Application Data Distribution in Edge Computing

A B. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Ravi Shankar
(170101053)

under the guidance of

Dr. Aryabartta Sahu

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Application Data Distribution in Edge Computing**" is a bonafide work of **Ravi Shankar (Roll No. 170101053)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Aryabartta Sahu**

Associate Professor,

April, 2021

Department of Computer Science & Engineering,

Guwahati.

Indian Institute of Technology Guwahati, Assam.

Acknowledgements

I would like to thank Dr.Aryabartta Sahu for his constant support and guidance throughout the project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Abstract

Edge computing is a distributed computing paradigm that brings computation and data storage closer to the user's geographical location to improve response times and save bandwidth. It also helps to power a variety of applications requiring low latency. These application data hosted on the cloud needs to be transferred to the respective edge servers in a specific area to help provide low latency app-functionalities to the users of that area. Meanwhile, these arbitrary heavy data transactions from the cloud to the edge servers result in high cost and time penalties. Thus, we need an application data distribution strategy that minimizes these penalties within the app-vendors' specific latency constraint.

In phase 1 work of BTP, we provide a refined formulation of an optimal approach to solve this Edge Data Distribution (EDD) problem using Integer Programming (IP) technique. Due to the time complexity limitation of the IP approach, we suggest an O(k) approximation algorithm based on network Steiner tree estimation (EDD-NSTE) for estimating solutions to dense large-scale EDD problems. Integer Programming and EDD-NSTE are

evaluated on a standard real-world EUA data set and the result demonstrates that EDD-NSTE significantly outperform with a performance margin of 86.67% over the other three representative approaches and the start of art approach.

In phase 2 work of BTP, we provide a modified formulation of the EDD problem to include end-to-end propagation delay guarantees. We provide an optimal approach to solve this problem using Integer Programming (IP) technique. Due to the time complexity limitation of the IP approach, we suggest a modified implementation of the EDD-NSTE algorithm to include this. Integer Programming and EDD-NSTE are then evaluated on standard real-world EUA and SLNDC datasets and the result demonstrates that EDD-NSTE significantly outperform other approaches with a performance margin of 80.35% over other approaches in comparison.

1.2 Introduction

Cloud computing is the practice of using a network of remote servers hosted on the internet to store, manage, and process data, rather than a local server or a personal computer. Other conventional network paradigms provided by cloud computing, which includes content-centric network, content-delivery network, and information-centric network also cannot handle the huge increase in the latency of the network and the congestion caused by the resources at the edge of the cloud [WWea20, PB08].

This is where the idea of edge computing kicks in which is basically the method of decoupling the computer services with the help of edge servers deployed at the base stations or access points that are geographically close to the users [DPW04, SCZ+16]. These computing and storage resources can then be easily hired by any mobile or IoT app-vendor for hosting their applications. The users can then get rid of the computational burden and energy overload from their resource-limited end-devices to the edge servers located nearby. Thus, from an app vendors' perspective, caching data on the edge servers can reduce both

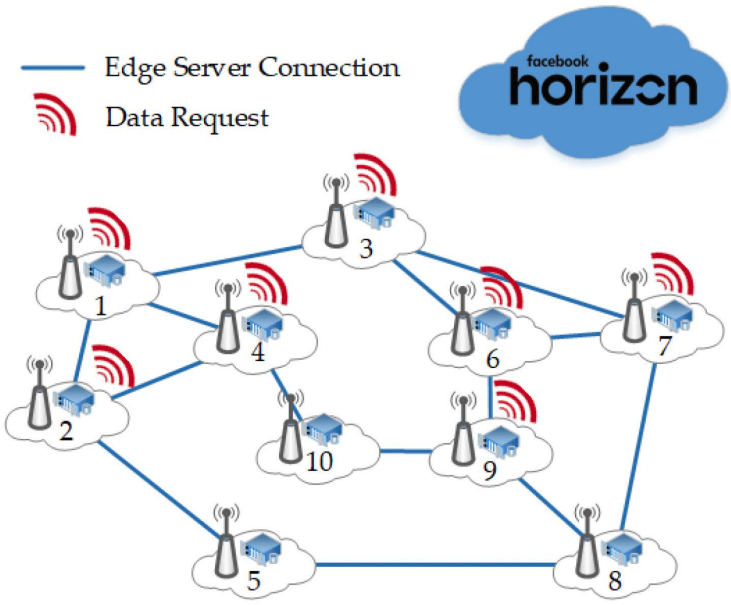

Fig. 1.1: An industry example

the latency of their users to fetch the data and the volume of their application data transmitted between the cloud and its users, thereby, reducing the transmission cost. But then a new challenge arises here as to how to distribute the application data onto the edge servers to minimize the cost incurred for the transactions of data between cloud-to-edge (C2E) and edge-to-edge (E2E) servers.

Facebook Horizon [FH] is an example of industrial application that can benefit considerably from caching their data onto the edge servers. Facebook users using Oculus headsets can access Virtual Reality (VR) videos and VR games on Facebook Horizon. VR users and their applications are highly latency-sensitive. Thus, caching the most trendy and popular VR videos and VR games onto the edge servers will allow the users near the geographical locations of those edge servers to access the application data with minimum latency which in turn will increase the VR experience, sensitivity, and performance. Figure 1.1 displays a simple graphical example of this idea, where it is required to distribute the application data to edge servers to decrease the data traffic and request-response time between the Facebook Horizon cloud servers and its users in those specific areas. Therefore, the edge data distribution is a must, and at the same time, a cost-effective method to distribute the

application data needs to be incorporated as otherwise the cost-ineffective or random app data distribution strategy can cost Facebook Horizon significantly. The cost-effective data distribution strategy should also consider a constraint on the time taken to distribute the application data to each of the edge servers, i.e., the Facebook Horizon latency limit.

1.3 Organization of The Report

This report is organized into six chapters. The first chapter includes the abstract and introduction of the report. Chapter 2 contains the previous works which are related to this research domain. Chapter 3 incorporates the problem formulation, solution strategies, and experimental evaluation of the *Edge Data Distribution (EDD) considering hop distances as the vendor's latency parameter*. Similarly, Chapter 4 contains the problem formulation, solution strategies, and experimental evaluation of the *Edge Data Distribution (EDD) considering propagation delay as the vendor's latency parameter*. Chapter 5 includes the conclusion of this research, and Chapter 6 contains the future works that can be extended on this research.

Chapter 2

Related Work

Cloud Computing is the practice of using a network of remote servers hosted on the internet to store, manage, and process data rather than a local server or a personal computer. Cloud-based storage makes it possible to save files to a remote database and retrieve them at user request. The scheduling of these requests is a crucial problem in cloud computing and, over the years, researchers have proposed various scheduling algorithms. In a recent research Zhao et al. [ZML18], the authors suggested a state-of-the-art scheduler, which not only takes into consideration the load balance but also the application properties for request scheduling.

On the other hand, Edge Computing is an extension of cloud computing which distributes computing resources and services to the edge servers of a particular region. The placement of these edge servers is a fundamental issue in edge computing. In Yao et al. [YBX$^+$17], the authors suggested a cost-effective method that uses 0-1 integer programming to help providers of edge infrastructure make correct decisions regarding the edge servers placement. Similarly, Hao Yin et al. [YZL$^+$17] suggested a decision support framework based on a flexible server placement, namely Tentacle, which aims to minimize the cost while maximizing the overall system performance. The cost-effective application user allocation is another fundamental problem in edge computing first studied in [LHA$^+$]. In research [LHGea20],

the authors formulated it as a bin packing problem and suggested a heuristic approach for approximating the sub-optimal solutions to this problem.

In the recent researches [CZP], [DGT$^+$], [ZLH$^+$18], [HFKT], [ZZ18] and [BSEB], researchers have proposed investigative new techniques and approaches for data caching in the edge computing environment. However, even after having an optimal edge server placement, application user allocation, and data caching, we still need to consider the fact that transmitting the application data from the cloud to the edge servers also contributes a considerable amount to the app vendors' expense structure. Thus, a cost-effective application data distribution strategy is needed to minimize and accurately estimate the app vendors' total cost. A recent research in Xia et al. [XCH$^+$21], authors have formulated this problem and suggested a heuristic state-of-the-art algorithm to estimate the total cost, including both the C2E and E2E transmissions. This paper we extends the idea, refined the formulation and introduces a solution approach named EDD-NSTE based on network Steiner tree estimation, for estimating the total cost associated with the edge data distribution problem from the app vendors' perspective.

Chapter 3

EDD considering hop distances as the vendor's latency parameter

3.1 Problem Formulation

In edge computing, adjacent edge servers at different stations can also communicate, share information as well as resources between them. Thus, edge servers of a specific geo-location can be combined to form a network, which can then be formulated as a graph. In this paper, we consider N edge servers of a specific area and model it as a graph G. Each node v of the graph G represents an edge-server v, and each edge of the graph connecting any two nodes u and v, i.e., $e_{(\mathrm{u},\ \mathrm{v})}$ represents the link between these two edge-servers. Thus, we use $G(V, E)$ to represent the graph, where V is the set of edge servers in the graph and E is the set of links. Let R denote the set of destination edge servers in the graph G, i.e., the edge servers which must be sent data directly or indirectly from the cloud servers within the vendor's specific latency constraint.

For example, in Figure 3.1, $N = 10$, which means we have 10 edge servers i.e., $V = \{1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10\}$. The set of edges $E = \{e_{(2,\ 3)},\ e_{(6,\ 8)},\ e_{(8,\ 7)},\ \cdots,\ e_{(2,\ 4)}\}$. Similarly, the set of destination edge servers $R = \{2,\ 3,\ 4,\ 5,\ 6,\ 8,\ 9\}$ shown as square box

Table 3.1: Summary of Common Notations

Notation	Description
G	graph representing a particular region
N	number of edge servers in a particular region
V	set of all the edge servers in a particular region
E	set of all edges in the graph, i.e, high-speed links
R	set of destination edge servers in the graph G
u, v	edge servers
$e_{(\mathrm{u,\ v})}$	link/edge between edge server nodes u and v
c	cloud server
S	set of binary variables indicating initial transit edge server
H	set of binary variables indicating edge servers visited during EDD process
T	set of binary variables indicating the data distribution path
γ	ratio of cost of C2E to cost of 1-Hop E2E
ρ	destination edge server density
δ	edge density
d_{limit}	vendors' latency constraint, or hop limit
D_{limit}	depth limit, $D_{\text{limit}} = d_{\text{limit}} + 1$
D_{v}	depth of edge server v, $D_{\mathrm{v}} = d_{\mathrm{v}} + 1$
K	depth limit, $K = D_{\text{limit}}$
G_{DT}	directed graph with cloud as the root and edges replaced by the shortest path between two nodes in graph G
$\overline{G}$	metric closure of graph G
$\overline{E}$	set of edges in the metric closure of graph G
$\overline{G}_{\mathrm{R}}$	subgraph of $\overline{G}$ induced over R
$mst(G)$	minimum spanning tree of G
$Triples$	set of all combination of 3 destination edge servers
G_{cv}	tree having cloud as the root and edges $e \in G \setminus \{c\}$
G_{opt}	optimal solution of the given EDD problem
$Cost(G)$	cost associated with any graph G
$v(z)$	centroid of a particular $Triple\ z$
$d(e)$	weight of edge e in the graph

in the Figure 3.1. Common notations used throughout the text in the paper are given in Table 3.1.

Lead cloud service providers like Google, Amazon, or Microsoft charge differently for

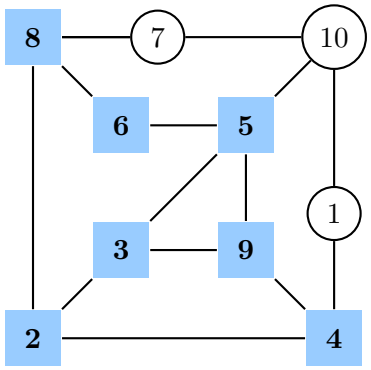

Fig. 3.1: EDD scenario with 10 edge servers

data transactions. For e.g., hosting applications in Microsoft Azure involves careful consideration of the costs generated by three main components i.e., computing, storage, and network. Similarly, for any edge infrastructure provider, the pricing models usually are different for cloud-to-edge and edge-to-edge transmissions. Thus, we need to define a relationship between these two transmissions, so that they can be incorporated easily into the generic optimization objective model. We define γ, to specify the ratio between the C2E and E2E transmission costs. For e.g., $\gamma = 20$, means the $Cost_{\text{C2E}}$ is 20 times more than $Cost_{\text{E2E}}$. The cloud-to-edge transmissions are much more expensive than the edge-to-edge transmissions mainly because the graph nodes are connected via high-speed transmission links and are also not that distant apart from each other. Since the transmission latency is measured by the number of hops it takes to reach the destination node from the source in graph G, the C2E cost can then be easily transformed to γ times of the 1-hop E2E transmission cost in the generic model. Now there are two possible scenarios for data transmissions in EDD, firstly C2E, where the data is sent from the cloud to some of the edge servers referred to as the *"initial transit edge servers"* and secondly E2E, where the data is transmitted between any two edge servers[1]. Thus, an EDD strategy comprises of two parts, a C2E and an E2E strategy.

[1]There is no necessity for initial transit edge servers to be a destination edge server and a destination edge server can also be an initial transit edge server as it may transfer data further to other servers.

- In the C2E strategy, the set $S = \{s_1, s_2, \cdots, s_{\mathrm{N}}\}$, where s_{v} $(1 \leq v \leq N)$ denotes the set of Boolean array representation of the initial transit edge servers, which receives data directly from the cloud.

$$s_{\mathrm{v}} = \begin{cases} 1, & \text{if } v \text{ is an initial transit edge server} \\ 0, & \text{if } v \text{ is not an initial transit edge server} \end{cases} \tag{3.1}$$

- Similarly, In the E2E strategy, the set $T = \{\tau_{(1,\,1)}, \tau_{(1,\,2)}, \cdots, \tau_{(\mathrm{N},\,\mathrm{N})}\}$, where $\tau_{(u,\,v)}$ $(u,\, v \in V)$ denotes whether the data is transmitted through edge $e_{(u,\,v)}$ in G.

$$\tau_{(u,\,v)} = \begin{cases} 1, & \text{if data is transmitted through } \mathrm{e}_{(\mathrm{u},\,\mathrm{v})} \\ 0, & \text{if data is not transmitted through } \mathrm{e}_{(\mathrm{u},\,\mathrm{v})} \end{cases} \tag{3.2}$$

  Since a reasonable EDD solution must link each destination edge server $v \in R$ to an initial transit edge server in S through edges in E, thus,

$$\mathit{Connected}(S,\ T,\ u,\ v) = \mathit{true},\ \forall v \in R,\ \exists\, u, s_{\mathrm{u}} = 1 \tag{3.3}$$

Now, the app-vendor has the liberty to regulate the EDD latency constraint as per the application specifics and requirements. Let d_{limit} represent the app vendors EDD time constraint. Thus, for each of the destination edge server v, edge-to-edge latency or data transmission latency or path length in hops between v and its parent initial transit edge server in S, i.e., d_{v} should not exceed this time or depth constraint.

$$0 \leq d_v \leq d_{limit},\, d_v \in Z^{+}, \forall v \in R \tag{3.4}$$

For example, consider the graph in Figure 4.2, the nodes of the graph represent the edge servers given by $V = \{1,\, 2,\, 3,\, 4,\, 5,\, 6,\, 7,\, 9\}$, where number of edge servers $N = 9$ and the

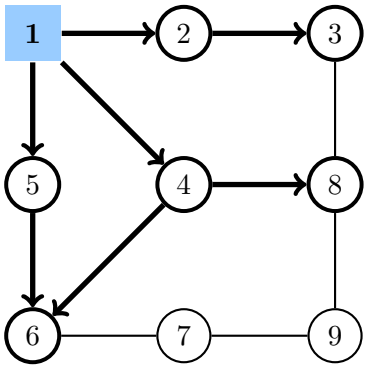

Fig. 3.2: EDD example to demonstrate d_{limit}

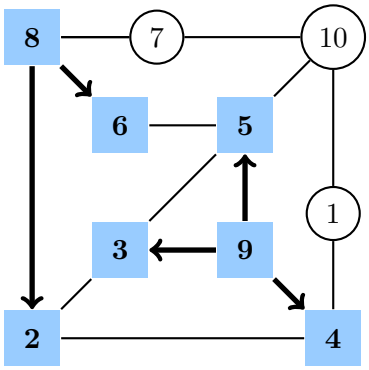

Fig. 3.3: Optimal solution using integer programming

set of high-speed links i.e., $E = \{e_{(1,\,2)},\, e_{(1,\,3)},\, e_{(1,\,4)},\, \cdots,\, e_{(7,\,9)}\}$. Now, let the EDD time constraint as per the vendor's application requirements be $d_{\text{limit}} = 2$, which means that it should take less than or equal to two hops for a destination edge server to receive the data from any initial transit edge server. In Figure 4.2, if node 1 is the only node selected as the initial transit edge server, then one possible E2E strategy is to select edges $\{e_{(1,\,2)},\, e_{(1,\,4)},\, e_{(1,\,5)},\, e_{(5,\,6)},\, e_{(2,\,3)},\, e_{(4,\,6)},\, e_{(4,\,8)}\}$. This means that in the set T_{E2E} the value of $\tau_{(1,\,2)} = \tau_{(1,\,4)} = \tau_{(1,\,5)} = \tau_{(2,\,3)} = \tau_{(4,\,6)} = \tau_{(5,\,6)} = \tau_{(4,\,8)} = 1$. Accordingly, we can also obtain the time constraint limit of each of these destination edge servers, i.e, $d_1 = 0$, $d_2 = d_4 = d_5 = 1$, and $d_6 = d_8 = d_3 = 2$.

Now, given an EDD latency constraint of d_{limit}, the app vendor's aim is to minimize the EDD cost, which consists of the part incurred by the C2E transmissions and the E2E transmissions.

Thus, the Edge Data Distribution (EDD) problem can be formulated as —

$$minimize\left\{Cost_{\text{C2E}}(S) + Cost_{\text{E2E}}(T)\right\} \tag{3.5}$$

while fulfilling the EDD latency constraint in Equation 4.6. As we can see, the Edge Data Distribution (EDD) is a constrained optimization problem based on a well-known NP-hard problem in graph theory known as the Steiner Tree [XCH+21].

3.2 Solution Strategy

In this section, we first present a refined formulation of the optimization approach given in Xia et al. [XCH+21] where the Edge Data Distribution (EDD) is formulated as a constrained integer programming problem which can then be solved by using any simple IP solver[2]. Since this method takes an exponential amount of time to produce results for large datasets, we designed an $O(k)$ approximation method, named EDD-NSTE that can estimate solutions to the EDD problems in polynomial time. Also, we did the theoretical analysis of the performance of the proposed EDD-NSTE based solution approach.

3.2.1 Edge Data Distribution (EDD) as an Integer Programming

The Edge Data Distribution (EDD) can be formulated as a constrained integer programming problem as given in [XCH+21]. Our refined formulation of the same is specified as below.

The solution for the EDD should minimize the cost incurred for data transmissions within the app vendor's latency constraint of d_{limit}. Thus, to model the EDD problem as a generalized constrained integer programming optimization problem firstly we add the cloud server c into V, and then add the edges from cloud c to each edge server in graph G. Now, we can formulate the C2E strategy of EDD, $S = \{s_1,\ s_2,\ s_3\ \cdots\ s_{\text{N}})$ by selecting edges in graph G, such that, $\text{T}_\text{c} = (\tau_{(\text{c},\ 1)},\ \tau_{(\text{c},\ 2)},\ \cdots,\ \tau_{(\text{c},\ \text{N})}\}$, where $\tau_{(\text{c},\ \text{v})}$ denotes if the data is transferred from cloud server c to the edge server v. Here, we combine the C2E and E2E strategy of EDD into one parameter

$$\begin{gathered} T = \{\tau_{(\text{c},\ 1)}, \tau_{(\text{c},\ 2)} \cdots \tau_{(\text{c},\ \text{N})}, \tau_{(1,\ 1)}, \cdots, \tau_{(\text{N},\ \text{N})}\}, \\ \textit{where},\ \tau_{(\text{u},\ \text{v})} \in (0,1)\ (u \in V, v \in V \setminus \{c\}) \end{gathered} \tag{3.6}$$

[2]IP solvers are the frameworks or tools used to solve integer programming problems. For E.g., GLPK, LP Solve, CLP, CBC, CVXOPT, SciPy, Gurobi Optimizer, CPLEX, XPRESS, MOSEK, Google OR Tools, PuLP, etc.

which indicates whether the edge $e_{(\mathrm{u,\ v})}$ is included in T. Secondly, we need to update the definition of EDD time constraint as the depth of the edge server in the graph with root as cloud. So, D_v represents the depth of the edge server v, where $D_v = d_v + 1$ and $D_c = 0$, as the cloud is the root of the graph. Thus, we can define the limit as

$$D_{\text{limit}} = d_{\text{limit}} + 1 \tag{3.7}$$

Also, we need to define a parameter for each of the edge servers,

$$H_{\mathrm{v}} = \begin{cases} 0, & \text{if v is not visited during the EDD process} \\ 1, & \text{if v is visited during the EDD process} \end{cases} \tag{3.8}$$

here, $H_{\mathrm{v}} = 1, \forall v \in R$ make sure that it include the destination edge servers into our solution. Also, $H_c = 1$, as cloud always be a part of the solution. Now, the constrained integer programming optimization problem can be formally expressed as follows,

$$min \left(\gamma \sum_{v \in V \setminus \{c\}} \tau_{(\mathrm{c,\ v})} \ + \sum_{u,v \in V \setminus \{c\}} \sum \tau_{(\mathrm{u,\ v})} \right) \tag{3.9}$$

subjected to constraints,

$$H_{\mathrm{v}} = 1, \forall v \in R \cup \{c\} \tag{3.10}$$

$$\tau_{(\mathrm{u,\ v})} \leq H_{\mathrm{u}} \cdot H_{\mathrm{v}}, \forall u \in V, v \in V \setminus \{c\} \tag{3.11}$$

$$\sum \tau_{(\mathrm{u,\ v})} = H_{\mathrm{v}}, \forall u \in V, v \in V \setminus \{c\} \tag{3.12}$$

$$\sum_{v \in V \setminus \{c\}} \tau_{(\mathrm{c,\ v})} \geq 1 \tag{3.13}$$

$$\tau_{(\mathrm{u,\ v})}, H_{\mathrm{v}} \in (0, 1) \tag{3.14}$$

$$\tau_{(\mathrm{c,\ v})} = 1 \ \forall v \in V \setminus \{c\}, D_{\mathrm{v}} = 1 \tag{3.15}$$

$$D_{\mathrm{c}} = 0, D_{\mathrm{c}} \leq D_{\mathrm{v}} \leq D_{\mathrm{limit}}, \forall v \in V \setminus \{c\} \tag{3.16}$$

$$D_{\mathrm{v}} - D_{\mathrm{u}} = 1, \forall u, v \in V, \tau_{(\mathrm{u},\ \mathrm{v})} = 1 \tag{3.17}$$

Here, in-order to minimize the total cost incurred by both the C2E and E2E edges given by Equation 4.10, we must satisfy the above constraints. Equation 4.13 make sure that, if edge $e_{(\mathrm{u},\ \mathrm{v})}$ is considered a part of the solution, or, $\tau_{(\mathrm{u},\ \mathrm{v})} = 1$, then both the nodes u and v must be visited, i.e., $H_{\mathrm{v}} = 1$ and $H_{\mathrm{u}} = 1$. Equation 4.14 suggests a constraint that the summation of $\tau_{(\mathrm{u},\ \mathrm{v})}$ of the edges incoming at the particular node v, must be equal to the H_{v} of that particular node, i.e., whether or not the node v is visited, if edges incoming onto it are considered a part of the solution. Equation 4.15 suggests that the solution must have atleast one cloud server, i.e., summation of $\tau_{(\mathrm{c},\ \mathrm{v})}$ must be greater than or equal to 1. Equation 3.15 makes sure that the node servers having depth of 1, must be connected to the cloud server. Equation 4.17 suggests that the depth of any edge server v, must be within $[0, d_{\mathrm{limit}}]$, and Equation 4.18 suggests that for nodes u, v connected through edge $e_{(\mathrm{u},\ \mathrm{v})}$ in the EDD solution must satisfy the depth difference of 1.

Now, this Integer Programming model is a generalized model and thus, for vendor's specific cost models say *cost(c, v)* and *cost(u, v)* can be simply incorporated into our Equation 4.10, as

$$min \left(\sum_{v \in V \setminus \{c\}} \tau_{(\mathrm{c},\ \mathrm{v})} \cdot cost_{(\mathrm{c},\ \mathrm{v})} \ + \sum_{u,v \in V \setminus \{c\}} \sum \tau_{(\mathrm{u},\ \mathrm{v})} \cdot cost_{(\mathrm{u},\ \mathrm{v})} \right) \tag{3.18}$$

Here, as we know $D_{\mathrm{limit}} \geq 1$, as otherwise the cloud cannot transfer data to any edge server, and for the base case of $D_{\mathrm{limit}} = 1$, the cost incurred will be γR times the cost of 1-hop E2E transmission.

Here in the refined formulation Equation 4.15, Equation 4.16, and Equation 3.15 are the new constraints added to the formulation of the integer programming method given

in [XCH+21]. These constraints narrow the possible solution search-space and also updates the final solution to contain all the edges, i.e., the C2E edges and the E2E edges.

Example to understand Integer Programming

Consider an example in Figure 4.3 that displays the optimal solution generated, if we solve the graph in Figure 3.1 using the integer programming approach, with EDD time constraint of $d_{limit} = 1$. The C2E strategy specify *node 8* and *node 9* as the initial transit edge servers which receives the data directly from the cloud server and thus, the cost incurred for the cloud to edge data transmission is $Cost_{C2E} = 2\gamma$ times of the 1-hop E2E transmission cost. Now, these nodes transmit the data to the other destination edge servers (represented by light blue square nodes) that can be reached within the given time constraint of d_{limit}. Thus, the cost incurred for the edge to edge data transmission is $Cost_{E2E} = 5$ times of the 1-hop E2E transmission cost. The integer programming approach selects the edges E = $\{e_{(c,\,8)}, e_{(c,\,9)}, e_{(8,\,2)}, e_{(8,\,6)}, e_{(9,\,3)}, e_{(9,\,4)}, e_{(9,\,5)}\}$, for data transmissions and thus, the optimal total cost incurred is the sum of $Cost_{C2E}$ and $Cost_{E2E}$, i.e., $2\gamma + 5$ times of the 1-hop E2E transmission cost.

3.2.2 Edge Data Distribution (EDD) as a Network Steiner Tree Estimation

The Edge Data Distribution (EDD) is an NP-Hard [XCH+21] problem. Thus, finding an optimal solution for large-scale EDD scenarios is troublesome due to the exponential time complexity of the algorithm. To address this issue, we propose an approach, named Edge Data Distribution as a Network Steiner Tree Estimation (EDD-NSTE), to estimate large-scale EDD solutions. The estimation ratio for EDD-NSTE is $O(k)$, which means that the ratio of the cost by EDD-NSTE solution and that of the optimal solution is $O(k)$ in the worst case, where k is constant. In our proposed approach, we calculates the estimated cost incurred for a given graph in two steps —

- First of all, we calculate a network Steiner tree approximation [Zel93a] with destina-

Algorithm 1: Network Steiner Tree Approximation

F $\leftarrow \overline{G}_{\mathrm{R}}$; W $\leftarrow \phi$; $Triples \leftarrow \{z \subset \mathrm{R} : \|z\| = 3\}$
foreach $z \in Triples$ **do**
    find v which **minimizes** the $\sum_{s \in z} d(v, s)$
    v(z) $\leftarrow$ v
    d(z) $\leftarrow \sum_{s \in z} d(v(z), s)$
while *true* **do**
    M $\leftarrow$ an MST of F; $FindSave(M)$
    find z $\in Triples$ which **maximizes**, $win = \max_{e \in z} save(e) + \min_{e \in z} save(e)$ - d(z)
    if $win \leq 0$ **then**
        break
    F $\leftarrow$ F[z]; put(W, v(z))
Find a steiner tree for R $\cup$ W in graph G using MST algorithm

tion edge servers R, as the set of distinguished vertices, in the given graph G.

- Then, we use an approach which estimates a rooted minimum Steiner tree, by adding cloud c to the graph, and then slicing and fine-tuning the rooted Steiner tree with the vendor's latency constraint of D_{limit}.

The proposed approach works as follows: Given a graph G, we first calculate the metric closure of the graph, denoted by $\overline{G}$, which is a complete graph of G, having edge lengths equal to the shortest distance between the vertices in G. $\overline{G}_{\mathrm{R}}$ denotes the subgraph of $\overline{G}$ induced by a vertex subset $R \subseteq V$, i.e., the subgraph on destination edge servers. We denote the minimum spanning tree (MST) of any graph G as $mst(G)$. For any triple set of vertices $z = (u, v, w)$, the graph $G[z]$ represent a graph resulting from contraction of

Algorithm 2: To calculate $FindSave(M)$

if $M \neq$ *empty* **then**
    $e \leftarrow$ max weight edge in M;
    $x \leftarrow d(e)$
    M_1, $M_2 \leftarrow$ the subgraphs linked through e
    foreach *vertex* v_1 *of* M_1 *and* v_2 *of* M_2 **do**
        $save(\overline{e}) \leftarrow x$, where $\overline{e} = e_{(v_1, v_2)}$
    $FindSave(M_1)$; $FindSave(M_2)$

Algorithm 3: EDD-NSTE Algorithm

Input: G, G_{DT}, D_{limit}, depth, cost, visited
Output: G_{final}

```
Construct the graph G_ST from Algorithm 1. Take the maximum connectivity node
  in the graph G_ST and connect it to the cloud to make it a rooted tree. Update the
  distances between consecutive nodes by the shortest path between them and let
  this graph be G_DT
Initialize depth[c] ← 0, visited[c] ← 1, depth[v] ← ∞, visited[v] ← 0, ∀ v ∈ V,
  D_limit ← d_limit + 1, Update G_DT to include the minimum cost path between any
  two vertices with edges in graph G
Update G_DT to form a directed rooted tree, with cloud as the root
Run a BFS to compute the minimum depth of each vertex v ∈ G_DT from cloud
foreach vertex v, in the path from root to leaf node and parent p in Depth-First
  Search Algorithm do
    if visited[v] ≠ 1 then
        if depth[v] ≤ D_limit and v ∈ R then
            add edge e_(p, v) in G_final
            visited[v] ← 1
        else if v ∉ R then
            add edge e_(p, v) in G_final
            visited[v] ← 1
        else if depth[v] > D_limit and v ∈ R then
            depth[v] ← 1, visited[v] ← 1
            add edge e_(c, v) in G_final
            foreach child node u of node v in graph G in DFS order do
                if depth[u] > depth[v] + 1 and depth[v] + 1 ≤ D_limit then
                    if e(v, u) ∉ G_DT then
                        update edge in G_DT from v to u
                        depth[u] ← depth[v] + 1
            run BFS and update depths of the vertices in G_DT
Run a DFS and remove unnecessary edges from graph G_final
Calculate total cost incurred as sum of cost[e], ∀ e ∈ G_final
```

two edges i.e., $e_{(u,\,v)}$ and $e_{(v,\,w)}$. *Triples* denotes the set of all combination of 3 destination edge servers, $v(z)$ denotes the centroid of a particular triple, i.e., the vertices other than the destination edge servers whose sum of distance from a triple is minimum and $d(e)$ denotes the weight of the edge e.

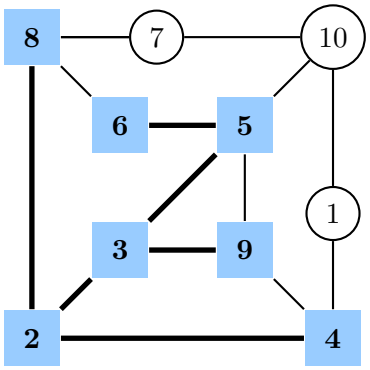

Fig. 3.4: Steiner Tree estimated by Algorithm 1

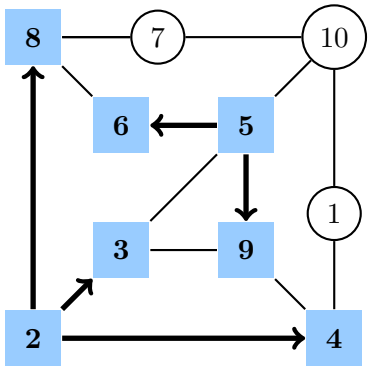

Fig. 3.5: Final EDD solution estimated by Algorithm 3

The Pseudo code of the proposed approach is given in Algorithm 1, which loops continuously to find the best possible reduction in the cost of the minimum spanning tree of F, where the initial value of $F = \overline{G}_{\mathrm{R}}$. The algorithm does so by adding the three edges of G having a common end node, i.e., the centroid, and consecutively removing the max-weight edge from each resulting cycle using the *save* matrix, from the MST of F. This class of algorithms is also known as the Triple Loss Contracting Algorithm [GHNP01b] [Zel93a]. The output of Algorithm 1 is an approximated Steiner tree G_{ST}, with vertices in R $\cup$ W and edges $e \subset \overline{E}$.

Here, we calculate *save* matrix for edges in the graph F, to estimate the maximum win for a given triple. To calculate the *save* matrix, a pseudo code for a recursive function $FindSave(M)$ is given in Algorithm 2. The implementation time complexity of this algorithm is $O(\|R\|^{2})$.

$$save(\overline{e}) = mst(F) - mst(F[\overline{e}]), \overline{e} = e_{(v_1,\, v_2)}, \forall\, v_1, v_2 \in F \tag{3.19}$$

Once we get an approximated Steiner tree, it is necessary to add cloud server c to the given graph by adding an edge from cloud to the maximum connectivity node in the approximated Steiner tree. Then we convert resulting graph into a rooted tree, with cloud as the root. We perform the same by a breadth-first search on G_{ST}. In this process, we also convert the edges between any two nodes of G_{ST} by the shortest path between two nodes

in graph G, as the Steiner tree G_{ST}, was approximated over the metric closure of graph G. Let this resultant directed graph be G_{DT}.

The pseudo code for slicing and fine-tuning the graph to satisfy the vendor's latency constraint is shown in Algorithm 3. In this approach, we take the directed Steiner tree G_{DT} and initialize the $depth[]$ and the $visited[]$ for each node of the graph, with $visited[c] = 1$. It then runs the Breadth-First Search to compute the minimum depths of each node assuming the cloud server as the root. After that, it runs the Depth-First Search and —

- for each unvisited destination edge server with depth less than or equal to D_{limit}, it adds the corresponding edge into the final solution G_{final}.
- for each unvisited edge server, which is not a destination edge server, it again adds the corresponding edge into the final solution G_{final}.
- for each unvisited destination edge server having a depth greater than D_{limit}, it connects that edge server directly to the cloud and adds edge from cloud to this edge server in G_{final}. It then runs a Depth-First Search to update the minimum depths of the remaining unvisited edge servers assuming this edge server as the root.

Finally, this algorithm removes the unnecessary edges from the graph G_{final} and calculates the cost of the solution as the sum of the cost of edges in G_{final}.

Example to understand EDD-NSTE algorithm

Let us consider the graph in the Figure 3.1, with vendors' latency constraint or hop-limit of $d_{\mathrm{limit}} = 1$. When we apply Algorithm 1 onto it, it selects edges $\{e_{(2,\,3)}, e_{(2,\,4)}, e_{(2,\,8)}, e_{(3,\,5)}, e_{(3,\,9)}, e_{(5,\,6)}\}$ to construct the estimated Steiner tree given in Figure 4.4, using the triple loss contracting mechanism, which in this case is also the optimal Steiner tree possible. Now, as we can see the *node 2* is the node with the max-connectivity in the estimated steiner tree, thus, we add this node to the cloud directly, as given in Figure 4.5 to make a rooted EDD solution. The *node 3*, *node 4*, and *node 8* can be reached within the 1-hop

distance from *node 2*, thus edges $\{e_{(c,\,2)},\ e_{(2,\,3)},\ e_{(2,\,4)},\ e_{(2,\,8)}\}$ are included into the final EDD solution. When the algorithm reaches *node 3*, it encounters two nodes, i.e., *node 5* and *node 9* exceeding the hop-limit. Now, since *node 5* is encountered first, the algorithm adds *node 5* to the cloud directly, which then updates the final EDD solution to include *node 6* and *node 9*, as they can be reached within vendors' latency constraint with *node 5* as the root. The final EDD solution then becomes $\{e_{(c,\,2)}, e_{(c,\,5)}, e_{(2,\,3)}, e_{(2,\,4)}, e_{(2,\,8)}, e_{(5,\,6)}, e_{(5,\,9)}\}$, which incurs a total cost of $2\gamma + 5$, including both the C2E and E2E edges.

Time Complexity analysis of EDD-NSTE algorithm

There has been a wealth of research in approximating the best Steiner Tree for a given general graph as mentioned in Gropl et al. [GHNP01b], with a lower bound of smaller than 1.01 [GHNP01a] theoretically. In this paper, we first introduced an algorithm to calculate the Network Steiner Tree approximation, based on the algorithm proposed in Zelikovsky et al. [Zel93a] which has a time complexity of $O(\|V\|\|E\| + R^4)$ and an approximation ratio of $O(11/6)$. The same approximation ratio can be achieved with a faster implementation of $O(\|R\|\,(\|E\| + \|V\|\|R\| + \|V\|\,log\|V\|))$ as mentioned in [Zel93b].

The algorithm EDD-NSTE has a time complexity of $O(\|V\| + \|E\|)$ in the worst case. Thus, the total time complexity of the EDD-NSTE Algorithm along with network Steiner tree estimation is $O((\|V\|+\|E\|)+\|V\|\|E\|+\|R\|^4)$ and in the worst case the time complexity becomes $O(\|V\|^4)$.

In the next subsection, we prove that the EDD-NSTE is an $O(k)$ approximation algorithm with experimental results suggesting that EDD-NSTE significantly outperforms the other three representative approaches in comparison with a performance margin of 86.67% (as per our experimental result given Section V).

EDD-NSTE is an O(k) approximation algorithm

As we discussed in the previous section, let tree G_{final} be the final EDD solution produced by Algorithm 3, G_{ST} be the minimum cost steiner tree approximated by Algorithm 1, and G_{cv} be the tree having cloud c as the root and edges $e_{(\text{c, v})}$, $\forall$ v $\in G_{\text{final}} \setminus \{c\}$. Also, let the cost associated with each of these trees be $Cost(G_{\text{final}})$, $Cost(G_{\text{ST}})$, and $Cost(G_{\text{cv}})$ respectively.

Let G_{opt} be the optimal solution for the given EDD problem and the optimal cost associated be $Cost(G_{\text{opt}})$, thus, as per the research in Zelikovsky et al. [Zel93a], we can write

$$Cost(G_{\text{ST}}) = \frac{11}{6} Cost(G_{\text{opt}}) \tag{3.20}$$

Now, let $v_{\text{o}} = c$ be the cloud server and v_{i} as the i^{th} edge server, where $i \in \{1, 2, \cdots, m\}$, which brings extra paths during the DFS iteration. Also, let $K = D_{\text{limit}}$ be the depth limit provided by the vendor. Thus, for any edge server there exists two possibilities, i.e.,

- v_{i} exceeds the latency constraint of D_{limit} and thus, we need to add an extra edge (c, v_{i}), which in turn adds a cost of $Cost_{(\text{G}_{\text{cv}})}(c, v_{\text{i}}) = \gamma$.
- $\exists$ a path (c, v_{i}) as a path from $(c, v_{\text{i-1}})$ and $(v_{\text{i-1}}, v_{\text{i}})$, such that v_{i} does not exceed the latency constraint, which in turn adds a cost of $Cost_{(\text{G}_{\text{final}})}(c, v_{\text{i}}) \leq Cost_{(\text{G}_{\text{cv}})}(c, v_{\text{i-1}}) + Cost_{(\text{G}_{\text{ST}})}(v_{\text{i-1}}, v_{\text{i}})$.

In the first case, when edge server v_{i} exceeds the latency constraint, then we must have

$$Cost_{(\text{G}_{\text{ST}})}(c, v_{\text{i}}) \geq \gamma + K \geq \left(1 + \frac{K}{\gamma}\right) Cost_{(\text{G}_{\text{cv}})}(c, v_{\text{i}}) \tag{3.21}$$

where, K denotes the latency constraint D_{limit}. Thus, we can obtain the combined equation as

$$\begin{gathered} \left(1 + \frac{K}{\gamma}\right) Cost_{(\text{G}_{\text{cv}})}(c, v_{\text{i}}) \leq \\ Cost_{(\text{G}_{\text{ST}})}(c, v_{\text{i}}) \leq Cost_{(\text{G}_{\text{cv}})}(c, v_{\text{i-1}}) + Cost_{(\text{G}_{\text{ST}})}(v_{\text{i-1}}, v_{\text{i}}) \end{gathered} \tag{3.22}$$

Summing the above equation for all of the m edge servers, i.e., v_{i}, where $i \in \{1, 2, ...m\}$, we have

$$\left(1 + \frac{K}{\gamma}\right) \sum_{i=1}^{m} Cost_{(\mathrm{G_{cv}})}(c, v_{\mathrm{i}}) \leq \sum_{i=1}^{m} Cost_{(\mathrm{G_{cv}})}(c, v_{\mathrm{i\text{-}1}}) + \sum_{i=1}^{m} Cost_{(\mathrm{G_{ST}})}(v_{\mathrm{i\text{-}1}}, v_{\mathrm{i}}) \tag{3.23}$$

Now, as we know from the basic inequality of positive numbers, that $\sum_{i=1}^{m-1} Cost_{(\mathrm{G_{cv}})}(c, v_{\mathrm{i}}) \leq \sum_{i=1}^{m} Cost_{(\mathrm{G_{cv}})}(c, v_{\mathrm{i}})$, so the above equation yields —

$$\frac{K}{\gamma} \sum_{i=1}^{m} Cost_{(\mathrm{G_{cv}})}(c, v_{\mathrm{i}}) \leq \sum_{i=1}^{m} Cost_{(\mathrm{G_{ST}})}(v_{\mathrm{i\text{-}1}}, v_{\mathrm{i}}) \tag{3.24}$$

$$\frac{K}{\gamma} Cost(G_{\mathrm{cv}}) \leq \sum_{i=1}^{m} Cost_{(\mathrm{G_{ST}})}(v_{\mathrm{i\text{-}1}}, v_{\mathrm{i}}) \tag{3.25}$$

For each edge server v_{i}, that changes its path in G_{final} during the DFS traversal without adding the path (c, v_{i}), the total cost of update is less than that of $Cost_{(\mathrm{G_{cv}})}(c, v_{\mathrm{i}})$ and thus, the cost after forming G_{ST} must not be more than $\sum_{i=1}^{m} Cost_{(\mathrm{G_{cv}})}(c, v_{\mathrm{i}})$. Now, each edge in G_{ST} is traversed at most twice during the DFS traversal, thus, we have

$$\sum_{i=1}^{m} Cost_{(\mathrm{G_{ST}})}(v_{\mathrm{i\text{-}1}}, v_{\mathrm{i}}) \leq 2 \cdot Cost(G_{\mathrm{ST}}) \tag{3.26}$$

Thus, combining both above equations, results in

$$Cost(G_{\mathrm{cv}}) \leq \frac{2\gamma}{K} Cost(G_{\mathrm{ST}}) \tag{3.27}$$

Finally, as we know, that the cost of the final EDD solution approximated by EDD-NSTE is

$$Cost(G_{\mathrm{final}}) \leq Cost(G_{\mathrm{cv}}) + Cost(G_{\mathrm{ST}}) \tag{3.28}$$

$$Cost(G_{\mathrm{final}}) \leq \frac{2\gamma}{K} Cost(G_{\mathrm{ST}}) + Cost(G_{\mathrm{ST}}) \tag{3.29}$$

$$Cost(G_{\text{final}}) \leq \left(\frac{2\gamma}{K} + 1\right) Cost(G_{\text{ST}}) \tag{3.30}$$

$$Cost(G_{\text{final}}) \leq \frac{11}{6}\left(\frac{2\gamma}{K} + 1\right) Cost(G_{\text{opt}}) \tag{3.31}$$

Now, since K and γ are both constants, we can safely assume $k = \frac{11}{6}\left(\frac{2\gamma}{K} + 1\right)$, and thus, EDD-NSTE is an $O(k)$ approximation algorithm.

As compared to Xia et al. [XCH+21], our approximation is lightly better as $\frac{11}{6}\left(\frac{2\gamma}{K} + 1\right) \leq 2\left(\frac{2\gamma}{K} + 1\right)$. But in actual run this produce even better result as compared to EDD-A [XCH+21].

3.3 Experimental Evaluation

We conducted experiments to evaluate the effectiveness of the proposed EDD-NSTE approach in comparisons with the state of art EDD-A [XCH+21], refined LP solution and other representative approaches. All these experiments were conducted on an Ubuntu 20.04.1 LTS machine equipped with Intel Core i5-7200U processor (4 CPU's, 2.50GHz) and 16GB RAM. The integer programming simulation was done on Google Collab (Python 3 Google Computer Engine Backend, 12.72GB RAM).

3.3.1 Simulation Settings and Approaches for comparison of the result

Description of the state of art solution technique (EDD-A) and other representative approaches are —

- **Greedy Connectivity (GC)** — In this approach, for each node, the approach define its connectivity which is equal to the number of destination servers that have not received the data yet and are reachable within the vendors' latency constraint. Then it select the node with the maximum connectivity and make it an initial transmit edge server which then transmits the data to other servers within the app vendors' latency limit of d_{limit} until all destination edge servers have received the data.

- **Random** — In this approach, it randomly select the initial transit edge servers which directly receive data from the cloud and then transmit it to the other servers within the app vendors' latency limit of d_{limit}, until all destination edge servers have received the data.

- **EDD Integer Programming** — As discussed above in the Section IV, we refine the formulation of given EDD problem into a 0-1 integer programming problem which can then be solved by using any simple IP solvers. This method take a huge amount of time to produce the result in the case of large datasets as is exponential in complexity.

- **EDD-A Algorithm** — This is state of the art approach given in [XCH+21], in this approach it first calculate a connectivity-oriented minimum Steiner tree (CMST) using an $O(2)$ approximation method. This method calculates a minimum Steiner tree having cost at most twice the optimal Steiner tree on the graph. It then uses a heuristic algorithm to calculate the minimum cost of transmission incurred using E2E and C2E edges in the CMST. This algorithm is proved to have an $O(k)$ approximation solution and a time complexity $O(\|V\|^3)$, in the worst case.

3.3.2 Dataset Used

The experiments were performed on a standard real-world EUA dataset [3], which contains the geo-locations of more than 1,400 edge-server base stations of Melbourne, Australia. The set of destination edge servers R is generated randomly. The edges between the nodes of the graph are also generated randomly using an online random connected graph model generator tool. In the experiments, the value of γ is set to 20 which is same setting as specified in [XCH+21].

[3] https://github.com/swinedge/eua-dataset

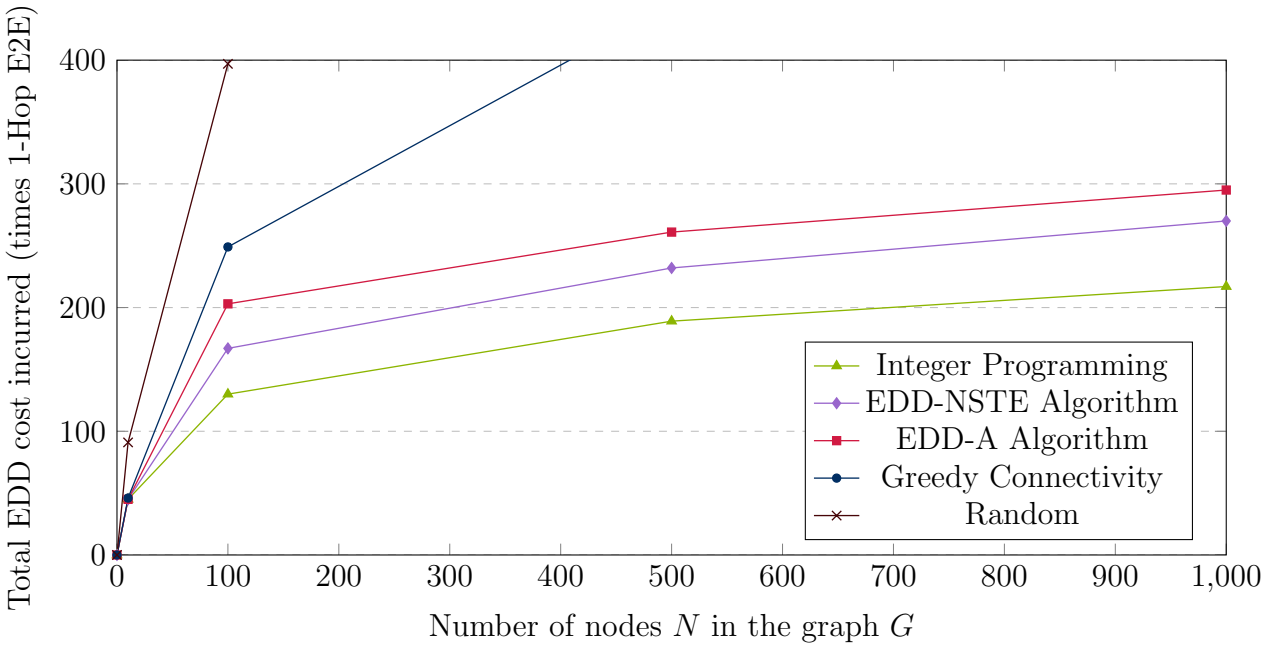

Fig. 3.6: N v/s Total EDD cost

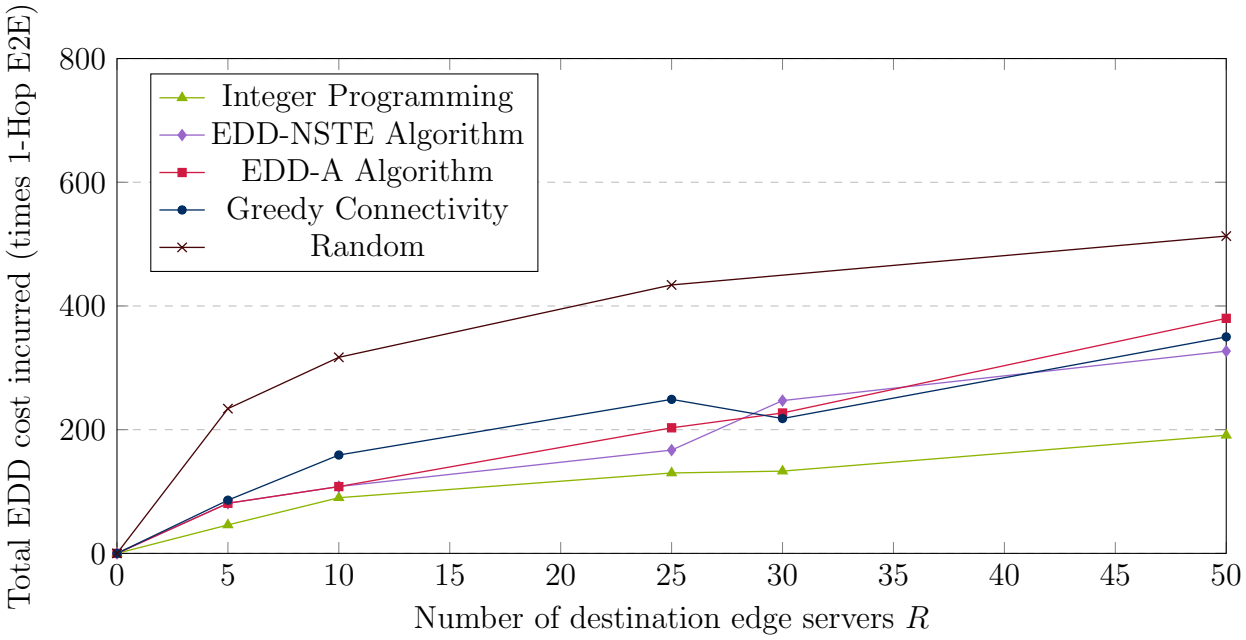

Fig. 3.7: R v/s Total EDD cost

3.3.3 Experiment Results

Different EDD scenarios and algorithms mentioned above in the section solution strategy and other approaches are simulated and compared to each other by varying different

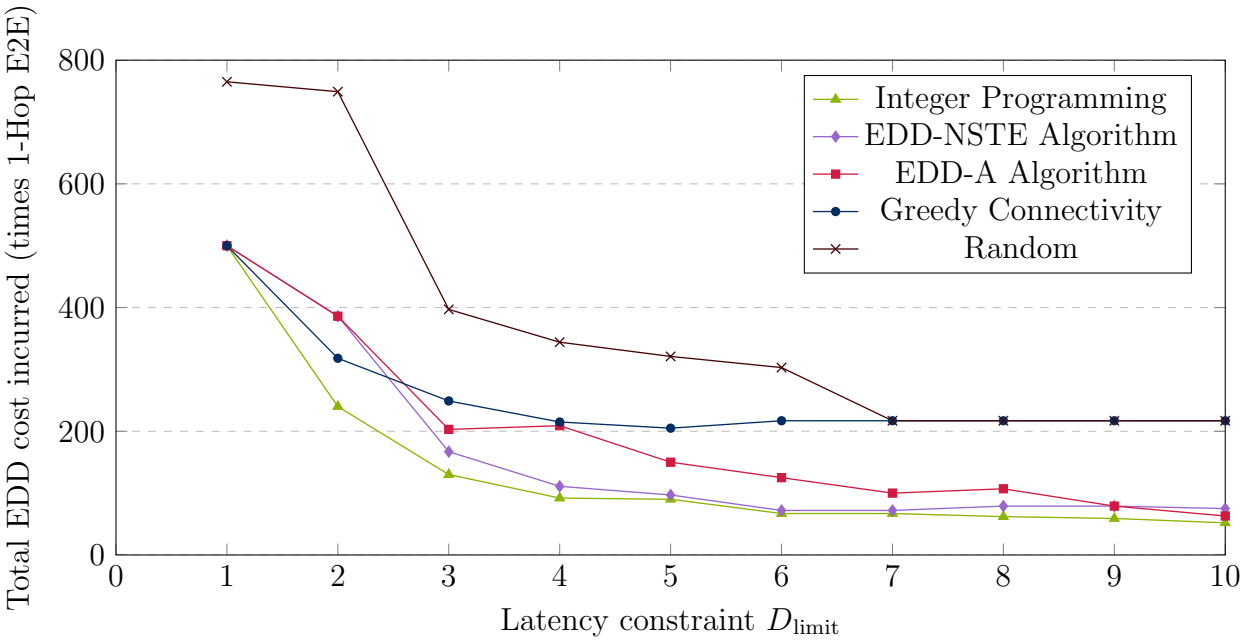

Fig. 3.8: Latency constraint D_{limit} v/s Total EDD cost, lower is better

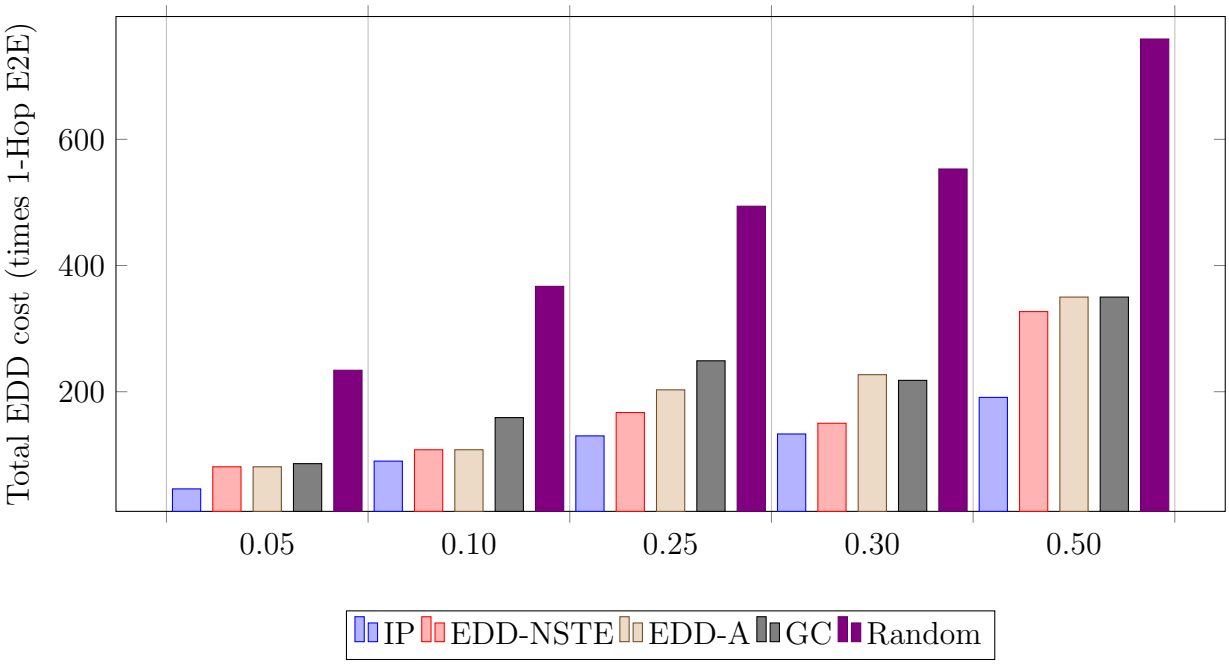

Fig. 3.9: ρ v/s Total EDD cost

parameters as mentioned below —

- The number of nodes N in the graph G.
- The number of destination edge servers R in the graph.

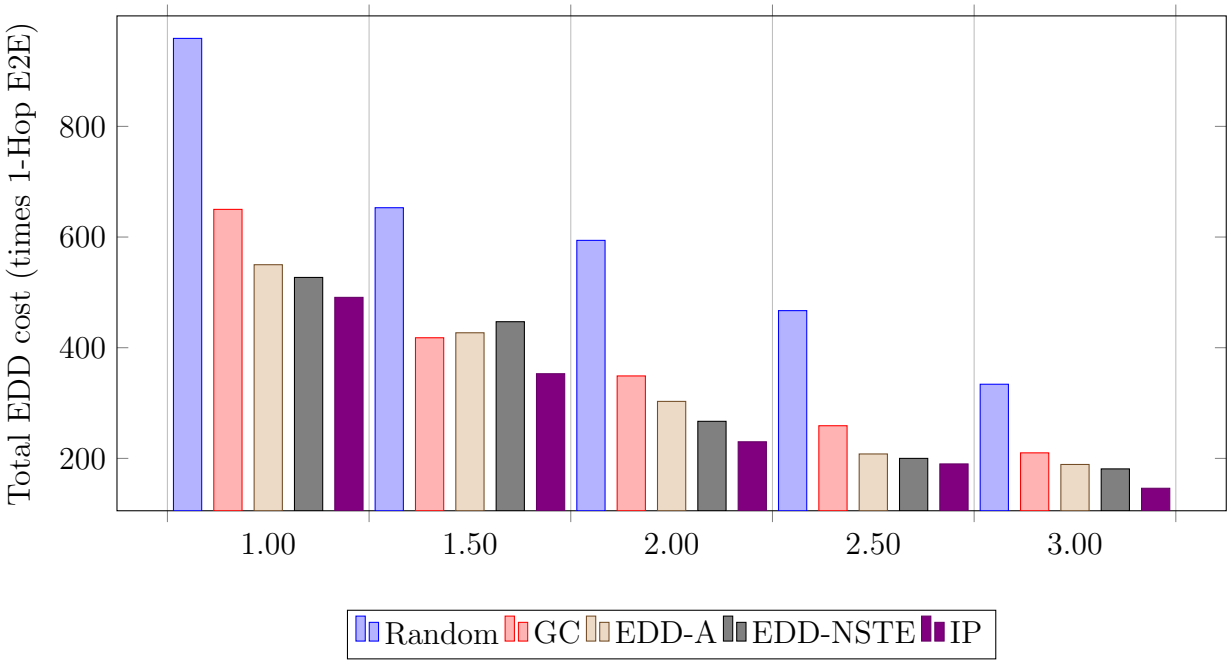

Fig. 3.10: δ v/s Total EDD cost, lower is better

- The vendor's latency constraint limit, i.e., D_{limit}.

Among the above three factors, we vary one of the factors while keeping the others constant and then compare the estimated total EDD cost incurred in each case with the optimal cost generated by the integer programming approach.

Figure 4.8, displays the relationship between the number of nodes N in the graph G v/s total EDD cost incurred by different simulation settings and approaches. The value of the other parameters such as $D_{\text{limit}} = 3$, $R = 25$ random nodes, and $\gamma = 20$ were fixed during the experiment. For $N < 100$, there is a very slight difference between other approaches but as the value of N increases the difference becomes significantly more. Resultant cost produced by EDD-NSTE approach is lower (is better) as compared to EDD-A in all the cases.

Similarly, Figure 4.9, displays the relationship between the number of destination edge servers R v/s the total EDD cost incurred by different other settings. For this experiment the value of other parameters such as $N = 100$, $D_{\text{limit}} = 3$ and $\gamma = 20$ were fixed during the experiment. For $\|R\| \leq 10$, the EDD cost incurred in all approaches except the Random

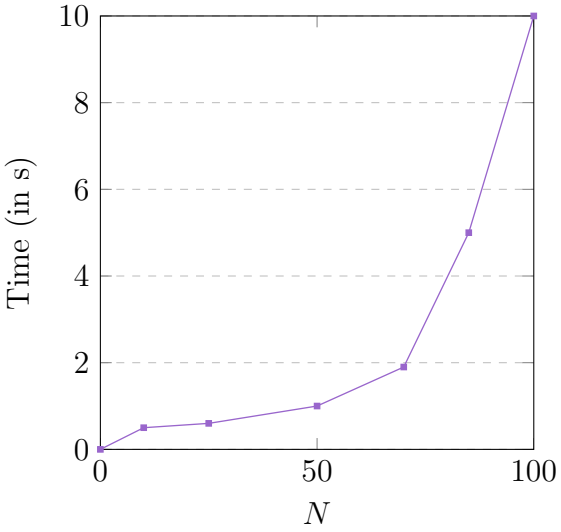

Fig. 3.11: N v/s Computational Overhead, lower is better

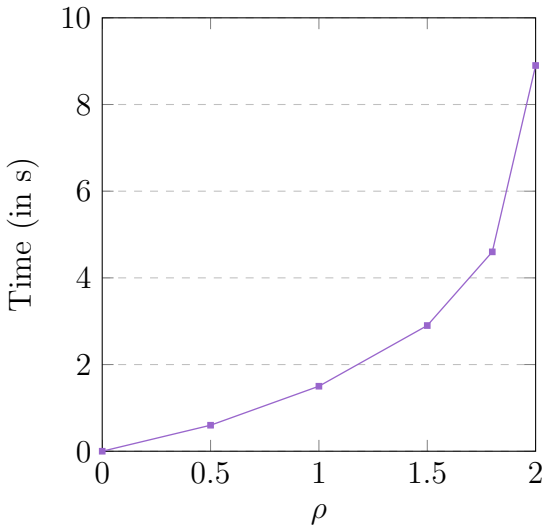

Fig. 3.12: ρ v/s Computational Overhead,lower is better

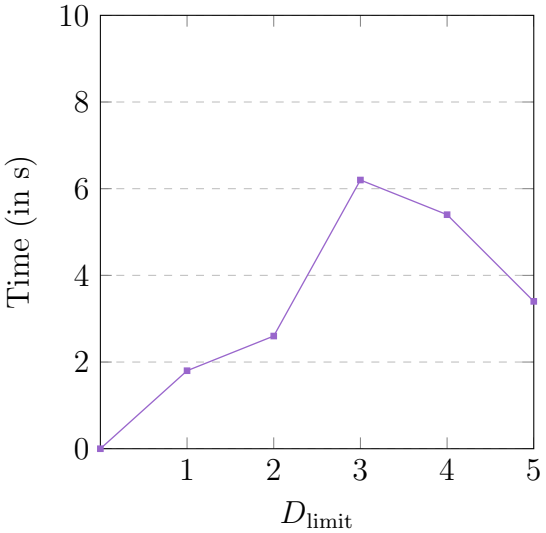

Fig. 3.13: D_{limit} v/s Computational Overhead, lower is better

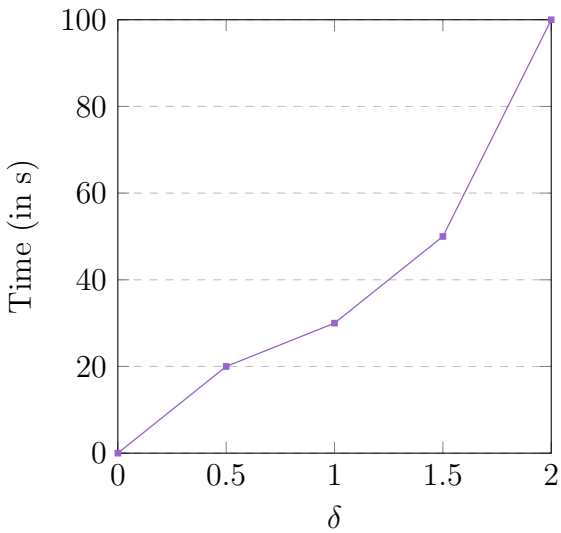

Fig. 3.14: δ v/s Computational Overhead, lower is better

approach results in the same cost but the number of destination edge servers increase in the graph the difference becomes quite broad. The Greedy Connectivity algorithm for the given dataset with $\|R\| = 30$, performs better than other approximation algorithms like EDD-A and EDD-NSTE. This may be an anomaly. However, the Greedy Algorithm, cannot guarantee to maintain a certain performance ratio on any given dataset, unlike EDD-A or EDD-NSTE. Resultant cost produced by EDD-NSTE approach is lower (is better) as

compared to EDD-A in most of the cases except $\|R\| = 30$.

Now, the only factor left to alter is D_{limit}, and the rest of the parameters are kept constant similar to the previous case. The number of destination edge servers $\|R\| = 25$ for this experiment. Figure 4.10, displays the relationship between the vendor's latency constraint D_{limit} v/s the total EDD cost incurred by different solution approaches. As we know, these two quantities are inversely related, i.e., with an increase in the value of latency constraint, the total EDD cost decreases. The total EDD cost reaches its maxima of γR when $D_{\text{limit}} = 1$, as in this case all the destination edge servers are connected directly to the cloud server to fulfill the vendors' latency constraint. Also, Random approach and Greedy Connectivity algorithms become constant for $N \geq 7$, as then almost both the Random or Greedy selection of servers is not able to reduce the EDD cost. Resultant cost produced by EDD-NSTE approach is lower (is better) as compared to EDD-A in all the value of D_{limit} from 2 to 9.

Here, we consider another parameter referred to as the destination edge server density. It is defined as the ratio of number of destination edge server upon total number of edge servers, i.e., Destination Edge Server Density (ρ) —

$$\rho = \frac{\|R\|}{\|V\|} \tag{3.32}$$

Figure 4.11, displays the bar-graph relationship between the total EDD cost and the density of the destination edge servers. With increase in the density, the total EDD cost associated with each of the approach tend to increase as a whole. Among the other algorithms and approaches in comparison, EDD-NSTE Algorithm performed well overall. As ρ increases the resultant cost difference between EDD-NSTE approach and LP approach is lower as compared to the resultant cost difference between EDD-A approach and LP approach.

Similarly, we consider another experimental parameter known as edge density. This

parameter helps to estimate the overall density of the graph. A very dense graph will have a higher value of δ than a sparse graph or a tree. Thus, it is defined as the total no of edges or high-speed links in the graph upon total no of edge servers, i.e, Edge Density (δ) —

$$\delta = \frac{\|E\|}{\|V\|} \tag{3.33}$$

Figure 4.12, displays the bar-graph relationship between the total EDD cost incurred and the density of the graph, or edge density. With the increase in the edge density, the total EDD cost associated with each of the simulations tends to decrease as now shorter paths from cloud to other edge servers exist and thus, any destination edge server can be reached within the vendors' latency constraint following these shorter paths, thereby, decreasing the number of initial transit edge servers, which in turn decreases the total EDD cost incurred as the $Cost_{\text{C2E}}$ is significantly greater than $Cost_{\text{E2E}}$. Here also, we see as ρ increases the resultant cost difference between EDD-NSTE approach and LP approach is lower as compared to the resultant cost difference between EDD-A approach and LP approach.

Figure 4.18 to Figure 4.21 displays the computational overhead of EDD-NSTE algorithm versus different other graph parameters like number of nodes(N), destination edge server density(ρ), the vendors' latency constraint(D_{limit}), and edge density(δ) respectively. As we can see, with an increase in N, δ or ρ, the computational overhead of EDD-NSTE increases, as its worst-case time complexity is $O(\|V\|^4)$, whereas, with an increase in D_{limit}, the computational overhead first increases and then decreases, as relaxing the vendors' latency constraint after a certain limit decreases the extra overhead caused by the slicing and pruning algorithm.

Chapter 4

EDD considering propagation delay as the vendor's latency parameter

4.1 Problem Formulation

The edge-server network is a physical network of edge-servers connected via transmission links. Using these links, the edge servers can communicate and share data with each other and their users. This network, where nodes of the graphs denote the edge servers, the edges represent the transmission links, and edge-weights representing the length of the links or the propagation delays between edge-servers is called a edge-server network. In this research, we consider an edge-server network of a particular geographical region and formulate it as a graph $G(V, E, W)$, with V being the number of edge servers, E as the set of edges or transmission links, and W is the weights associated with each edge.

In Figure 4.1, we have $N = 10$, which means we have 10 edge servers i.e., $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. The set of edges $E = \{e_{(1,\,4)},\ e_{(1,\,2)},\ e_{(1,\,3)}\ \cdots\ e_{(9,\,8)}\}$. The weights corresponding to these edges are $W = \{100, 5, 6, \cdots 20\}$. Similarly, the set of destination edge servers $R = \{3, 7, 6, 9, 1, 4, 2\}$. The common notations used in this paper are listed in Table 3.1.

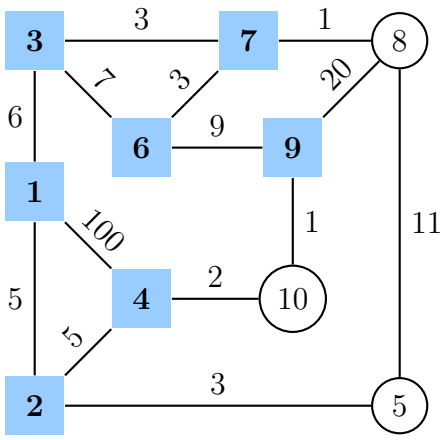

Fig. 4.1: EDD scenario with 10 edge servers

Different companies charge differently for different data transactions. For e.g., the cost of hosting applications in Amazon AWS depends on the individual user's usage, as it offers a pay-as-you-go approach. Similarly, from an app vendor's perspective, the cost consists of two components, the cloud-to-edge (C2E) transactions, and the edge-to-edge (E2E) transactions.

Now, since there are two possible scenarios for the data transmission in EDD, we will consider each one of them separately. Firstly, we have the cloud-to-edge (C2E) transactions, where data is transferred between the cloud to some of the edge servers called the *"initial transit edge servers"*. Secondly, we have the edge-to-edge (E2E) transactions, where the data is transferred between the different edge servers. Thus, an EDD solution strategy must also comprise two parts, i.e., a C2E strategy, and an E2E strategy.

In the C2E strategy, we define the set $S = (s_1, s_2 \cdots s_{\mathrm{N}})$, where s_{v} $(1 \leq v \leq N)$ denotes the set of boolean array representation of the initial transit edge servers, i.e., the nodes which receives data directly from the cloud.

$$s_{\mathrm{v}} = \begin{cases} 1, & \text{if } v \text{ is an initial transit edge server} \\ 0, & \text{if } v \text{ is not an initial transit edge server} \end{cases} \tag{4.1}$$

Similarly, In the E2E strategy, we define the set $T = (\tau_{(1,\,1)},\, \tau_{(1,\,2)} \cdots\, \tau_{(\mathrm{N},\,\mathrm{N})})$, where $\tau_{(u,\,v)}$ $(u,\, v \in V)$ denotes whether the data is transmitted through edge $e_{(u,\,v)}$ in the graph G.

$$\tau_{(u,\,v)} = \begin{cases} 1, & \text{if data is transmitted through } \mathrm{e}_{(\mathrm{u},\,\mathrm{v})} \\ 0, & \text{if data is not transmitted through } \mathrm{e}_{(\mathrm{u},\,\mathrm{v})} \end{cases} \tag{4.2}$$

Since a reasonable EDD strategy must connect each destination edge server $v \in R$ to an initial transit edge server in S through edges in E, thus,

$$\mathit{Connected}(S,\ T,\ u,\ v) = \mathit{true},\ \forall v \in R,\ \exists\, u, s_{\mathrm{u}} = 1 \tag{4.3}$$

Now, here we consider the length of the transmission links for the delay constraint, as the propagation delay (in time) is directly proportional to the link-length (L_{link}), assuming the speed (s_{medium}) in the medium as constant, i.e.,

$$P_{\mathrm{delay}} = \frac{L_{\mathrm{link}}}{s_{\mathrm{medium}}} \tag{4.4}$$

We here assume that the bandwidth delay and queuing delay at the nodes is negligible, and thus the vendors' latency or delay constraint will depend just on the total length of the links.

Also, the mode of transmission between the cloud server and each of the edge server is same. Thus, we denote the propagation delay between them as a constant, i.e., the weights of these edges in the graph $G(V, E, W)$ will also be a constant denoted by λ, thus,

$$W_{(\mathrm{c},\,\mathrm{v})} = \lambda, \forall\, v \in V \setminus \{c\} \tag{4.5}$$

Now, the app-vendor has the liberty to regulate the EDD delay constraint as per the application specifics and requirements. Let P_{limit} represent the app vendors' EDD propagation

delay constraint, and L_{limit} represent the equivalent EDD length constraint. Thus, for each of the destination edge server v, edge-to-edge latency or data transmission latency or total path length between v and its connected parent initial transit edge server in S, i.e., L_{v} must not exceed this delay constraint.

$$0 \leq L_v \leq L_{limit}, L_v \in Z^+, \forall v \in R \tag{4.6}$$

For example, consider the graph in Figure 4.2, the nodes of the graph represent the edge servers given by $V = \{1, 2, 3, 4, 5, 6, 7, 9\}$, where number of edge servers $N = 9$ and the set of high-speed links i.e., $E = \{e_{(1,\,2)}, e_{(1,\,3)}, e_{(1,\,4)} \cdots e_{(7,\,9)}\}$. Now, let the EDD length constraint as per the vendor's application requirements be $L_{\text{limit}} = 12$. This means each destination edge server must receive data within the maximum path length of 12, where path starts from an initial transit edge server. In Figure 4.2, if node 4 is the only edge server selected as the initial transit edge server, then one possible E2E strategy is to select edges $\{e_{(1,\,2)}, e_{(1,\,4)}, e_{(1,\,5)}, e_{(2,\,3)}, e_{(4,\,8)}\}$. This means that in the set T_{E2E} there is $\tau_{(1,\,2)} = \tau_{(1,\,4)} = \tau_{(1,\,5)} = \tau_{(2,\,3)} = \tau_{(4,\,8)} = 1$. Accordingly, we can also obtain the time constraint limit of each of these destination edge servers, i.e, $L_1 = 1$, $L_2 = 3$, $L_4 = 0$, $L_5 = 10$, $L_8 = 10$, and $L_3 = 10$.

Thus, the Edge Data Distribution (EDD) problem can be formulated as —

$$minimize \left(Cost_{\text{C2E}}(S) + Cost_{\text{E2E}}(T)\right) \tag{4.7}$$

while fulfilling the EDD EDD length constraint in Equation 4.6.

As we can see, the Edge Data Distribution (EDD) is a constrained optimization problem based on a well-known NP-Hard problem in graph theory known as the Steiner Tree [XCH+21].

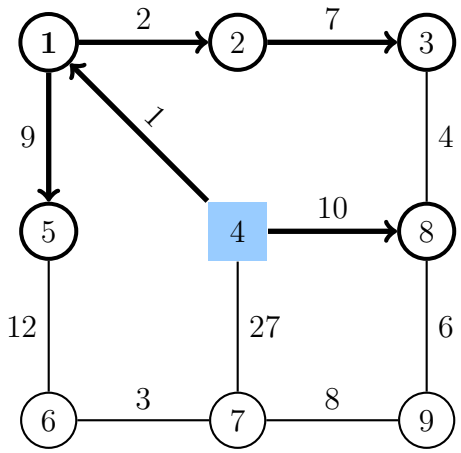

Fig. 4.2: EDD example to demonstrate L_{limit}

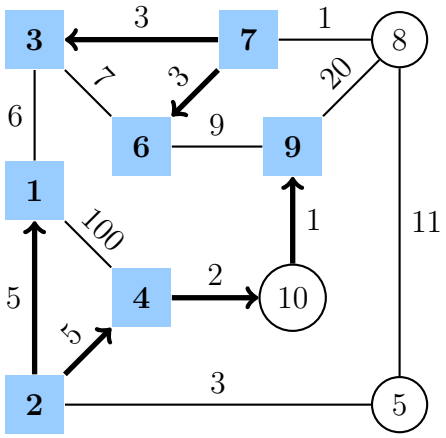

Fig. 4.3: Optimal Solution using integer programming

4.2 Solution Strategy

In this section, we first present a modified implementation of the optimization approach given in [XCH+21] where the Edge Data Distribution (EDD) is formulated as a constrained integer programming problem which can then be solved by using any simple IP solver[1]. Since this method takes an exponential amount of time to produce results for large datasets, we will design an O(k) approximation method, named EDD-NSTE that can estimate solutions to the EDD problems in polynomial time. This will be followed by a theoretical analysis of the performance of the EDD-NSTE algorithm.

4.2.1 Edge Data Distribution (EDD) as an Integer Programming

The Edge Data Distribution (EDD) can be formulated as a constrained integer programming problem as given in [XCH+21]. Our modified implementation of the same is specified as below.

The solution to the EDD problem must minimize the cost incurred for data transmissions while fulfilling the app vendor's EDD length constraint of L_{limit}. Thus, to model the EDD

[1]IP solvers are the frameworks or tools used to solve integer programming problems. For E.g., GLPK, LP Solve, CLP, CBC, CVXOPT, SciPy, Gurobi Optimizer, CPLEX, XPRESS, MOSEK, Google OR Tools, PuLP, etc.

problem as a generalized constrained integer programming optimization problem firstly we add the cloud c into V, and then add the edges from cloud c to each edge server in graph G. Now, we can formulate the C2E strategy of EDD, $S = (s_1,\ s_2,\ s_3\ \cdots\ s_{\mathrm{N}})$ by selecting edges in graph G, such that, $\mathrm{T_c} = (\tau_{(\mathrm{c},\ 1)},\ \tau_{(\mathrm{c},\ 2)}\ \cdots\ \tau_{(\mathrm{c},\ \mathrm{N})})$, where $\tau_{(\mathrm{c},\ \mathrm{v})}$ denotes whether the data is transmitted from cloud server c to the edge server v. Here, we combine the C2E and E2E strategy of EDD into one parameter

$$\begin{gathered} T = (\tau_{(\mathrm{c},\ 1)}, \tau_{(\mathrm{c},\ 2)} \cdots \tau_{(\mathrm{c},\ \mathrm{N})}, \tau_{(1,\ 1)} \cdots \tau_{(\mathrm{N},\ \mathrm{N})}), \\ where,\ \tau_{(\mathrm{u},\ \mathrm{v})} \in (0,1)\ (u \in V, v \in V \setminus \{c\}) \end{gathered} \tag{4.8}$$

which indicates whether the edge $e_{(\mathrm{u},\ \mathrm{v})}$ is included in T. We also need to define a parameter for each of the edge servers, i.e.,

$$H_{\mathrm{v}} = \begin{cases} 0, & \text{if v is not visited during the EDD process} \\ 1, & \text{if v is visited during the EDD process} \end{cases} \tag{4.9}$$

here, $H_{\mathrm{v}} = 1, \forall v \in R$ will make sure that we include the destination edge servers into our solution. Also, $H_c = 1$, as cloud will always be a part of the solution. Now, the constrained integer programming optimization problem can be formally expressed as follows,

$$min \left(\sum_{v \in V \setminus \{c\}} \tau_{(\mathrm{c},\ \mathrm{v})} \cdot W_{(\mathrm{c},\ \mathrm{v})} \ + \sum_{u,v \in V \setminus \{c\}} \sum \tau_{(\mathrm{u},\ \mathrm{v})} \cdot W_{(\mathrm{u},\ \mathrm{v})} \right) \tag{4.10}$$

subjected to constraints,

$$H_{\mathrm{v}} = 1, \forall v \in R \cup \{c\} \tag{4.11}$$

$$W_{(\mathrm{c},\ \mathrm{v})} = \lambda, \forall\ v \in V \setminus \{c\} \tag{4.12}$$

$$\tau_{(\mathrm{u},\ \mathrm{v})} \leq H_{\mathrm{u}} \cdot H_{\mathrm{v}}, \forall u \in V, v \in V \setminus \{c\} \tag{4.13}$$

$$\sum \tau_{(\mathrm{u},\ \mathrm{v})} = H_{\mathrm{v}}, \forall u \in V, v \in V \setminus \{c\} \tag{4.14}$$

$$\sum_{v \in V \setminus \{c\}} \tau_{(\mathrm{c},\ \mathrm{v})} \geq 1 \tag{4.15}$$

$$\tau_{(\mathrm{u},\ \mathrm{v})}, H_{\mathrm{v}} \in (0, 1) \tag{4.16}$$

$$L_{\mathrm{c}} = 0, L_{\mathrm{c}} \leq L_{\mathrm{v}} \leq L_{\mathrm{limit}}, \forall v \in V \setminus \{c\} \tag{4.17}$$

$$L_{\mathrm{v}} - L_{\mathrm{u}} = W_{(\mathrm{u},\ \mathrm{v})}, \forall u, v \in V, \tau_{(\mathrm{u},\ \mathrm{v})} = 1 \tag{4.18}$$

Here, in-order to minimize the total cost incurred by both the C2E and E2E edges given by Equation 4.10, we must satisfy the above constraints. The Equation 4.12 makes sure that the weight of the edge from cloud to each of the edge server is equal to λ. The Equation 4.13 will make sure that, if edge $e_{(\mathrm{u},\ \mathrm{v})}$ is considered a part of the solution, or, $\tau_{(\mathrm{u},\ \mathrm{v})} = 1$, then both the nodes u and v must be visited, i.e., $H_{\mathrm{v}} = 1$ and $H_{\mathrm{u}} = 1$. Equation 4.14 suggests a constraint that the summation of $\tau_{(\mathrm{u},\ \mathrm{v})}$ of the edges incoming at the particular node v, must be equal to the H_{v} of that particular node, i.e., whether or not the node v is visited, if edges incoming onto it are considered a part of the solution. Equation 4.15 suggests that the solution must have atleast one cloud server, i.e., summation of $\tau_{(\mathrm{c},\ \mathrm{v})}$ must be greater than or equal to 1. Equation 4.17 suggests that the depth of any edge server v, must be within $[0, L_{\mathrm{limit}}]$, and Equation 4.18 suggests that for nodes u, v connected through edge $e_{(\mathrm{u},\ \mathrm{v})}$ in the EDD solution must satisfy the depth difference of 1.

Example to understand Integer Programming

Consider an example in Figure 4.3 that displays the optimal solution generated, if we solve the graph in Figure 3.1 using the integer programming approach, with EDD length constraint of $L_{\mathrm{limit}} = 110$, and $W_{(\mathrm{c},\ \mathrm{v})} = 100, \forall v \in V \setminus \{c\}$. The C2E strategy will specify *node 2* and *node 7* as the initial transit edge servers which receives the data directly from the cloud server and thus, the cost incurred for the cloud to edge data transmission is $Cost_{C2E} = 200$. Now, these nodes will transmit the data to the other destination edge

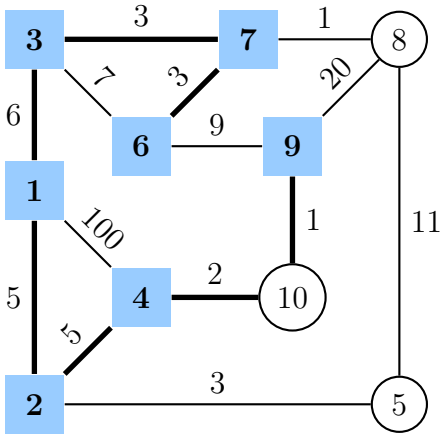

Fig. 4.4: Steiner Tree estimated by Algorithm 1

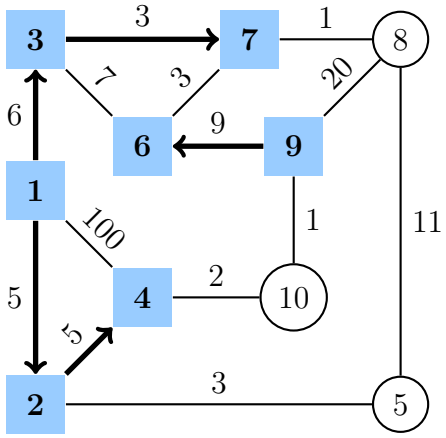

Fig. 4.5: Final EDD solution estimated by Algorithm 3

servers (represented by light blue square nodes) that can be reached within the given time constraint of L_{limit}. Thus, the cost incurred for the edge to edge data transmission is $Cost_{E2E}$ = *19*. The integer programming approach selects the edges E = {$e_{(c, 2)}$, $e_{(c, 7)}$, $e_{(2, 1)}$, $e_{(2, 4)}$, $e_{(4, 10)}$, $e_{(10, 9)}$, $e_{(7, 3)}$, $e_{(7, 6)}$}, for data transmissions and thus, the optimal total cost incurred is the sum of $Cost_{C2E}$ and $Cost_{E2E}$, i.e., 219.

4.2.2 Edge Data Distribution (EDD) as a Network Steiner Tree Estimation

This method is similar to as the method defined in the previous chapter. Algorithm 1 and Algorithm 2 are used in the similar fashion, after that we slice and fine-tune the graph to obtain the final EDD solution.

The algorithm for slicing and fine-tuning the graph to satisfy the vendor's EDD length constraint is presented in Algorithm 4. In this algorithm, we take the directed steiner tree G_{DT} and initialize the *pathLength*[] and the *visited*[] for each node of the graph, with $visited[c] = 1$, $pathLength[c] = 1$, and $pathLength[v] = \infty, \forall v \in V$. It then runs the Breadth-First Search to compute the minimum length of path of each node from the root node, assuming the cloud server as the root. After that, it runs the Depth-First Search and —

- for each unvisited destination edge server with path length less than or equal to L_{limit},

Algorithm 4: EDD-NSTE Algorithm

Input: G, L_{limit}, pathLength, W, visited
Output: G_{final}

Construct the graph G_{ST} from Algorithm 1. Take the maximum connectivity node in the graph G_{ST} and connect it to the cloud to make it a rooted tree. Update the distances between consecutive nodes by the shortest path between them and let this graph be G_{DT}
Initialize pathLength[c] $\leftarrow 0$, visited[c] $\leftarrow 1$, pathLength[v] $\leftarrow \infty$, visited[v] $\leftarrow 0$, $\forall$ v $\in$ V, Update G_{DT} to include the minimum cost path between any two vertices with edges in graph G
Update G_{DT} to form a directed rooted tree, with cloud as the root
Run a Breadth-First Search Algorithm to compute the minimum depth of each vertex v $\in G_{DT}$ from cloud
foreach *vertex v, in the path from root to leaf node and parent p in Depth-First Search Algorithm* **do**
    if *visited[v]* $\neq$ *1* **then**
        if *pathLength[v]* $\leq L_{limit}$ *and* $v \in R$ **then**
            add edge $e_{(p, v)}$ in G_{final}
            visited[v] $\leftarrow 1$
        else if $v \notin R$ **then**
            add edge $e_{(p, v)}$ in G_{final}
            visited[v] $\leftarrow 1$
        else if *pathLength[v]* $> L_{limit}$ *and* $v \in R$ **then**
            pathLength[v] $\leftarrow W_{(c, v)}$, visited[v] $\leftarrow 1$
            add edge $e_{(c, v)}$ in G_{final}
            foreach *child node u of node v in graph G in Depth-First Search order* **do**
                if *pathLength[u] > pathLength[v] +* $W_{(u, v)}$ *and pathLength[v] +* $W_{(u, v)} \leq L_{limit}$ **then**
                    if $e(v, u) \notin G_{DT}$ **then**
                        update edge in G_{DT} from v to u
                        pathLength[u] $\leftarrow$ pathLength[v] + $W_{(u, v)}$
            run a Breadth-First Search Algorithm and update depths of the vertices in G_{DT}
Run a Depth-First Search and remove the unnecessary edges from the graph G_{final}
Calculate the total cost incurred as the sum of $W_{(u, v)}$ $\forall$ $e_{(u, v)} \in G_{final}$

it adds the corresponding edge into the final solution G_{final}.

- for each unvisited edge server, which is not a destination edge server, it again adds

the corresponding edge into the final solution G_{final}.

- for each unvisited destination edge server having a depth greater than L_{limit}, it connects that edge server directly to the cloud and adds edge from cloud to this edge server in G_{final}. It then runs a Depth-First Search to update the minimum depths of the remaining unvisited edge servers assuming this edge server as the root.

Finally, this algorithm removes the unnecessary edges from the graph G_{final} and calculates the cost of the solution as the sum of the cost of edges in G_{final}.

Example to understand the EDD-NSTE algorithm

Consider the graph in the Figure 3.1, with data transmission latency $L_{\text{limit}} = 110$, and $W_{(\text{c, v})} = \lambda = 100, \forall v \in V \setminus \{c\}$. When we apply 2Algorithm 1 onto it, it selects edges $\{\text{e}_{(2,\ 1)},\ \text{e}_{(2,\ 4)},\ \text{e}_{(1,\ 3)},\ \text{e}_{(4,\ 10)},\ \text{e}_{(10,\ 9)},\ \text{e}_{(3,\ 7)},\ \text{e}_{(7,\ 6)}\}$ to construct the estimated steiner tree given in Figure 4.4, using the triple loss contracting mechanism, which in this case is also the optimal steiner tree possible. Now, as we can see the *node 1* is the node with the max-connectivity in the estimated steiner tree, thus, we add this node to the cloud directly, as given in Figure 4.5 to make a rooted EDD solution. The *node 2*, *node 3*, *node 4*, and *node 7* can be reached within the 110 unit distance from the cloud, thus edges $\{\text{e}_{(\text{c},\ 1)},\ \text{e}_{(1,\ 2)},\ \text{e}_{(1,\ 3)},\ \text{e}_{(2,\ 4)},\ \text{e}_{(3,\ 7)}\}$ are included into the final EDD solution. When the algorithm reaches *node 10*, it encounters *node 9* exceeding the L_{limit}. Now, since *node 9* is encountered first, the algorithm adds *node 9* to the cloud directly, which then updates the final EDD solution to include *node 6* and *node 9*, as they can be reached within data transmission constraint with *node 9* as the root. The final EDD solution then becomes $\{\text{e}_{(\text{c},\ 1)},\ \text{e}_{(\text{c},\ 9)},\ \text{e}_{(1,\ 2)},\ \text{e}_{(1,\ 3)},\ \text{e}_{(2,\ 4)},\ \text{e}_{(3,\ 7)},\ \text{e}_{(9,\ 6)}\}$, which incurs a total cost of 228 distance units, including both the C2E and E2E edges.

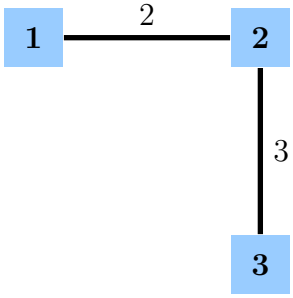

Fig. 4.6: EDD-A considering hop distances

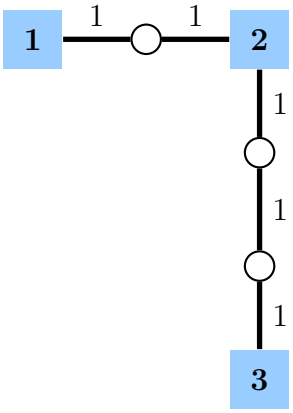

Fig. 4.7: EDD-A considering propagation delays

EDD-NSTE is an O(k) approximation algorithm

The same proof will work in this case as well with $K = L_{\text{limit}} - \gamma$, instead of $K = D_{\text{limit}}$ as was defined in the previous case.

4.3 Experimental Evaluation

We experimentally evaluated the performance of above methods with the other simulation settings and approaches in comparison as discussed below. All experimental were conducted on an Ubuntu 20.04.1 LTS machine equipped with Intel Core i5-7200U processor (4 CPU's, 2.50GHz) and 16GB RAM. The integer programming simulation was done on Google Collab (Python 3 Google Computer Engine Backend, 12.72GB RAM).

4.3.1 Simulation Settings and Approaches for comparison of the result

We compared our experimental results with the other optimization techniques and representative approaches —

- **Greedy Connectivity (GC)** — In this approach, for each node, the approach define its connectivity which is equal to the number of destination servers that have not received the data yet and are reachable within the vendors' EDD length constraint.

Then it select the node with the maximum connectivity and make it an initial transmit edge server which then transmits the data to other servers within the app vendors' EDD length limit of L_{limit} until all destination edge servers have received the data.

- **Random** — In this approach, it randomly select the initial transit edge servers which directly receive data from the cloud and then transmit it to the other servers within the app vendors' EDD length limit of L_{limit}, until all destination edge servers have received the data.

- **EDD Integer Programming** — As discussed above in the Section IV, we refine the formulation of given EDD problem into a 0-1 integer programming problem which can then be solved by using any simple IP solvers. This method take a huge amount of time to produce the result in the case of large datasets as is exponential in complexity.

- **EDD-A Algorithm** — This is state of the art approach given in [XCH+21], in this approach it first calculate a connectivity-oriented minimum Steiner tree (CMST) using an $O(2)$ approximation method. This method calculates a minimum Steiner tree having cost at most twice the optimal Steiner tree on the graph. It then uses a heuristic algorithm to calculate the minimum cost of transmission incurred using E2E and C2E edges in the CMST. This algorithm is proved to have an $O(k)$ approximation solution and a time complexity $O(\|V\|^3)$, in the worst case.

  To compare this algorithm in the propagation delay scenario we have introduced temporary nodes in the graph which corresponds to the 1-hop distance in the edge-weight graph, i.e., Figure 4.6 represents the graph with 2 nodes and edge weights. In the scenario of hop distances these edge-weights can simply be taken as 1, but in the case of propagation delay we need to convert this graph into Figure 4.7 where temporary nodes between the two nodes will be same as the 1-hop distance edge length. Thus, now we can compare the results against EDD-A with $D_{\text{limit}} = L_{\text{limit}}$.

4.3.2 Dataset Used

The experiments were conducted on a widely used EUA[2] and SLNDC[3] Dataset. The set of destination edge servers R is generated randomly. The links between the edge-servers are also generated randomly using an online random connected graph model generator tool. In the experiments, $\gamma = 20$ is taken.

4.3.3 Experiment Results

Different EDD scenarios and algorithms mentioned above in the section solution strategy and other approaches are simulated and compared to each other by varying different parameters as mentioned below —

- The number of nodes N in the graph G.
- The number of destination edge servers R in the graph.
- The vendor's EDD length limit, i.e., L_{limit}.

Among the above three factors, we vary one of the factors while keeping the others constant and then compare the estimated total EDD cost incurred in each case with the optimal cost generated by the integer programming approach. The experiment is done in two sets —

- Set 1 — Above factors are varied and the results are calculated on EUA Dataset.
- Set 2 — Above factors are varied and the results are calculated on SLNDC Dataset.

Figure 4.8 and Figure 4.13, displays the relationship between the number of nodes N in the graph G v/s total EDD cost incurred by different simulation settings and approaches. The value of the other parameters such as $L_{\text{limit}} = 150$, $R = 25$ random nodes, $\gamma = 100$, and weights of the edges are in the range [1, 50] were fixed during the experiment for

[2]https://github.com/swinedge/eua-dataset
[3]https://snap.stanford.edu/data/

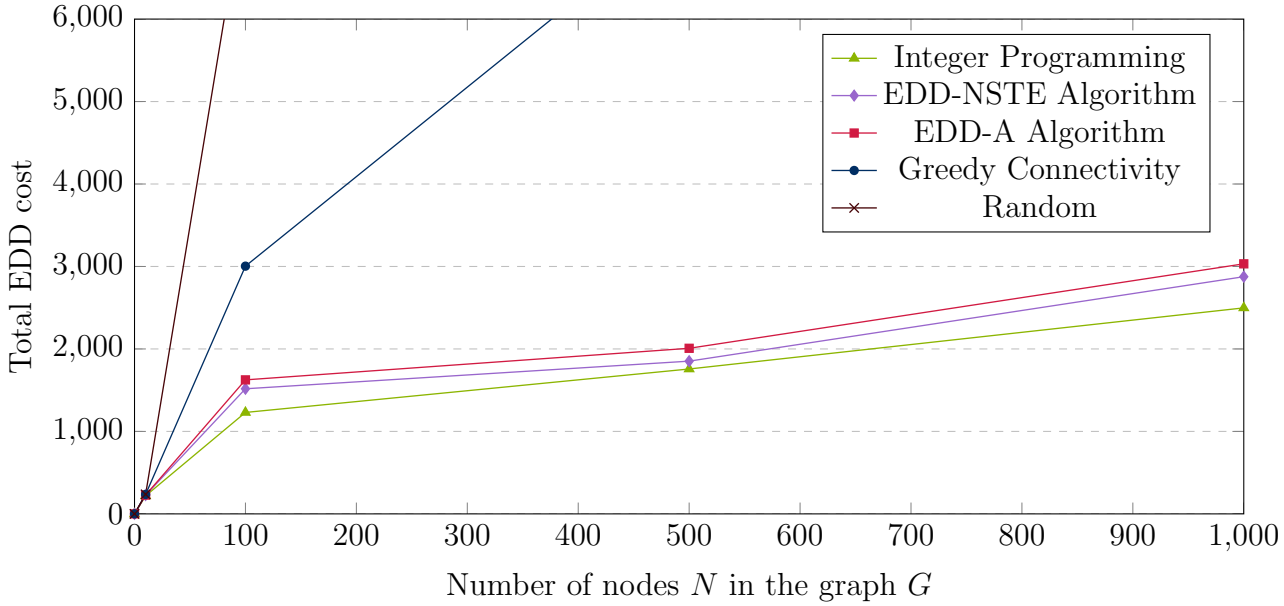

Fig. 4.8: Set 1: N v/s Total EDD cost

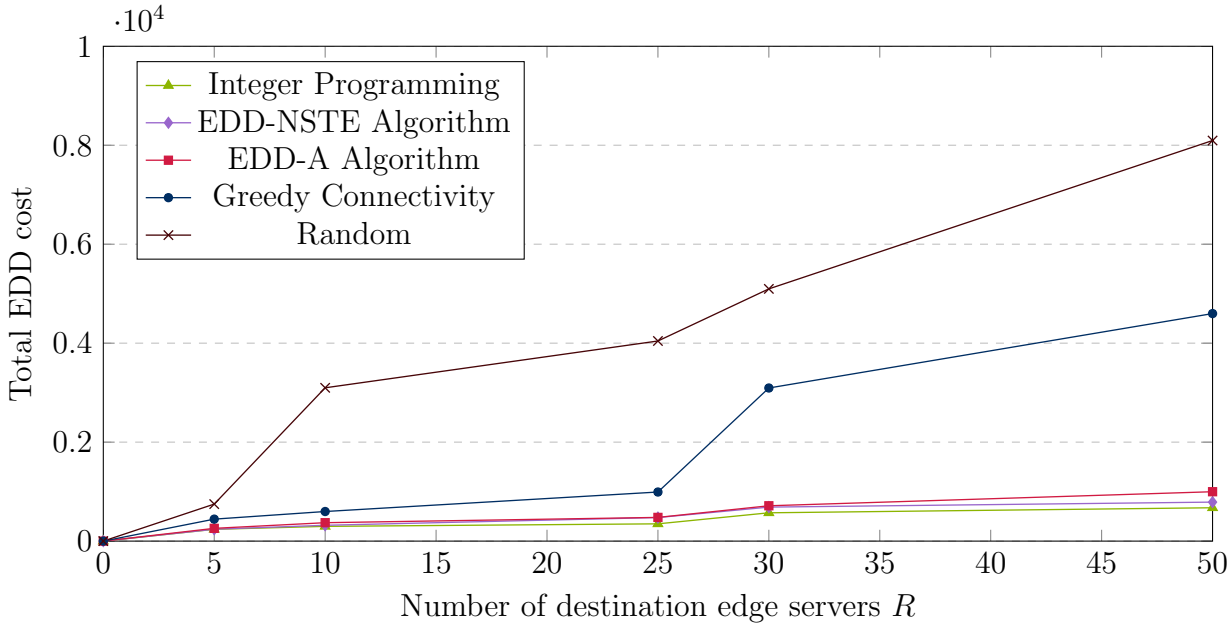

Fig. 4.9: Set 1: R v/s Total EDD cost

both the sets. As we can see that for both sets of experiments EDD-A and EDD-NSTE perform better than any other heuristic approach. For $N < 100$ there is a very slight difference between other approaches but as we increase the value of N, the difference becomes significantly more, and overall EDD-NSTE performs better than other heuristic

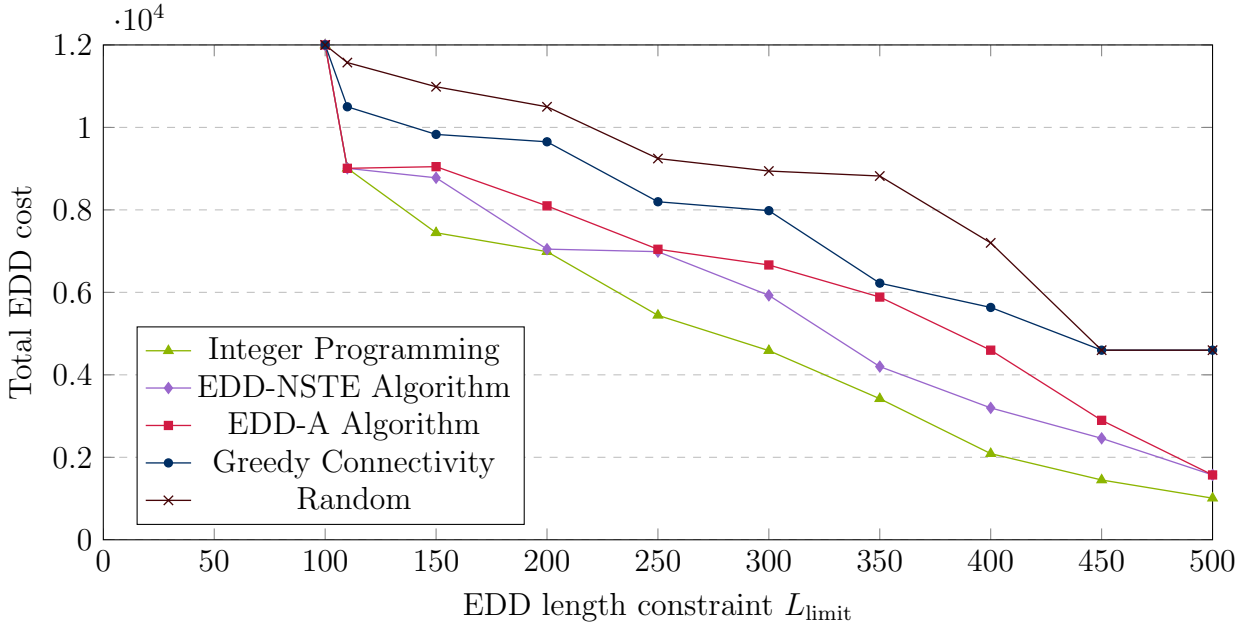

Fig. 4.10: Set 1: EDD length constraint L_{limit} v/s Total EDD cost

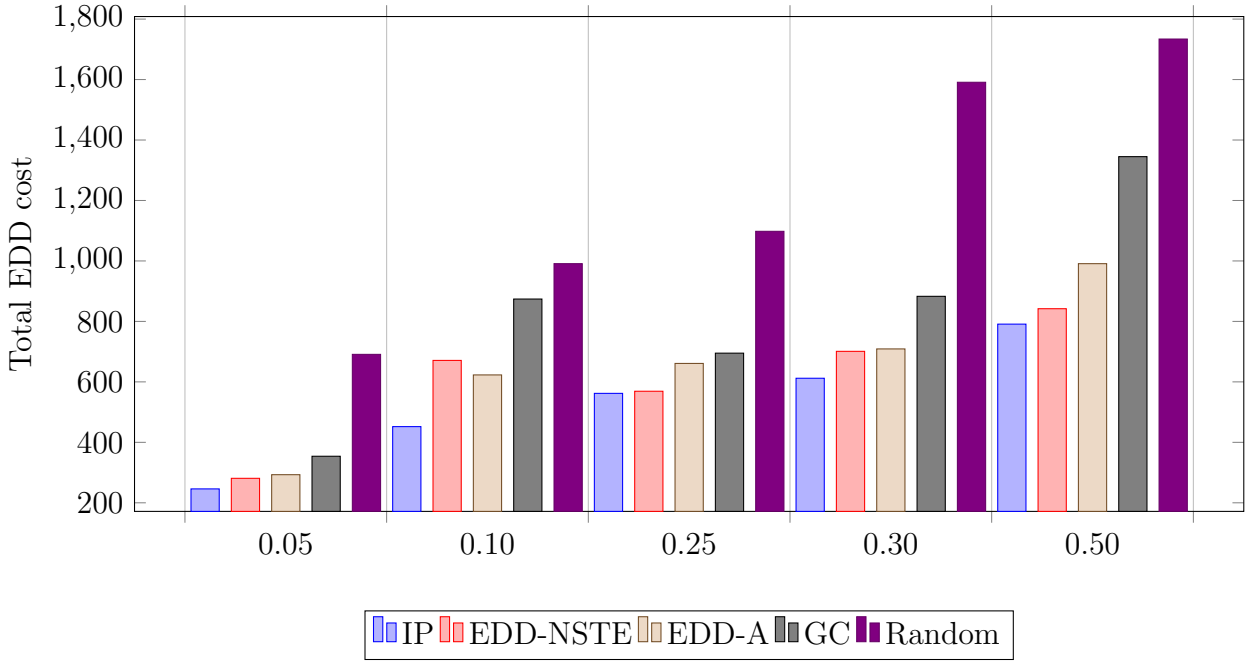

Fig. 4.11: Set 1: ρ v/s Total EDD cost

methods.

Similarly, Figure 4.9 and Figure 4.14, displays the relationship between the number of destination edge servers R v/s the total EDD cost incurred by different other settings. For

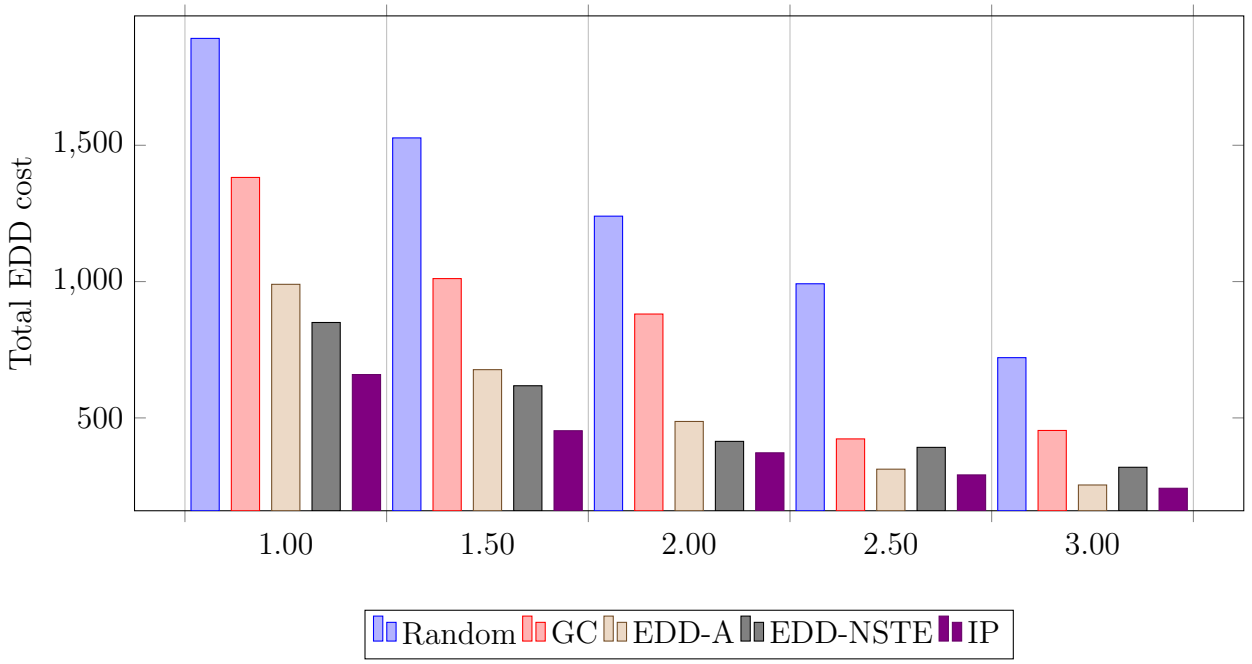

Fig. 4.12: Set 1: δ v/s Total EDD cost

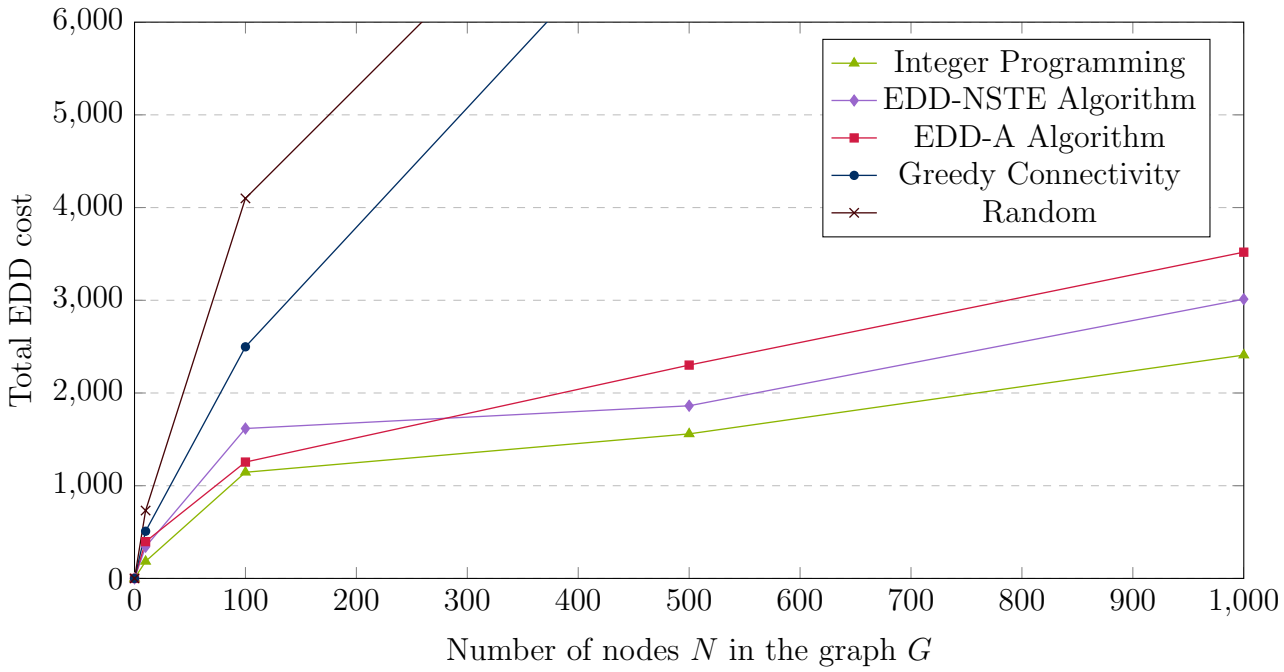

Fig. 4.13: Set 2: N v/s Total EDD cost

this experiment the value of other parameters such as $N = 100$, $L_{\text{limit}} = 150$, $\gamma = 100$, and weight of the edges are in the range [1, 50] were fixed during the experiment. As we can see that in Set 1, EDD-NSTE and EDD-A perform very close to the optimal solution,

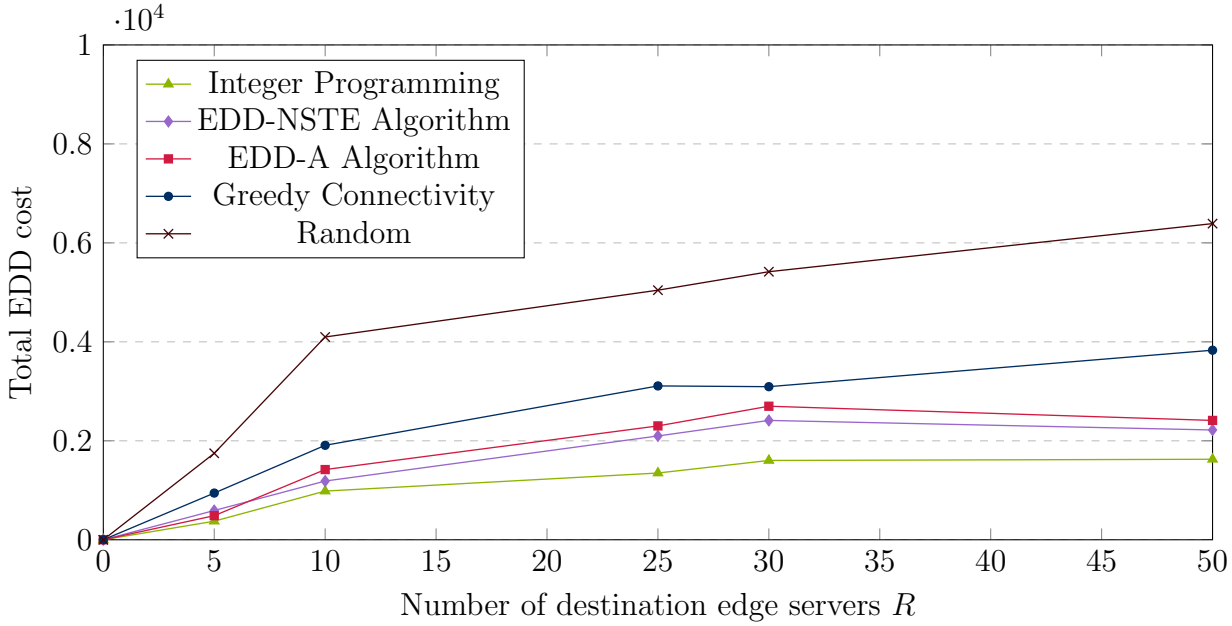

Fig. 4.14: Set 2: R v/s Total EDD cost

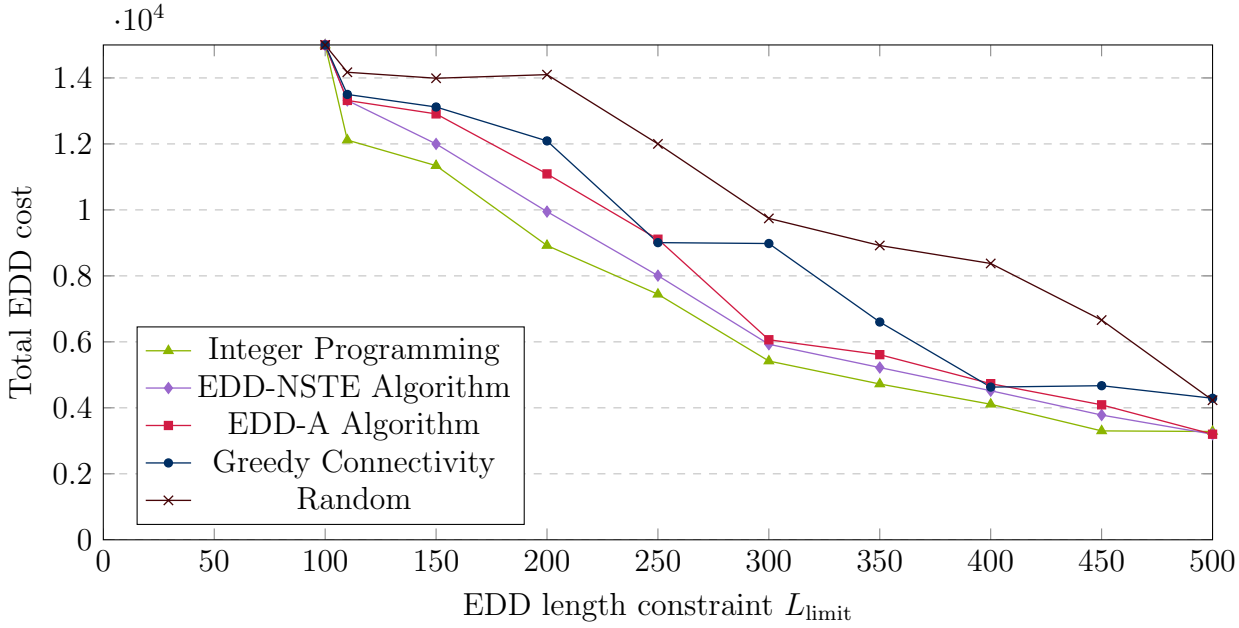

Fig. 4.15: Set 2: EDD length constraint L_{limit} v/s Total EDD cost

and in Set 2, there is a considerable difference between these two heuristics. Now, the only factor left to alter is L_{limit}, and the rest of the parameters are kept constant similar to the previous case. The number of destination edge servers $\|R\| = 25$ for this experiment. Figure 4.10 and Figure 4.15, displays the relationship between the vendor's EDD length

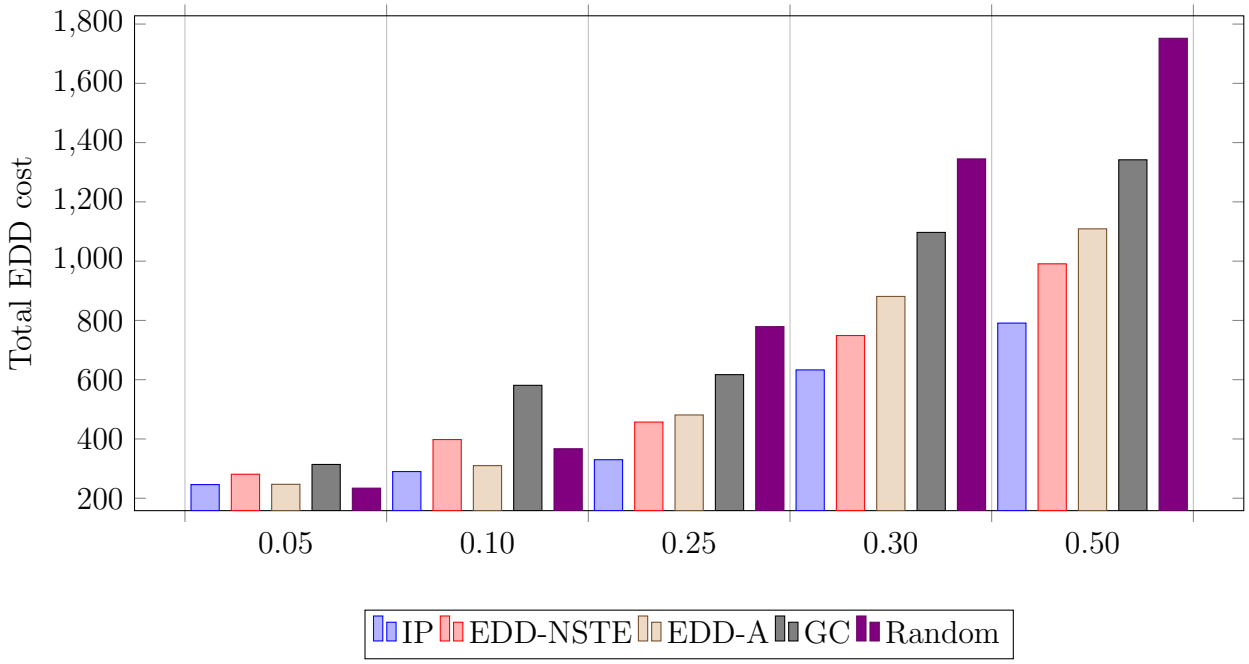

Fig. 4.16: Set 2: ρ v/s Total EDD cost

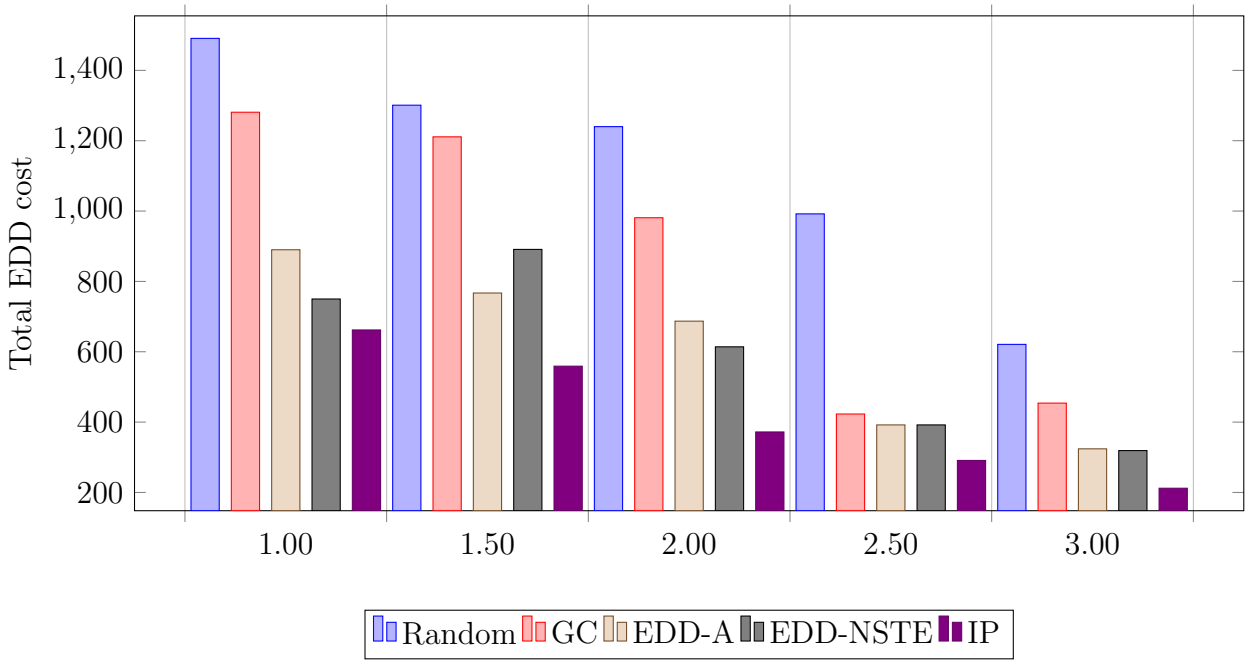

Fig. 4.17: Set 2: δ v/s Total EDD cost

constraint L_{limit} v/s the total EDD cost incurred by different simulations. As we know, these two quantities are inversely related, i.e., with an increase in the value of EDD length constraint, the total EDD cost decreases. The total EDD cost reaches its maxima of γR

when $L_{\text{limit}} = \gamma$, as in this case all the destination edge servers are connected directly to the cloud server to fulfill the vendors' EDD length constraint. Also, as we can see that the Random and Greedy Connectivity algorithms become constant for $N \geq 450$ in Set 1, as then almost any random or greedy selection of servers is not able to improve on the EDD cost.

Here, we consider another parameter referred to as the destination edge server density. It is defined as the ratio of number of destination edge server upon total number of edge servers, i.e., Destination Edge Server Density (ρ) —

$$\rho = \frac{\|R\|}{\|V\|} \tag{4.19}$$

Figure 4.11 and Figure 4.16, displays the bar-graph relationship between the total EDD cost and the density of the destination edge servers. As we can see, with the increase in density, the total EDD cost associated with each of the simulations tends to increase as a whole. Among the other algorithms and approaches in comparison, EDD-NSTE Algorithm performed well overall.

Similarly, we consider another experimental parameter known as edge density. This parameter helps to estimate the overall density of the graph. A very dense graph will have a higher value of δ than a sparse graph or a tree. Thus, it is defined as the total no of edges or high-speed links in the graph upon total no of edge servers, i.e, Edge Density (δ) —

$$\delta = \frac{\|E\|}{\|V\|} \tag{4.20}$$

Figure 4.12 and Figure 4.17, displays the bar-graph relationship between the total EDD cost incurred and the density of the graph, or edge density. As we can see, with the increase in the edge density, the total EDD cost associated with each of the simulations tends to decrease as now shorter paths from cloud to other edge servers exist and thus, any destination edge server can be reached within the vendors' EDD length constraint following

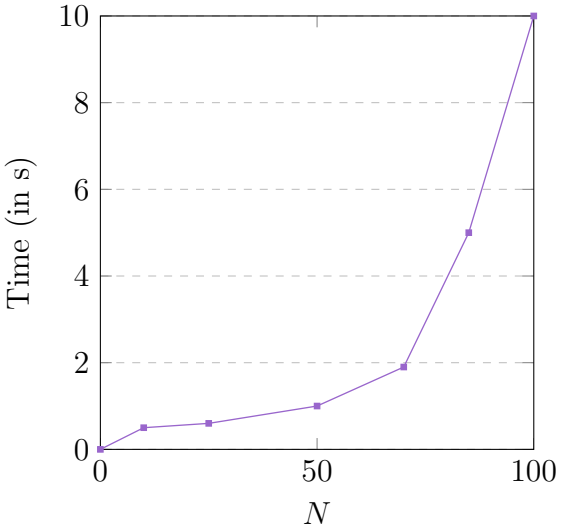

Fig. 4.18: N v/s Computational Overhead

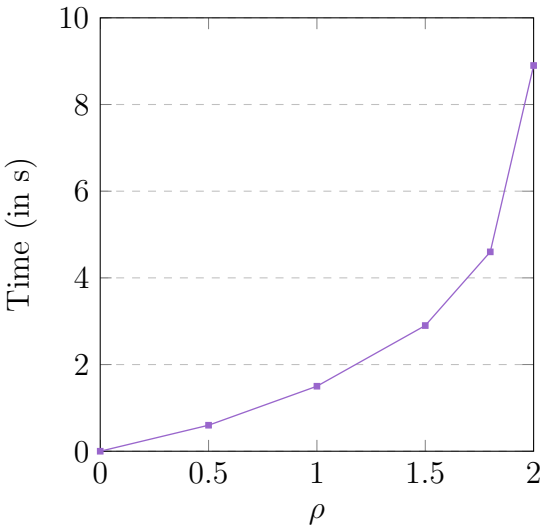

Fig. 4.19: ρ v/s Computational Overhead

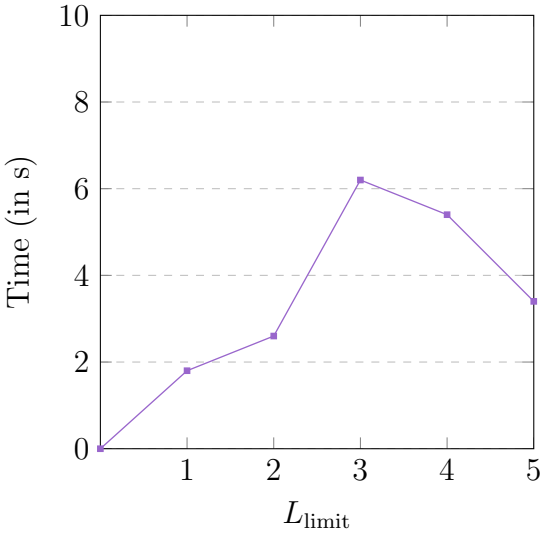

Fig. 4.20: L_{limit} v/s Computational Overhead

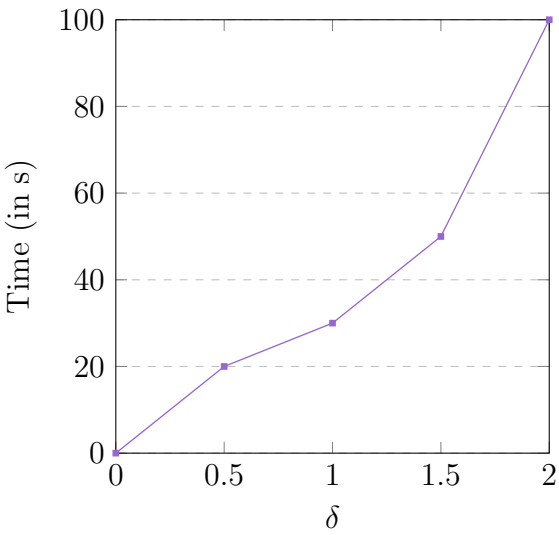

Fig. 4.21: δ v/s Computational Overhead

these shorter paths, thereby, decreasing the number of initial transit edge servers, which in turn decreases the total EDD cost incurred as the $Cost_{\text{C2E}}$ is significantly greater than $Cost_{\text{E2E}}$.

Figure 4.18 to Figure 4.21 displays the computational overhead of EDD-NSTE algorithm versus different other graph parameters like number of nodes(N), destination edge server density(ρ), the vendors' EDD length constraint(L_{limit}), and edge density(δ) respectively.

As we can see, with an increase in N, δ or ρ, the computational overhead of EDD-NSTE increases, as its worst-case time complexity is $O(\|V\|^4)$, whereas, with an increase in L_{limit}, the computational overhead first increases and then decreases, as relaxing the vendors' EDD length constraint after a certain limit decreases the extra overhead caused by the slicing and pruning algorithm.

Chapter 5

Conclusion

The EDD-NSTE algorithm suggested in this paper performs considerably better than the other simulations and approaches implemented for the edge data distribution problem. The total EDD cost approximated by this algorithm is closer to the optimal solution than the basic hard-coded approaches like Greedy Connectivity or Random. The algorithm also performs significantly better than the last best EDD-A algorithm suggested in [XCH+21]. The EDD-A algorithm is an $O(k)$ approximation algorithm[1] and the EDD-NSTE algorithm is an $O(k')$ approximation algorithm where $k' < k$ and thus, the performance of EDD-NSTE is better than EDD-A, which coincides with the experimental evaluations done so far.

EDD-NSTE algorithm produced better solutions than EDD-A or other approaches in more than 86.67% of the test cases(considering hop distance as the vendor's latency parameter), and 80.35% of the test cases(considering propagation delay as the vendor's latency parameter), ranging from very dense graphs to sparse graphs or trees.

Greedy Connectivity may produce solutions better than EDD-A or EDD-NSTE in some of the cases, but the greedy solutions are just an ad-hoc solution and thereby, cannot always guarantee to produce results that closer to the actual solution whereas the latter algorithms are restricted by their approximation ratio to always produce solutions below

[1] $O(k)$ approximation algorithm means that solution produced by the algorithm is atmost k times the optimal solution.

a certain maximum value for a given graph, thus they perform better than these methods on an average.

Chapter 6

Future Work

- The lower bound to approximate Steiner Tree in general graphs is smaller than 1.01 theoretically. The current best algorithm has an approximation ratio of (1.39+epsilon). Thus, implementing a better way of estimating the Steiner Tree can help improve the approximation ratio and so, we can work on incorporating these methods in our edge data distribution solution.

- We can also try and implement a heuristic algorithm completely independent from Steiner Tree approximation that can estimate better solutions.

- Also, we can try and include other delays into the EDD problem formulation like transmission delays and queuing delays and make it efficient for end-to-end delay guarantees.

References

[BSEB] Martin Breitbach, Dominik Schäfer, Janick Edinger, and Christian Becker. Context-aware data and task placement in edge computing environments. In *Proc. IEEE Int. Conf. Pervasive Comput. Commun.*, pp. 1-10, 2019.

[CZP] Xuanyu Cao, Junshan Zhang, and H. Vincent Poor. An optimal auction mechanism for mobile edge caching. In *Proc. 38th IEEE Int. Conf. Distrib. Comput. Syst.*, pp. 388-399, 2018.

[DGT$^+$] Utsav Drolia, Katherine Guo, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan.

[DPW04] A. Davis, J. Parikh, and W. E. Weihl. Edge Computing: Extending Enterprise Applications to the Edge of the Internet. WWW Alt. '04, page 180–187, 2004.

[FH] https://www.oculus.com/facebook-horizon/.

[GHNP01a] Clemens Gröpl, Stefan Hougardy, Till Nierhoff, and Hans Jürgen Prömel. Lower bounds for approximation algorithms for the steiner tree problem. *Springer Verlag Berlin Heidelberg*, 46(5):217–228, 2001.

[GHNP01b] Clemens Gröpl, Stefan Hougardy, Till Nierhoff, and Jürgen Prömel. Approximation algorithms for the steiner tree problems in graphs. *Springer Verlag Berlin Heidelberg*, 46(5):235–279, 2001.

[HFKT] Raluca Halalai, Pascal Felber, Anne-Marie Kermarrec, and François Taïani. Agar: A caching system for erasure-coded data. In *Proc. 37th IEEE Int. Conf. Distrib. Comput. Syst.*, pp. 23-33, 2017.

[LHA+] Phu Lai, Qiang He, Mohamed Abdelrazek, Feifei Chen, and et al. Optimal edge user allocation in edge computing with variable sized vector bin packing. In *Proc. Int. Conf. Service-Oriented Comput.*, pp. 230-245, 2018.

[LHGea20] Phu Lai, Qiang He, John Grundy, and et al. Cost-effective app user allocation in an edge computing environment. *IEEE Transactions on Cloud Computing*, 31(15):1–13, 2020.

[PB08] Mukaddim Pathan and Rajkumar Buyya. *A Taxonomy of CDNs*, pages 33–77. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[SCZ+16] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[WWea20] Bastiaan Wissingh, Christopher A. Wood, and et al. Information-centric networking (ICN): content-centric networking (ccnx) and named data networking (NDN) terminology. *RFC*, 8793:1–17, 2020.

[XCH+21] HXiaoyu Xia, Feifei Chen, Qiang He, John C. Grundy, Mohamed Abdelrazek, and Hai Jin. Cost-Effective App Data Distribution in Edge Computing. *IEEE Tran. on Parallel and Distributed Systems*, 32(1):31–43, 2021.

[YBX+17] Hong Yao, Changmin Bai, Muzhou Xiong, Deze Zeng, and Zhangjie Fu. Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing. *Concurrency Comp. Pract. Exp.*, 29(16), 2017,.

[YZL+17] Hao Yin, Xu Zhang, Hongqiang H. Liu, Yan Luo, Chen Tian, Shuoyao Zhao, and Feng Li. Edge provisioning with flexible server placement. *IEEE Trans. on Parallel Distribution System*, 28(4):1031–1045, 2017.

[Zel93a] Alexander Z Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9(5):463–470, 1993.

[Zel93b] Alexander Z Zelikovsky. A faster approximation algorithm for the steiner tree problem in graphs. *Information Processing Letters*, 46(5):79–83, 1993.

[ZLH+18] Ke Zhang, Supeng Leng, Yejun He, Sabita Maharjan, and Yan Zhang. Cooperative Content Caching in 5G Networks with Mobile Edge Computing. *IEEE Wireless Communications*, 25(3):80–87, 2018.

[ZML18] Dongfang Zhao, Mohamed Mohamed, and Heiko Ludwig. Locality-aware scheduling for containers in cloud computing. *IEEE Transactions on Cloud Computing*, 8(2):635–646, 2018.

[ZZ18] Xi Zhang and Qixuan Zhu. Hierarchical caching for statistical QoS guaranteed multimedia transmissions over 5G edge computing mobile wireless networks. *IEEE Wireless Comm.*, 25(3):12–20, 2018.

Turnitin Originality Report

Processed on: 17-Apr-2021 15:29 IST
ID: 1559310113
Word Count: 14770
Submitted: 5

BTP THESIS By Ravi SHANKAR

Similarity Index	Similarity by Source	
20%	Internet Sources:	8%
	Publications:	18%
	Student Papers:	N/A

9% match (publications)
Xiaoyu Xia, Feifei Chen, Qiang He, John Grundy, Mohamed Abdelrazek, Hai Jin. "Cost-Effective App Data Distribution in Edge Computing", IEEE Transactions on Parallel and Distributed Systems, 2020

1% match (publications)
Xiaoyu Xia, Feifei Chen, Qiang He, John C. Grundy, Mohamed Abdelrazek, Hai Jin. "Cost-Effective App Data Distribution in Edge Computing", IEEE Transactions on Parallel and Distributed Systems, 2021

1% match (Internet from 29-Jul-2014)
http://152.3.140.5/~abhinath/btp.pdf

1% match (Internet from 23-Nov-2020)
https://ieeexplore.ieee.org/document/9145634/

1% match (publications)
Handbook of Combinatorial Optimization, 2013.

< 1% match (Internet from 09-Mar-2021)
https://dokumen.pub/service-oriented-computing-17th-international-conference-icsoc-2019-toulouse-france-october-2831-2019-proceedings-1st-ed-2019-978-3-030-33701-8-978-3-030-33702-5.html

< 1% match (Internet from 25-Aug-2020)
https://mafiadoc.com/ieee-paper-template-in-a4-v1-ijarcsse_5980292e1723dded563a3b76.html

< 1% match (Internet from 20-May-2020)
https://epdf.pub/handbook-of-approximation-algorithms-and-metaheuristics-chapman-amp-hall-crc-com.html

< 1% match (publications)
"Service-Oriented Computing", Springer Science and Business Media LLC, 2020

< 1% match (Internet from 06-Feb-2020)
https://www.ijert.org/coordinated-pss-and-statcom-controller-for-damping-low-frequency-oscillations-in-power-systems

< 1% match (publications)
Chen, . "TOPOLOGICAL ANALYSIS OF LINEAR SYSTEMS", Graph Theory And Its Engineering Applications, 1997.

< 1% match (publications)
Xiaoyu Xia, Feifei Chen, Qiang He, Guangming Cui, Phu Lai, Mohamed Abdelrazek, John Grundy, Hai Jin. "Graph-based data caching optimization for edge computing", Future Generation Computer Systems, 2020

< 1% match (publications)
A. Z. Zelikovsky. "An 11/6-approximation algorithm for the network steiner problem", Algorithmica, 1993

< 1% match (publications)
Chin Hau Hoo, , Akash Kumar, and Yajun Ha. "ParaLaR: A parallel FPGA router based on Lagrangian relaxation", 2015 25th International Conference on Field Programmable Logic and Applications (FPL), 2015.

< 1% match (Internet from 20-Feb-2021)
https://energiforsk.se/media/29219/industrial-internet-of-things-in-nuclear-energiforskrapport-2021-726.pdf

< 1% match (publications)

Magnús M. Halldórsson, Kazuo Iwano, Naoki Katoh, Takeshi Tokuyama. "Finding Subsets Maximizing Minimum Structures", SIAM Journal on Discrete Mathematics, 1999

< 1% match (publications)
Chao, . "Spanning Trees and Optimization Problems", Discrete Mathematics and Its Applications, 2004.

< 1% match ()
https://en.wikipedia.org/wiki?curid=498304

< 1% match (publications)
Feifei Chen, Jingwen Zhou, Xiaoyu Xia, Hai Jin, Qiang He. "Optimal Application Deployment in Mobile Edge Computing Environment", 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), 2020

< 1% match (Internet from 25-Nov-2020)
https://realpython.com/linear-programming-python/

< 1% match (Internet from 31-Jan-2020)
https://www.scribd.com/document/242690516/The-Design-of-Approximation-Algorithms-David-P-Williamson-David-B-Shmoys

< 1% match (publications)
"Algorithms and Computation", Springer Science and Business Media LLC, 2011

< 1% match (publications)
Xiaoyu Xia, Feifei Chen, John Grundy, Mohamed Abdelrazek, Hai Jin, Qiang He. "Constrained App Data Caching over Edge Server Graphs in Edge Computing Environment", IEEE Transactions on Services Computing, 2021

< 1% match (publications)
Phu Lai, Qiang He, John Grundy, Feifei Chen, Mohamed Abdelrazek, John G Hosking, Yun Yang. "Cost-Effective App User Allocation in an Edge Computing Environment", IEEE Transactions on Cloud Computing, 2020

< 1% match (Internet from 05-Nov-2020)
https://drops.dagstuhl.de/opus/volltexte/lipics-complete/lipics-vol75-sea2017-complete.pdf

< 1% match (Internet from 25-Nov-2020)
https://vibdoc.com/introduction-to-algorithms-2e.html

< 1% match (publications)
"Service-Oriented Computing", Springer Science and Business Media LLC, 2019

< 1% match (publications)
Upadhyayula, S.. "Spanning tree based algorithms for low latency and energy efficient data aggregation enhanced convergecast (DAC) in wireless sensor networks", Ad Hoc Networks, 200707

< 1% match (publications)
"Algorithms and Data Structures", Springer Science and Business Media LLC, 2003

< 1% match (publications)
"Collaborative Computing: Networking, Applications and Worksharing", Springer Science and Business Media LLC, 2021

< 1% match (Internet from 17-Nov-2009)
http://www.civil.northwestern.edu/docs/Nie/nie-diss.pdf

< 1% match (Internet from 22-Jan-2019)
http://ishitvtech.in/pdf/4_3_Network.pdf

< 1% match ()
http://arxiv.org/abs/2012.08718

< 1% match (Internet from 08-Mar-2014)
http://ee.yazd.ac.ir/saadat/temporary/mobile/Wireless%20information%20Network.pdf

< 1% match (publications)
Lecture Notes in Computer Science, 2015.

< 1% match (publications)
"Handbook of Optimization in Telecommunications", Springer Science and Business Media LLC, 2006

< 1% match (publications)

"Big Data", Springer Science and Business Media LLC, 2019

< 1% match (publications)
Refael Hassin, R. Ravi, F. Sibel Salman. "Approximation Algorithms for a Capacitated Network Design Problem", Algorithmica, 2003

< 1% match (publications)
Weiwei Ding, Jin Wang, Xiumin Wang, Ruimin Zhao, Yanqin Zhu. "Optimal Flow Allocation and Linear Network Coding Design for Multiple Multicasts under the Requirements of Information Theoretical Security", 2016 IEEE Trustcom/BigDataSE/ISPA, 2016

< 1% match (Internet from 14-Nov-2020)
https://www.investopedia.com/terms/c/cloud-computing.asp#:~:text=Key%20Takeaways-,Cloud%20computing%20is%20the%20delivery%20of%20different%20services%20th

< 1% match (Internet from 01-Jul-2003)
http://maths.abdn.ac.uk/~crabb/alg/graph.txt

< 1% match (Internet from 25-Mar-2019)
https://ub-madoc.bib.uni-mannheim.de/48832/1/Dissertation_DominikSchaefer.pdf

< 1% match (Internet from 10-Dec-2017)
https://link.springer.com/content/pdf/10.1007%2Fb11837.pdf

< 1% match (publications)
Lecture Notes in Computer Science, 2007.

< 1% match (publications)
"Graph-Theoretic Concepts in Computer Science", Springer Science and Business Media LLC, 2001

< 1% match (publications)
Qiang He, Cheng Wang, Guangming Cui, Bo Li, Rui Zhou, Qingguo Zhou, Yang Xiang, Hai Jin, Yun Yang. "A Game-Theoretical Approach for MitigatingEdge DDoS Attack", IEEE Transactions on Dependable and Secure Computing, 2021

< 1% match (publications)
D. Kagaris, S. Tragoudas. "Retiming-based partial scan", IEEE Transactions on Computers, 1996

< 1% match (Internet from 06-Feb-2020)
https://link.springer.com/chapter/10.1007%2F978-3-030-33702-5_37

< 1% match (Internet from 15-Oct-2020)
https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-019-1494-1

< 1% match (publications)
"Algorithms and Architectures for Parallel Processing", Springer Science and Business Media LLC, 2020

< 1% match (publications)
Xiaoyu Xia, Feifei Chen, Guangming Cui, Mohamed Abdelrazek, John Grundy, Hai Jin, Qiang He. "Budgeted Data Caching based on k-Median in Mobile Edge Computing", 2020 IEEE International Conference on Web Services (ICWS), 2020

< 1% match (publications)
Ying Liu, Qiang He, Dequan Zheng, Mingwei Zhang, Feifei Chen, Bin Zhang. "Data Caching Optimization in the Edge Computing Environment", 2019 IEEE International Conference on Web Services (ICWS), 2019

< 1% match (Internet from 10-Nov-2010)
http://noonrd.org/Dissertations/abdelal_g_ETD.pdf

< 1% match (Internet from 20-Nov-2016)
http://cadair.aber.ac.uk/dspace/bitstream/handle/2160/13780/chen_z.pdf?isAllowed=y&sequence=2

< 1% match (publications)
"Combinatorial Optimization and Applications", Springer Science and Business Media LLC, 2015

< 1% match (publications)
Xiaoyu Xia, Feifei Chen, Qiang He, John Grundy, Mohamed Abdelrazek, Hai Jin. "Online Collaborative Data Caching in Edge Computing", IEEE Transactions on Parallel and Distributed Systems, 2020

< 1% match (publications)

Qiang He, Guangming Cui, Xuyun Zhang, Feifei Chen, Shuiguang Deng, Hai Jin, Yanhui Li, Yun Yang. "A Game-Theoretical Approach for User Allocation in Edge Computing Environment", IEEE Transactions on Parallel and Distributed Systems, 2020

< 1% match (publications)
Bo Li, Qiang He, Guangming Cui, Xiaoyu Xia, Feifei Chen, Hai Jin, Yun Yang. "READ: Robustness-oriented Edge Application Deployment in Edge Computing Environment", IEEE Transactions on Services Computing, 2020

< 1% match (publications)
Ju Ren, Deyu Zhang, Shiwen He, Yaoxue Zhang, Tao Li. "A Survey on End-Edge-Cloud Orchestrated Network Computing Paradigms", ACM Computing Surveys, 2019

< 1% match (publications)
Abbas Mehrabi, Matti Siekkinen, Antti Yla-Jaaski. "QoE-traffic Optimization Through Collaborative Edge Caching in Adaptive Mobile Video Streaming", IEEE Access, 2018

< 1% match (publications)
Munirul M. Haque. "", IEEE Transactions on Systems Man and Cybernetics Part C (Applications and Reviews), 5/2008

< 1% match (publications)
Lecture Notes in Computer Science, 2003.

< 1% match (publications)
Georgios Stamoulis. "The multi-budgeted and weighted bounded degree metric Steiner network problem", Journal of Parallel and Distributed Computing, 2017

< 1% match (publications)
"Service-Oriented Computing", Springer Science and Business Media LLC, 2018

< 1% match (publications)
Marian Kyryk, Nazar Pleskanka, Mariana Pleskanka, Petro Nykonchuk. "Load Balancing Method in Edge Computing", 2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), 2020

Application Data Distribution in Edge Computing A B. Tech Project Report Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelor of Technology by Ravi Shankar (170101053) under the guidance of Dr. Aryabartta Sahu to the DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI GUWAHATI - 781039, ASSAM 2 CERTIFICATE This is to certify that the work contained in this thesis entitled “Application Data Distribution in Edge Computing” is a bonafide work of Ravi Shankar (Roll No. 170101053), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree. April, 2021 Guwahati. Supervisor: Dr. Aryabartta Sahu Associate Professor, Department of Computer Science & Engineering, Indian Institute of Technology Guwahati, Assam. i ii Acknowledgements I would like to thank Dr. Aryabartta Sahu for his constant support and guidance through- out the project. iii iv Contents List of Figures List of Tables 1 Introduction 1.1 Abstract . . . . . 1.2 Introduction . . . . . 1.3 Organization of The Report . . . . . 2 Related Work vii ix 1 1 2 4 5 3 EDD considering hop distances as the vendor’s latency parameter 7 3.1 Problem Formulation . . . . . 7 3.2 Solution Strategy . . . . . 12 3.2.1 Edge Data Distribution (EDD) as an Integer Programming . . . . . 12 3.2.2 Edge Data Distribution (EDD) as a Network Steiner Tree Estimation 15 3.3 Experimental Evaluation . . . . . 23 3.3.1 Simulation Settings and Approaches for comparison of the result . . 23 3.3.2 Dataset Used . . . . . 24 3.3.3 Experiment Results . . . . . 25 4 EDD considering propagation delay as the vendor’s latency parameter 31 v 4.1 Problem Formulation . . . . . 31 4.2 Solution Strategy . . . . . 35 4.2.1 Edge Data Distribution (EDD) as an Integer Programming . . . . . 35 4.2.2 Edge Data Distribution (EDD) as a Network Steiner Tree Estimation 38 4.3 Experimental Evaluation . . . . . 41 4.3.1 Simulation Settings and Approaches for comparison of the result . . 41 4.3.2 Dataset Used . . . . . 43 4.3.3 Experiment Results . . . . . 43 5 Conclusion 53 6 Future Work 55 References 57 vi List of Figures 1 .1 An industry example . . . . . 3.1 EDD scenario with 10 edge servers . . . . . 3.2 EDD example to demonstrate dlimit . . . . . 3.3 Optimal solution using integer programming . . . . . 3.4 Steiner Tree estimated by Algorithm 1 . . . . . 3.5 Final EDD solution estimated by Algorithm 3 . . . . . 3.6 N v/s Total EDD cost . . . . . 3.7 R v/s Total EDD cost . . . . . 3.8 Latency constraint Dlimit v/s Total EDD cost, lower is better . . . . . 3.9 ρ v/s Total EDD cost . . . . . 3.10 δ v/s Total EDD cost, lower is better . . . . . 3.11 N v/s Computational Overhead, lower is better . . . . . .

. . 3.12 ρ v/s Computational Overhead,lower is better . . . . . 3.13 Dlimit v/s Computational Overhead, lower is better . . . . . 3.14 δ v/s Computational Overhead, lower is better . . . . . 4.1 EDD scenario with 10 edge servers . . . . . 4.2 EDD example to demonstrate Llimit . . . . . 4.3 Optimal Solution using integer programming . . . . . 4.4 Steiner Tree estimated by Algorithm 1 . . . . . 3 9 11 11 18 18 25 25 26 26 27 28 28 28 28 32 35 35 38 vii 4.5 Final EDD solution estimated by Algorithm 3 . . . . . 4.6 EDD-A considering hop distances . . . . . 4.7 EDD-A considering propagation delays . . . . . . 4.8 Set 1: N v/s Total EDD cost . . . . . 4.9 Set 1: R v/s Total EDD cost . . . . . 4.10 Set 1: EDD length constraint Llimit v/s Total EDD cost . . . . . 4.11 Set 1: ρ v/s Total EDD cost . . . . . 4.12 Set 1: δ v/s Total EDD cost . . . . . 4.13 Set 2: N v/s Total EDD cost . . . . . 4.14 Set 2: R v/s Total EDD cost . . . . . 4.15 Set 2: EDD length constraint Llimit v/s Total EDD cost . . . . . 4.16 Set 2: ρ v/s Total EDD cost . . . . . 4.17 Set 2: δ v/s Total EDD cost . . . . . 4.18 N v/s Computational Overhead . . . . . 4.19 ρ v/s Computational Overhead . . . . . 4.20 Llimit v/s Computational Overhead . . . . . 4.21 δ v/s Computational Overhead . . . . . 38 41 41 44 44 45 45 46 46 47 47 48 48 50 50 50 50 viii List of Tables 3.1 Summary of Common Notations . . . . . 8 ix x Chapter 1 Intro duction 1.1 Abstract Edge computing is a distributed computing paradigm that brings computation and data storage closer to the user's geographical location to improve response times and save band- width. It also helps to power a variety of applications requiring low latency. These appli- cation data hosted on the cloud needs to be transferred to the respective edge servers in a specific area to help provide low latency app-functionalities to the users of that area. Mean- while, these arbitrary heavy data transactions from the cloud to the edge servers result in high cost and time penalties. Thus, we need an application data distribution strategy that minimizes these penalties within the app-vendors' specific latency constraint. In phase 1 work of BTP, we provide a refined formulation of an optimal approach to solve this Edge Data Distribution (EDD) problem using Integer Programming (IP) technique. Due to the time complexity limitation of the IP approach, we suggest an O(k) approxi- mation algorithm based on network Steiner tree estimation (EDD-NSTE) for estimating solutions to dense large-scale EDD problems. Integer Programming and EDD- NSTE are 1 evaluated on a standard real-world EUA data set and the result demonstrates that EDD- NSTE significantly outperform with a performance margin of 86.67% over the other three representative approaches and the start of art approach. In phase 2 work of BTP, we provide a modified formulation of the EDD problem to include end-to-end propagation delay guarantees. We provide an optimal approach to solve this problem using Integer Programming (IP) technique. Due to the time complexity limitation of the IP approach, we suggest a modified implementation of the EDD-NSTE algorithm to include this. Integer Programming and EDD- NSTE are then evaluated on standard real-world EUA and SLNDC datasets and the result demonstrates that EDD- NSTE significantly outperform other approaches with a performance margin of 80.35% over other approaches in comparison. 1.2 Introduction Cloud computing is the practice of using a network of remote servers hosted on the internet to store, manage, and process data, rather than a local server or a personal computer. Other conventional network paradigms provided by cloud computing, which includes content- centric network, content-delivery network, and information-centric network also cannot handle the huge increase in the latency of the network and the congestion caused by the resources at the edge of the cloud [WWea20, PB08]. This is where the idea of edge computing kicks in which is basically the method of decoupling the computer services with the help of edge servers deployed at the base sta- tions or access points that are geographically close to the users [DPW04, SCZ+16]. These computing and storage resources can then be easily hired by any mobile or IoT app-vendor for hosting their applications. The users can then get rid of the computational burden and energy overload from their resource-limited end-devices to the edge servers located nearby. Thus, from an app vendors' perspective, caching data on the edge servers can reduce both 2 Fig. 1.1: An industry example the latency of their users to fetch the data and the volume of their application data trans- mitted between the cloud and its users, thereby, reducing the transmission cost. But then a new challenge arises here as to how to distribute the application data onto the edge servers to minimize the cost incurred for the transactions of data between cloud-to-edge (C2E) and edge-to-edge (E2E) servers. Facebook Horizon [FH] is an example of industrial application that can benefit consid- erably from caching their data onto the edge servers. Facebook users using Oculus headsets can access Virtual Reality (VR) videos and VR games on Facebook Horizon. VR users and their applications are highly latency-sensitive. Thus, caching the most trendy and popular VR videos and VR games onto the edge servers will allow the users near the geographical locations of those edge servers to access the application data with minimum latency which in turn will increase the VR experience, sensitivity, and performance. Figure 1.1 displays a simple graphical example of this idea, where it is required to distribute the application data to edge servers to decrease the data traffic and request-response time between the Facebook Horizon cloud servers and its users in those specific areas. Therefore, the edge data distribution is a must, and at the same time, a cost-effective method to distribute the 3 application data needs to be incorporated as otherwise the cost-ineffective or random app data distribution strategy can cost Facebook Horizon significantly. The cost-effective data distribution strategy should also consider a constraint on the time taken to distribute the application data to each of the edge servers, i.e., the Facebook Horizon latency limit. 1.3 Organization of The Report This report is organized into six chapters. The first chapter includes the abstract and introduction of the report. Chapter 2 contains the previous works which are related to this research domain. Chapter 3 incorporates the problem formulation, solution strate- gies, and experimental evaluation of the Edge Data Distribution (EDD) considering hop distances as the

vendor's latency parameter. Similarly, Chapter 4 contains the problem formulation, solution strategies, and experimental evaluation of the Edge Data Distribu- tion (EDD) considering propagation delay as the vendor's latency parameter. Chapter 5 includes the conclusion of this research, and Chapter 6 contains the future works that can be extended on this research. 4 Chapter 2 Related Work Cloud Computing is the practice of using a network of remote servers hosted on the inter- net to store, manage, and process data rather than a local server or a personal computer. Cloud- based storage makes it possible to save files to a remote database and retrieve them at user request. The scheduling of these requests is a crucial problem in cloud computing and, over the years, researchers have proposed various scheduling algorithms. In a recent research Zhao et al. [ZML18], the authors suggested a state-of-the-art scheduler, which not only takes into consideration the load balance but also the application properties for request scheduling. On the other hand, Edge Computing is an extension of cloud computing which distributes computing resources and services to the edge servers of a particular region. The placement of these edge servers is a fundamental issue in edge computing. In Yao et al. [YBX+17], the authors suggested a cost-effective method that uses 0-1 integer programming to help providers of edge infrastructure make correct decisions regarding the edge servers place- ment. Similarly, Hao Yin et al. [YZL+17] suggested a decision support framework based on a flexible server placement, namely Tentacle, which aims to minimize the cost while maximiz- ing the overall system performance. The cost-effective application user allocation is another fundamental problem in edge computing first studied in [LHA+]. In research [LHGea20], 5 the authors formulated it as a bin packing problem and suggested a heuristic approach for approximating the sub -optimal solutions to this problem. In the recent researches [CZP], [DGT+], [ZLH+18], [HFKT], [ZZ18] and [BSEB], researchers have proposed investigative new techniques and approaches for data caching in the edge computing environment. However, even after having an optimal edge server placement, application user allocation, and data caching, we still need to consider the fact that trans- mitting the application data from the cloud to the edge servers also contributes a consider- able amount to the app vendors' expense structure. Thus, a cost-effective application data distribution strategy is needed to minimize and accurately estimate the app vendors' total cost. A recent research in Xia et al. [XCH+21], authors have formulated this problem and suggested a heuristic state-of-the-art algorithm to estimate the total cost, including both the C2E and E2E transmissions. This paper we extends the idea, refined the formulation and introduces a solution approach named EDD-NSTE based on network Steiner tree es- timation, for estimating the total cost associated with the edge data distribution problem from the app vendors' perspective. Chapter 3 EDD considering hop distances as the vendor's latency parameter 3.1 Problem Formulation In edge computing, adjacent edge servers at different stations can also communicate, share information as well as resources between them. Thus, edge servers of a specific geo-location can be combined to form a network, which can then be formulated as a graph. In this paper, we consider N edge servers of a specific area and model it as a graph G. Each node v of the graph G represents an edge-server v, and each edge of the graph connecting any two nodes u and v, i.e., e(u, v) represents the link between these two edge-servers. Thus, we use G(V, E) to represent the graph, where V is the set of edge servers in the graph and E is the set of links. Let R denote the set of destination edge servers in the graph G, i.e., the edge servers which must be sent data directly or indirectly from the cloud servers within the vendor's specific latency constraint. For example, in Figure 3.1, N = 10, which means we have 10 edge servers i.e., V = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. The set of edges E = {e(2, 3), e(6, 8), e(8, 7), · · · , e(2, 4)}. Similarly, the set of destination edge servers R = {2, 3, 4, 5, 6, 8, 9} shown as square box Table 3.1: Summary of Common Notations Notation Description G N V E R u, v e(u, v) c S H T γ ρ δ dlimit Dlimit Dv K GDT G E GR mst(G) T riples Gcv Gopt Cost(G) v(z) d(e) graph representing a particular region number of edge servers in a particular region set of all the edge servers in a particular region set of all edges in the graph, i .e, high-speed links set of destination edge servers in the graph G edge servers link/edge between edge server nodes u and v cloud server set of binary variables indicating initial transit edge server set of binary variables indicating edge servers visited during EDD process set of binary variables indicating the data distribution path ratio of cost of C2E to cost of 1-Hop E2E destination edge server density edge density vendors' latency constraint, or hop limit depth limit, Dlimit = dlimit + 1 depth of edge server v, Dv = dv + 1 depth limit, K = Dlimit directed graph with cloud as the root and edges replaced by the shortest path between two nodes in graph G metric closure of graph G set of edges in the metric closure of graph G subgraph of G induced over R minimum spanning tree of G set of all combination of 3 destination edge servers tree having cloud as the root and edges e ∈ G \ {c} optimal solution of the given EDD problem cost associated with any graph G centroid of a particular T riple z weight of edge e in the graph in the Figure 3.1. Common notations used throughout the text in the paper are given in Table 3.1. Lead cloud service providers like Google, Amazon, or Microsoft charge differently for 8 7 10 6 5 1 3 9 2 4 Fig. 3.1: EDD scenario with 10 edge servers data transactions. For e.g., hosting applications in Microsoft Azure involves careful con- sideration of the costs generated by three main components i.e., computing, storage, and network. Similarly, for any edge infrastructure provider, the pricing models usually are different for cloud-to-edge and edge-to-edge transmissions. Thus, we need to define a rela- tionship between these two transmissions, so that they can be incorporated easily into the generic optimization objective model. We define γ, to specify the ratio between the C2E and E2E transmission costs. For e.g., γ = 20, means the CostC2E is 20 times more than CostE2E. The cloud-to-edge transmissions are much more expensive than the edge-to-edge transmissions mainly because the graph nodes are connected via high-speed transmission links and are also not that distant apart from each other. Since the transmission latency is measured by the number of hops it takes to reach the destination node from the source in graph G, the C2E cost can then be easily transformed to γ times of the 1-hop E2E transmission cost in the generic model. Now there are two possible scenarios for data transmissions in EDD, firstly C2E, where the data is sent from

the cloud to some of the edge servers referred to as the "initial transit edge servers" and secondly E2E, where the data is transmitted between any two edge servers1. Thus, an EDD strategy comprises of two parts, a C2E and an E2E strategy. 1There is no necessity for initial transit edge servers to be a destination edge server and a destination edge server can also be an initial transit edge server as it may transfer data further to other servers. • In the C2E strategy, the set S = {s1, s2, · · · , sN}, where sv (1 ≤ v ≤ N) denotes the set of Boolean array representation of the initial transit edge servers, which receives data directly from the cloud. sv = ⎧ 1, ⎨ 0, if v is an initial transit edge server (3.1) if v is not an initial transit edge server ⎩ • Similarly, In the E2E strategy, the set T = {τ (1, 1), τ (1, 2), · · · , τ (N, N)}, where τ (u, v) (u, v ∈ V) denotes whether the data is transmitted through edge e(u, v) in G. τ (u, v) = ⎧ 1, if data is transmitted through e(u, v) (3.2) ⎨ 0, if data is not transmitted through e(u, v) ⎩ Since a reasonable EDD solution must link each destination edge server v ∈ R to an initial transit edge server in S through edges in E, thus, Connected(S, T, u, v) = true, ∀v ∈ R, ∃ u, su = 1 (3.3) Now, the app-vendor has the liberty to regulate the EDD latency constraint as per the application specifics and requirements. Let dlimit represent the app vendors EDD time constraint. Thus, for each of the destination edge server v, edge -to-edge latency or data transmission latency or path length in hops between v and its parent initial transit edge server in S, i.e., dv should not exceed this time or depth constraint. 0 ≤ dv ≤ dlimit, dv ∈ Z+, ∀v ∈ R (3.4) For example, consider the graph in Figure 4.2, the nodes of the graph represent the edge servers given by V = {1, 2, 3, 4, 5, 6, 7, 9}, where number of edge servers N = 9 and the 1 2 3 8 7 10 6 5 5 4 8 1 3 9 6 7 9 2 4 Fig. 3.2: EDD example to demonstrate Fig. 3.3: Optimal solution using integer dlimit programming set of high-speed links i.e., E = {e(1, 2), e(1, 3), e(1, 4), · · · , e(7, 9)}. Now, let the EDD time constraint as per the vendor's application requirements be dlimit = 2, which means that it should take less than or equal to two hops for a destination edge server to receive the data from any initial transit edge server. In Figure 4.2, if node 1 is the only node selected as the initial transit edge server, then one possible E2E strategy is to select edges {e(1, 2), e(1, 4), e(1, 5), e(5, 6), e(2, 3), e(4, 6), e(4, 8)}. This means that in the set TE2E the value of τ (1, 2) = τ (1, 4) = τ (1, 5) = τ (2, 3) = τ (4, 6) = τ (5, 6) = τ (4, 8) = 1. Accordingly, we can also obtain the time constraint limit of each of these destination edge servers, i.e, d1 = 0, d2 = d4 = d5 = 1, and d6 = d8 = d3 = 2. Now, given an EDD latency constraint of dlimit, the app vendor's aim is to minimize the EDD cost, which consists of the part incurred by the C2E transmissions and the E2E transmissions. Thus, the Edge Data Distribution (EDD) problem can be formulated as — minimize CostC2E(S) + CostE2E(T) (3 .5) { } while fulfilling the EDD latency constraint in Equation 4.6. As we can see, the Edge Data Distribution (EDD) is a constrained optimization problem based on a well-known NP- hard problem in graph theory known as the Steiner Tree [XCH+21]. 3.2 Solution Strategy In this section, we first present a refined formulation of the optimization approach given in Xia et al. [XCH+21] where the Edge Data Distribution (EDD) is formulated as a constrained integer programming problem which can then be solved by using any simple IP solver2. Since this method takes an exponential amount of time to produce results for large datasets, we designed an O(k) approximation method, named EDD-NSTE that can estimate solutions to the EDD problems in polynomial time. Also, we did the theoretical analysis of the performance of the proposed EDD-NSTE based solution approach. 3.2.1 Edge Data Distribution (EDD) as an Integer Programming The Edge Data Distribution (EDD) can be formulated as a constrained integer program- ming problem as given in [XCH+21]. Our refined formulation of the same is specified as below. The solution for the EDD should minimize the cost incurred for data transmissions within the app vendor's latency constraint of dlimit. Thus, to model the EDD problem as a generalized constrained integer programming optimization problem firstly we add the cloud server c into V , and then add the edges from cloud c to each edge server in graph G. Now, we can formulate the C2E strategy of EDD, S = {s1, s2, s3 · · · sN) by selecting edges in graph G, such that, Tc = {τ (c, 1), τ (c, 2), · · · , τ (c, N)}, where τ (c, v) denotes if the data is transferred from cloud server c to the edge server v. Here, we combine the C2E and E2E strategy of EDD into one parameter T = {τ (c, 1), τ (c, 2) · · · τ (c, N), τ (1, 1), · · · , τ (N, N)}, (3.6) where, τ (u, v) ∈ (0, 1) (u ∈ V, v ∈ V \ {c}) 2IP solvers are the frameworks or tools used to solve integer programming problems. For E.g., GLPK, LP Solve, CLP, CBC, CVXOPT, SciPy, Gurobi Optimizer, CPLEX, XPRESS, MOSEK, Google OR Tools, PuLP, etc. which indicates whether the edge e(u, v) is included in T . Secondly, we need to update the definition of EDD time constraint as the depth of the edge server in the graph with root as cloud. So, Dv represents the depth of the edge server v, where Dv = dv + 1 and Dc = 0, as the cloud is the root of the graph. Thus, we can define the limit as Dlimit = dlimit + 1 (3.7) Also, we need to define a parameter for each of the edge servers, Hv = ⎧ 0, if v is not visited during the EDD process (3.8) ⎨ 1, if v is visited during the EDD process here, H v = 1, ∀v ∈ R make sure that it include the destination edge servers into our ⎩ solution. Also, Hc = 1, as cloud always be a part of the solution. Now, the constrained integer programming optimization problem can be formally expressed as follows, min ⎛ γ τ (c, v) + (3.9) ⎝ v ∈ V∑\{c} u∑,v∈V∑\{c} τ (u, v)⎞ ⎠ subjected to constraints, H v = 1, ∀v ∈ R ∪ {c} τ (u, v) ≤ H u · H v, ∀u ∈ V, v ∈ V \ {c} τ (u, v) = H v, ∀u ∈ V, v ∈ V \ {c} ∑ τ (c, v) ≥ 1 v∈∑V\{c} τ (u, v), H v ∈ (0, 1) τ (c, v) = 1 ∀v ∈ V \ {c}, Dv = 1 (3.10) (3.11) (3.12) (3.13) (3.14) (3.15) Dc = 0, Dc ≤ Dv ≤ Dlimit, ∀v ∈ V \ {c} (3.16) Dv − Du = 1, ∀u, v ∈ V, τ (u, v) = 1 (3.17) Here, in-order to minimize the total cost incurred by both the C2E and E2E edges given by Equation 4.10, we must satisfy the above constraints. Equation 4.13 make sure that, if edge e(u, v) is considered a part of the solution, or, τ(u, v) = 1, then both the nodes u and v must be visited, i.e., Hv = 1 and Hu = 1. Equation 4.14 suggests a constraint that the summation of τ(u, v) of the edges incoming at the particular node v, must be equal to the Hv of that particular node, i.e., whether or not the node v is visited, if edges incoming onto it are considered a part of the solution. Equation 4.15 suggests that the solution must have atleast one cloud server, i.e., summation of τ(c, v) must be greater than or equal to 1. Equation 3.15 makes sure that the node servers having depth of 1, must be connected to the cloud server. Equation 4.17 suggests that the depth of any edge server v, must be within

[0,dlimit], and Equation 4.18 suggests that for nodes u, v connected through edge e(u, v) in the EDD solution must satisfy the depth difference of 1. Now, this Integer Programming model is a generalized model and thus, for vendor's specific cost models say cost(c, v) and cost(u, v) can be simply incorporated into our Equation 4.10, as min ⌈ τ (c, v) · cost (c, v) + τ (u, v) · cost (u, v)⌋ (3.18) ⌊ v ∈∑V\{c} u ∑ ,v∈V∑\{ c} ⌋ Here, as we know Dlimit ≥ 1, as otherwise the cloud cannot transfer data to any edge server, and for the base case of Dlimit = 1, the cost incurred will be γR times the cost of 1-hop E2E transmission. Here in the refined formulation Equation 4.15, Equation 4.16, and Equation 3.15 are the new constraints added to the formulation of the integer programming method given 14 in [XCH+21]. These constraints narrow the possible solution search-space and also updates the final solution to contain all the edges, i.e., the C2E edges and the E2E edges. Example to understand Integer Programming Consider an example in Figure 4.3 that displays the optimal solution generated, if we solve the graph in Figure 3.1 using the integer programming approach, with EDD time constraint of dlimit = 1. The C2E strategy specify node 8 and node 9 as the initial transit edge servers which receives the data directly from the cloud server and thus, the cost incurred for the cloud to edge data transmission is CostC2E = 2 γ times of the 1-hop E2E transmission cost. Now, these nodes transmit the data to the other destination edge servers (represented by light blue square nodes) that can be reached within the given time constraint of dlimit. Thus, the cost incurred for the edge to edge data transmission is CostE2E = 5 times of the 1-hop E2E transmission cost. The integer programming approach selects the edges E = {e (c, 8), e(c, 9), e(8, 2), e(8, 6), e(9, 3), e(9, 4), e(9, 5)}, for data transmissions and thus, the optimal total cost incurred is the sum of CostC2E and CostE2E, i.e., 2 γ + 5 times of the 1-hop E2E transmission cost. 3.2.2 Edge Data Distribution (EDD) as a Network Steiner Tree Estimation The Edge Data Distribution (EDD) is an NP-Hard [XCH+21] problem. Thus, finding an optimal solution for large-scale EDD scenarios is troublesome due to the exponential time complexity of the algorithm. To address this issue, we propose an approach, named Edge Data Distribution as a Network Steiner Tree Estimation (EDD-NSTE), to estimate large- scale EDD solutions. The estimation ratio for EDD- NSTE is O(k), which means that the ratio of the cost by EDD- NSTE solution and that of the optimal solution is O(k) in the worst case, where k is constant. In our proposed approach, we calculates the estimated cost incurred for a given graph in two steps — • First of all, we calculate a network Steiner tree approximation [Zel93a] with destina- 15 Algorithm 1: Network Steiner Tree Approximation 1 F ← GR; W ← φ; T riples ← {z ⊂ R : ‖z‖ = 3} 2 foreach z ∈ T riples do 3 find v which minimizes the s ∈z d(v, s) 4 v(z) ← v 5 d(z) ← s∈z d(v(z), s) ∑ 6 while true ∑do 7 M ← an MST of F; F indSave(M) 8 find z ∈ T riples which maximizes, win = max save(e) + min save(e) - d(z) e∈z e∈z 9 if win ≤ 0 then 10 break 11 F ← F[z]; put (W, v(z)) 12 Find a steiner tree for R ∪ W in graph G using MST algorithm tion edge servers R, as the set of distinguished vertices, in the given graph G. • Then, we use an approach which estimates a rooted minimum Steiner tree, by adding cloud c to the graph, and then slicing and fine-tuning the rooted Steiner tree with the vendor's latency constraint of Dlimit. The proposed approach works as follows: Given a graph G, we first calculate the metric closure of the graph, denoted by G, which is a complete graph of G, having edge lengths equal to the shortest distance between the vertices in G. GR denotes the subgraph of G induced by a vertex subset R ⊆ V , i .e., the subgraph on destination edge servers. We denote the minimum spanning tree (MST) of any graph G as mst (G). For any triple set of vertices z = (u, v, w), the graph G[z] represent a graph resulting from contraction of Algorithm 2: To calculate FindSave(M) 1 if M ≠ empty then 2 e ← max weight edge in M; 3 x ← d(e) 4 M1, M2 ← the subgraphs linked through e 5 foreach vertex v1 of M1 and v2 of M2 do 6 save(e) ← x, where e = e(v1, v2) 7 FindSave(M1); FindSave(M2) Algorithm 3: EDD-NSTE Algorithm Input: G, GDT, Dlimit, depth, cost, visited Output: Gfinal 1 Construct the graph GST from Algorithm 1. Take the maximum connectivity node in the graph GST and connect it to the cloud to make it a rooted tree. Update the distances between consecutive nodes by the shortest path between them and let this graph be GDT 2 Initialize depth[c] ← 0, visited[c] ← 1, depth[v] ← ∞, visited[v] ← 0, ∀ v ∈ V, Dlimit ← dlimit + 1, Update GDT to include the minimum cost path between any two vertices with edges in graph G 3 Update GDT to form a directed rooted tree, with cloud as the root 4 Run a BFS to compute the minimum depth of each vertex v ∈ GDT from cloud 5 foreach vertex v, in the path from root to leaf node and parent p in Depth-First Search Algorithm do 6 if visited [v] ≠ 1 then 7 if depth [v] ≤ Dlimit and v ∈ R then 8 add edge e(p, v) in Gfinal 9 visited[v] ← 1 10 else if v ∈/ R then 11 add edge e(p, v) in Gfinal 12 visited[v] ← 1 13 else if depth[v] > D limit and v ∈ R then 14 depth[v] ← 1, visited[v] ← 1 15 add edge e(c, v) in Gfinal 16 foreach child node u of node v in graph G in DFS order do 17 if depth[u] > depth[v] + 1 and depth[v] + 1 ≤ Dlimit then 18 if e(v, u) ∈/ GDT then 19 update edge in GDT from v to u 20 depth[u] ← depth[v] + 1 21 run BFS and update depths of the vertices in GDT 22 Run a DFS and remove unnecessary edges from graph Gfinal 23 Calculate total cost incurred as sum of cost[e], ∀ e ∈ Gfinal two edges i.e., e(u, v) and e(v, w). T riples denotes the set of all combination of 3 destination edge servers, v(z) denotes the centroid of a particular triple, i.e., the vertices other than the destination edge servers whose sum of distance from a triple is minimum and d(e) denotes the weight of the edge e. 8 7 10 6 5 1 3 9 2 4 Fig. 3.4: Steiner Tree estimated by Algorithm 1 8 7 10 6 5 1 3 9 2 4 Fig. 3.5: Final EDD solution estimated by Algorithm 3 The Pseudo code of the proposed approach is given in Algorithm 1, which loops contin- uously to find the best possible reduction in the cost of the minimum spanning tree of F, where the initial value of F = GR. The algorithm does so by adding the three edges of G having a common end node, i.e., the centroid, and consecutively removing the max-weight edge from each resulting cycle using the save matrix, from the MST of F. This class of algorithms is also known as the Triple Loss Contracting Algorithm [GHNP01b] [Zel93a]. The output of Algorithm 1 is an approximated Steiner tree GST, with vertices in R ∪ W and edges e ⊂ E. Here, we calculate save matrix for edges in the graph F, to estimate the maximum win for a given triple. To calculate the save matrix, a pseudo code for a recursive function FindSave(M) is given in Algorithm 2. The implementation time complexity of this algo-

rithm is O(‖ R‖2). save(e) = mst(F) − mst(F [e]), e = e(v1, v2), ∀ v1, v2 ∈ F (3.19) Once we get an approximated Steiner tree, it is necessary to add cloud server c to the given graph by adding an edge from cloud to the maximum connectivity node in the approximated Steiner tree. Then we convert resulting graph into a rooted tree, with cloud as the root. We perform the same by a breadth-first search on GST. In this process, we also convert the edges between any two nodes of GST by the shortest path between two nodes 18 in graph G, as the Steiner tree GST, was approximated over the metric closure of graph G. Let this resultant directed graph be GDT. The pseudo code for slicing and fine-tuning the graph to satisfy the vendor's latency constraint is shown in Algorithm 3. In this approach, we take the directed Steiner tree GDT and initialize the depth[] and the visited[] for each node of the graph, with visited[c] = 1. It then runs the Breadth-First Search to compute the minimum depths of each node assuming the cloud server as the root. After that, it runs the Depth-First Search and — • for each unvisited destination edge server with depth less than or equal to Dlimit, it adds the corresponding edge into the final solution Gfinal. • for each unvisited edge server, which is not a destination edge server, it again adds the corresponding edge into the final solution Gfinal. • for each unvisited destination edge server having a depth greater than Dlimit, it con- nects that edge server directly to the cloud and adds edge from cloud to this edge server in Gfinal. It then runs a Depth-First Search to update the minimum depths of the remaining unvisited edge servers assuming this edge server as the root. Finally, this algorithm removes the unnecessary edges from the graph Gfinal and calculates the cost of the solution as the sum of the cost of edges in Gfinal. Example to understand EDD-NSTE algorithm Let us consider the graph in the Figure 3. 1, with vendors' latency constraint or hop-limit of dlimit = 1. When we apply Algorithm 1 onto it, it selects edges {e(2, 3), e(2, 4), e(2, 8), e(3, 5), e(3, 9), e(5, 6)} to construct the estimated Steiner tree given in Figure 4.4, using the triple loss contracting mechanism, which in this case is also the optimal Steiner tree possible. Now, as we can see the node 2 is the node with the max-connectivity in the estimated steiner tree, thus, we add this node to the cloud directly, as given in Figure 4.5 to make a rooted EDD solution. The node 3, node 4, and node 8 can be reached within the 1-hop 19 distance from node 2, thus edges {e(c, 2), e(2, 3), e(2, 4), e(2, 8)} are included into the final EDD solution. When the algorithm reaches node 3, it encounters two nodes, i.e., node 5 and node 9 exceeding the hop-limit. Now, since node 5 is encountered first, the algorithm adds node 5 to the cloud directly, which then updates the final EDD solution to include node 6 and node 9, as they can be reached within vendors' latency constraint with node 5 as the root. The final EDD solution then becomes {e(c, 2), e(c, 5), e(2, 3), e(2, 4), e(2, 8), e(5, 6), e(5, 9)}, which incurs a total cost of 2γ + 5, including both the C2E and E2E edges. Time Complexity analysis of EDD-NSTE algorithm There has been a wealth of research in approximating the best Steiner Tree for a given general graph as mentioned in Gropl et al. [GHNP01b], with a lower bound of smaller than 1.01 [GHNP01a] theoretically. In this paper, we first introduced an algorithm to calculate the Network Steiner Tree approximation, based on the algorithm proposed in Zelikovsky et al. [Zel93a] which has a time complexity of O(‖ V ‖‖E‖ + R4) and an approximation ratio of O(11/6). The same approximation ratio can be achieved with a faster implementation of O(‖R‖ (‖E‖ +‖V ‖‖R‖ +‖V ‖ log‖V ‖)) as mentioned in [Zel93b]. The algorithm EDD-NSTE has a time complexity of O(‖ V ‖ +‖E‖) in the worst case. Thus, the total time complexity of the EDD-NSTE Algorithm along with network Steiner tree estimation is O((‖V ‖+‖E‖)+‖V ‖‖E‖+‖R‖ 4) and in the worst case the time complexity becomes O(‖ V ‖ 4). In the next subsection, we prove that the EDD- NSTE is an O(k) approximation algo- rithm with experimental results suggesting that EDD-NSTE significantly outperforms the other three representative approaches in comparison with a performance margin of 86.67% (as per our experimental result given Section V). EDD-NSTE is an O(k) approximation algorithm As we discussed in the previous section, let tree Gfinal be the final EDD solution produced by Algorithm 3, GST be the minimum cost steiner tree approximated by Algorithm 1, and Gcv be the tree having cloud c as the root and edges e(c, v) , ∀ v ∈ Gfinal \ {c}. Also, let the cost associated with each of these trees be Cost(Gfinal), Cost(GST), and Cost(Gcv) respectively. Let Gopt be the optimal solution for the given EDD problem and the optimal cost associated be Cost(Gopt), thus, as per the research in Zelikovsky et al. [Zel93a], we can write C ost(GST) = 11 6 C ost(Gopt) (3.20) Now, let vo = c be the cloud server and vi as the ith edge server, where i ∈ {1, 2, · · · , m}, which brings extra paths during the DFS iteration. Also, let K = Dlimit be the depth limit provided by the vendor. Thus, for any edge server there exists two possibilities, i.e., • vi exceeds the latency constraint of Dlimit and thus, we need to add an extra edge (c, vi), which in turn adds a cost of Cost(Gcv)(c, vi) = γ. • ∃ a path (c, vi) as a path from (c, vi-1) and (vi-1, vi), such that vi does not exceed the latency constraint, which in turn adds a cost of Cost(Gfinal)(c, vi) ≤ Cost(Gcv)(c, vi-1) + Cost(GST)(vi-1, vi). In the first case, when edge server vi exceeds the latency constraint, then we must have Cost(GST)(c, vi) ≥ γ + K ≥ 1 + Cost(Gcv)(c, vi) K γ (3.21) ( ) where, K denotes the latency constraint Dlimit. Thus, we can obtain the combined equation as 1 + K γ Cost(Gcv)(c, vi) ≤ ( ) (3.22) Cost(GST)(c, vi) ≤ Cost(Gcv)(c, vi-1) + Cost(GST)(vi-1, vi) 21 Summing the above equation for all of the m edge servers, i.e., vi, where i ∈ {1, 2, ...m}, we have m 1 + K γ Cost(Gcv)(c, vi) ≤ m ( ) ∑i=1 m (3.23) Cost(Gcv)(c, vi-1) + Cost(GST)(vi-1, vi) ∑i=1 ∑i=1 Now, as we know from the basic inequality of positive numbers, that mi=−11 Cost(Gcv)(c, vi) ≤ i=1 Cost(Gcv)(c, vi), so the above equation yields — m ∑ ∑ K m m γ Cost(Gcv)(c, vi) ≤ Cost(GST)(vi-1, vi) (3.24) ∑i=1 ∑i=1 K m γ Cost(Gcv) ≤ Cost(GST)(vi-1, vi) (3.25) ∑i=1 For each edge server vi, that changes its path in Gfinal during the DFS traversal without adding the path (c, vi), the total cost of update is less than that of Cost(Gcv)(c, vi) and thus, the cost after forming GST must not be more than mi=1 Cost(Gcv)(c, vi). Now, each edge in GST is traversed at most twice during the DFS tra∑versal, thus, we have m Cost(GST)(vi-1, vi) ≤ 2 · Cost(GST) ∑i=1 (3.26) Thus, combining both above equations, results in Cost(Gcv) ≤ Cost(GST) 2γ K (3.27) Finally, as we know, that the cost of the final EDD solution approximated by EDD-NSTE is Cost(Gfinal) ≤ Cost(Gcv) + Cost(GST) (3.28) Cost(Gfinal) ≤ Cost(GST) + Cost(GST) 2γ K (3.29) Cost(Gfinal) ≤ 2γ (K + 1 Cost(GST)) Cost(Gfinal) ≤ 11 2γ 6 + 1 Cost(Gopt) (K) (3.30) (3.31) Now,

since K and γ are both constants, we can safely assume k = 161 2Kγ + 1 , and thus, EDD- NSTE is an O(k) approximation algorithm. ( ) As compared to Xia et al. [XCH+21], our approximation is lightly better as 161 2Kγ + 1 ≤ 2 2Kγ + 1 . But in actual run this produce even better result as compared(to ED)D- A([XCH+)21]. 3.3 Experimental Evaluation We conducted experiments to evaluate the effectiveness of the proposed EDD-NSTE ap- proach in comparisons with the state of art EDD-A [XCH+21], refined LP solution and other representative approaches. All these experiments were conducted on an Ubuntu 20.04.1 LTS machine equipped with Intel Core i5- 7200U processor (4 CPU's, 2. 50GHz) and 16GB RAM. The integer programming simulation was done on Google Collab (Python 3 Google Computer Engine Backend, 12.72GB RAM). 3.3.1 Simulation Settings and Approaches for comparison of the result Description of the state of art solution technique (EDD-A) and other representative ap- proaches are — • Greedy Connectivity (GC) — In this approach, for each node, the approach define its connectivity which is equal to the number of destination servers that have not received the data yet and are reachable within the vendors' latency constraint. Then it select the node with the maximum connectivity and make it an initial transmit edge server which then transmits the data to other servers within the app vendors' latency limit of dlimit until all destination edge servers have received the data. 23 • Random — In this approach, it randomly select the initial transit edge servers which directly receive data from the cloud and then transmit it to the other servers within the app vendors' latency limit of dlimit, until all destination edge servers have received the data. • EDD Integer Programming — As discussed above in the Section IV, we refine the formulation of given EDD problem into a 0-1 integer programming problem which can then be solved by using any simple IP solvers. This method take a huge amount of time to produce the result in the case of large datasets as is exponential in complexity. • EDD-A Algorithm — This is state of the art approach given in [XCH+21], in this approach it first calculate a connectivity-oriented minimum Steiner tree (CMST) using an O(2) approximation method. This method calculates a minimum Steiner tree having cost at most twice the optimal Steiner tree on the graph. It then uses a heuristic algorithm to calculate the minimum cost of transmission incurred using E2E and C2E edges in the CMST. This algorithm is proved to have an O(k) approximation solution and a time complexity O(‖V ‖ 3), in the worst case. 3.3.2 Dataset Used The experiments were performed on a standard real-world EUA dataset 3, which contains the geo -locations of more than 1,400 edge-server base stations of Melbourne, Australia. The set of destination edge servers R is generated randomly. The edges between the nodes of the graph are also generated randomly using an online random connected graph model generator tool. In the experiments, the value of γ is set to 20 which is same setting as specified in [XCH+21]. 3 https://github.com/swinedge/eua-dataset T otal EDD cost incurred (times 1-Hop E2E) 400 T otal EDD cost incurred (times 1-Hop E2E) 800 0 100 200 300 400 500 600 700 Number of nodes N in the graph G Fig. 3.6: N v/s Total EDD cost 800 900 1,000 300 200 Integer Programming 100 EDD-NSTE Algorithm EDD-A Algorithm Greedy Connectivity Random 0 600 400 200 0 Integer Programming EDD-NSTE Algorithm EDD-A Algorithm Greedy Connectivity Random 0 5 10 15 20 25 30 35 40 45 50 Number of destination edge servers R Fig. 3.7: R v/s Total EDD cost 3.3.3 Experiment Results Different EDD scenarios and algorithms mentioned above in the section solution strat- egy and other approaches are simulated and compared to each other by varying different 25 T otal EDD cost incurred (times 1-Hop E2E) 800 Integer Programming EDD-NSTE Algorithm 600 EDD-A Algorithm Greedy Connectivity Random 400 200 0 0 Fig. 3.8: Latency constraint Dlimit v/s Total EDD cost, lower is better 1 2 3 4 5 6 7 8 Latency constraint Dlimit 9 10 T otal EDD cost (times 1-Hop E2E) 600 400 200 0.05 0.10 0.25 0.30 0.50 IP EDD-NSTE EDD-A GC Random Fig. 3.9: ρ v/s Total EDD cost parameters as mentioned below — • The number of nodes N in the graph G. • The number of destination edge servers R in the graph. 26 T otal EDD cost (times 1-Hop E2E) 800 600 400 200 1.00 1.50 2.00 2.50 3.00 Random GC EDD-A EDD-NSTE IP Fig. 3.10: δ v/s Total EDD cost, lower is better • The vendor's latency constraint limit, i.e., Dlimit. Among the above three factors, we vary one of the factors while keeping the others constant and then compare the estimated total EDD cost incurred in each case with the optimal cost generated by the integer programming approach. Figure 4.8, displays the relationship between the number of nodes N in the graph G v/ s total EDD cost incurred by different simulation settings and approaches. The value of the other parameters such as Dlimit = 3, R = 25 random nodes, and γ = 20 were fixed during the experiment. For N < 100, there is a very slight difference between other approaches but as the value of N increases the difference becomes significantly more. Resultant cost produced by EDD-NSTE approach is lower (is better) as compared to EDD-A in all the cases. Similarly, Figure 4.9, displays the relationship between the number of destination edge servers R v/s the total EDD cost incurred by different other settings. For this experiment the value of other parameters such as N = 100, Dlimit = 3 and γ = 20 were fixed during the experiment. For ‖R‖ ≤ 10, the EDD cost incurred in all approaches except the Random 27 10 10 Time (in s) 8 6 4 Time (in s) 8 6 4 2 2 0 0 50 N 100 0 0 0.5 1 ρ 1.5 2 Fig. 3.11: N v/s Computational Overhead, lower is better 10 Fig. 3.12: ρ v/s Computational Overhead,lower is better 100 8 Time (in s) 6 4 Time (in s) 2 0 1 2 3 4 5 Dlimit 80 60 40 20 0 0 0.5 1 1.5 2 δ Fig. 3.13: Dlimit v/s Computational Fig. 3.14: δ v/s Computational Overhead, lower is better Overhead, lower is better approach results in the same cost but the number of destination edge servers increase in the graph the difference becomes quite broad. The Greedy Connectivity algorithm for the given dataset with ‖R‖ = 30, performs better than other approximation algorithms like EDD- A and EDD-NSTE. This may be an anomaly. However, the Greedy Algorithm, cannot guarantee to maintain a certain performance ratio on any given dataset, unlike EDD-A or EDD-NSTE. Resultant cost produced by EDD-NSTE approach is lower (is better) as compared to EDD-A in most of the cases except ‖R‖ = 30. Now, the only factor left to alter is Dlimit, and the rest of the parameters are kept constant similar to the previous case. The number of destination edge servers ‖R‖ = 25 for this experiment. Figure 4.10, displays the relationship between the vendor's latency constraint Dlimit v/s the total EDD cost incurred by different solution approaches. As we know, these two quantities are inversely related, i.e., with

an increase in the value of latency constraint, the total EDD cost decreases. The total EDD cost reaches its maxima of γR when Dlimit = 1, as in this case all the destination edge servers are connected directly to the cloud server to fulfill the vendors' latency constraint. Also, Random approach and Greedy Connectivity algorithms become constant for N ≥ 7, as then almost both the Random or Greedy selection of servers is not able to reduce the EDD cost. Resultant cost produced by EDD-NSTE approach is lower (is better) as compared to EDD-A in all the value of Dlimit from 2 to 9. Here, we consider another parameter referred to as the destination edge server density. It is defined as the ratio of number of destination edge server upon total number of edge servers, i.e., Destination Edge Server Density (ρ) — ρ = ‖V ‖ ‖R‖ (3.32) Figure 4.11, displays the bar-graph relationship between the total EDD cost and the density of the destination edge servers. With increase in the density, the total EDD cost associated with each of the approach tend to increase as a whole. Among the other algo- rithms and approaches in comparison, EDD-NSTE Algorithm performed well overall. As ρ increases the resultant cost difference between EDD-NSTE approach and LP approach is lower as compared to the resultant cost difference between EDD-A approach and LP approach. Similarly, we consider another experimental parameter known as edge density. This 29 parameter helps to estimate the overall density of the graph. A very dense graph will have a higher value of δ than a sparse graph or a tree. Thus, it is defined as the total no of edges or high-speed links in the graph upon total no of edge servers, i.e, Edge Density (δ) — δ = ‖V ‖ ‖E‖ (3.33) Figure 4.12, displays the bar-graph relationship between the total EDD cost incurred and the density of the graph, or edge density. With the increase in the edge density, the total EDD cost associated with each of the simulations tends to decrease as now shorter paths from cloud to other edge servers exist and thus, any destination edge server can be reached within the vendors' latency constraint following these shorter paths, thereby, decreasing the number of initial transit edge servers, which in turn decreases the total EDD cost incurred as the CostC2E is significantly greater than CostE2E. Here also, we see as ρ increases the resultant cost difference between EDD-NSTE approach and LP approach is lower as compared to the resultant cost difference between EDD-A approach and LP approach. Figure 4.18 to Figure 4.21 displays the computational overhead of EDD-NSTE algorithm versus different other graph parameters like number of nodes(N), destination edge server density(ρ), the vendors' latency constraint(Dlimit), and edge density(δ) respectively. As we can see, with an increase in N , δ or ρ, the computational overhead of EDD-NSTE increases, as its worst-case time complexity is O(‖V ‖ 4), whereas, with an increase in Dlimit, the computational overhead first increases and then decreases, as relaxing the vendors' latency constraint after a certain limit decreases the extra overhead caused by the slicing and pruning algorithm. Chapter 4 EDD considering propagation delay as the vendor's latency parameter 4.1 Problem Formulation The edge-server network is a physical network of edge-servers connected via transmission links. Using these links, the edge servers can communicate and share data with each other and their users. This network, where nodes of the graphs denote the edge servers, the edges represent the transmission links, and edge-weights representing the length of the links or the propagation delays between edge-servers is called a edge-server network. In this research, we consider an edge-server network of a particular geographical region and formulate it as a graph G(V, E, W), with V being the number of edge servers, E as the set of edges or transmission links, and W is the weights associated with each edge. In Figure 4.1, we have N = 10, which means we have 10 edge servers i.e., V = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. The set of edges E = {e (1, 4), e(1, 2), e(1, 3) · · · e(9, 8)}. The weights corresponding to these edges are W = {100, 5, 6, · · · 20}. Similarly, the set of destination edge servers R = {3, 7, 6, 9, 1, 4, 2}. The common notations used in this paper are listed in Table 3.1. 3 3 7 1 7 8 3 2 0 6 6 9 9 1 1 0 0 1 11 5 4 2 10 5 2 3 5 Fig. 4.1: EDD scenario with 10 edge servers Different companies charge differently for different data transactions. For e.g., the cost of hosting applications in Amazon AWS depends on the individual user's usage, as it offers a pay-as-you-go approach. Similarly, from an app vendor's perspective, the cost consists of two components, the cloud-to-edge (C2E) transactions, and the edge-to-edge (E2E) transactions. Now, since there are two possible scenarios for the data transmission in EDD, we will consider each one of them separately. Firstly, we have the cloud-to-edge (C2E) transactions, where data is transferred between the cloud to some of the edge servers called the "initial transit edge servers". Secondly, we have the edge- to-edge (E2E) transactions, where the data is transferred between the different edge servers. Thus, an EDD solution strategy must also comprise two parts, i.e., a C2E strategy, and an E2E strategy. In the C2E strategy, we define the set S = (s1, s2 · · · sN), where sv (1 ≤ v ≤ N) denotes the set of boolean array representation of the initial transit edge servers, i.e., the nodes which receives data directly from the cloud. sv = ⎧ 1, if v is an initial transit edge server (4.1) ⎨ 0, if v is not an initial transit edge server ⎩ Similarly, In the E2E strategy, we define the set T = (τ (1, 1), τ (1, 2) · · · τ (N, N)), where τ (u, v) (u, v ∈ V) denotes whether the data is transmitted through edge e(u, v) in the graph G. 1, if data is transmitted through e(u, v) τ (u, v) = ⎧ (4.2) ⎨ 0, if data is not transmitted through e(u, v) Since a reasonable EDD strategy must connect each destination edge server v ∈ R to an ⎩ initial transit edge server in S through edges in E, thus, Connected(S, T, u, v) = true, ∀v ∈ R, ∃ u, su = 1 (4.3) speed (smedium) in the medium as constant, i.e., Now, here we consider the length of the transmission links for the delay constraint, as the propagation delay (in time) is directly proportional to the link-length (Llink), assuming the Llink P delay = smedium (4.4) We here assume that the bandwidth delay and queuing delay at the nodes is negligible, and thus the vendors' latency or delay constraint will depend just on the total length of the links. Also, the mode of transmission between the cloud server and each of the edge server is same. Thus, we denote the propagation delay between them as a constant, i.e., the weights of these edges in the graph G(V, E, W) will also be a constant denoted by λ, thus, W (c, v) = λ, ∀ v ∈ V \ {c} (4.5) Now, the app-vendor has the liberty to regulate the EDD delay constraint as per the appli- cation specifics and requirements. Let P limit represent the app vendors' EDD propagation delay constraint, and Llimit represent the equivalent EDD length

constraint. Thus, for each of the destination edge server v, edge -to-edge latency or data transmission latency or total path length between v and its connected parent initial transit edge server in S, i.e., Lv must not exceed this delay constraint. 0 ≤ Lv ≤ Llimit, Lv ∈ Z+, ∀v ∈ R (4.6) For example, consider the graph in Figure 4.2, the nodes of the graph represent the edge servers given by V = {1, 2, 3, 4, 5, 6, 7, 9}, where number of edge servers N = 9 and the set of high-speed links i.e., E = {e(1, 2), e(1, 3), e(1, 4) · · · e(7, 9)}. Now, let the EDD length constraint as per the vendor's application requirements be Llimit = 12. This means each destination edge server must receive data within the maximum path length of 12, where path starts from an initial transit edge server. In Figure 4.2, if node 4 is the only edge server selected as the initial transit edge server, then one possible E2E strategy is to select edges {e(1, 2), e(1, 4), e(1, 5), e(2, 3), e(4, 8)}. This means that in the set TE2E there is τ (1, 2) = τ (1, 4) = τ (1, 5) = τ (2, 3) = τ (4, 8) = 1. Accordingly, we can also obtain the time constraint limit of each of these destination edge servers, i.e, L1 = 1, L2 = 3, L4 = 0, L5 = 10, L8 = 10, and L3 = 10. Thus, the Edge Data Distribution (EDD) problem can be formulated as — minimize CostC2E(S) + CostE2E(T) (4.7) ( ) while fulfilling the EDD EDD length constraint in Equation 4.6. As we can see, the Edge Data Distribution (EDD) is a constrained optimization prob- lem based on a well-known NP- Hard problem in graph theory known as the Steiner Tree [XCH+21]. 1 2 2 7 3 9 1 4 5 4 10 8 12 27 6 6 3 7 8 9 3 3 7 7 8 3 1 2 0 6 6 9 9 1 100 11 1 5 4 2 10 5 2 3 5 Fig. 4.2: EDD example to demonstrate Fig. 4.3: Optimal Solution using integer Llimit programming 4.2 Solution Strategy In this section, we first present a modified implementation of the optimization approach given in [XCH+21] where the Edge Data Distribution (EDD) is formulated as a constrained integer programming problem which can then be solved by using any simple IP solver1. Since this method takes an exponential amount of time to produce results for large datasets, we will design an O(k) approximation method, named EDD-NSTE that can estimate so- lutions to the EDD problems in polynomial time. This will be followed by a theoretical analysis of the performance of the EDD-NSTE algorithm. 4.2.1 Edge Data Distribution (EDD) as an Integer Programming The Edge Data Distribution (EDD) can be formulated as a constrained integer program- ming problem as given in [XCH+21]. Our modified implementation of the same is specified as below. The solution to the EDD problem must minimize the cost incurred for data transmissions while fulfilling the app vendor's EDD length constraint of Llimit. Thus, to model the EDD 1IP solvers are the frameworks or tools used to solve integer programming problems. For E.g., GLPK, LP Solve, CLP, CBC, CVXOPT, SciPy, Gurobi Optimizer, CPLEX, XPRESS, MOSEK, Google OR Tools, PuLP, etc. problem as a generalized constrained integer programming optimization problem firstly we add the cloud c into V , and then add the edges from cloud c to each edge server in graph G. Now, we can formulate the C2E strategy of EDD, S = (s1, s2, s3 · · · sN) by selecting edges in graph G, such that, Tc = (τ (c, 1), τ (c, 2) · · · τ (c, N)), where τ (c, v) denotes whether the data is transmitted from cloud server c to the edge server v. Here, we combine the C2E and E2E strategy of EDD into one parameter T = (τ (c, 1), τ (c, 2) · · · τ (c, N), τ (1, 1) · · · τ (N, N)), (4.8) where, τ (u, v) ∈ (0, 1) (u ∈ V, v ∈ V \ {c}) which indicates whether the edge e(u, v) is included in T . We also need to define a parameter for each of the edge servers, i.e., 0, if v is not visited during the EDD process Hv = ⌈ (4.9) ⏐ ⏐ 1, if v is visited during the EDD process here, H v = 1, ∀v ∈ R will make sure that we include the destination edge servers into our ⏐ ⏐ ⌊ solution. Also, Hc = 1, as cloud will always be a part of the solution. Now, the constrained integer programming optimization problem can be formally expressed as follows, min ⌈ τ (c, v) · W (c, v) + ⌊ v ∈∑V\{c} τ (u, v) · W (u, v)⌋ u∑,v∈V ∑\{c} ⌋ subjected to constraints, H v = 1, ∀v ∈ R ∪ {c} W (c, v) = λ, ∀ v ∈ V \ {c}τ (u, v) ≤ H u · H v , ∀u ∈ V , v ∈ V \ {c} τ (u, v) = H v , ∀u ∈ V , v ∈ V \ {c} ∑ 36 (4.10) (4.11) (4.12) (4.13) (4.14) τ(c, v) ≥ 1 v∈∑V\{c} τ(u, v), Hv ∈ (0,1) Lc = 0, Lc ≤ Lv ≤ Llimit, ∀v ∈ V \ {c} Lv − Lu = W (u, v) , ∀u, v ∈ V , τ (u, v) = 1 (4.15) (4.16) (4.17) (4.18) Here, in-order to minimize the total cost incurred by both the C2E and E2E edges given by Equation 4.10, we must satisfy the above constraints. The Equation 4.12 makes sure that the weight of the edge from cloud to each of the edge server is equal to λ. The Equation 4.13 will make sure that, if edge e (u, v) is considered a part of the solution, or, τ(u, v) = 1, then both the nodes u and v must be visited, i.e., Hv = 1 and Hu = 1. Equation 4.14 suggests a constraint that the summation of τ(u, v) of the edges incoming at the particular node v, must be equal to the Hv of that particular node, i.e., whether or not the node v is visited, if edges incoming onto it are considered a part of the solution. Equation 4.15 suggests that the solution must have atleast one cloud server, i.e., summation of τ(c, v) must be greater than or equal to 1. Equation 4.17 suggests that the depth of any edge server v, must be within [0,Llimit], and Equation 4.18 suggests that for nodes u, v connected through edge e(u, v) in the EDD solution must satisfy the depth difference of 1. Example to understand Integer Programming Consider an example in Figure 4.3 that displays the optimal solution generated, if we solve the graph in Figure 3.1 using the integer programming approach, with EDD length constraint of Llimit = 110, and W (c, v) = 100, ∀v ∈ V \ {c}. The C2E strategy will specify node 2 and node 7 as the initial transit edge servers which receives the data directly from the cloud server and thus, the cost incurred for the cloud to edge data transmission is CostC2E = 200. Now, these nodes will transmit the data to the other destination edge 3 3 7 7 1 3 2 0 8 6 6 9 9 5 1 1 0 0 4 2 1 10 11 5 2 3 5 Fig. 4.4: Steiner Tree estimated by Algorithm 1 3 3 7 7 1 3 2 0 8 6 6 9 9 1 1 0 0 1 11 5 4 2 10 5 2 3 5 Fig. 4.5: Final EDD solution estimated by Algorithm 3 servers (represented by light blue square nodes) that can be reached within the given time constraint of Llimit. Thus, the cost incurred for the edge to edge data transmission is CostE2E = 19. The integer programming approach selects the edges E = {e(c, 2), e(c, 7), e(2, 1), e(2, 4), e(4, 10), e(10, 9), e(7, 3), e(7, 6)}, for data transmissions and thus, the optimal total cost incurred is the sum of CostC2E and CostE2E, i.e., 219. 4.2.2 Edge Data Distribution (EDD) as a Network Steiner Tree Estimation This method is similar to as the method defined in the previous chapter. Algorithm 1 and Algorithm 2 are used in the similar fashion, after that we slice and fine-tune the graph to obtain the final EDD solution. The algorithm for slicing and fine-tuning the graph to satisfy the vendor's EDD length constraint is presented in Algorithm 4. In this algorithm,

we take the directed steiner tree GDT and initialize the pathLength[] and the visited[] for each node of the graph, with visited[c] = 1, pathLength[c] = 1, and pathLength[v] = ∞, ∀v ∈ V . It then runs the Breadth-First Search to compute the minimum length of path of each node from the root node, assuming the cloud server as the root. After that, it runs the Depth-First Search and — • for each unvisited destination edge server with path length less than or equal to Llimit, 38 Algorithm 4: EDD-NSTE Algorithm Input: G, Llimit, pathLength, W, visited Output: Gfinal 1 Construct the graph GST from Algorithm 1. Take the maximum connectivity node in the graph GST and connect it to the cloud to make it a rooted tree. Update the distances between consecutive nodes by the shortest path between them and let this graph be GDT 2 Initialize pathLength[c] ← 0, visited[c] ← 1, pathLength[v] ← ∞, visited[v] ← 0, ∀ v ∈ V, Update GDT to include the minimum cost path between any two vertices with edges in graph G 3 Update GDT to form a directed rooted tree, with cloud as the root 4 Run a Breadth-First Search Algorithm to compute the minimum depth of each vertex v ∈ GDT from cloud 5 foreach vertex v, in the path from root to leaf node and parent p in Depth-First Search Algorithm do 6 if visited [v] ≠ 1 then 7 if pathLength [v] ≤ Llimit and v ∈ R then 8 add edge e(p, v) in Gfinal 9 visited[v] ← 1 10 else if v ∈/ R then 11 add edge e(p, v) in Gfinal 12 visited[v] ← 1 13 else if pathLength[v] > Llimit and v ∈ R then 14 pathLength[v] ← W(c, v), visited[v] ← 1 15 add edge e(c, v) in Gfinal 16 foreach child node u of node v in graph G in Depth-First Search order do 17 if pathLength[u] > pathLength [v] + W(u, v) and pathLength [v] + W(u, v) ≤ Llimit then 18 if e(v, u) ∈/ GDT then 19 update edge in GDT from v to u 20 pathLength[u] ← pathLength[v] + W(u, v) 21 run a Breadth-First Search Algorithm and update depths of the vertices in GDT 22 Run a Depth-First Search and remove the unnecessary edges from the graph Gfinal 23 Calculate the total cost incurred as the sum of W(u, v) ∀ e(u, v) ∈ Gfinal it adds the corresponding edge into the final solution Gfinal. • for each unvisited edge server, which is not a destination edge server, it again adds the corresponding edge into the final solution Gfinal. • for each unvisited destination edge server having a depth greater than Llimit, it con- nects that edge server directly to the cloud and adds edge from cloud to this edge server in Gfinal. It then runs a Depth-First Search to update the minimum depths of the remaining unvisited edge servers assuming this edge server as the root. Finally, this algorithm removes the unnecessary edges from the graph Gfinal and calculates the cost of the solution as the sum of the cost of edges in Gfinal. Example to understand the EDD-NSTE algorithm Consider the graph in the Figure 3. 1, with data transmission latency Llimit = 110, and W (c, v) = λ = 100, ∀v ∈ V \ {c}. When we apply 2Algorithm 1 onto it, it selects edges {e(2, 1), e(2, 4), e(1, 3), e(4, 10), e(10, 9), e(3, 7), e(7, 6)} to construct the estimated steiner tree given in Figure 4.4, using the triple loss contracting mechanism, which in this case is also the optimal steiner tree possible. Now, as we can see the node 1 is the node with the max- connectivity in the estimated steiner tree, thus, we add this node to the cloud directly, as given in Figure 4.5 to make a rooted EDD solution. The node 2, node 3, node 4, and node 7 can be reached within the 110 unit distance from the cloud, thus edges {e(c, 1), e(1, 2), e(1, 3), e(2, 4), e(3, 7)} are included into the final EDD solution. When the algorithm reaches node 10, it encounters node 9 exceeding the Llimit. Now, since node 9 is encountered first, the algorithm adds node 9 to the cloud directly, which then updates the final EDD solution to include node 6 and node 9, as they can be reached within data transmission constraint with node 9 as the root. The final EDD solution then becomes {e(c, 1), e(c, 9), e(1, 2), e(1, 3), e(2, 4), e(3, 7), e(9, 6)}, which incurs a total cost of 228 distance units, including both the C2E and E2E edges. 40 1 1 1 2 1 2 2 1 1 3 1 3 Fig. 4.6: EDD-A considering hop distances 3 Fig. 4.7: EDD-A considering propagation delays EDD-NSTE is an O(k) approximation algorithm The same proof will work in this case as well with K = Llimit − γ, instead of K = Dlimit as was defined in the previous case. 4.3 Experimental Evaluation We experimentally evaluated the performance of above methods with the other simulation settings and approaches in comparison as discussed below. All experimental were conducted on an Ubuntu 20.04.1 LTS machine equipped with Intel Core i5- 7200U processor (4 CPU's, 2. 50GHz) and 16GB RAM. The integer programming simulation was done on Google Collab (Python 3 Google Computer Engine Backend, 12.72GB RAM). 4.3.1 Simulation Settings and Approaches for comparison of the result We compared our experimental results with the other optimization techniques and repre- sentative approaches — • Greedy Connectivity (GC) — In this approach, for each node, the approach define its connectivity which is equal to the number of destination servers that have not received the data yet and are reachable within the vendors' EDD length constraint. 41 Then it select the node with the maximum connectivity and make it an initial transmit edge server which then transmits the data to other servers within the app vendors' EDD length limit of Llimit until all destination edge servers have received the data. • Random — In this approach, it randomly select the initial transit edge servers which directly receive data from the cloud and then transmit it to the other servers within the app vendors' EDD length limit of Llimit, until all destination edge servers have received the data. • EDD Integer Programming — As discussed above in the Section IV, we refine the formulation of given EDD problem into a 0-1 integer programming problem which can then be solved by using any simple IP solvers. This method take a huge amount of time to produce the result in the case of large datasets as is exponential in complexity. • EDD-A Algorithm — This is state of the art approach given in [XCH+21], in this approach it first calculate a connectivity-oriented minimum Steiner tree (CMST) using an O(2) approximation method. This method calculates a minimum Steiner tree having cost at most twice the optimal Steiner tree on the graph. It then uses a heuristic algorithm to calculate the minimum cost of transmission incurred using E2E and C2E edges in the CMST. This algorithm is proved to have an O(k) approximation solution and a time complexity O(‖V ‖ 3), in the worst case. To compare this algorithm in the propagation delay scenario we have introduced temporary nodes in the graph which corresponds to the 1-hop distance in the edge- weight graph, i.e., Figure 4.6 represents the graph with 2 nodes and edge weights. In the scenario of hop distances these edge-weights can simply be taken as 1, but in the case of propagation delay we need to convert this graph into Figure 4.7 where temporary nodes between the two nodes will be same as the 1-hop distance edge

length. Thus, now we can compare the results against EDD-A with Dlimit = Llimit. 4.3.2 Dataset Used The experiments were conducted on a widely used EUA2 and SLNDC3 Dataset. The set of destination edge servers R is generated randomly. The links between the edge-servers are also generated randomly using an online random connected graph model generator tool. In the experiments, γ = 20 is taken. 4.3.3 Experiment Results Different EDD scenarios and algorithms mentioned above in the section solution strategy and other approaches are simulated and compared to each other by varying different parameters as mentioned below — • The number of nodes N in the graph G. • The number of destination edge servers R in the graph. • The vendor's EDD length limit, i.e., Llimit. Among the above three factors, we vary one of the factors while keeping the others constant and then compare the estimated total EDD cost incurred in each case with the optimal cost generated by the integer programming approach. The experiment is done in two sets — • Set 1 — Above factors are varied and the results are calculated on EUA Dataset. • Set 2 — Above factors are varied and the results are calculated on SLNDC Dataset. Figure 4.8 and Figure 4.13, displays the relationship between the number of nodes N in the graph G v/ s total EDD cost incurred by different simulation settings and approaches. The value of the other parameters such as Llimit = 150, R = 25 random nodes, γ = 100, and weights of the edges are in the range [1, 50] were fixed during the experiment for 2https://github.com/swinedge/eua-dataset 3https://snap.stanford.edu/data/ 6,000 Integer Programming 5,000 EDD-NSTE Algorithm EDD-A Algorithm T otal EDD cost 4,000 Greedy Connectivity Random 3,000 2,000 1,000 0 0 100 200 300 400 500 600 700 800 900 1,000 Number of nodes N in the graph G Fig. 4.8: Set 1: N v/s Total EDD cost 1 ·104 Integer Programming 0.8 EDD-NSTE Algorithm EDD-A Algorithm T otal EDD cost Greedy Connectivity 0.6 Random 0.4 0.2 0 0 5 10 15 20 25 30 35 40 45 50 Number of destination edge servers R Fig. 4.9: Set 1: R v/s Total EDD cost both the sets. As we can see that for both sets of experiments EDD-A and EDD-NSTE perform better than any other heuristic approach. For N < 100 there is a very slight difference between other approaches but ", IEEE Transactions on Systems Man and Cybernetics Part C (Applications and Reviews), 5/2008">as we increase the value of N , the difference becomes significantly more, and overall EDD-NSTE performs better than other heuristic 1.2 ·104 1 T otal EDD cost 0.8 0.6 0.4 0.2 0 Integer Programming EDD-NSTE Algorithm EDD-A Algorithm Greedy Connectivity Random 0 50 100 150 EDD length constraint Llimit 200 250 300 350 400 450 500 Fig. 4.10: Set 1: EDD length constraint Llimit v/s Total EDD cost 1,800 1,600 1,400 T otal EDD cost 1,200 1,000 800 600 400 200 0.05 0.10 0.25 0.30 0.50 IP EDD-NSTE EDD-A GC Random Fig. 4.11: Set 1: ρ v/s Total EDD cost methods. Similarly, Figure 4.9 and Figure 4.14, displays the relationship between the number of destination edge servers R v/s the total EDD cost incurred by different other settings. For T otal EDD cost 1,500 1,000 500 1.00 1.50 2.00 2.50 3.00 Random GC EDD-A EDD-NSTE IP Fig. 4.12: Set 1: δ v/s Total EDD cost 6,000 Integer Programming 5,000 EDD-NSTE Algorithm EDD-A Algorithm T otal EDD cost 4,000 Greedy Connectivity Random 3,000 2,000 1,000 0 0 100 200 300 400 500 600 700 800 900 1,000 Number of nodes N in the graph G Fig. 4.13: Set 2: N v/s Total EDD cost this experiment the value of other parameters such as N = 100, Llimit = 150, γ = 100, and weight of the edges are in the range [1, 50] were fixed during the experiment. As we can see that in Set 1, EDD-NSTE and EDD-A perform very close to the optimal solution, 46 1 ·104 Integer Programming 0.8 EDD-NSTE Algorithm EDD-A Algorithm T otal EDD cost Greedy Connectivity 0.6 Random 0.4 0.2 0 0 5 10 15 20 25 30 35 40 45 50 Number of destination edge servers R Fig. 4.14: Set 2: R v/s Total EDD cost ·104 1.4 1.2 T otal EDD cost 1 0.8 0.6 Integer Programming 0.4 EDD-NSTE Algorithm EDD-A Algorithm 0.2 Greedy Connectivity Random 0 0 50 100 150 200 250 300 350 400 450 500 EDD length constraint Llimit Fig. 4.15: Set 2: EDD length constraint Llimit v/s Total EDD cost and in Set 2, there is a considerable difference between these two heuristics. Now, the only factor left to alter is Llimit, and the rest of the parameters are kept constant similar to the previous case. The number of destination edge servers ‖R‖ = 25 for this experiment. Figure 4.10 and Figure 4.15, displays the relationship between the vendor's EDD length 1,800 1,600 T otal EDD cost 1,400 1,200 1,000 800 600 400 200 0.05 0.10 0.25 0.30 0.50 IP EDD-NSTE EDD-A GC Random Fig. 4.16: Set 2: ρ v/s Total EDD cost 1,400 T otal EDD cost 1,200 1,000 800 600 400 200 1.00 1.50 2.00 2.50 3.00 Random GC EDD-A EDD-NSTE IP Fig. 4.17: Set 2: δ v/s Total EDD cost constraint Llimit v/s the total EDD cost incurred by different simulations. As we know, these two quantities are inversely related, i.e., with an increase in the value of EDD length constraint, the total EDD cost decreases. The total EDD cost reaches its maxima of γR when Llimit = γ, as in this case all the destination edge servers are connected directly to the cloud server to fulfill the vendors' EDD length constraint. Also, as we can see that the Random and Greedy Connectivity algorithms become constant for N ≥ 450 in Set 1, as then almost any random or greedy selection of servers is not able to improve on the EDD cost. Here, we consider another parameter referred to as the destination edge server density. It is defined as the ratio of number of destination edge server upon total number of edge servers, i.e., Destination Edge Server Density (ρ) — ‖R‖ ρ = ‖V ‖ (4.19) Figure 4.11 and Figure 4.16, displays the bar-graph relationship between the total EDD cost and the density of the destination edge servers. As we can see, with the increase in density, the total EDD cost associated with each of the simulations tends to increase as a whole. Among the other algorithms and approaches in comparison, EDD-NSTE Algorithm performed well overall. Similarly, we consider another experimental parameter known as edge density. This pa- rameter helps to estimate the overall density of the graph. A very dense graph will have a higher value of δ than a sparse graph or a tree. Thus, it is defined as the total no of edges or high-speed links in the graph upon total no of edge servers, i.e, Edge Density (δ) — δ = ‖V ‖ ‖E‖ (4.20) Figure 4.12 and Figure 4.17, displays the bar-graph relationship between the total EDD cost incurred and the density of the graph, or edge density. As we can see, with the increase in the edge density, the total EDD cost associated with each of the simulations tends to decrease as now shorter paths from cloud to other edge servers exist and thus, any destination edge server can be reached within the vendors' EDD length constraint following 49 10 10 Time (in s) 8 6 4 Time (in s) 8 6 4 2 2

0 0 50 N 100 0 0 0.5 1 ρ 1.5 2 Fig. 4.18: N v/s Computational Fig. 4.19: ρ v/s Computational Overhead Overhead 10 100 8 Time (in s) 6 4 Time (in s) 2 0 1 2 3 4 5 Llimit 80 60 40 20 0 0 0.5 1 1.5 2 δ Fig. 4.20: Llimit v/s Computational Fig. 4.21: δ v/s Computational Overhead Overhead these shorter paths, thereby, decreasing the number of initial transit edge servers, which in turn decreases the total EDD cost incurred as the CostC2E is significantly greater than CostE2E. Figure 4.18 to Figure 4.21 displays the computational overhead of EDD-NSTE algorithm versus different other graph parameters like number of nodes(N), destination edge server density(ρ), the vendors' EDD length constraint(Llimit), and edge density(δ) respectively. 50 As we can see, with an increase in N , δ or ρ, the computational overhead of EDD-NSTE increases, as its worst-case time complexity is O(‖V ‖ 4), whereas, with an increase in Llimit, the computational overhead first increases and then decreases, as relaxing the vendors' EDD length constraint after a certain limit decreases the extra overhead caused by the slicing and pruning algorithm. Chapter 5 Conclusion The EDD-NSTE algorithm suggested in this paper performs considerably better than the other simulations and approaches implemented for the edge data distribution problem. The total EDD cost approximated by this algorithm is closer to the optimal solution than the basic hard-coded approaches like Greedy Connectivity or Random. The algorithm also performs significantly better than the last best EDD-A algorithm suggested in [XCH+21]. The EDD-A algorithm is an O(k) approximation algorithm1 and the EDD-NSTE algorithm is an O(k') approximation algorithm where k'< k and thus, the performance of EDD-NSTE is better than EDD-A, which coincides with the experimental evaluations done so far. EDD-NSTE algorithm produced better solutions than EDD-A or other approaches in more than 86.67% of the test cases(considering hop distance as the vendor's latency param- eter), and 80.35% of the test cases(considering propagation delay as the vendor's latency parameter), ranging from very dense graphs to sparse graphs or trees. Greedy Connectivity may produce solutions better than EDD-A or EDD-NSTE in some of the cases, but the greedy solutions are just an ad-hoc solution and thereby, cannot always guarantee to produce results that closer to the actual solution whereas the latter algorithms are restricted by their approximation ratio to always produce solutions below 1O(k) approximation algorithm means that solution produced by the algorithm is atmost k times the optimal solution. a certain maximum value for a given graph, thus they perform better than these methods on an average. Chapter 6 Future Work • The lower bound to approximate Steiner Tree in general graphs is smaller than 1.01 theoretically. The current best algorithm has an approximation ratio of (1.39+ep- silon). Thus, implementing a better way of estimating the Steiner Tree can help improve the approximation ratio and so, we can work on incorporating these meth- ods in our edge data distribution solution. • We can also try and implement a heuristic algorithm completely independent from Steiner Tree approximation that can estimate better solutions. • Also, we can try and include other delays into the EDD problem formulation like transmission delays and queuing delays and make it efficient for end-to-end delay guarantees. References [BSEB] Martin Breitbach, Dominik Schäfer, Janick Edinger, and Christian Becker. Context-aware data and task placement in edge computing environments. In Proc. IEEE Int. Conf. Pervasive Comput. Commun., pp. 1-10, 2019. [CZP] Xuanyu Cao, Junshan Zhang, and H. Vincent Poor. An optimal auction mechanism for mobile edge caching. In Proc. 38th IEEE Int. Conf. Distrib. Comput. Syst., pp. 388-399, 2018. [DGT+] Utsav Drolia, Katherine Guo, Jiaqi Tan, Ra jeev Gandhi, and Priya Narasimhan. [DPW04] A. Davis, J. Parikh, and W. E. Weihl. Edge Computing: Extending Enterprise Applications to the Edge of the Internet. WWW Alt. '04, page 180–187, 2004. [FH] https://www.oculus.com/facebook-horizon/. [GHNP01a] Clemens Gröpl, Stefan Hougardy, Till Nierhoff, and Hans Jürgen Prömel. Lower bounds for approximation algorithms for the steiner tree problem. Springer Verlag Berlin Heidelberg, 46(5):217–228, 2001. [GHNP01b] Clemens Gröpl, Stefan Hougardy, Till Nierhoff, and Jürgen Prömel. Approx- imation algorithms for the steiner tree problems in graphs. Springer Verlag Berlin Heidelberg, 46(5):235–279, 2001. [HFKT] Raluca Halalai, Pascal Felber, Anne-Marie Kermarrec, and François Taïani. Agar: A caching system for erasure-coded data. In Proc. 37th IEEE Int. Conf. Distrib. Comput. Syst., pp. 23-33, 2017. [LHA+] Phu Lai, Qiang He, Mohamed Abdelrazek, Feifei Chen, and et al. Optimal edge user allocation in edge computing with variable sized vector bin packing. In Proc. Int. Conf. Service-Oriented Comput., pp. 230-245, 2018. [LHGea20] Phu Lai, Qiang He, John Grundy, and et al. Cost-effective app user allocation in an edge computing environment. IEEE Transactions on Cloud Computing, 31(15):1–13, 2020. [PB08] Mukaddim Pathan and Rajkumar Buyya. A Taxonomy of CDNs, pages 33–77. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. [SCZ+16] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge Computing: Vision and Challenges. IEEE Internet of Things Journal, 3(5):637–646, 2016. [WWea20] Bastiaan Wissingh, Christopher A. Wood, and et al. Information-centric net- working (ICN): content-centric networking (ccnx) and named data networking (NDN) terminology. RFC, 8793:1–17, 2020. [XCH+21] HXiaoyu Xia, Feifei Chen, Qiang He, John C. Grundy, Mohamed Abdelrazek, and Hai Jin. Cost-Effective App Data Distribution in Edge Computing. IEEE Tran. on Parallel and Distributed Systems, 32(1):31–43, 2021. [YBX+17] Hong Yao, Changmin Bai, Muzhou Xiong, Deze Zeng, and Zhangjie Fu. Het- erogeneous cloudlet deployment and user-cloudlet association toward cost ef- fective fog computing. Concurrency Comp. Pract. Exp., 29(16), 2017,. [YZL+ 17] [Zel93a] [Zel93b] [ZLH+ 18] [ZML18] [ZZ18] Hao Yin, Xu Zhang, Hongqiang H. Liu, Yan Luo, Chen Tian, Shuoyao Zhao, and Feng Li. Edge provisioning with flexible server placement. IEEE Trans. on Parallel Distribution System, 28(4):1031–1045, 2017. Alexander Z Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. Algorithmica, 9(5):463–470, 1993. Alexander Z Zelikovsky. A faster approximation algorithm for the steiner tree problem in graphs. Information Processing Letters, 46(5):79–83, 1993. Ke Zhang, Supeng Leng, Yejun He, Sabita Maharjan, and Yan Zhang. Coop-erative Content Caching in 5G Networks with Mobile Edge Computing. IEEE Wireless Communications, 25(3):80–87, 2018. Dongfang Zhao, Mohamed Mohamed, and Heiko Ludwig. Locality-aware scheduling for containers in cloud computing. IEEE Transactions on Cloud Computing, 8(2):635–646, 2018. Xi Zhang and

Qixuan Zhu. Hierarchical caching for statistical QoS guaranteed multimedia transmissions over 5G edge computing mobile wireless networks. IEEE Wireless Comm., 25(3):12–20, 2018. 6 7 8 9 10 11 12 13 16 17 20 22 24 28 30 31 32 33 34 35 37 39 42 43 44 45 47 48 51 52 53 54 55 56 57 58 59