A Decentralized Approach for Resource Discovery using Metadata Replication in Edge Networks

Ilir Murturi and Schahram Dustdar, *Fellow, IEEE*

Abstract—Recent advancements in distributed systems have enabled deploying low-latency edge applications (i.e., IoT applications) in proximity to the end-users, respectively, in edge networks. The stringent requirements combined with heterogeneous, resource-constrained and dynamic edge networks make the deployment process a challenging task. Besides that, the lack of resource discovery features make it particularly difficult to fully exploit available resources (i.e., computational, storage, and IoT resources) provided by low-powered edge devices. To that end, this article proposes a decentralized resource discovery mechanism that enables discovering resources in an automatic manner in edge networks. Through replicating resource descriptions (i.e., metadata), edge devices exchange information about available resources within their scope in a peer-to-peer manner. To handle the resource discovery complexity, we propose a solution to built edge networks as a flat model and enable edge devices to be organized in clusters. Our approach supports the system in coping with the dynamicity and uncertainty of edge networks. We discuss the architecture, processes of the approach, and the experiments we conducted on a testbed to validate its feasibility on resource-constrained edge networks.

Index Terms—Edge computing, Internet of Things, decentralized, resource discovery

1 Introduction

Researchers from academia and industry stakeholders suggest adding more computational resources (i.e., perceived as *edge devices*) in proximity to the end-users to overcome high-latency issues between the cloud and the Internet of Things (IoT) domain [1]. Edge devices are low-powered computer entities featuring different capabilities; resources available may differ in terms of computational capabilities and IoT resources attached to them. A wide range of available resources at the edge has introduced new opportunities such as deploying low-latency, privacy-awareness, and resilient *edge applications* (e.g., IoT applications). In this regard, many studies have been carried to exploit edge networks for various purposes (i.e., from processing sensory data streams to EdgeAI applications) [2]. Notably, we consider edge networks as resource-constrained, heterogeneous, and dynamic environments where multiple low-powered edge devices in proximity are connected. In this sense, we may have various edge networks (e.g., smart building, smart home, drone network, etc.) where end-users may deploy different edge applications.

In the past few years, computer scientists have been mostly focused on proposing multiple techniques for resource allocation problems to minimize latency and maximize resource utilization at the edge. Notably, today's applications are not monolithic; they are divided into multiple independent deployable tasks. Each task may have various resource requirements that need to be fulfilled by available edge devices upon deployment. Tasks are characterized by requirements such as computational (i.e., processing, memory, storage), energy, or bandwidth. However, resource allocation approaches often overlook the dependence between tasks and IoT resources (e.g., sensors and actuators)[3]. Additionally, despite the numerous advantages introduced by edge networks, communication between edge devices and the network organization has been neglected by many research papers [4], [5]. According to the paper [6], the communication and network organization type of a platform affects the functionality of the final applications deployed at the edge infrastructure.

Very few research works consider IoT resources as an application requirement that needs to be fulfilled when deploying them at the edge [7], [8]. For example, computing a local weather forecast in a smart agriculture setting may require various IoT resources such as temperature and humidity readings from available sensors across a crop field [5]. In such a scenario, application tasks dependent on particular IoT resources must be deployed on edge devices providing those resources. Notably, edge devices are not equipped with the same capabilities, and such a stringent constraint reduces the number of eligible deployments at the edge. For example, the allocation technique [7] tries to overcome the problem by enabling sharing IoT resource information within neighbor nodes. Similarly, the proposed solution [8] acknowledges the problem; however, it faces latency issues, and it considers a limited number of edge devices in the network topology. Nevertheless, both approaches do not address issues related to the communication between edge devices, network organization, and resource discovery.

• *The authors are with Distributed Systems Group, TU Wien, 1040 Vienna, Austria. E-mail: {i.murturi, dustdar}@dsg.tuwien.ac.at.*

Manuscript received 27 Apr. 2020; revised 6 Jan. 2021; accepted 12 May 2021. Date of publication 20 May 2021; date of current version 7 Oct. 2022. (Corresponding author: Ilir Murturi.)
Digital Object Identifier no. 10.1109/TSC.2021.3082305

To overcome these shortcomings, we discuss two major issues. First, edge networks should be designated to handle the complexity of discovering resources in a decentralized and automatic manner. Thus, we design edge networks in a flat model where edge devices in certain proximity are connected in a peer-to-peer (P2P) way. A set of edge devices form a *cluster*; while multiple connected clusters form an edge network, respectively *an edge neighborhood*. Besides that, we introduce system coordinators with their corresponding functionalities to organize edge devices and support the resource discovery process in an edge neighborhood. Second, resource managements' fundamental objective is to discover resources available at the edge [9]. Edge devices provide heterogeneous resources and are equipped with a rich set of IoT resources. We refer to heterogeneous resources as computational, sensing, context data, or other domain-specific resources. Naturally, performing a resource discovery algorithm for each resource on the entire network is possible. However, such a process is computationally intensive, and resources are discovered by querying the entire network based on a keyword [10]. Thus, we advocate that exchanging information about available resources between edge devices in an automatic manner enables: i) sharing resources across the whole system and ii) performing complex queries by edge devices locally.

In this paper, we extend the framework of [11] with a methodology to built edge networks as a flat model and enable edge devices to be organized in clusters. Our proposed framework enables discovering heterogeneous resources and make them available at the runtime. A resource is described by providing certain core information about the functionality and its properties. This type of description, known as the resource's metadata, is replicated among edge devices and stored in a decentralized manner. Furthermore, we treat privacy aspects based on each edge device's resource preferences, ensuring that not all resources are advertised across the whole system. Specifically, our concrete contributions are as follows:

- We develop a prototype enabling edge devices to be connected in a P2P manner and form an edge network. Our approach is built on top of the Kademlia Protocol [12] used as the communication protocol between edges devices. To enable scaling of our approach, we propose a solution to break down edge networks into clusters as well as introduce new coordinators to handle the complexity to discover resources automatically.
- We advocate decentralization as the system can operate without a static entity for discovering available resources. Essentially, coordinators are placed dynamically and run on the most suitable edge devices providing various services. Our approach supports the system in coping with the environment's dynamicity and uncertainty and continuously re-evaluates coordinators' placements.
- To validate the approach's feasibility, we show that the prototype's footprint is limited to hardware resources and network bandwidth. We evaluate our prototype on a testbed composed of low-powered ARM-based edge devices.

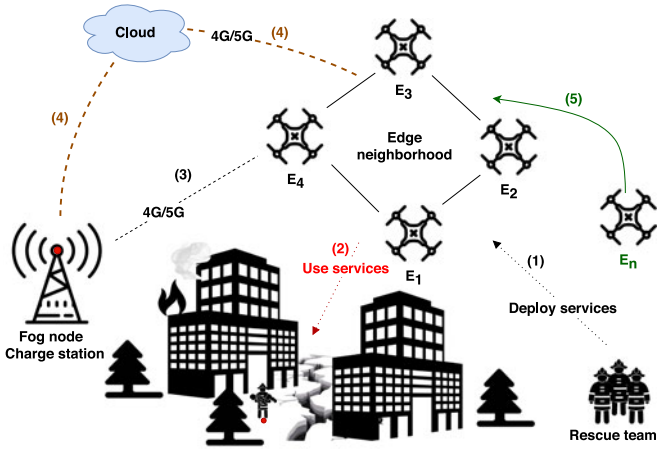

Fig. 1. IoT safety application [13].

The rest of the paper is structured as follows. After a motivating example used throughout this paper in Section 2, related work is presented in Section 3. Section 4 presents our approach to organize edge devices in edge neighborhoods and resource modeling. Section 5 describes in detail the proposed algorithm in charge of determining system coordinators and the framework for automatic resource discovery in edge neighborhoods. Section 6 provides the experiment results to evaluate the proposed solution. Finally, Section 7 concludes the paper and outlines future work directions.

2 Motivation Scenario

To motivate our discussion, we consider emergencies such as natural disasters (e.g., earthquakes, fires, floods) in a city [13]. Various city areas can be affected by natural disasters such as earthquakes, which can destroy infrastructure, cause injury or death, and trap people under buildings. In such situations, time is valuable, and drones could be used to analyze the situation and assist rescue teams in locating and communicating with victims trapped under a collapsed building. In such a scenario, multiple connected drones are essentially edge devices forming an *edge neighborhood*. Drones flying over the city's affected areas (i.e., neighborhoods) assist rescue teams in locating people trapped under a collapsed structure. Each drone is equipped with various computation capabilities and integrated sensors (e.g., radar sensors, infrared cameras, electronic noses, etc.). We consider drones as multipurpose devices where the rescue teams may deploy various services depending on the emergency. Furthermore, we consider three-tier Edge-Cloud infrastructure (i.e., cloud, fog, and edge) [14]. Fog devices (i.e., server-graded hosts) placed in base stations provide computational and storage capabilities to edge neighborhoods. In addition to that, base stations may provide docker charge stations for charging drones. Cloud hosts can be used to store data for long terms.

We assume that drones are connected via a wireless connection provided by the ground users (i.e., rescue team) or by drones [15] covering a particular city area (e.g., neighborhood). Based on the situation seen in Fig. 1, we assume that the rescue team deploys (1) a public safety IoT service that detects a dangerous zone in the affected area (i.e., discovering cracks, smoke, hazardous gases, etc.). Such a service

aims at helping rescue teams (2) find a safe path, and avoid danger zones. The service is dependent on various resources such as multiple infrared cameras, radar sensors, and an electronic nose that are integrated into various drones. Since each drone is a potential candidate to run the service, it is then evident that each edge devices should be able to automatically discover resources in a decentralized manner and make them available at runtime. In such use case scenarios, we cannot depend on the service availability [16] offered from physically static entities (3-4). Additionally, edge-based systems with centralized architecture cannot run properly due to the network dynamicity (i.e., drones may join (5) and leave often).

3 Related Work

We divide related work into two categories. First, we review P2P approaches and communication types used at the edge. Afterwards, we review related techniques on resource management as they apply to IoT.

3.1 Communication in Edge Networks

Many studies propose various allocation techniques to facilitate the deployment of IoT applications in edge networks. In this context, many platforms use different communication types aiming to achieve particular system goals. The current literature recognizes and briefly discusses communication types at the edge [17], [18]. According to the paper [6], three main types of communications in Edge Computing are identified: hierarchical, P2P, and hybrid.

Notably, P2P approaches have shown great potential to handle edge infrastructures in a scalable manner [19]. Therefore, a lot of research has been conducted in this context, resulting in many approaches that aim at organizing edge devices using different communication types [20]. Due to their fully distributed nature, P2P architectures are both scalable, reliable [21], and fault-tolerant [22]. Many edge-based platforms use P2P to organize edge devices within the edge network [23]. Similarly, Cabrera *et al.* [24] propose a P2P-based fog platform that enables storing data generated from IoT devices. In the proposed approach, data is stored among fog devices in a distributed manner. A fog device can restore corrupted data by asking other fog device in the network. In the mentioned works, devices are organized in a P2P manner and are assumed to have partial view [25] or full view of the network (i.e., $\mathcal{O}(1)$ protocols). In contrast to the mentioned works, our approach provides a decentralized mechanism to organize edge devices in clusters. Our solution determines the most suitable edge devices to place coordinators in edge networks and adapts to the network changes that may occur. Nevertheless, the proposed solution makes edge neighborhoods autonomous environments and less dependent on centralized nodes.

3.2 Resource Management

Performing a resource discovery algorithm for each resource on the entire edge neighborhood is computationally demanding. For instance, queries like discovering all cameras in a certain area are becoming highly desirable for IoT applications. In the general sense, such system behavior in decentralized common DHT protocols is hardly achievable. Even though some DHT-based approaches support discovering data through multi-attribute queries [26]; however, such methods remain unsuitable in IoT systems and edge networks due to the high content lookup latency. In addition to that, resource discovery is a critical challenge for IoT application performance.

Service-based discovery has been widely studied [27], [28]. Paganelli *et al.* [10] introduce a DHT-based IoT service discovery that supports multi-attribute and range queries. Furthermore, the proposed solution enables real-time monitoring of resource positions since it updates resource location periodically. Santos *et al.* [22] propose a resource discovery service for resource provisioning in fog environments. The proposed solution is based on DHTs and enables exchanging provisioning information about the available resources (i.e., performance metrics, workloads, etc.). However, in contrast to our approach, the proposed solution does not address privacy aspects, does not consider discovering IoT resources, and no actual provisioning mechanisms are discussed.

Resource allocation and management have been widely studied both in cloud and fog computing [17]. A taxonomy of resource management at the edge is presented in [9]. Up to now, many factors have been considered including time (e.g., computation [29]), data size [30], cost (e.g., networking and deployment[31], execution [32]), deploying self-adaptive IoT systems [33], user-application context [32], etc., which have been found to play important roles in resource and service provisioning. Jain *et al.* [34] propose a solution where the IoT application is divided into multiple tasks annotated with location information. The application is decomposed into fragments and deployed to the corresponding individual compute nodes based on the annotation. In contrast to the mentioned research papers, the resource discovery aspect has been mostly ignored. Furthermore, none of the papers have addressed privacy aspects when considering resource discovery.

Our approach's second novelty lies in a decentralized mechanism for automatic resource discovery in edge networks. Discovering resources at once represents a feasible and optimal solution for edge neighborhoods. Through replicating metadata between edge devices, we enable end-users or edge applications to perform locally various complex queries. Moreover, our resource discovery mechanism considers resource privacy preferences ensuring that not all resources are advertised across the whole system.

4 Edge Networks and Resource Modeling

This section introduces our approach to organize edge devices in an edge neighborhood. We describe basic definitions and then discuss communication protocol between edge devices. Subsequently, we discuss architecture modeling and resource modeling in Section 4.4.

4.1 Definitions

We refer to an *edge neighborhood* as a resource-constrained edge network, which is comprised of edge devices placed close to each other (see Fig. 2). Edge neighborhoods are formed in various geographical areas and within different

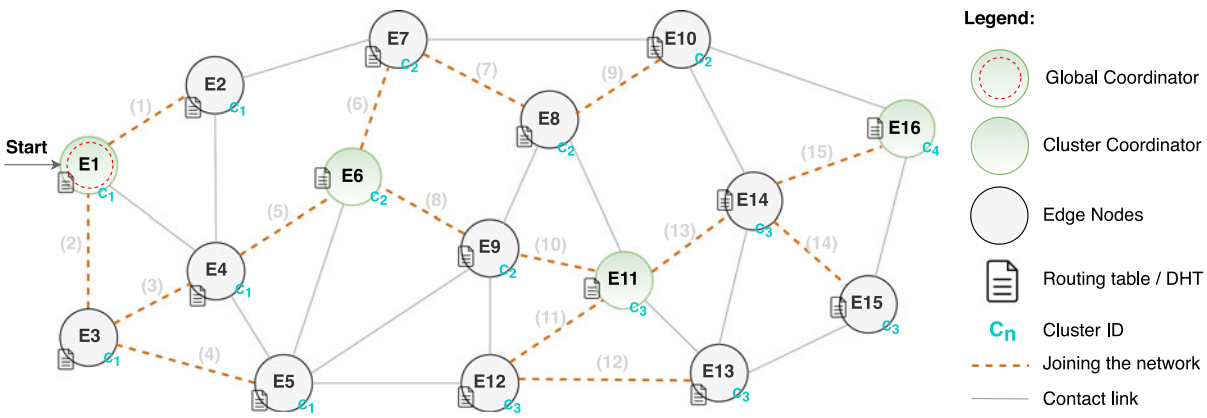

Fig. 2. An example of an edge neighborhood consisting of sixteen devices organized in four clusters.

contexts (i.e., smart homes, drones network, etc.). Notably, they may vary in their sizes and settings; thus, our proposed system is configurable by the system designer. In our conception, edge devices are low-powered, heterogeneous, and resource-constrained computational entities in the system. Within the system context, edge devices may provide multiple functionalities (e.g., act as a client device, server device, and bootstrap device). Furthermore, edge devices are grouped in clusters to promote scaling, reduce bandwidth consumption, and manage the difficulty of discovering resources in an automatic manner in edge neighborhoods. Subsequently, edge devices within the same cluster are considered *neighbor devices*, as well as clusters close to each other are considered as *neighbor clusters*. Each cluster consists of a finite number of edge devices, and each device belongs to one and only one cluster at a time.

In the system context, each edge device may serve specific roles such as *i) cluster coordinator* and *ii) global coordinator*. Such roles are dynamically assigned to the edge devices based on their resource capabilities (discussed in Section 5.2). In Fig. 2, each cluster has a coordinator device (i.e., with a green circle) and a single global coordinator device (i.e., with red and green circles). We assume that cluster coordinators act as *superpeers* [35]. Each cluster coordinator keeps track of the other coordinators and devices within the same cluster (i.e., IP addresses, port). Similarly, edge devices in the same cluster store information for one another and are always aware of their cluster coordinator and global coordinator. A cluster coordinator may have various responsibilities for a subset of devices (i.e., monitoring, discovering resources, etc.). The global coordinator is responsible for monitoring coordinators, exchange resource descriptions between clusters, and orchestrate edge applications in Edge-Cloud infrastructure (i.e., controlling elasticity, migrating tasks, etc.). However, it remains the future work to provide a complete solution for the coordinators introduced in this paper. In this paper, we focus on three main aspects: i) organizing edge devices in the edge neighborhood, ii) determining the most suitable devices to place the global and cluster coordinators, and iii) enabling automatic resource discovery on heterogeneous and dynamic edge neighborhoods.

Each edge device may serve as the entrance door into the edge neighborhood. Essentially, edge devices provide core functionalities to assign newly added edge devices in the available clusters or create new clusters in the edge neighborhood. We introduce three functionalities to identify the maximum number of edge devices in a cluster. First, the system designer may define a system-wide parameter to bound the maximum size of clusters. Second, the system designer may configure cluster coordinators to generate random cluster size (i.e., not higher than the system-wide parameter). And third, the system designer may define a system-wide threshold value specifying the maximum CPU utilization for cluster coordinators. As a result, depending on the functionality enabled, we may have edge neighborhoods with different cluster sizes. The number of clusters in an edge neighborhood is not bounded.

4.2 Communication Protocol

In our proposed architecture, communications between edge devices is realized through implementing the Kademlia Protocol [12]. We have outlined [11] our main reasons to use Kademlia as the communication protocol between edge devices. Kademlia is a distributed hash table (DHT) for decentralized P2P computer networks. Essentially, DHT is a data storage that is kept consistent between all edge devices within the whole edge network. Essentially, when an edge device updates its local DHT, the changes are propagated to all other devices, allowing them to be queried and manipulated again. Likewise, information about current cluster coordinators and the global coordinator is stored in DHT.

Each edge device implements a distributed routing table and stores data in Kademlia buckets ordered by the local device's distance. The routing table size is bounded by $\mathcal{O}(log_2(l/k))$ where l is the number of edge devices in the edge neighborhood, and k is the bucket size. Once a bucket is full, it starts replacing unresponsive devices in favor of incoming devices. The routing table size in our proposed approach is configurable, and it depends on the expected edge neighborhood size. We consider edge devices as resource-constrained computers (i.e., in terms of computational and storage capabilities); thus, edge neighborhoods' size is not expected to be massive.

4.3 Forming an Edge Neighborhood

Fig. 2 shows an illustration of how our solution organizes edge devices in an edge neighborhood. Initially, the edge neighborhood is formed with an edge device `E1`. Since it is

the only device in the edge neighborhood, it is automatically assigned to cluster C1 and determined as the cluster coordinator $C1_{coord}$ and the global coordinator G_{coord}. At the same time, E1 serves as an entrance door into the edge neighborhood. We assume edge devices progressively join the edge neighborhood and each edge device also becomes another bootstrap device En_b (i.e., n is a unique random ID). It is a common approach to keep a list of always-running edge devices to allow new edge devices to join an edge neighborhood. In our case, E1 is the first device contacted by E2 to join the edge neighborhood. The edge device E2 is assigned to cluster C1 by E1 in coordination with $C1_{coord}$. Furthermore, E2 is supplied with DHTs and the complete routing table from E1. The process of adding new edge devices in an edge neighborhood is presented in Algorithm 1.

Algorithm 1. Process of Adding a New Edge Device

```
   Input: E_new
   Output: Adding E_new to the edge neighborhood
 1: C_M = this.MaxClusterSize()
 2: C_S = this.CurrentClusterSize()
 3: C_N = this.OtherClusters()
 4: booleanflag = false
 5: if C_M < C_S then
 6:   En_b.addDevice(E_new, this.cluster())
 7: else
 8:   if C_N ! = null then
 9:     C_clo = this.findMostClosest(C_N)
10:     foreach c ∈ C_clo do
11:       if c.size() < C_N.cluster(c).maxSize() then
12:         En_b.addDevice(E_new, c.cluster())
13:         flag = true
14:         break
15:       end
16:     end
17:   end
18: end
19: if flag = false then
20:   En_b.addDevice(E_new, newCluster())
21: end
22: this.updateRoutingTable()
23: this.updateDHT()
```

The process continues with adding a new edge device E_{new} by contacting En_b. Once E_{new} request to join the neighborhood, bootstrap device En_b initially provides a unique random ID (i.e., 160-bit). Afterward, En_b retrieves information from its cluster coordinator Cn_{coord} where to assign E_{new}. A cluster coordinator Cn_{coord} (i.e., referred as this) maintain information regarding: i) own cluster maximum size C_M (line 1), the current cluster size C_S (line 2), and other available clusters C_N (line 3). C_N provides clusters with available places where E_{new} can be assigned.

The MaxClusterSize() function is configurable and enables the system designer to define the maximum number of edge devices per cluster. Such value can be determined by i) the system-wide parameter (e.g., five edge devices per cluster or random value), implying that when the cluster exceeds the maximum allowed devices, more clusters with their corresponding coordinators should be designated to handle the resource discovering complexity, and ii) the CPU utilization threshold (e.g., CPU utilization set to 35 percent). When the system-wide parameter is set and random cluster size is disabled, edge devices are grouped into clusters of the same size (as illustrated in Fig. 2). Edge devices are grouped into clusters of different sizes when the CPU utilization threshold is set. More specifically, when C_M and C_S are equal, E_{new} is assigned to one of the existing clusters, or a new cluster is created. Nevertheless, both options can be used at the same time. However, the option that is violated decides whether or not the cluster can scale further. Furthermore, if the condition in (line 5) is not violated, E_{new} is assigned to the current cluster of the En_b (line 6).

Neighbor clusters are found through using a system call (i.e., *traceroute command*), which estimates proximity with other cluster coordinators (i.e., using hop count and latency). The function findMostClosest(C_N) uses traceroute command and returns the most suitable clusters that E_{new} can be assigned (line 9). This is especially useful in edge neighborhoods running on different networks. Finally, Cn_{coord} provides information to En_b to assign the E_{new} to the particular cluster if and only if condition in (line 11) is not violated (lines 11-15). Otherwise, when there are no clusters with free places, then E_{new} is assigned to a newly created cluster by Cn_{coord} (line 20). Note that each cluster in the edge neighborhood has a unique ID generated by En_b when the cluster is created. Notably, the bootstrap device's task is to cooperate with the cluster coordinator to assign a cluster ID to the newly joined devices. Finally, once E_{new} joins the edge neighborhood, Cn_{coord} updates the routing table and other DHTs (lines 22-23).

4.4 Resource Modelling

We assume that an edge device may contain multiple resources (i.e., computational, sensing, or context data) represented as microservices [36]. When invoked remotely, such microservices yield resources; however, resource information needs to be shared among edge devices in the edge neighborhood beforehand. An essential step towards discovering these resources in the edge network is resource modeling at design time. To ensure automatic resource discovery in an edge setting, two types of resources must be modeled: *i) IoT resources* (i.e., sensors, actuators, etc.) and *ii) edge devices*.

To accomplish our goal in a pervasive environment, we have outlined resource representation structure in [11]. The resource structure provides seven main properties such as: i) resource identification, ii) resource connectivity, iii) resource capability, iv) resource accessibility, v) resource output, vi) resource location, and vii) resource administrative domain. Our resource structure is similar to the ontology-based structure proposed by Barnaghi *et al.* [37]. Unlike the ontology-based approach, we format resource descriptions in a JavaScript Object Notation (JSON). We advocate that exchanging metadata over JSON is a lightweight process, machine-readable, and provides rich information about resources. Besides that, size of metadata in JSON is very small. Thus, the replication process across many edge devices organized in clusters is feasible and does not degrade the overall network performance.

5 A Decentralized Edge-to-Edge Resource Discovery

In this section, we first discuss the overall design goals for the proposed approach in edge networks. Then, we discuss the process of determining the global coordinator and cluster coordinators. Next we explain in detail the framework for sharing edge device resources based on their privacy preferences. Finally, we discuss edge device failures, dynamicity and uncertainty of edge neighborhoods.

5.1 Design Goals

Edge Computing introduces effective ways to overcome many of the limitations faced by the cloud [3]. Nevertheless, the paradigm alone also presents some limitations (i.e., computation capability and latency). To address such challenges, we identify three design goals that need to be established by any edge-based system:

- *Latency-aware.* The Edge Computing paradigm aims at providing low-latency services for the endpoint devices and the end-users. As a result, determining system coordinators at the edge neighborhood quickly is essential to fulfilling such a stringent requirement.
- *Dynamic Network.* An edge neighborhood changes with time (i.e., new edge devices can be added or excluded). Hence, the edge-based systems should be able to utilize newly added resources at the edge flexibly.
- *Adaptability.* Edge-based systems should be able to adapt to unexpected changes that might occur in the edge neighborhood. Thus, the coordinators should be dynamically placed among available edge devices and continuously re-evaluate placement decisions.

5.2 The Process of Determining Coordinators

The process to determine coordinators in a distributed system should be carried out in system background, encapsulated from the end user's perspective, but indispensable for the correct and efficient execution of distributed tasks. System coordinators' role is versatile and can range between orchestrating applications, monitoring resources, or distributing data between devices. We define two leading roles, such as *i) cluster coordinator* and *ii) global coordinator*. Such roles are dynamically assigned to edge devices based on their resource capabilities. The process to determine new coordinators is triggered by an event when the global or a cluster coordinator experiences high utilization in specific hardware resources (i.e., CPU, memory, or storage) and requests to transfer leadership to other devices. The process to determine a new cluster coordinator occurs only between edge devices within the same cluster. The process to determine a new global coordinator occurs between cluster coordinators. The latter essentially consists of two phases: first, cluster coordinators are determined; second, new cluster coordinators determine the global coordinator. In Algorithm 2, we present the process to determine system coordinators.

The proposed algorithm runs on each edge device separately. The algorithm takes three inputs: i) an edge device hardware metrics denoted with ϕ_i, ii) the deadline to find a solution denoted with θ, and iii) the process type denoted with σ_i. The process to determine coordinators is designed by considering hardware metrics and bandwidth of edge devices. We consider hardware metrics both statically (e.g., CPU cores, storage capacity, etc.) and dynamically (e.g., current CPU load, current storage, etc.). Such hardware information can be monitored using *Hyperic Sigar* [38] while *Assolo* [39] enables collecting bandwidth probes. Notably, the algorithm gets only the current device metrics ν_{this} specified at design time (line 2). In our case, we consider metrics θ_i related to the CPU utilization degree. However, such a parameter is configurable based on system requirements. The deadline θ is given at design time (e.g., $\theta = 50$ ms). The third input σ_i specify the process type: i) determining the global coordinator (σ = `global`) and ii) determining a cluster coordinator (σ = `cluster`).

Algorithm 2. Process of Determining Coordinators

```
   Input: φ_i , θ, σ_i
   Output: C_coord , G_coord
 1: t = 0
 2: ν_this = GetDeviceMetrics(φ_i)
 3: round = Random()
 4: Initial_Message(round, σ_i)
 5: Parameter_Message(ν_this, round, σ_i, γ)
 6: I ← Receive_Parameter_Messages()
 7: Solution_Found ← False
 8: while t < θ or ! Solution_Found do
 9:    ν = I.getBest()
10:    if ν < ν_this and ν.round = round then
11:       Solution_Found ← True
12:       Set(E_this, EdgeMode)
13:       Set(En, Coordinator, σ_i)
14: end
15: if !Solution_Found then
16:    Set(E_this, Coordinator, σ_i)
17: end
18: E_this.updateDHT()
```

A unique random signature (i.e., SHA-1) called `round` is used to make each process unique (line 3). In lines (4-5), we define two types of messages exchanged between devices: *i) initial message* and *ii) parameter message*. First, the initiating coordinator (i.e., process initiator device) probes which edge devices are up and running in the edge neighborhood. Then, it sends an initial message only to the edge devices that responded on time. The initial message essentially contains a list of all participating devices (i.e., only those edge devices that replied) and a unique `round` value assigned to a process round. Second, once receiving an initial message and the list of participants, the process initiator device sends a parameter message containing its local metrics to all participating devices. The parameter message contains local hardware metrics, utilization values in percentage, a process round value, and timestamp when the process to determine the new coordinator is started γ. The timestamp γ is used to ensure that the metrics are not older than the process initiation time. Each edge device responds to the initial message and sends its parameter message to other edge devices, including the initiating coordinator (line 6). Notice

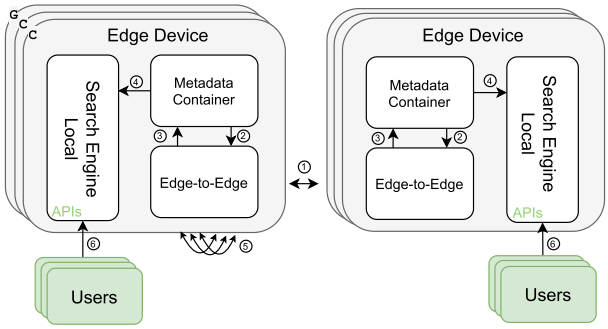

Fig. 3. Resource discovery through metadata replication: High-level architecture.

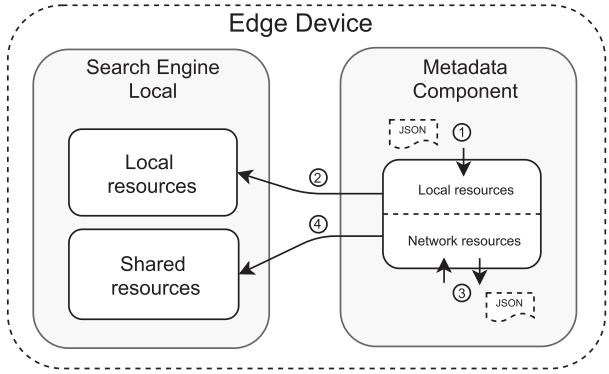

Fig. 4. Processing metadata on the edge.

that the same proposed algorithm is used to determine the global and cluster coordinators.

After exchanging performance metrics, edge devices develop the same result independently. Each device compares received parameter metrics and determines which device is most suitable to serve as a cluster/global coordinator (lines 8-14). To find the most suitable edge device, we compare the number of CPU cores, clock speed, and current utilization (line 9). $\mathtt{E_{this}}$ changes the status to the `Edge-Mode` (i.e., edge device only shares resources) only when an edge device `En` with better performance metrics is found. Line 16 is executed only when a solution is not found within the time θ, and the coordinator role remains in the current edge device $\mathtt{E_{this}}$ since no better edge device is found. Furthermore, each edge device maintains a complete overview of all processes and saves the information in a DHT. The result is propagated to all edge devices within the edge neighborhood by using the DHT. The latter step is executed by all edge devices, which on the one side creates some redundancy but also improves the stability of propagating results (line 18).

5.3 System Architecture and Resource Discovery

We propose a framework for enabling automatic discovery of heterogeneous resources in edge networks [11]. Fig. 3 illustrates three main components of the framework: Edge-to-Edge Communication, the Metadata Container, and the Local Search Engine facility. The Edge-to-Edge Communication component implements communication between edge devices (i.e., Kademlia Protocol), organizes edge devices in clusters, determines coordinators, and exchanges resource descriptions in an edge neighborhood (1). The Metadata Container is responsible for analyzing resource descriptions based on their privacy preference defined by the system designer at design time. Besides that, the component is responsible for sharing resource descriptions system-wide (2), processing received resource descriptions from other edge devices (3), and store resource descriptions locally (4). The Local Search Engine component (4) is based on CouchDB document-oriented NoSQL database [40]. This component subsequently stores data locally and, through APIs, enables users to query stored content (6).

Referring to Fig. 3, our resource discovery mechanism allows edge devices within the same cluster to exchange metadata through their cluster coordinator. Once coordinators are determined, edge devices are ordered to share their public metadata document with their cluster coordinator (1). Other edge devices contact their cluster coordinator to retrieve all public metadata documents contained within their cluster. This may happen once a new edge device is connected to the edge neighborhood or when an edge device wants to refresh its current storage. The frequency to refresh metadata documents is also configurable at design time. Besides that, the global coordinator regularly exchanges metadata documents with the cluster coordinators (5).

In Fig. 4, we present the process to analyze resource descriptions and the process to share them in edge neighborhood. As mentioned before, we consider edge devices as resource-constrained devices with a set of built-in sensors and actuators. Each resource is described by providing certain core information about the functionality and properties (see Section 4.4). We assume that resource descriptions (i.e., metadata) are provided by edge device manufacturers in JSON format. The Metadata Container is responsible for analyzing metadata in the edge device. The component stores resource descriptions based on privacy preference i) public resources (i.e., shared resources (4)) and ii) private resources (i.e., local resources (2)). Public resource descriptions are merged into a single metadata document and named with the edge device ID. The public metadata document is shared with the corresponding cluster coordinator system-wide (3). The local search engine component separates resource descriptions into private ones and those that are shared system-wide.

5.4 Edge Device Failures

Consider a situation when a cluster coordinator triggers a new process to determine the new coordinator in the cluster. An edge device under a high workload may fail to participate in determining cluster coordinators or the global coordinator. Even though not participating in a determination process, edge devices continuously update their local DHTs with the closest devices. Besides that, those who have failed and re-joined the edge neighborhood may also contain some obsolete information (i.e., DHT is not up-to-date, coordinator information is outdated). However, edge devices must update their local DHTs with the closest devices after joining the edge neighborhood.

Another situation may arise when a specific cluster coordinator fails. The global coordinator is responsible for detecting such failure and trigger a new process to determine the coordinator. As discussed previously, edge

devices within the cluster are responsible for determining their new coordinator. When the global coordinator fails, one of the cluster coordinators triggers a new global coordinator determination process. Notably, each cluster coordinator verifies whether the global coordinator has failed and responds to the determination process. Furthermore, the resource discovery process is not repeated when edge devices frequently leave and join the edge neighborhood (i.e., due to the connectivity issues). Moreover, when an edge device went offline, the metadata documents remain stored in the other devices for some time. This is especially useful in unstable edge neighborhoods (e.g., wireless), prone to momentary loss of connection.

6 Evaluation

In this section, we first discuss our evaluation setup environment, prototype details, and limitations. Next, we experimentally evaluate the approach's effectiveness by running multiple experiments and checking the proposed solution's behavior in different situations. We assess our proposed solution in terms of hardware and bandwidth consumption during runtime. Then, through a use case, we show how the proposed discovery mechanism increases the number of eligible deployments when deploying edge applications in an edge neighborhood. Finally, we conclude with a discussion in Section 6.4.

6.1 Setup, Prototype Details, and Limitations

To assess the proposed approach's feasibility, we developed a prototype that implements the Kademlia Protocol and core functionalities to enable the automatic discovery of heterogeneous resources in edge networks. The prototype is written in Java, and it is tested on a testbed composed of (a) edge devices (i.e., Raspberry Pi 3 Model B V1.2) with 4ARM Cortex-A53 CPU at 1.2 GHz, 1 GB of RAM, and 16 GB disk storage, and (b) virtual edge device instances. To evaluate our prototype, we exploited the testbed (i.e., edge neighborhood) composed of 60 edge devices placed close to each other. Edge devices contain multiple resource descriptions generated randomly at design time. Furthermore, edge devices in the testbed are connected through a wireless connection with a nominal speed of 10 Mbps and 5 Mbps in download and upload. We assume that every edge device trusts all other devices, and they all belong to the same local administrative domain.

In distributed systems, discovering devices is a significant challenge. In our current implementation, edge devices have an open designation port that listens for possible future connections. To join the edge neighborhood, edge devices require to know at least one device (i.e., IP, port) currently up and running. We acknowledge that the current implementation represents a limitation, and further investigations are required to develop an advanced approach to discover running edge devices. We acknowledge that IoT resources (e.g., sensors) can also be connected on edge devices using various end-to-end communication protocols (e.g., Zigbee, etc.). However, in this paper, we treat communication and operational aspects of IoT resources as orthogonal to our approach; we assume edge devices are equipped with a set of IoT resources. Furthermore, we acknowledge that using the Java Virtual Machine (JVM) environment is resource-expensive. However, within this paper, we aim to show the approach's feasibility in resource-constrained edge neighborhoods.

TABLE 1
Edge Neighborhood

Neighborhood	Edge Devices (per cluster)	Total
C1	15	15
C2	20	35
C3	15	50
C4	10	60

6.2 Experiments and Results

We evaluate our prototype on a testbed, whose size progressively increases to 60 edge devices as presented in Table 1. The cluster coordinators can determine their cluster sizes based on i) the CPU utilization threshold (i.e., configured to 35 percent), ii) the system-wide parameter (i.e., configured to 30 devices per cluster), and iii) the random value (i.e., not bigger than the system-wide parameter). Furthermore, the routing table size is set to $k = 20$. We monitor edge devices through the *nmon* tool [41] and retrieve information regarding the hardware utilization and the data received and sent between edge devices.

The goal of the first experiment is to assess the prototype's footprint on hardware resources and bandwidth usage. Notably, we focus on resource consumption to determine the system coordinators and discover resources in the edge neighborhood. For each cluster created, we monitor the global coordinator for up to 60 seconds in the edge neighborhood (i.e., running on RPi3). We also monitor bootstrap devices (i.e., RPi3s) when a new request arrives to join the edge neighborhood. Notably, we observe that the time required to join the edge neighborhood and assign it to the existing cluster is between $0.07-0.2$ seconds in all test cases. However, when a new cluster is required to be created, the average latency is slightly higher between $0.5-1.02$ seconds, but reasonable for the resource-constrained edge neighborhood. Specifically, the overall CPU and memory utilization remains almost similar. Concretely, the average CPU consumption is around 2.5 percent, while the overall memory consumption is around 5 MB. In all test cases, edge devices have been successfully assigned to the corresponding clusters, or new clusters are created when required. Table 2 presents a detailed overview of the resource consumption and latency for edge devices to join the neighborhood.

Fig. 5 plots the overall CPU utilization during the process to determine the global coordinator (i.e., the global and cluster coordinators are determined) and synchronization processes running in the background. Essentially, we simulate the situation when the global coordinator is overloaded, and the function to determine the new global coordinator is automatically triggered. We repeat the process more than 10 times for each test case (i.e., clusters). As shown in Fig. 5, the overall CPU utilization is slightly increased when adding more edge devices/clusters in the edge neighborhood. Specifically, the average CPU consumption during the

TABLE 2
Average Latency and Resource Consumption to Join the Edge Neighborhood

Clusters	Latency (avg.)	CPU (avg.)	RAM (avg.)
C1	71.25 ms	2%	2 MB
C2	82 ms	2.5%	3 MB
C3	137.36 ms	2.5%	2 MB
C4	193.6 ms	3%	5 MB

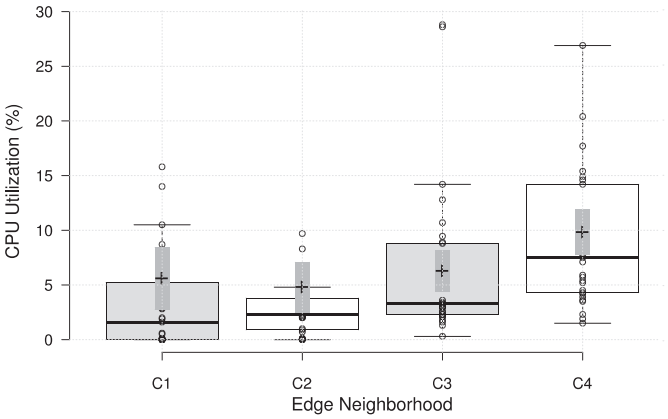

Fig. 5. Analysis of the CPU utilization during the process to determine the system coordinators.

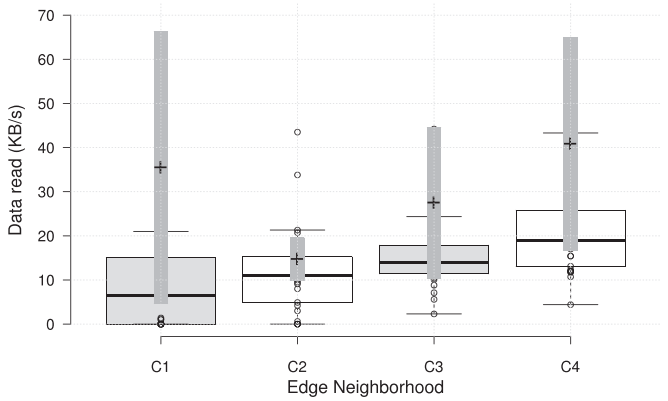

Fig. 6. Analysis of the data received in kilobytes per second by the global coordinator.

process to determine the system coordinators is around 10 percent, while the overall memory consumption is around 5 MB. The most important aspect is the accuracy of assigning edge devices to a particular cluster or forming new clusters when needed (i.e., when the cluster size is exceeded). In all test cases, creating new clusters was successful, including newly joined edge devices added to their adequate clusters. Since the overhead imposed by our approach is small, the experiments show that the proposed approach is feasible to operate on low-powered and battery-powered edge devices. Notably, the proposed approach has shown a very contained impact on hardware resources and bandwidth usage. To obtain consistent results, for each experiment, we calculated an 83 percent confidence interval of means.

Figs. 6 and 7 plot the overall maximum and minimum data transfer/received by the global coordinator during the process to determine new system coordinators and the synchronization process. We consider analyzing the bandwidth

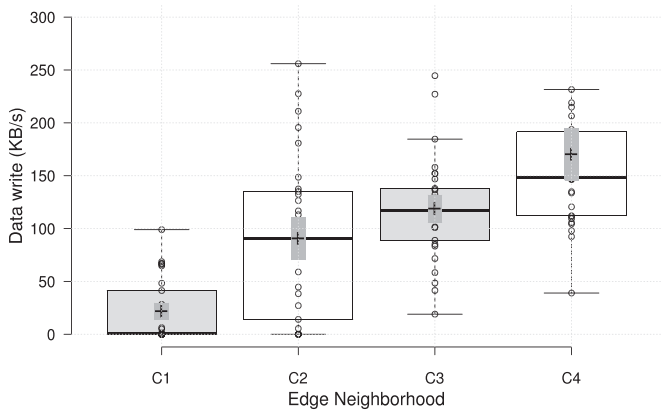

Fig. 7. Analysis of the data transfer in kilobytes per second by the global coordinator.

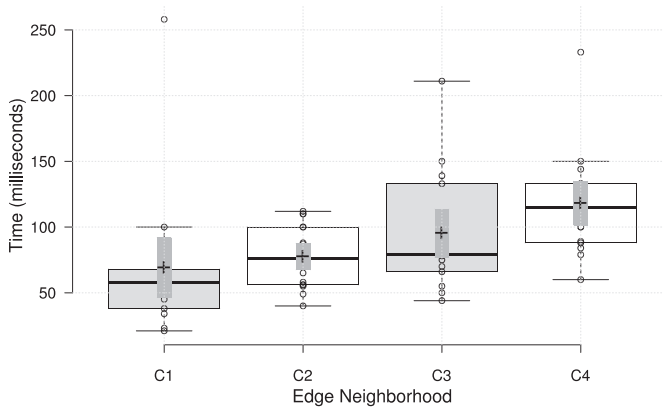

Fig. 8. Time required to determine coordinators.

due to the relationship between network traffic and the load on the memory system of an edge device [15]. Furthermore, in resource-constrained edge neighborhoods, the energy supply and bandwidth are among the main resource constraints of edge devices [42]. Therefore, it is necessary to know the total data size transferred/received (i.e., metadata sharing, processes to determine coordinators, synchronizing processes, etc.) between edge devices during the runtime. Notably, the maximum and minimum values vary depending on the number of edge devices in the neighborhood. As shown in Fig. 7, the data transfer is slightly increased by adding more devices in the edge neighborhood. The slight increase occurs due to the increased number of edge devices that impose the formation of new coordinators. Thus, newly formed coordinators synchronize their routing tables and their DHTs with the global coordinator.

The goal of the second experiment is to assess the time complexity to determine coordinators and place them on the edge neighborhood's most suited edge devices. Concretely, we show how the proposed mechanism discussed in Section IV and Section V performs to determine coordinators. The experiment results illustrated in Fig. 8 show that the proposed approach in all test cases finds an edge device with the most suitable computation resources to assign the global coordinator. Besides that, the proposed approach successfully determines cluster coordinators for each cluster.

In Fig. 9, we analyze the percentage of responsive edge devices during the process to determine the cluster

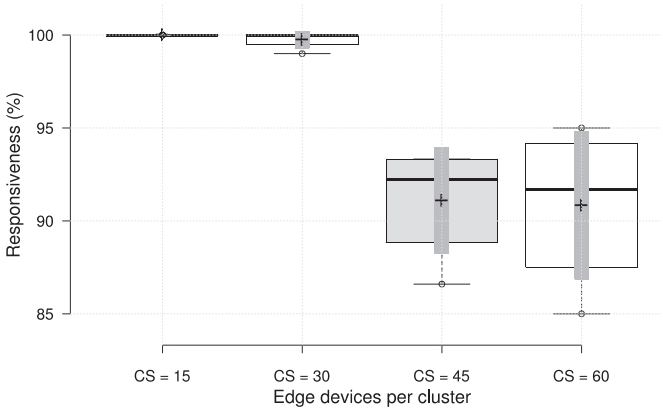

Fig. 9. Percentage of responsive edge devices during the process to determine coordinator on clusters with different sizes.

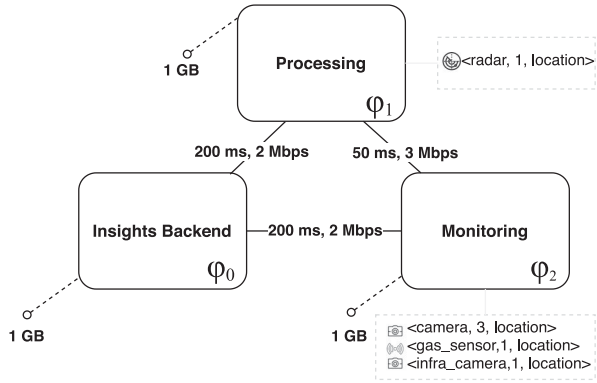

Fig. 10. IoT safety application.

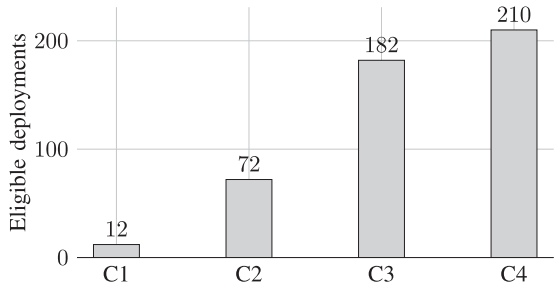

Fig. 11. Generating deployments plans for the IoT safety application in an edge neighborhood (see Fig. 2).

coordinator on an edge neighborhood with a single cluster and different sizes (e.g., CS = 15, etc.). We repeat the process more than five times for each test case. Notably, we observe that the time required to join the edge neighborhood and assign it to the existing cluster remains similar to the previous results (i.e., between $0.07-0.2$ seconds in all test cases). The number of edge devices that respond to this process is critical since it enables us to find the most suitable edge device to place the cluster coordinator. In clusters with more than 40 edge devices, the percentage of responsiveness is decreased to 92 percent. In contrast, clusters up to 30 edge devices show a higher rate of responsiveness. To that end, we configure the system-wide parameter to bound the maximum number of edge devices per cluster (i.e., 30 edge devices).

6.3 Use case: Deploying IoT Safety Application

The third experiment aims at demonstrating the proposed mechanism's function in the application deployment process. For the demonstration purpose, we adopt and extended parts of the FogTorchΠ simulator [43] to generate deployment plans in the edge infrastructure. FogTorchΠ was originally proposed to support IoT designers in making decisions about where to deploy application components at the edge. The approach allows to specify IoT resources as an application requirement that must be met before applications can be deployed.

We assume edge devices containing a set of built-in IoT resources (i.e., cameras, radars, gas sensors, etc.) that can be accessed remotely through APIs. Recalling our motivation scenario, consider the IoT safety application that provides a service to rescue teams (as discussed in Section II). As illustrated in Fig. 10, the IoT safety application comprises three components: i) the insights backend component (φ_0), ii) the monitoring component (φ_2), and iii) the processing component (φ_1). The insights backend component (φ_0) enables users to interact with the service. The monitoring component (φ_2) is responsible for monitoring resources specified at design time. The processing component (φ_1) analyzes collected data into meaningful results and provides it to the end-users through the insight component. We assume deploying IoT safety application in the edge neighborhood as presented in Fig. 2. Notice that only a single application component can be executed in parallel on edge devices. The other application requirements depicted in Fig. 10 are assumed to be met by all edge devices.

In our given edge neighborhood, the edge device E_2 provides the radar resource with privacy preference set to private. This means that the φ_1 can be deployed only in E_2. The rest of IoT resources (e.g., cameras, gas sensors, etc.) are remotely accessible, and their privacy preference is set to public. Through the metadata replication process, edge devices exchange public resources in the edge neighborhood. To this end, each edge device is considered a potential candidate to execute one of the two other components. Meanwhile, each edge device shares private resources only with the global coordinator, responsible for generating deployment plans for the IoT safety application. Fig. 11 shows the total number of eligible deployments plans generated through FogTorchΠ combined with our resource discovery mechanism. It is evident that φ_0 and φ_2 components dependent on particular resources can be executed on all other edge devices. More precisely, each edge device knows how to access resources on other edge devices due to metadata replication.

Unlike our approach, FogTorchΠ does not support resource privacy preferences, does not provide a mechanism to discover resources (i.e., supports resource sharing with neighbor devices), and does not support the dynamic changes in their environments. Besides that, evaluating the deployment mechanism is out of this paper's scope, as we use it to demonstrate that the discovery mechanism allows us to completely exploit available resources in edge neighborhoods.

6.4 Discussion

We have demonstrated that by using our resource discovery mechanism, discovering heterogeneous resources in an

edge setting increases the number of eligible deployments for applications dependent on IoT resources. We showed that organizing edge devices in clusters and discovering resources through the edge replication process is performant and feasible on a testbed with low-powered ARM-based devices. Our results showed that the overhead introduced by the proposed decentralized resource discovery mechanism is very low for realistic edge neighborhood sizes.

Our approach within the given testbed can operate with larger cluster sizes. However, edge devices may not react in time to participate in processes to determine coordinators in large cluster sizes. In such cases, the Kademlia Protocol requires much more processing capabilities to process faster incoming requests (i.e., updating routing tables, updating DHTs, resource metadata, etc.). Nonetheless, we plan to improve and optimize the current mechanism to support forming larger clusters in edge neighborhoods. We plan to investigate the maximum edge neighborhood size and the routing table size acceptable for low-powered edge devices.

Our approach does not provide a mechanism to detect which edge devices fail to send their parameter metrics and ask them to resend their messages. In addition to that, since multiple edge devices in an edge network come to the same result independently, it is necessary to introduce an additional mechanism to verify the result. In our approach, we overcome such an issue by distributing results using the DHT. This creates some redundancy - but also improves the stability of propagating results. However, using consensus algorithms where edge devices pick random participants to verify their final result is highly desirable. Thus, some assumptions underlying our methodology must be further explored.

7 Conclusion and Future Work

Edge networks provide a seamless opportunity for deploying various edge applications providing multiple services to the end-users and the surrounding IoT devices. However, to deploy edge applications dependent on IoT resources, we require novel lightweight and decentralized mechanisms to automatically discover heterogeneous resources at the edge. To that end, we introduced a decentralized mechanism that enables edge devices to connect in a P2P manner, organize edge devices in clusters, and support automatic discovery of heterogeneous resources in edge networks. The proposed approach support resource discovery based on resource privacy preferences. Furthermore, we evaluated our approach in a testbed composed of a set of low-powered edge devices. Throughout the experiments, we showed the feasibility of the proposed approach to run on low-powered edge devices.

We believe that the proposed approach paves the way for utilizing available resources, leading to accomplish the promised high-quality and low-latency services deployed in edge networks (i.e., edge neighborhoods). As future work, we aim at providing a complete technical solution for the global and cluster coordinator; this includes both technical and architectural aspects. In addition to that, we plan to investigate techniques that will enable edge devices to discover nearby devices that allow them to join an edge neighborhood in an automatic manner. Finally, we plan to investigate several assumptions made in this paper.

Acknowledgments

This work was supported in part by the Research Cluster Smart Communities and Technologies (Smart CT), TU Wien, and in part by the EU's Horizon 2020 Research and Innovation Programme under Grant 871525.

References

[1] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, May 2016.
[2] S. Dustdar and I. Murturi, "Towards distributed edge-based systems," in *Proc. IEEE Second Int. Conf. Cogn. Mach. Intell.*, 2020, pp. 1–9.
[3] C. Avasalcai, I. Murturi, and S. Dustdar, "Edge and fog: A survey, use cases, and future challenges," *Fog Computing: Theory and Practice*. Hoboken, NJ, USA: Wiley, 2020.
[4] C. Avasalcai and S. Dustdar, "Latency-aware decentralized resource management for IoT applications," in *Proc. 8th Int. Conf. Internet Things*, 2018, pp. 1–4.
[5] C. Tsigkanos, I. Murturi, and S. Dustdar, "Dependable resource coordination on the edge at runtime," *Proc. IEEE*, vol. 107, no. 8, pp. 1520–1536, Aug. 2019.
[6] V. Karagiannis, "Compute node communication in the fog: Survey and research challenges," in *Proc. Workshop Fog Comput. Internet Things*, pp. 36–40, 2019.
[7] A. Brogi and S. Forti, "QoS-Aware deployment of IoT applications through the fog," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1185–1192, Oct. 2017.
[8] C. Avasalcai, C. Tsigkanos, and S. Dustdar, "Decentralized resource auctioning for latency-sensitive edge computing," in *Proc. IEEE Int. Conf. Edge Comput.*, 2019, pp. 72–76.
[9] K. Toczé and S. N.-Tehrani, "A taxonomy for management and optimization of multiple resources in edge computing," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 7476201:1–7476201:23, 2018.
[10] F. Paganelli and D. Parlanti, "A DHT-based discovery service for the Internet of Things," *J. Comput. Netw. Commun.*, vol. 2012, 2012, Art no. 107041.
[11] I. Murturi, C. Avasalcai, C. Tsigkanos, and S. Dustdar, "Edge-to-edge resource discovery using metadata replication," in *Proc. IEEE 3rd Int. Conf. Fog Edge Comput.*, 2019, pp. 1–6.
[12] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proc. Int. Workshop Peer-to-Peer Syst.*, 2002, pp. 53–65.
[13] S. Dustdar and I. Murturi, *Towards IoT Processes on the Edge*. New York, NY, USA: Springer, 2021, pp. 167–178.
[14] S. Dustdar and I. Murturi, "Towards distributed edge-based systems," in *Proc. IEEE 2nd Int. Conf. Cogn. Mach. Intell.*, 2020, pp. 1–9.
[15] M. Mozaffari, W. Saad, M. Bennis, and M. Debbah, "Communications and control for wireless drone-based antenna array," *IEEE Trans. Commun.*, vol. 67, no. 1, pp. 820–834, Jan. 2019.
[16] Y. Shibata, N. Uchida, and N. Shiratori, "Analysis of and proposal for a disaster information network from experience of the great east japan earthquake," *IEEE Commun. Mag.*, vol. 52, no. 3, pp. 44–50, Mar. 2014.
[17] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Proc. Internet Everything*, 2018, pp. 103–130.
[18] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. First Ed. MCC Workshop Mobile Cloud Comput.*, 2012, pp. 13–16.
[19] A. Lebre, J. Pastor, A. Simonet, and F. Desprez, "Revising openstack to operate fog/edge computing infrastructures," in *Proc. IEEE Int. Conf. Cloud Eng.*, 2017, pp. 138–148.
[20] F. Rizzo, G. Luca Spoto , P. Brizzi, D. Bonino, G. Di Bella , and P. Castrogiovanni, "Beekup: A distributed and safe P2P storage framework for IoE applications," in *Proc. 20th Conf. Innov. Clouds, Internet Netw.*, 2017, pp. 44–51.
[21] S. Ioannidis and P. Marbach, "Absence of evidence as evidence of absence: A simple mechanism for scalable P2P search," in *Proc. IEEE INFOCOM*, 2009, pp. 576–584.
[22] J. Santos, T. Wauters, B. Volckaert, and F. D. Turck, "Towards dynamic fog resource provisioning for smart city applications," in *Proc. 14th Int. Conf. Netw. Serv. Manage.*, 2018, pp. 290–294.

[23] G. Tato, M. Bertier, and C. Tedeschi, "Koala: Towards lazy and locality-aware overlays for decentralized clouds," in *Proc. IEEE 2nd Int. Conf. Fog Edge Comput.*, 2018, pp. 1–10.
[24] J. A. C. Guerrero, D. E. Lucani, and F. H. P. Fitzek, "On network coded distributed storage: How to repair in a fog of unreliable peers," in *Proc. Inter. Symp. Wirel. Commun. Syst.*, 2016, pp. 188–193.
[25] G. Tato, M. Bertier, and C. Tedeschi, "Designing overlay networks for decentralized clouds," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci.*, 2017, pp. 391–396.
[26] Y. Zhao *et al.*, "A new DHT supporting multi-attribute queries for grid information services," in *Proc. IEEE 10th Int. Conf. High Perfom. Comput. Commun., IEEE Int. Conf. Embedded Ubiquitous Comput.*, 2013, pp. 1675–1680.
[27] J. Kim and J. W. Lee, "OpenioT: An open service framework for the Internet of Things," in *Proc. IEEE World Forum Internet Things*, 2014, pp. 89–93.
[28] E. Wang and R. Chow, "What can i do here? IoT service discovery in smart cities," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops*, 2016, pp. 1–6.
[29] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Trans. Comput.*, vol. 65, no. 12, pp. 3702–3712, Dec. 2016.
[30] H. Shi, N. Chen, and R. Deters, "Combining mobile and fog computing: Using CoAP to link mobile device clouds with fog computing," in *Proc. IEEE Int. Conf. Data Sci. Data Intensive Syst.*, 2015, pp. 564–571.
[31] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang, "Cost efficient resource management in fog computing supported medical cyber-physical system," *IEEE Trans. Emerg. Top. Comput.*, vol. 5, no. 1, pp. 108–119, Jan.–Mar. 2017.
[32] M. A. Hassan, M. Xiao, Q. Wei, and S. Chen, "Help your mobile applications with fog computing," in *Proc. 12th Annu. IEEE Int. Conf. Sens., Commun., Netw.-Workshops (SECON Workshops)*, 2015, pp. 1–6.
[33] F. Alkhabbas, I. Murturi, R. Spalazzese, P. Davidsson, and S. Dustda, "A goal-driven approach for deploying self-adaptive IoT systems," in *Proc. IEEE Int. Conf. Softw. Archit.*, 2020, pp. 146–156.
[34] R. Jain and S. Tata, "Cloud to edge: distributed deployment of process-aware IoT applications," in *Proc. IEEE Int. Conf. Edge Comput.*, 2017, pp. 182–189.
[35] G. P. Jesi, A. Montresor, and O. Babaoglu, "Proximity-aware superpeer overlay topologies," in *Proc. IEEE Int. Workshop Self-Managed. Netw., Syst., Serv.*, 2006, pp. 43–57.
[36] A. Bouguettaya *et al.*, "A service computing manifesto: the next 10 years," *Commun. ACM*, vol. 60, no. 4, pp. 64–72, 2017.
[37] S. De, P. Barnaghi, M. Bauer and S. Meissner, "Service modeling for the Internet of Things," in *Proc. Federated Conf. Comput. Sci. Inf. Syst.*, 2011, pp. 949–955.
[38] Hyperic Sigar. Accessed: Jan. 2020. [Online]. Available: https://github.com/hyperic/sigar/wiki/overview
[39] E. Goldoni, G. Rossi and A. Torelli, "Assolo, a new method for available bandwidth estimation," in *Proc. Fourth Int. Conf. Internet Monit. Protection*, 2009, pp. 130–136.
[40] CouchDB. Accessed: Jan. 2020. [Online]. Available: https://www.couchdb.com/
[41] IBM. Nmon. Accessed: Jan. 2020. [Online]. Available: https://developer.ibm.com/technologies/systems/articles/aunmon_analyser/
[42] C. Bae and W. E. Stark, "A tradeoff between energy and bandwidth efficiency in wireless networks," in *Proc. MILCOM IEEE Mil. Commun. Conf.*, 2007, pp. 1–7.
[43] A. Brogi, S. Forti, C. Guerrero, and I. Lera, "How to place your apps in the fog-state of the art and open challenges," 2019, *arXiv:1901.05717*.

Ilir Murturi received the MSc degree in computer engineering from the Faculty of Electrical and Computer Engineering, University of Prishtina, Prishtina, Kosovo. He is currently working toward the PhD degree in edge computing under the supervision of Professor S. Dustdar with the Distributed Systems Group, TU Wien, Vienna, Austria. His current research interests include the Internet of Things, edge computing, crowdsourcing, privacy, and smart cities.

Schahram Dustdar (Fellow, IEEE) is currently a professor of computer science with the Distributed Systems Group, TU Wien, Vienna, Austria. He is currently an elected member of the Academia Europaea, where he is also the chairman of the Informatics Section. He is currently on the editorial board of the *IEEE Internet Computing* and *IEEE Computer Magazine*. He is also the coeditor-in-chief of the *ACM Transactions on Internet of Things* and the editor-in-chief of Computing (Springer). He is currently an associate editor for the *IEEE Transactions on Services Computing*, *IEEE Transactions on Cloud Computing*, *ACM Transactions on the Web*, and *ACM Transactions on Internet Technology*. He was the recipient of the ACM Distinguished Scientist Award in 2009, the IBM Faculty Award in 2012, and the IEEE TCSVC Outstanding Leadership Award for outstanding leadership in services computing in 2018.