n$ (Set #1.1) (b) $time$ vs. $density$ (Set #1.2) (c) $time$ vs. $ratio$ (Set #1.3) (d) $time$ vs. $limit$ (Set #1.4)

Fig. 5. Efficiency comparison in Set #1.

because few destination edge servers have to receive the data directly from the cloud.

Fig. 4c shows the effectiveness results of experiment Set #1.3, where *EDD-IP and EDD-A achieve the best and second best performance*, respectively. The advantage of EDD-IP over EDD-A is 16.09 percent. The advantages of EDD-A over MMR, GC and Random are 13.79, 17.24 and 44.77 percent, respectively. As $ratio$ increases from 0.2 to 1.0, all five approaches need more costs to distribute the data. This is expected because a larger number of destination edge servers will certainly incur higher costs of C2E and/or E2E transmissions.

Fig. 4d shows the results of experiment Set #1.4. *EDD-IP outperforms EDD-A* by 15.56 percent, while EDD-A outperforms MMR, GC and Random by 11.27, 11.79 and 28.20 percent, respectively. As $limit$ increases, the costs of all five approaches decrease. The main reason is that a less stringent EDD time constraint relies less on C2E transmissions which are faster but more expensive than E2E transmissions.

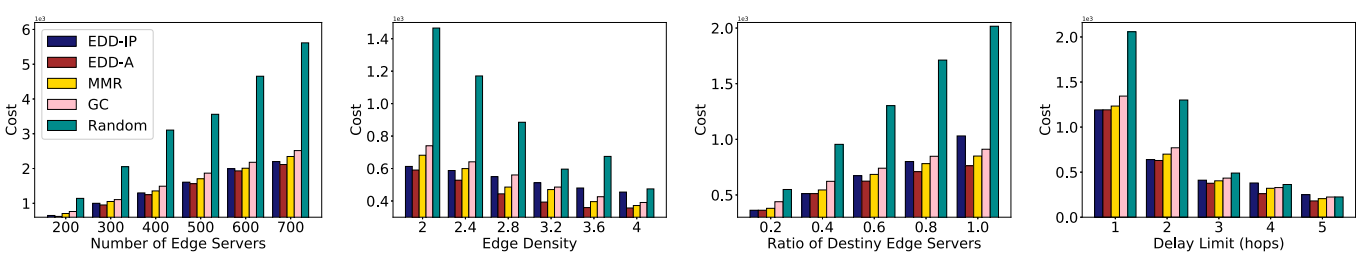
*Efficiency*. The efficiency results of Set #1 is presented in Fig. 5. As demonstrated, *EDD-IP is much more computationally expensive than all other approaches*. This validates the $\mathcal{NP}$-hardness of the EDD problem - excessive computational overheads are inevitable for finding the optimal solutions to large-scale EDD problems. As demonstrated in Fig. 5a, in largest-scale EDD scenarios with 700 edge servers, EDD-IP takes 39.96 seconds to find the optimal solution in Set #1.1. Moreover, EDD-IP takes up to 71.03 seconds in Set #1.2. When the edge density increases from 1.0 to 2.0, the size of the solution space for EDD-IP to explore becomes larger quickly. Thus, *its computational overhead increases significantly with the increase in edge density*. Similar phenomena are observed in Fig. 5c. Interestingly, the computational overhead of EDD-IP in Fig. 5d increases when the EDD time constraint $limit$ increases from 1 to 3, then decreases after that. With $limit > 3$, EDD-IP can find the optimal solution that requires as few as 1 initial transit edge servers. This eases the EDD-IP's pain in inspecting the possible solutions that require multiple initial transit edge servers. As a result, EDD-IP's computational overhead decreases.

Compared with EDD-IP, *the computational overheads of EDD-A stay at a very low level*, taking less than 8.06 seconds in average to find a solution in the entire Set #1. To further evaluate the performance of EDD-A, we present and discuss the results of Set #2, i.e., large-scale experiments, in Section 5.2.2.

### 5.2.2 Experiment Set #2

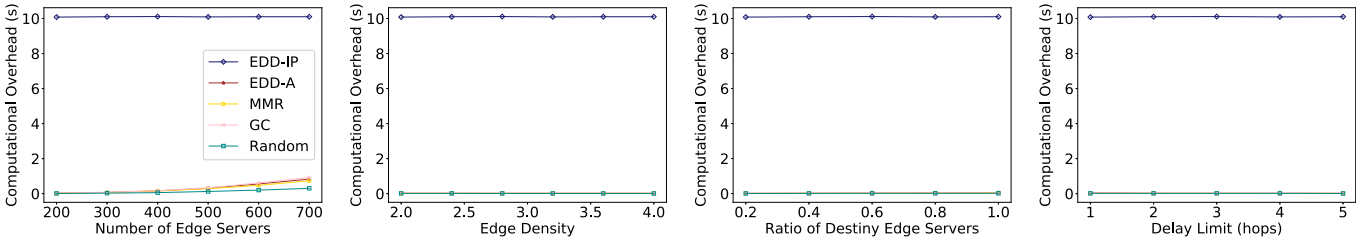
*Effectiveness*. Fig. 6 demonstrates the EDD costs achieved by EDD-IP, EDD-A, MMR, GC and Random in experiment Sets #2.1 to #2.4 . Overall, their trends in achieving a low cost are similar to Fig. 4 with the same reasons for their performance changes as discussed in Section 5.2.1. Since the maximum running time of EDD-IP is limited in Set #2, its solution is not always the real optimal solution. Thus, EDD-A achieves the best performance in the entire Set #2. In Fig. 6a, the costs increase from 650.3 to 2,200.1 by 238.82 percent for EDD-IP, from 623.4 to 2,115.0 by 239.27 percent for EDD-A, from 706.2 to 2,346.2 by 232.23 percent for MMR, from 764.6 to 2,515.8 by 229.03 percent for GC, and from 1,142.0 to 5,614.6 by 391.65 percent for Random. Fig. 6b shows that the advantages of EDD-A are 29.86 percent over EDD-IP, 11.08 percent over MMR, 17.63 percent over GC and 49.26 percent by Random on average in experiment Set #2.2 where the edge density $d$ increases from 2.0 to 4.0. The performance differences between the five approaches demonstrated in Figs. 6c and 6d are similar to those demonstrated in Figs. 4c and 6d. Thus, it is not discussed in detail here. Fig. 8a depicts that the cost achieved by EDD-IP decreases in Set #2.5 when the running time limit $T_{limit}$ increases. This is because given more running time, EDD-IP can search for more possible solutions to achieve a lower cost.

*Efficiency*. Fig. 7 depicts the efficiency of each approach in experiment Sets #2.1 to #2.4. In such large-scale experiments,




(a) $cost$ vs. $n$ (Set #2.1) (b) $cost$ vs. $density$ (Set #2.2) (c) $cost$ vs. $ratio$ (Set #2.3) (d) $cost$ vs. $limit$ (Set #2.4)

Fig. 6. Effectiveness comparison in Sets #2.1 - #2.4.




(a) $time$ vs. $n$ (Set #2.1) (b) $time$ vs. $density$ (Set #2.2) (c) $time$ vs. $ratio$ (Set #2.3) (d) $time$ vs. $limit$ (Set #2.4 )

Fig. 7. Efficiency comparison in Sets #2.1 - #2.4.

EDD-A, MMR, GC and Random can still find their solutions very fast. In Set #2.1, EDD-A takes a little more time than MMR and Random when the number of edge servers increases. This is the performance price to pay for EDD-A's advantages over these approaches in terms of effectiveness as shown and discussed above. With the significant advantages of EDD-A, especially where there are more edge servers, a higher edge density, a higher ratio of destination edge servers and a lower EDD time constraint, it is worth running EDD-A in real-world applications. As shown in Fig. 8b, the computational overheads of EDD-IP are slightly higher than its running time limit. The reason is that after EDD-IP stops searching for possible solutions when $T_{limit}$ is reached, it still needs some time to find the best solution among those possible solutions.

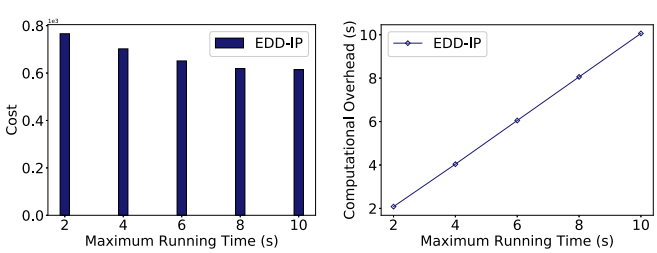
#### 5.2.3 Conclusion

The experimental results demonstrated and discussed above indicate that EDD-IP is suitable for solving EDD problems of reasonable sizes. To solve large-scale EDD problems, EDD-A is more practical for its high efficiency in finding solutions very close to the optimal ones.

### 5.3 Threats to Validity

#### 5.3.1 Construct Validity

The main threats to the construct validity are threefold: 1) the comparison with MMR, GC and Random may not suffice to comprehensively evaluate EDD-IP and EDD-A; 2) the dataset used in the experiments may not represent all real-world scenarios exactly; 3) the performance of our approaches in real-world EDD scenarios may not be exactly the same as in our experiments. To minimize these threats, we varied setting parameters in both sets of experiments, as summarized in Table 2, to simulate a broad range of various EDD scenarios. This has allowed us to evaluate our approaches more comprehensively. Moreover, we evaluated EDD-IP and EDD-A by not only the comparison with three representative approaches but also the demonstration of how the changes in the setting parameters impacted their performance. In this way, the experimental results can be used as guidelines on the estimation of the performance of our approaches in real-world applications.




(a) $cost$ vs. $T_{limit}$ (Set #2.5) (b) $cost$ vs. $T_{limit}$ (Set #2.5)

Fig. 8. Effectiveness and efficiency of EDD-IP in Set #2.5.

#### 5.3.2 External Validity

The main threat to the external validity of the evaluation is whether EDD-IP and EDD-A can be generalized and applied in other application scenarios in the edge computing environment. To tackle this threat, we modeled the problem and evaluated the performance of EDD-IP and EDD-A in a generic manner - using the number of hops to represent the data transmission latency between edge servers and a ratio $\gamma$ to indicate the difference between the C2E transmission cost and the E2E transmission cost. In this way, the evaluation results can be interpreted with specific latency and cost models. We also varied parameters to change the size and the complexity of the EDD problem. This way, the representativeness and comprehensiveness of the evaluation are ensured. The above mitigates the threat to external validity.

#### 5.3.3 Conclusion Validity

The lack of statistical tests, e.g., chi-square tests, is the major threat to conclusion validity in our paper. To compensate this threat, we have conducted comprehensive and intensive experiments to cover various scenarios in different size and complexity. In addition, every time a parameter changes, we repeat the experiment for 100 times and calculate the averaged results. This led to a large number of test cases, which tend to result in a small p-value in the chi-square tests and lower the practical significance of the test results [31]. For example, in experiment Set #1, there were a total of 2,200 runs. This number is not even close to the number of observation samples that concern Lin *et al.* in [31]. This way, the threat to the conclusion validity due to the lack of statistical tests might be high but is not significant.

## 6 Related Work

Edge computing was proposed by Cisco in 2012 as a new computing paradigm. As an extension of cloud computing, edge computing distributes cloud-like computing resources and services to the edge of the cloud [24]. Applications, services and data can now be deployed on edge servers to

offer end-users ultra-low latency. It offers new opportunities and in the meantime raises many new challenges, e.g., edge server placement, edge user allocation, computation offloading, edge application deployment, edge data caching and edge data distribution.

Edge server placement is a fundamental problem in edge computing. In [32], the authors focused on minimizing the cost incurred during edge server deployment. They designed a cost-effective method that employs integer programming to help edge infrastructure providers make decisions on edge server placements. Similarly, Yin *et al.* [33] presented a decision support framework based on flexible placement, namely Tentacle. It aims to minimize the cost of edge infrastructure deployment while maximizing the overall system performance.

The edge user allocation problem in the edge computing environment was first studied in [29]. The authors of [29] modeled this problem as a variable sized vector bin packing problem to maximize the number of allocated app users, while minimizing the cost of hiring edge servers. He *et al.* [3] tackled a similar problem in edge user allocation with the aim to find a near-optimal solution in an efficient manner. They proposed EUAGame, a game-theoretic approach that employs a decentralized algorithm to find the Nash equilibrium of the game as the solution to the problem.

The problem of computation offloading has been intensively studied with consideration of edge servers' energy efficiency, offloading cost and joint service with caching [5], [32], [34], [35]. Xu *et al.* [5] proposed an online algorithm, namely OREO, to efficiently improve offloading performance jointly with service caching. OREO was developed based on Lyapunov optimization to reduce computation offloading latency while keeping energy consumption low. In [36], Wang *et al.* considered the computation offloading problem in wireless cellular networks with mobile edge computing. They modeled this problem as a convex problem and then decomposed it to be solved in a distributed manner.

Application deployment is another problem in the edge computing environment that has attracted increasing attention from researchers. A number of approaches have been proposed to determine optimal deployment strategies with different objectives, such as maximizing the user or request coverage [37], minimizing the deployment cost [38], and maximizing the profit [39]. For example, Wang *et al.* [38] focused on an application migration problem and proposed a Markov decision process based framework for migrating application instances between edge-clouds, with the aim to minimize the average migration cost and the transmission cost over time. Mahmud *et al.* [39] proposed a new pricing model deploying applications on fog instances. They also proposed a user compensation method based on SLA violation. Based on the new pricing model and the user compensation method, an approach was proposed to find optimal application deployment strategies that fulfil resource constraints like processing cores and memory.

In recent years, researchers have started to propose and investigate new ideas and techniques for data caching in the edge computing environment. Cao *et al.*[14] presented an optimal auction mechanism to maximize service provider's revenue based on cache allocation and user valuation reports. They proposed computationally-efficient approaches to apply the auction mechanism based on data retrieval and delivery costs. The authors of [11] proposed a caching system named Cachier for recognition of applications in an MEC environment. Cachier coordinates the loading balance between edge servers and the cloud to minimize the data retrieval latency in a dynamical manner. Breitbach *et al.* proposed a data management system for edge computing environments by decoupling data placement based on task scheduling [40]. The system adjusted the data replica placement cost to achieve the balance between data management overhead and execution delay. In [41], the authors focused on data-intensive IoT workflows in a collaborative edge and cloud computing environment. They also formulated the problem as a 1-0 integer programming model and provided a variant of the intelligent swarm optimization algorithm to solve the problem.

However, existing research has not considered the fact that transmitting the data on cloud and edge computing infrastructure is also a large component in app vendors' cost structure in the edge computing environment. Without considering this component, app vendors will not be able to realistically evaluate the costs of caching their app data on edge servers. To the best of our knowledge, this paper makes the first attempt to solve the edge data distribution problem from the app vendor' perspective in the edge computing environment. Its aim is to minimize the cost of distributing data from the cloud to edge server, with consideration of the costs incurred during C2E and E2E transmissions, while fulfilling the app vendor's time constraint for data distribution.

## 7 Conclusion

In this paper, we formulated the edge data distribution problem in the edge computing environment as a constrained optimization problem from the app vendor's perspective. We proved that the EDD problem is $\mathcal{NP}$-hard. To solve this problem, we proposed an optimal approach named EDD-IP based on the Integer Programming technique to minimize the cost incurred during data distribution. As the EDD problem is $\mathcal{NP}$-hard, we also provided an approximation approach named EDD-A for finding approximate solutions to large-scale EDD problems efficiently. Extensive experiments were conducted on a widely-used real-world dataset to evaluate the performance of the proposed approaches. The results showed that our approaches significantly outperformed the representative approaches in various EDD scenarios.

This research has established the foundation for the EDD problem and opened up a number of future research directions. In our future work, we will consider the robustness and fault-tolerance of EDD strategies, and more dynamic and heterogeneous aspects of edge servers.

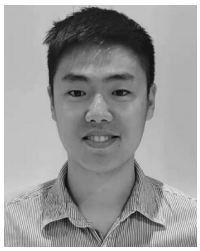
## Acknowledgments

This research was funded in part by the Australian Research Council Discovery Projects No. DP180100212, DP200102491, and Laureate Fellowship FL190100035.

## References

[1] A. Osseiran *et al.*, "The foundation of the mobile and wireless communications system for 2020 and beyond: Challenges, enablers and technology solutions," in *Proc. IEEE 77th Veh. Technol. Conf.*, 2013, pp. 1–5.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[3] Q. He *et al.*, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 3, pp. 515–529, Mar. 2020.
[4] T. X. Tran, M.-P. Hosseini, and D. Pompili, "Mobile edge computing: Recent efforts and five key research directions," *IEEE COMSOC MMTC Commun.-Frontiers*, vol. 12, no. 4, pp. 29–33, 2017.
[5] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 207–215.
[6] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
[7] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.
[8] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.
[9] L. Yang, H. Zhang, M. Li, J. Guo, and H. Ji, "Mobile edge computing empowered energy efficient task offloading in 5G," *IEEE Trans. Veh. Technol.*, vol. 67, no. 7, pp. 6398–6409, Jul. 2018.
[10] J. Zhao, W. Gao, Y. Wang, and G. Cao, "Delay-constrained caching in cognitive radio networks," *IEEE Trans. Mobile Comput.*, vol. 15, no. 3, pp. 627–640, Mar. 2016.
[11] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in *Proc. 37th IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 276–286.
[12] X. Zhang and Q. Zhu, "Hierarchical caching for statistical QoS guaranteed multimedia transmissions over 5G edge computing mobile wireless networks," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 12–20, Jun. 2018.
[13] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Cooperative content caching in 5G networks with mobile edge computing," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 80–87, Jun. 2018.
[14] X. Cao, J. Zhang, and H. V. Poor, "An optimal auction mechanism for mobile edge caching," in *Proc. 38th IEEE Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 388–399.
[15] R. Halalai, P. Felber, A.-M. Kermarrec, and F. Taïani, "Agar: A caching system for erasure-coded data," in *Proc. 37th IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 23–33.
[16] Y. Wang, B. Veeravalli, and C.-K. Tham, "On data staging algorithms for shared data accesses in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 4, pp. 825–838, Apr. 2013.
[17] Q. Zhang, Q. Zhang, W. Shi, and H. Zhong, "Firework: Data processing and sharing for hybrid cloud-edge analytics," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 9, pp. 2004–2017, Sep. 2018.
[18] Y. Liu, M. Dong, K. Ota, and A. Liu, "ActiveTrust: Secure and trustable routing in wireless sensor networks," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 9, pp. 2013–2027, Sep. 2016.
[19] K. Gai, L. Qiu, M. Chen, H. Zhao, and M. Qiu, "SA-EAST: Security-aware efficient data transmission for its in mobile heterogeneous cloud computing," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 2, pp. 1–22, 2017.
[20] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through VM migration and transmission power control," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 810–819, May 2017.
[21] H. Cai, B. Xu, L. Jiang, and A. V. Vasilakos, "IoT-based big data storage systems in cloud computing: Perspectives and challenges," *IEEE Internet Things J.*, vol. 4, no. 1, pp. 75–87, Feb. 2017.
[22] H. Guo and J. Liu, "Collaborative computation offloading for multi-access edge computing over fiber-wireless networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4514–4526, May 2018.
[23] M. Patel *et al.*, "Mobile-edge computing introductory technical white paper," *White Paper, Mobile-Edge Comput. (MEC) Industry Initiative*, pp. 1089–7801, 2014.
[24] M. Yannuzzi *et al.*, "A new era for cities with fog computing," *IEEE Internet Comput.*, vol. 21, no. 2, pp. 54–67, Mar./Apr. 2017.
[25] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*. Berlin, Germany: Springer, 1972, pp. 85–103.
[26] F. K. Hwang and D. S. Richards, "Steiner tree problems," *Networks*, vol. 22, no. 1, pp. 55–89, 1992.
[27] H. Takahashi, "An approximate solution for the steiner problem in graphs," *Math. Japonica*, vol. 6, pp. 573–577, 1990.
[28] G. Xue, "Minimum-cost QoS multicast and unicast routing in communication networks," *IEEE Trans. Commun.*, vol. 51, no. 5, pp. 817–824, May 2003.
[29] P. Lai *et al.*, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Proc. Int. Conf. Service-Oriented Comput.*, 2018, pp. 230–245.
[30] P. Erdös and A. Rényi, "On random graphs publ," *Math. Debrecen*, vol. 6, pp. 290–297, 1959.
[31] M. Lin, H. C. Lucas Jr, and G. Shmueli, "Research commentary—too big to fail: Large samples and the p-value problem," *Inf. Syst. Res.*, vol. 24, no. 4, pp. 906–917, 2013.
[32] H. Yao, C. Bai, M. Xiong, D. Zeng, and Z. Fu, "Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing," *Concurrency Comput. Pract. Experience*, vol. 29, no. 16, 2017, Art. no. e3975.
[33] H. Yin *et al.*, "Edge provisioning with flexible server placement," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1031–1045, Apr. 2017.
[34] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1784–1797, Mar. 2018.
[35] S. Jošilo and G. Dán, "A game theoretic analysis of selfish mobile computation offloading," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[36] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 16, no. 8, pp. 4924–4938, Aug. 2017.
[37] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *Proc. 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 365–375.
[38] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.
[39] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Profit-aware application placement for integrated fog–cloud computing environments," *J. Parallel Distrib. Comput.*, vol. 135, pp. 177–190, 2020.
[40] M. Breitbach, D. Schäfer, J. Edinger, and C. Becker, "Context-aware data and task placement in edge computing environments," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, 2019, pp. 1–10.
[41] Y. Shao, C. Li, and H. Tang, "A data replica placement strategy for IoT workflows in collaborative edge and cloud environments," *Comput. Netw.*, vol. 148, pp. 46–59, 2019.



**Xiaoyu Xia** received the master's degree from The University of Melbourne, Australia, in 2015. He is currently working toward the PhD degree at Deakin University, Australia. His research interests include edge computing, service computing, and software engineering.

**Feifei Chen** received the PhD degree from the Swinburne University of Technology, Australia, in 2015. She is currently a lecturer at Deakin University, Australia. Her research interests include software engineering, cloud computing, and green computing.

**Qiang He** (Member, IEEE) received the first PhD degree from the Swinburne University of Technology, Australia, in 2009 and the second PhD degree in computer science and engineering from the Huazhong University of Science and Technology, China, in 2010. He is a senior lecturer at Swinburne, Australia. His research interests include service computing, software engineering, cloud computing and edge computing. For more information please visit https://sites.google.com/site/heqiang/

**John C. Grundy** (Senior Member, IEEE) received the BSc (Hons), MSc, and PhD degrees in computer science from the University of Auckland, New Zealand. He is currently an australian laureate fellow and a professor of software engineering at Monash University, Melbourne, Australia. He is an associate editor of the *IEEE Transactions on Software Engineering*, *Automated Software Engineering Journal*, and *IEEE Software*. His current interests include domain-specific visual languages, model-driven engineering, large-scale systems engineering, and software engineering education. For more information please visit https://sites.google.com/site/johncgrundy/

**Mohamed Abdelrazek** is an associate professor of software engineering and IoT at Deakin University, Australia. Before joining Deakin University, Australia, in 2015, he worked as a senior research fellow at the Swinburne University of Technology, Australia and Swinburne-NICTA software innovation lab (SSIL). Before 2010, he was the head of Software Development Department at Microtech. For more information please visit at https://sites.google.com/site/mohamedalmorsy/

**Hai Jin** (Fellow, IEEE) received the PhD degree in computer engineering from the Huazhong University of Science and Technology, China, in 1994. He is a Cheung Kung scholars chair professor of computer science and engineering at the Huazhong University of Science and Technology (HUST), China. His research interests include computer architecture, virtualization technology, cluster computing and cloud computing, peer-to-peer computing, network storage, and network security.