2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems

Kinaara: Distributed discovery and allocation of mobile edge resources

Ahmed Salem*, Theodoros Salonidis§, Nirmit Desai§, and Tamer Nadeem¶
* Old Dominion University, Norfolk, VA, USA {asalem}@odu.edu
§ IBM T. J. Watson Research Center, Yorktown Heights, NY, USA {tsaloni, nirmit.desai}@us.ibm.com
¶ Virginia Commonwealth University, Richmond, VA , USA {tnadeem}@vcu.edu

***Abstract*—Edge computing is an increasingly popular paradigm, wherein computation comes closer to the sources of data. A key challenge for edge computing is discovering and utilizing the heterogeneous resources of the vast number of mobile devices at the network edge. Mobile edge devices hide behind private networks, they are mobile and their owners hesitate to share them due to privacy considerations. We propose *Kinaara*[1] a framework for discovering and allocating collective resources for edge computing applications. First, *Kinaara* uses a multi-tier architecture to organize geographically proximal edge devices in clusters and provide their resources via trusted mediator entities. Second, it uses a novel resource representation to name and encode devices in terms of the resources advertised by their users. Third, it uses a novel distributed indexing scheme to organize devices in a proximal cluster on a ring logical structure based on their resource similarity. Our extensive experimental evaluation on a large Wi-Fi mobility dataset of 150K users and 345 APs show that compared to existing approaches *Kinaara* reduces resource discovery overhead by up to 70% and is robust to device mobility.**

I. Introduction

Edge computing is a new paradigm that addresses the explosive mobile data growth by pushing computation at the network edge close to the data sources. Compared to cloud computing which transports all mobile data to cloud servers for processing, edge computing reduces delays, network bandwidth and mobile device energy consumption. Most existing edge computing approaches use compute resources in the fixed edge network infrastructure instead of mobile devices. On the edge, however, the network infrastructure may not be able to keep up with the explosive edge data growth and real time application requirements [1], [2].

The collective compute, storage, connectivity, and sensing resources of network edge mobile devices such as smartphones, wearables, and vehicles, offer a new opportunity to extend the boundaries of the network edge closer to the data sources, not only for computation but also for other capabilities such as sensing, storage and connectivity. The IoT phenomenon amplifies the untapped potential that fuels rapid proliferation of edge devices in multiple forms.

We envision that in the future, edge computing architectures will enable seamless usage of collective edge resources and create ecosystems where device users provide their resources for a potential compensation in return. One early instance of this vision is CrowdSensing applications which focus on deployment of sensing tasks within single device boundaries. Complex applications beyond sensing, and leveraging a collective of heterogeneous resources and devices is a natural evolution step. As a motivating example, consider the possibility of being able to see in real-time, expected wait times, noise levels, and lighting conditions at a restaurant, without deploying any dedicated devices or infrastructure. Participating devices of the patrons at the restaurant can sense noise levels via their microphones, estimate wait times by discovering the presence of other smartphones in the proximal area, and report on lighting conditions based on a light sensor available on some of the devices there. Other examples include searching for a criminal or a lost child in an amber alert area via computer vision on a collective set of devices, reconstructing an auto accident scene by analyzing the dash-cam videos of the cars at the spot at the time of accident, or reporting timely news in an emergency situation through cameras of people on-site. A wide range of similar possibilities exist for public safety, retail shopping experience, entertainment, augmented reality and healthcare at home, among others.

Despite its potential, harnessing the collective resources of edge devices entails multiple challenges. First, the billions of edge devices owned by millions of individuals and enterprises are hidden behind firewalls, making it hard to discover their resources. Second, most of these devices are mobile, resulting in dynamic changes to the state of discovered resources. Third, security and privacy concerns of sending their data to the cloud often deprive device owners of the opportunity to share and potentially monetize their resources.

We present *Kinaara* a framework for discovering and allocating collective compute and sensing resources for edge computing applications. *Kinaara* has three fundamental design principles. First, it uses a multi-tier architecture to organize geographically proximal edge devices in clusters and provide their collective resources via trusted mediator entities. Second, it uses a novel resource representation to name and encode devices in terms of their user advertised resources. Third, it uses a novel distributed indexing scheme to organize devices in a proximal cluster based on their resource similarity.

Kinaara possesses multiple distinguishing features. First, *Kinaara* enables scalable and fast discovery of collective edge resources without revealing the identities of individual edge devices or exposing them to a public network.

Second, *Kinaara* enables a rich set of resource discovery primitives. *Kinaara* queries can identify multiple devices that satisfy a wide range of requirements particular to edge applications, including exact, aggregate and range query matches, multiple and heterogeneous resources and devices, and co-location constraints of resources on devices.

Third, the *Kinaara* indexing scheme's design, a ring logical

[1] *Kinaara* means edge in Hindi

2155-6814/17 $31.00 © 2017 IEEE
DOI 10.1109/MASS.2017.10

structure, effectively handles device mobility and changes in resource status, by encoding multiple resources and their features in a single key and by employing efficient basic ring operations (join/leave/lookup) that incur minimum overhead.

Fourth, *Kinaara* empowers edge application developers to express application resource requirements that go beyond single device boundaries. For example, in a video analytics application, the developer can express requirements such as number of cameras and their resolutions for video acquisition and collective CPU and memory requirements (which may exceed single device capabilities) required for timely video processing. *Kinaara* also enables application users to specify application context such as location and execution duration. Based on user-provided context and application requirements, the *Kinaara* resource discovery and allocation mechanism can automatically identify a set of matching devices and allocate resources for deploying the application components to these devices. During execution, the mediators act as aggregation points that disseminate results from edge devices to users.

We evaluate *Kinaara* using extensive simulations and mobility traces from a large public WiFi Internet dataset with 150K users and 345 Wi-Fi Access Points (APs). The evaluations include *Kinaara* performance under different system parameters and mobility dynamics, and comparison with alternative resource discovery approaches. Our evaluation shows that *Kinaara* reduces discovery overhead by 70% and 40% compared to Chord[3] and a Centralized scheme and that performance degrades gracefully under high mobility.

II. Related Work

The vision of Edge Computing has been initially articulated through Cloudlets [4] and fog computing [5], which focus on the compute resources of the edge network infrastructure instead of mobile devices. Recent work has proposed using computation of clusters of mobile devices [6], [7]. Resource discovery has not been the focus of the Edge Computing work. Most work has focused on problems such as migration [4], decreasing data communication overhead [8], computation offloading [9], scheduling [6], or fault tolerance [7].

Grid Computing systems addressed distributed resource discovery and typically employed P2P overlays[10]. *P2P* systems designed mainly for file sharing applications also considered file discovery, which maps to resources in our case [3], [11]. The key space of such approaches is typically generated with hash functions and therefore devices are not grouped based on resource similarity. In addition, P2P discovery approaches targets exact match queries and, as we will show in the evaluation section are not efficient for searching collective and heterogeneous resources.

Resource discovery has been recently addressed in the context of Internet of Things (IoT), which is a form of edge computing. Recent IoT standards, i.e., AllJoyn [12] and IoTivity [13], offer name based discovery using IP multicasting, where a device either advertise its presence or identify others. These approaches do not support a notions of single or collective resource discovery as *Kinaara*, however, their open source is programmable. These approaches also operate under a single WLAN scenarios and scaling them to larger networks is challenging. Moreover, they require enhancements to execute low latency applications. The work in [14] proposed a global resource registry that provides open APIs for search and resource matching. This is a centralized approach and has scalability issues. Furthermore, it targets returning the addresses of individual devices instead of using a mediator that exposes just their resources. This work offers a visionary approach without any evaluation metric. A recent work proposed a hierarchical architecture where distributed gateways index devices connected to them and report them to the cloud [15]. This approach reaches to the gateway in a distributed fashion, while the gateway indexes the resources connected to it in a centralized manner. This approach introduces single point of failure. It also supports simple exact match queries based on resource type and location and does not support complex queries for collective and heterogeneous device sensing and computation resources like *Kinaara*. DRAGON uses a tree structures to index resources and support range queries [16]. The aims at indexing each resource separately to guarantee efficient tree search. Maintaining this large tree is not robust to mobility and requires efficient aggregation between multiple devices, which consume the offered by edge resources.

CrowdSensing is similar to *Kinaara*'s envisioned applications. It leverages sensing capabilities of edge devices for multiple applications. Examples include reducing power consumption from redundant sensing [17] and I/O virtualization [18]. Discovery is either not addressed or achieved through a third party solutions, e.g., Amazon Mechanical Turk (AMK), which does not expose the resources per device [19]. Also, they require human in the loop, which is different than the applications envisioned by *Kinaara* where complex computation and sensing applications with real time requirements running on the edge and there is collective resource usage that may exceed the device boundaries.

In summary, although the above approaches were efficient in their domain, using them on *Kinaara* edge computing applications would result in communication overhead and raise security concerns of exposing devices to public networks. In this work, we introduce mediators to eliminate redundant communication and hide the identity of collaborating devices. In addition, previous work does not efficiently support scalable and efficient resource discovery of collective and heterogeneous resources with dynamic values and device mobility.

III. *Kinaara*

Kinaara follows the edge computing paradigm in utilizing compute resources of network components at the edge [20], [21], [22]. *Kinaara*, however, goes one step further by harnessing the collaborative compute and sensing capabilities of user devices in addition to compute capabilities of edge network infrastructure devices. Unlike offloading techniques, *Kinaara*'s design supports applications that run on the edge using edge resources. In this section, we describe the *Kinaara* three-layer

architecture, show how *Kinaara* handles device mobility, and provide an overview of the resource discovery mechanisms.

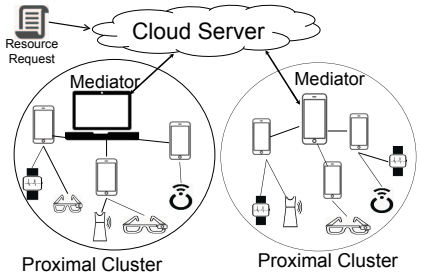


Fig. 1: *Kinaara*'s Edge Computing Architecture

A. Architecture

Kinaara consists of a three-layer architecture, consisting of a cloud server, mediators and edge devices. *Kinaara* users interact with *Kinaara* using a mobile App to authenticate, grant access to their device resources at their convenient time slots, and issue resource requests.

Edge devices and proximal clusters. At the lowest layer are *edge devices* that opt-in to *Kinaara* to share their compute and sensing resources. Each *Kinaara* edge device may advertise local or non-IP connected resources. For example, a smartphone may advertise a gyroscope either as an on-board sensor or a wearable sensor connected to it via Bluetooth. We say that devices sharing a certain context and a degree of trust among them form a *proximal cluster*, where devices communicate with each other either in a peer-to-peer fashion or via a gateway. For example, devices in a retail store, a restaurant, or a home would form proximal clusters.

Kinaara supports multiple flavors of proximal clusters. The simplest proximal cluster can be a single wireless LAN where devices communicate using wireless AP(s), Wi-Fi Direct or Bluetooth. This is common for IoT scenarios and is currently being standardized, e.g., AllJoyn [12]. Further, *Kinaara* envisions proximal clusters wherein devices across multiple locations need to collaborate. For example, video surveillance camera over a larger area may collaborate to infer that a theft is taking place, although such an inference is not possible based on any single video stream.

Mediators are proximal cluster managers playing a key role in orchestrating the cluster resources without the need for a cloud. Mediators have the following responsibilities: (a) establish a communication channel to the cloud service as well as other mediators, (b) aggregate and expose the cluster resources to cloud and other mediators while protecting the identities of edge devices and their owners, (c) discover and register new devices and their resources in the cluster as well as detect their departure, and (d) respond to requests for resources. A mediator is a logical entity, it can be physically hosted on any of the edge devices, e.g., laptop, AP, WLAN controller, or smartphone. Although a mediator can sit in an AP, its area of operation, i.e., cluster, spans beyond the AP limits. For example, in an enterprise, e.g., mall, covered by multiple APs a single mediator brings together all resources using inter-AP communication, e.g., AllJoyn [12]. Further mediator to mediator interactions offer a wider variety of resources for edge applications.

If a mediator device leaves the cluster, a decentralized consensus algorithm elects a new mediator. Since mediator election is expensive, the election algorithm takes into account the expected mobility of a candidate mediator and elects a device having the least expected mobility. Since such algorithms are well-known, this paper does not provide details beyond using a k-leader election algorithm [23].

Cloud server is a repository for sensing and computations tasks (applications). It deploys them on proximal clusters and acts as a communication channel between proximal clusters. Also, it receives user requests for executing applications with known resource requirements. During resource discovery, it probes mediators for resources in their clusters. After resource discovery, it notifies the mediators to reserve resources and send them the application code for execution.

Users. *Kinaara* engages three kinds of users: (a) device owners, (b) application users, and (c) application developers. Application developers submit resource requests to the cloud service, allowing them to develop and deploy applications without having direct access to the sensing and computing resources. The device owners opt-in to *Kinaara* by registering their devices and specifying expected incentives. Smart contracts and blockchain [24] can be readily applied for this purpose. Users acquire the application from the cloud server.

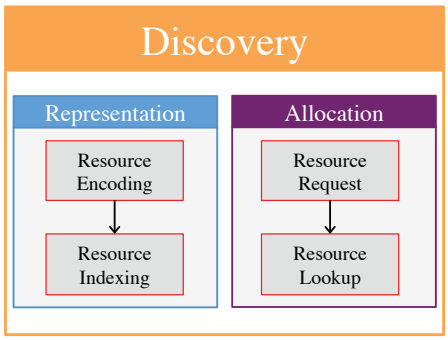


Fig. 2: *Kinaara* Phases of Resource Discovery

B. Mobility Management

Since edge devices are mobile, it is common for them to move across proximal clusters, e.g., a device owner leaving the home cluster to join the work cluster. Mediators detect device departures using heartbeat and network timeout techniques. Similarly, the device itself detects loss of connection to a cluster and starts discovering other clusters via standard service discovery protocols [25]. If a device leaves the cluster while executing an application, the *mediator* assigns the same application component to a different edge device. This transition may cause loss of data and computation that *Kinaara* handles in a variety of ways depending on the application requirements. Mediator notifies the application of such events and the application handles such losses. For example, loss of an

application component processing a video stream requires no special handling given that the missed video frames.

C. Resource Discovery

In the remainder of the paper, we will focus on resource discovery and its modules shown in Fig. 2.

Kinaara discovers resources at the network's edge based on their availability and readiness to execute an application. As shown in Fig. 2, the discovery process has two phases: Resource representation and resource allocation. *Resource Representation* (§ IV) names and indexes devices within a proximal cluster; *Resource Allocation* (§ V), analyzes resource requirements per incoming request and discovers devices matching these requirements.

Throughout both phases, *Kinaara* uses the following components: a *key* to represent resources and their attributes for each edge device; a *Similarity Table* wherein each device holds pointers to other devices with similar resources; and a *Similarity Function* used to identify how similar two or more keys are to each other, e.g., via Hamming distance.

IV. Resource Representation

Devices consist of a set of computing, memory, and sensing resources. Each characterized by a set of features. For example, (a) a camera resource has frame rate and resolution features, (b) an accelerometer resource has sampling frequency and accuracy features. A key challenge in representing such resources is the sheer heterogeneity of the types of device resources and features. Another challenge is in enabling scalable search for suitable resources across all proximal clusters. In this section, we describe the novel techniques of resource encoding and resource indexing that enable *Kinaara*.

A. Resource Encoding

The main novelty of *Kinaara* resource encoding scheme is to enable each *mediator* to encode a small subset of all resource types and features. The cloud service maintains a dictionary of all resource types and features recognized by *Kinaara*. The dictionary maps each resource type, feature types, and feature values to a unique binary code, e.g., $\{camera \rightarrow 1011000110011111\}$. When a *mediator* opts-in to *Kinaara*, the cloud service shares the current dictionary. Each *mediator* chooses the maximum number of resource types it encodes, for scalability, and comes up with its own local dictionary. For example, if a *mediator* chooses to support a maximum of 8 resource types, it allocates three bits for the mapping of first 8 resources it encounters in the proximal cluster. Thus, the domain of the local dictionary is a much smaller subset of the global dictionary, which is dynamically constructed as *Kinaara* discovers edge devices. For example, assume the first resource witnessed by the *mediator* is a camera, the local dictionary would be $\{camera \rightarrow 000\}$. When the *mediator* communicates its aggregated resources to cloud service or other *mediators*, it maps the local codes to the global ones. In case of expanding the cluster supported resources, mediator informs all devices to modify their keys by padding 0s to the MSB per resource. The padded 0s double the resource bits to minimize the repetition of such low performance operation.

Similar to the resource encoding described above, each feature associated with a resource type is also encoded via a dictionary mapping. For simplicity, without loss of generality, we assume two feature types: *binary* and *range*. *Binary* features indicate whether or not the corresponding feature is available on the resource or not, e.g., camera flash. *Range* features discretize a domain of values to a small number of categories wherein the domain itself can be discrete or continuous. For example, discretize a range of camera resolution features into "high", "medium", or "low".

The codes for a resource and its associated features are combined into a *chunk* via their concatenation in a predefined order. Similarly, chunks corresponding to all resources offered by a device in the ascending order of resource codes are combined to form a *key*. Note that *Kinaara* does not require the *key* to be unique, i.e., devices with identical resources will have identical *keys*. For example, device A with key "**001**0111**011**11" consists of two resources camera (code="001") and accelerometer (code="011") shown in bold. The camera supports a resolution of 8 Mpixel images (code="0111") and frame-rate 10 fps (code="1"), while the accelerometer reads with frequency 10 Hz (code="11").

Finally, resource similarity plays a major role in their indexing strategy, as described later. *Kinaara* employs hamming distance between *keys* estimate similarity of the resources offered by two devices. If the *keys* correspond to devices with no common resources, their distance is deemed infinite.

B. Resource Indexing

Kinaara indexes resources in a distributed fashion to avoid maintaining replicas for centralized indexing and cope with *mediator's* mobility. In this component, we explain how devices join and leave *Kinaara* using their keys. *Kinaara* creates a link between resource similar devices through a structure we call *ring* to enable easy resource lookup. For example, a request for a camera from a device whose camera is currently busy is forwarded to another device with a similar camera instead of denying the request. In this work, similarity expresses how close resource features are between devices and is measured by the sum of Hamming distances between keys.

In *Kinaara*, resources are dynamic in *Magnitude* and *Availability*. *Magnitude* represents the number of resources that an edge device selects to share at any time. *Availability* shows whether a resource is currently busy. Modifying the magnitude requires new key generation, then joining the ring as a new device. *Kinaara* detects availability during resource discovery. Our experiments showed that representing the magnitude only in the key yields better performance. *Kinaara*'s indexes clusters through a ring supporting device join and leave operations.

1) Kinaara's Ring: Inside a proximal cluster, *Kinaara* sorts devices on a logical ring by their resource similarity. In order to maintain the ring structure each device carries a direct link to its successor and predecessor. Moreover, each device carries a data structure called *Similarity Table*, whose entries

are the most resource similar devices on the ring. This structure creates a link between similar resources on the ring enhancing the lookup operation. To mitigate an oversized *similarity table*, *mediators* limit similarity entries to logN of cluster devices.

During the initiation of the ring, *Kinaara* will have similarity gaps, where non-similar devices are placed next to each other on the ring. The reason is that initially, devices with different resources may be joining the ring. Our assumption is that the number of device will be much larger than the number of distinct resources within each device. After multiple devices join, the distance gaps between devices will be smoothed out as more devices with similar resources join.

```
 1: Function deviceJoin (Node n, Key k, Node nearestNode){
 2:   IF (simFunction (n.key, k) <= 1)
 3:     appendNode(n,k)
 4:   else
 5:     IF( isEmpty(n.successor) )
 6:       appendNode(nearestNode, k)
 7:     else
 8:       nearestNode = nearestNode.closer(n)
 9:       deviceJoin(n.successor, k, nearestNode)
10:     end IF
11:   end IF
12: end Function
13:
14: Function appendNode(Node n, Key k)
15:     Node newN = new Node(k)
16:     newN.successor = n.successor
17:     newN.predecessor = n
18:     n.successor = newN
19:     (newN.successor).predecessor = newN
20:       newN.similarityTable.add("0000")
21:     Loop ( count(newN.simTable) < length(k))
22:         newN.simTable.add(newN.successor.simTable)
23:         IF ( count(newN.simTable) < length(k) )
24:           newN.simTable.add(newN.predecessor.simTable)
25:         end IF
26:     end Loop
27:     announcePresence(newNode)
28: end Function
29:
30: Function announcePresence(Node n)
31:     Loop i = 1:length(n.similarityTable)
32:       Node p = n.similarityTable(i)
33:       IF (simFunction(p.simEntriess, n.key) < simFunction (p.key, n.key))
34:         p.similarityEntry.updateSimTable(n.key)
35:       ELSE
36:         p.updateSimilarityTable(n.key)
37:       END IF
38:     END LOOP
39: END Function
```

Fig. 3: *Kinaara* Device Join Algorithm

2) Device Join: An edge device joining the ring sends the *mediator* resources he is offering with time slots. The mediator generates a *key* and forwards the join request to the closest device in its *similarity table*. This is repeated till finding the closest device, placing the new device after, creating links to predecessor and successor, and building the *similarity table*.

Throughout our search for the right location on the ring, *Kinaara* maintains a variable called "*nearestNode*", that is updated every hop between two edge devices if the current device is more similar than the previous. When the similarity function could not find closer entries, we use *nearestNode* as the new location. In case of duplicate keys, we place them as successors and the lookup could fetch their resources.

Fig. 3 depicts the details of device join algorithm. After *Kinaara* finds the location for the new device on the ring (lines 2-9), it creates a two-way pointer with successor and predecessor (lines 15-19). Then builds its *Similarity table*, the first entry is always the mediator to maintain a direct link, i.e., "0000", (line 20). Then device's spatial locality in terms of resources enables borrowing the similarity table entries from the successor to save the effort looking through the whole cluster. *Kinaara* applies Hamming distance similarity check on the borrowed table entries, the new device selects candidates for its similarity entries by iterating between predecessors and successors repeatedly till filling its table entries (lines 21-26).

Moreover, the new device announces its presence to cluster devices (line 27) by reaching out to its similarity entries and their entries and contact eligible devices to include the new device in their table (lines 33-37).

3) Device Leave: Detected by successor and predecessor once their direct link with the device drops. They establish a link together to mend the broken ring using a knowledge every device keeps, i.e., successor of successor and predecessor of predecessor. Since the device who left is still in other similarity tables, we remove these links individually whenever any results in a communication failure. We chose this approach over broadcasting a device leaving event to eliminate the overhead imposed by reaching out to the whole cluster.

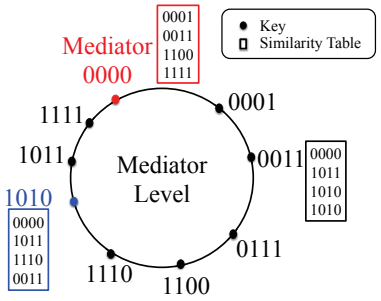


Fig. 4: *Kinaara* Sample Ring

C. Indexing Scenario

Fig. 4 shows a sample ring in *Kinaara*. For simplicity each device holds a single resource represented by 4 bits chunk (resource and features), this can be extrapolated on the application's need. When new device "1010" joins, the *mediator* searches his similarity table for the most similar device and forwards its control to its similarity table, i.e., "0011", which has a closer device, i.e., "1011". Hence, the new device request is forwarded to new ring location and after checking its *similarity table* and the two following hops, it finds his location prior to 1011. Therefore, the new device is placed on the ring and readjust pointers of "1110" and "1011" such that it is the successor of the former and predecessor of the latter. Meanwhile, it points to "1110", and "1011" as predecessor and successors respectively. Building a 4 entry similarity table, mediator comes first and the rest is borrowed from neighbors. Now, the ring is ready to for resource requests.

D. Mediator-to-Mediator Interaction

In an effort to incorporate the largest possible resources per application, *Kinaara* employs direct mediator interaction. This enables applications to fetch resources from surrounding mediators either because of resource shortage or expanding the geographical area of operation.

Similar to devices in *Kinaara*, mediators follow the same procedures in establishing a ring, except for key generation and similarity table. In a mediator ring, *Kinaara* uses only available resources in the key generation excluding features. Our experiments showed mobility to cause high feature variation compared to resources, hence, ignored to generate a mediator key that is less susceptible to changes. In the mediators ring, similarity tables holds two extra columns, *location* and *LRU*. The location holds geographical location (i.e., latitude, longitude and elevation) enabling applications to expand their operations in specific locations. The LRU field holds the latest time of calling the corresponding mediator. Eliminating features from mediator keys generate higher redundancy, hence, LRU ensures fair distribution of resources on applications.

The mediator's life cycle on the ring has three phases: join, modify, and leave. After establishing the device ring, a mediator can generate its key and use it to *join* in the mediators ring and create the corresponding similarity table. In case of a new resource showing up at a mediator, a new key is the generated and another round of mediator advertisement takes place to *modify* the previous key. When a mediator *leaves* its cluster, we follow the same lazy approach as devices to discover the failure through failed communication (§ IV-B3). Then, a new mediator is elected and joins the mediators ring.

V. Resource Allocation

On an established ring, *Kinaara* is ready to receive resource requests. To initiate resource allocation, *Kinaara* goes through two steps *resource request*, and *lookup*. On *mediators*, the former generates a request out of the application's required resources, while the latter performs the resource lookup.

TABLE I: Search Query Prefix

Feature	Value
Count	4 bits, 4 devices per increment
Priority	1 bits showing optional or mandatory
Diversity	1 bit, "1" = resources on a single device

A. Resource Request

Mediators intercept incoming requests to create a resource query for discovery. The request is an application that includes a manifest file with detailed resource and feature requirements, like android app's structure, which *Kinaara* uses to generate a *search query* with the same format as *keys* (§IV-A), except for a prefix. Table I shows features of the prefix: *Count*, required amount per resource; *Priority*, whether the resource is mandatory or optional; and *Diversity*, says which resources must be discovered on the same device.

Range Queries targets a range of features per resource. *Kinaara* codes its features in small ranges, if the query range is wider than the feature ranges, it is split among them generating multiple requests corresponding to the request's feature ranges.

```
 1: function lookup(Node n, SearchKey s, Path p)
 2:   loop r = s.resources [1 to m]
 3:     if (n.checkAvailability(r))
 4:       s.remove(r)
 5:       n.remove(r)
 6:       p.add(n,r)
 7:       r.hold(t)
 8:     end if
 9:   end loop
10:   if (not empty (s.resouces))
11:     lookup (n.successor, s, p)
12:   else
13:     return p
14:   end if
15: end lookup
```

Fig. 5: *Kinaara* Lookup Algorithm

B. Resource Lookup

Fig. 5 depicts *Kinaara*'s resource lookup algorithm. The *mediators* initiate lookup comparing chunks from the search query against its similarity entries. Then, hop to the most resource similar device (line 3). On each hop, *Kinaara* checks the resource availability and searches the similarity entries for similar devices to visit (lines 4-7). Available resources are held for a *Holding Period*, waiting for an application to execute, otherwise they are free for other resource requests.

In case of finding all or subset of the request on a device, the following operations are preformed: *First*, the resources found by the request are removed by eliminating their portion of the search query; *Second*, the selected resources are temporarily marked as *on-hold* until the mediator decides on whether to recruit or not. During the holding time, these resources are viewed as busy by other resource requests. If these resources are not utilized by the mediator, they become available after a *Holding period* timeout. *Finally*, this procedure is repeated with the upcoming similarities until forming a path of devices to fulfill the resource request (lines 2-9). The lookup algorithm has two outputs: *Path Length*, the number of devices visited to match the resource request; and *Discovered Devices*, whose resources are *on-hold* waiting for recruitment.

In designing the lookup operation, we had to consider two aspects, the similarity function and lookup style. *Similarity function* compares search strings with device keys as shown in the lookup algorithm Fig. 5, we compared multiple token comparison algorithms and chose Hamming distance for consistency and low latency. *Lookup style* defines how to select the next hop between proximal cluster devices. It can be either looking through the current node's similarity entries one by one or the closest entry in each visited node till establishing the path. The former is a Breadth First Search (BFS), while the latter is Depth First Search (DFS). As the similarity between devices and number of hops are inversely proportional, *Kinaara* favors the BFS approach, as it utilizes the spatial locality created on the *similarity table*.

C. Mediator Allocation

In case an application needs borrowing resources from other mediators, the current mediator searches its mediator similarity

table for the corresponding resources and selects a target. The target mediator receives a request similar cloud server's resource request, except the source is another mediator. Mediators comply to such requests as explained earlier and reports their resources back to the requesting mediator. The requesting mediator appends the returned results to the list of *Discovered Devices* to be used by the edge application.

VI. Implementation and Evaluation

In this section, we evaluate *Kinaara* through simulation using two kinds of experiments: (a) *Kinaara* against similar approaches to validate our design choices, e.g., efficient lookup; (b) *Kinaara*'s performance under challenging conditions, e.g., different resource availability, variable resource request rate per minute, and intensive mobility patterns.

In our experiments, we measured two metrics: *Path Length* and *Discovered devices* previously explained in § V-B. We used the following set of parameters: *Cluster Size*, number of resources (8per device) in a proximal cluster; *Request Success*, discovered resources as a percentage of the requested ones; *Request Rate (RR)*, number of received resource requests per minute; *Holding Time (HT)*, waiting period for application code, if passed resources are available for discovery; and *Mobility Rate*, resources joining/leaving a cluster per minute.

In our simulations, we used 16 different types of resources. To capture device heterogeneity, each device advertises 8 randomly selected resource types. We assembled resource queries by choosing a random sample from our resource types and selecting their prefix. We considered any device a successful match if at least one of its resources matches a search query resource. Unless otherwise mentioned, each experiment simulated 1-day per cluster size with *request rate*=1 per minute and *holding time*=2 minutes. Also, each plot is an average of 100 runs. Our key findings include:

- *Kinaara*'s distributed design results in ≈40% smaller path length compared to a *Centralized* scheme.
- *Kinaara* results in a significantly smaller *path length*, i.e., 70%, compared to Chord.
- Mobility handling in *Kinaara*' imposes low overhead on discovery within clusters up to 3000 resources.

A. Experimental Setup

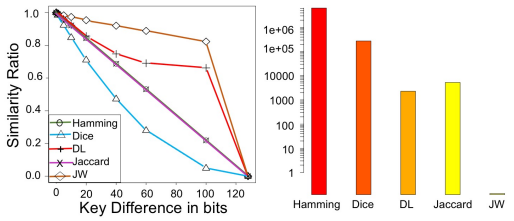
1) Kinaara Simulator: We implemented a Java-based platform of 27 classes (2500LOC) to simulate *Kinaara* modules, underlying network, and the experimental setup. We used an i7 2.2GHz quad core, 8GB RAM machine running OSX 10.9.

2) Dataset: We used WiFiDog, an extensive 6-year dataset of user mobility traces from public WiFi Access Points (APs) [26]. It contains 2,177,835 records of data with 149,861 users moving between 345 WiFi APs. An average of 5732 users visited each AP, each spending an average of 78min/AP. Also, on average each user visited 14 different APs. In our simulations, we assume the WiFiDog users are *Kinaara* edge devices and the mediators are deployed at the WiFi APs.

3) Reference Approaches: We compared *Kinaara* to two alternative approaches *Centralized* and Chord [3]. In *Centralized*, *mediators* maintain a dataset of all edge devices in a cluster, on a single table, without using the ring structure. *Mediator* can lookup his dataset to locate devices, but must query them sequentially for availability. Assuming *mediator's* up-to-date knowledge of resource availability would fit the purpose of discovery, but impractical in predicting the application's execution time. This is required to determine the resource availability for hosting edge applications. *Chord*, P2P approach to locating files over a large number of nodes. A node in Chord looks up a query through hopping to one of its known nodes, no knowledge of inter-node similarity. Since Chord has a similar strategy to *Kinaara*, we compared them to evaluate the similarity table and resource based keying design.

B. Kinaara Clusters in WiFiDog

To understand the need for *Kinaara*, we analyzed the WiFiDog dataset, for real-time device coexistence. In *Kinaara*, resources are related to the number of available devices, hence, we needed to explore the device availability. In WiFiDog users spend an average of 78min/AP, hence, we split the dataset to time intervals nearly half this time, i.e., 40mins, and studied those carefully. Our findings showed ≈12K time intervals where *Kinaara*'s ring holds at least 1000 distinct devices (8000 resources) from collocated APs, either through single or multiple mediator scenarios. This means that for those time intervals *Kinaara* ring would experience join and leave operations but at least 1000 devices stayed for the whole period. Investigating APs in the corresponding intervals, we collected those resources from as low as 9APs, which is a low number in enterprise networks, e.g., mall. Results from real traces as WiFiDog shows the feasibility and need for a distributed approach to manage those resources.



(a) Similarity Ratio (b) Throughput (ops/sec)

Fig. 6: Examining similarity functions for *Kinaara*

C. Similarity Function

We examined five token based similarity functions: Hamming Distance [27], Dice's Coefficient [28], Damerau-Levenshtein (DL) [29], Jaccard [29], and Jaro-Winkler [29].

In this experiment, we started with two identical 128-bit strings. We had 128 iterations to alter one of them, each flipping one random bit that was not previously flipped. At the end of each iteration, we compared the two strings using the functions mentioned above. Fig. 6a depicts the string comparison at each iteration. Although Hamming distance and

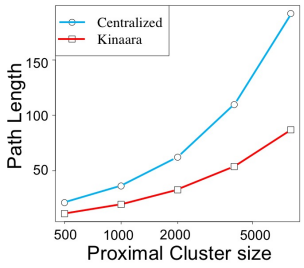


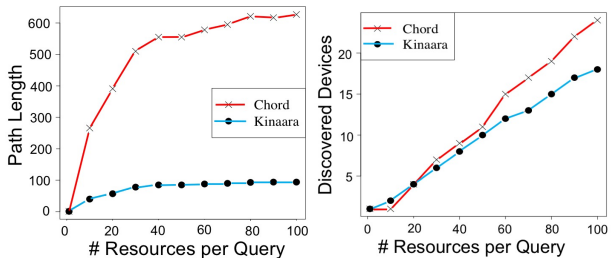
Fig. 7: Centralized Vs. *Kinaara* Resource Discovery

Jaccard were consistent with key changes, we chose Hamming distance for its throughput superiority as shown in Fig. 6b.

D. Centralized Vs. Distributed Resource Discovery

In this experiment, we compared *Kinaara* to the Centralized approach explained earlier. Centralized places all the logic on the mediator without using rings or similarity tables. When receiving a resource request, mediator shortlists devices with the required resources and contacts them one by one to check their resource availability. In this experiment we mimic real life scenarios where mediator has no control on application execution, hence, no way to predict if a resource is busy or finished executing its application. In this experiment, we modeled a busy resource as the summation of two parameters: (a) *holding time* of 2 minutes and (b) *App execution time*, a random period between 1-10mins.

Fig. 7 shows *Kinaara* to visit fewer number of devices, as *Centralized* has higher failure rate from visiting devices with busy resource. Fig. 7 shows *Kinaara* to visit fewer devices, e.g., 40% in 2K resource cluster, with 90% *request success*.



(a) Path Length (b) Discovered Devices

Fig. 8: *Kinaara*'s selective lookup vs. Chord

E. Efficient Path Length

In this experiment, we investigate the number of device hops needed to fulfill a search query. We compare *Kinaara* to Chord's open source code [3] with both subject to the same input. *Kinaara* selects its next device to look for resources through resource similarity comparison with the search query, while Chord hops sequentially between devices till finding its target. Using randomly generated queries, we performed this experiment on a 1K device cluster (8000 resources, a practical number from § VI-B).

Fig. 8b shows Chord and *Kinaara* to discover almost the same number of devices. *Kinaara*, however, visits fewer devices, which results in a customized resource lookup, that can reach 70% fewer device visits as shown in Fig. 8a. This experiment shows that the similarity table significantly decreases the number of device hops per search query.

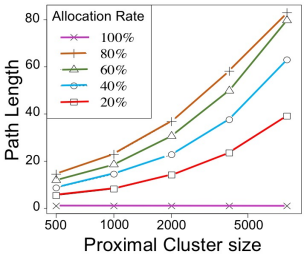


Fig. 9: Cluster Allocation, impact of increasing resource usage ratio on discovery

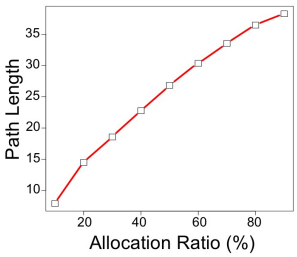
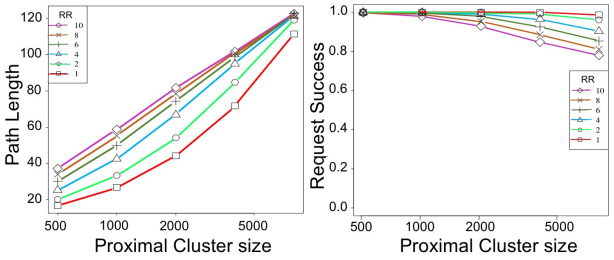


Fig. 10: Path Length under different Cluster Allocations on a 2k resource cluster.

F. Cluster Allocation

In this experiment, we measured the impact of a congested cluster over *Kinaara*. This occurs when resources are either scarce or pre-allocated to other applications. In this experiments, we used variable cluster sizes, each was subject to an assumption that a portion of its resources was pre-allocated.

Fig. 9 showed an increase in the path length to fulfill the same search query as the initial assumption decreased available resources. We monitored a decreasing rate of the path length increase as shown in Fig. 10, while achieving the same *request success*. This shows another benefit of using a key that conveys available resources and a similarity table that rectifies the scope of lookup to the most similar peer, not just any peer.



(a) Path Length (b) Request Success

Fig. 11: Impact of Request Rate (RR) on resource discovery

G. Resource Request Rate (RR)

In this experiment, we monitored the impact of multiple *Request Rates* (RR) on path length and request success.

Fig. 11a shows the path length to increase as the *request rate* increases, however, the spaces between the curves decrease implying a controlled *path length* increase. The *path length* behavior came along with a decrease *request success* due to discovery failure under intensive requests as shown in Fig. 11b. Increasing the *request rate* holds large number of resources, for example *request rate*=10 on a 2000 resource network results in 400 resource request per minute, which leads to requesting the whole network resources in 20 minutes. Therefore a *mediator* has to control *Kinaara request rate* while considering the number of resources available in the cluster. Also, they have

to report to *cloud server* that a *proximal cluster* is receiving many requests asking to target other clusters.

Fig. 12: Mobility Rate and its impact on discovery.

H. Mobility Rate

In this experiment, we measured the impact of variable mobility patterns on *Kinaara*. Mobility pattern is various devices joining and leaving the cluster that does not belong to WiFiDog dataset. This means that within each time interval, i.e., 1 minute, a percentage of the cluster devices move in or out. This scenario shows the impact on discovery by *Kinaara*'s device leave strategy (§ IV-B3) that does not remove entries in similarity tables for devices that left the cluster.

We investigated multiple mobility patterns, i.e., 10-80%, and all had similar impact on *Kinaara*, Fig. 12 depicts the zero and 20% patterns. We witnessed low impact on clusters up to 3000 resources, while path length increased between 20-45% in larger clusters. Hence, mediators should consider mobility prior to expanding the cluster size from neighboring APs.

VII. Conclusions

In this work, we presented *Kinaara*, a distributed resource discovery solution for mobile edge networks. First, we proposed the concept of keys encoding the resources of each device and designed and implemented the Resource Encoder that maps resources to keys. Second, we used keys to sort devices on a logical ring structure, each with a data structure, i.e., similarity table, pointing to devices with similar keys, i.e., resources. Third, we leveraged previous modules for an optimized resource lookup. Finally, we evaluated *Kinaara* against other approaches and under stress testing conditions. Our evaluation showed *Kinaara* to efficiently discover resources per incoming request, showing *Kinaara* 70% better than Chord and 40% than Centralized. In terms of mobility, *Kinaara* had low overhead at high mobility rates in clusters up to 3K resource compared to stationary modes.

Acknowledgment

This research is partially supported by the National Science Foundation under grant CNS-1454285. A major part of this work was performed during Ahmed Salem's internship at the IBM T.J. Watson Research Center.

References

[1] "Data Captured by IoT Connections," http://tinyurl.com/j7olhz3.
[2] "Rich data and Increasing value of IoT," http://tinyurl.com/pw38uzu.
[3] I. Stoica *et al.*, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.
[4] M. Satyanarayanan *et al.*, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.
[5] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," *Big Data and Internet of Things: A Roadmap for Smart Environments*, 2014.
[6] Q. Ning, C.-A. Chen, R. Stoleru, and C. Chen, "Mobile storm: Distributed real-time stream processing for mobile clouds," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*. IEEE, 2015, pp. 139–145.
[7] H. Wang and L.-S. Peh, "Mobistreams: A reliable distributed stream processing system for mobile devices," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, 2014.
[8] N. Zhang *et al.*, "Gameon: P2p gaming on public transport," in *Proceedings of the 13th Annual Int'l Conference on Mobile Systems, Applications, and Services*. ACM, 2015, pp. 105–119.
[9] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th Int'l conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.
[10] N. Navimipour *et al.*, "Resource discovery mechanisms in grid systems: A survey," *J. Netw. Comput. Appl.*, 2014.
[11] S. Ratnasamy *et al.*, "A scalable content-addressable network," in *Proceedings of the 2001 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, USA.
[12] "AllJoyn," http://tinyurl.com/zheat7u.
[13] "Iotivity, finding resources," http://tinyurl.com/j94d37g.
[14] S. K. Datta *et al.*, "Resource discovery in internet of things: Current trends and future standardization aspects," in *Proceedings of IEEE 2nd World Forum of Internet of Things (WF-IoT)*, 2015.
[15] Y. Fathy, P. Barnaghi, S. Enshaeifar, and R. Tafazolli, "A Distributed In-network Indexing Mechanism for the Internet of Things," in *Proceedings of the 3rd IEEE World Forum on Internet of Things*, 2016.
[16] E. Carlini, A. Lulli, and L. Ricci, "dragon: Multidimensional range queries on distributed aggregation trees," *Future Generation Computer Systems*, vol. 55, pp. 101–115, 2016.
[17] Y. Lee, Y. Ju, C. Min, S. Kang, I. Hwang, and J. Song, "Comon: cooperative ambience monitoring platform with continuity and benefit awareness," in *Proceedings of the 10th Int'l conference on Mobile systems, applications, and services*. ACM, 2012, pp. 43–56.
[18] A. A Sani *et al.*, "Rio: a system solution for sharing i/o between mobile systems," in *Proceedings of the 12th annual Int'l conference on Mobile systems, applications, and services*. ACM, 2014, pp. 259–272.
[19] M.-R. Ra *et al.*, "Medusa: A programming framework for crowd-sensing applications," in *Proceedings of the 10th Int'l conference on Mobile systems, applications, and services*. ACM, 2012, pp. 337–350.
[20] A. Ahmed and A. Ejaz, "A survey on mobile edge computing," in *Proceedings of the 10th Int'l Conference on Intelligent Systems and Control (ISCO)*, 2016.
[21] A. Salem, N. Desai, T. Salonidis, and T. Nadeem, "Resource hunting on the edge," in *Proceedings of the Edge Computing (SEC), IEEE/ACM Symposium*, 2016, pp. 83–84.
[22] A. Salem and T. Nadeem, "Lamen: leveraging resources on anonymous mobile edge nodes," in *Proceedings of the Eighth Wireless of the Students, by the Students, and for the Students Workshop*. ACM, 2016, pp. 15–17.
[23] V. Raychoudhury *et al.*, "Top k-leader election in mobile ad hoc networks," *Pervasive and Mobile Computing*, vol. 13, 2014.
[24] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 839–858.
[25] F. Baccelli, N. Khude, R. Laroia, J. Li, T. Richardson, S. Shakkottai, S. Tavildar, and X. Wu, "On the design of device-to-device autonomous discovery," in *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on*. IEEE, 2012, pp. 1–9.
[26] M. Lenczner and A. G. Hoen, "CRAWDAD dataset ilesansfil/wifidog (v. 2015-11-06)," Downloaded from http://tinyurl.com/zxorzl7, Nov. 2015.
[27] M. Norouzi *et al.*, "Hamming distance metric learning," in *In Proceedings of Advances in neural information processing systems*, 2012.
[28] D. Lin *et al.*, "An information-theoretic definition of similarity." in *ICML*, vol. 98, no. 1998. Citeseer, 1998, pp. 296–304.
[29] Y. Sun, L. Ma, and S. Wang, "A comparative evaluation of string similarity metrics for ontology alignment," *JOURNAL OF INFORMATION &COMPUTATIONAL SCIENCE*, vol. 12, no. 3, pp. 957–964, 2015.