Multiple Peer Chord Rings Approach for Device Discovery in IoT Environment

Binh Minh Nguyen[1],Hong-Nhat Quoc Hoang[1],Giang Nguyen[2], Tuyet-Trinh Vu[1], and Hieu Le[1]

[1] School of Information and Communication Technology, Hanoi University of Science and Technology, Ha Noi, Viet Nam
minhnb@soict.hust.edu.vn, nhat.hqh@gmail.com, trinhvt@soict.hust.edu.vn, hieulq19@gmail.com

[2] Institute of Informatics, Slovak Academy of Sciences, Slovakia
giang.ui@savba.sk

Abstract

Recently, with the advances of information and electrical technologies, Internet of Thing (IoT) has moved from being a far vision to an increasing market reality. IoT promises to connect all physical "things" via a dynamic global network infrastructure and creates intelligent behaviors in different contexts without human manipulations. In line with this development, the number of IoT solutions also have augmented exponentially covering many domains from health care, urban management to life quality improvement and so forth. However separate appearance of these solutions today has led to barriers in enlarging smart environments. It is well-known that in the very near future, IoT requires the capabilities of high scale and dynamic adaptation with integration of many new and existing smart contexts together. Hence, the need of having a comprehensive management method for the large-scale IoT environments is urgent at present. In this paper, we focus on designing an overlay management architecture based on Chord algorithm to manage things in heterogeneous space. The architecture is a collection of multiple peer Chord rings. Thus, each of these rings is considered as a smart context. We also bring forward operation mechanisms for the architecture such as thing identification, key insert/remove, key lookup and node joining/leave. Our proposed architecture is simulated and evaluated with diverse properties. The experiments show feasibility of our approach in reality meeting scalability, robustness and maintainability requirements in managing and controlling cyber-physical systems.

Keywords: IoT, Chord, multiple peer rings, distributed hash table, overlay management network, discovery mechanism, thing naming service, object discovery service

1 Introduction

Internet of Things (IoT) has emerged as a core factor of forth industrial evolution, where all things connect each other to share data and perform automatically tasks in specific contexts. Nowadays, advances in IoT create a huge potential for programmers to build novel value-added services by exploiting multiple capabilities (e.g. sensing, communication, computing, storage, and analytic) [1]. Thus, a lot of existing IoT solutions have been developed such as smart building management, traffic control, and health monitoring and so on. However, it is also well-known that the IoT solutions require the ability of very high scale and dynamic adaptation to be able to expand on large space such as towards the development of smart cities or even smart country in the near future.

In IoT environment, each thing is identified by an unique ID. Generally, a set of things joins together in a smart context (e.g. smart room or home) that will be managed by a specific IoT platform. The duty of platforms is to provide a common and unified architecture to coordinate entire system operations. In the small space, at logical management layer of IoT platform, things often are modeled in a hierarchical form consisting of several levels of root, parent and child nodes. Nonetheless, in larger environments, due to the heterogeneous of network, maybe each device or object has more than one link to connect to others and some of them can participant in many contexts at the same time. In this way, the hierarchical architecture at logical management layer may not be sufficient, especially in mash-up scenarios that contain millions of devices belonging to different IoT solutions integrated together. The reason is that we cannot determine what and where are root, parent, and child nodes to control and manage efficiently. While some devices can be looked up quickly, other thing control data flows will get stuck. Standing from perspective of logical management layer, an efficient mechanism for organizing and discovering devices in large IoT environments is the urgent requirement today.

In this paper, we propose a novel approach to manage things and thus provide discovery mechanisms for them in large smart contexts based on Chord algorithm[2]. In this direction, we consider a large IoT environment as an integration of multiple logical peer Chord rings, which could be intersected together by one or many share nodes. Each node on the ring is considered as an IoT gateway and plays the role of transmitter and connector to sensors that can connect to one gateway at the same time. In this proposed logical management architecture, by using DHT, we identify each ring, gateway, and sensor by an unique hashed ID under several features to distinguish each other. We also propose operation mechanisms including discovery, join in and out for both gateways (i.e. ring nodes) and sensors in the complex architecture. With the approach, we can solve the diverse and complicated links problem among things in real smart environments and hence enables IoT systems to manage and control things efficiently. The approach also meets scalability, robustness and maintainability requirements for large and mash-up smart contexts. Our work distinguishes with existing studies in four main aspects: adopting Chord to build and identify multiple peer rings for things (i.e. gateways and sensors) in a single unified architecture, managing multi-shared nodes among rings, controlling joining and leave of things in the architecture, and high range discovery query mechanisms for things.

The rest of the paper is organized as follows. Section 2 presents a typical IoT scenario, which puts motivations for our study. We discuss some related work in section 3 to highlight the contributions of our research in comparison with other researches. In section 4, we describe architecture design for the overlay thing management system with multiple peer Chord rings that operate together at the same time. Also in this section, we introduce main mechanisms to suit the proposed architecture including thing identification, key insert/remove, key lookup and node joining/leave. Theory discussion and analyses of the system operation also are presented

in this section. Section 5 provides our experiments, gained results and observations with our proposals through simulation in order to demonstrate the efficiency and feasibility of the architecture. Finally, section 6 concludes and figures out some future directions.

2 Scenario and Motivation

In this part, we give a concrete IoT scenario deployed in a skyscraper, which has many floors, rooms, corridors, and lobbies) to highlight the problem of thing discovery in fact. The IoT skyscraper scenario has the following characteristics:

- There are many environmental sensors, electric devices and physical things such as temperature, humidity, moving, lights, air-conditions, routers, smart TVs, clocks, photo frames, lamps, fans, curtains, windows, doors and so on that are deployed inside the building.
- Networking is heterogeneous and ad-hoc. While few things use wireless, few others exploit power line communication (PLC) or cable to send and receive data, a huge number of small sensors only interacts through Bluetooth (i.e. very short range communication). The common feature is that most of things do not connect directly to the Internet or cloud for data processing [3]. The things need intermediate devices to share data and connect to global. The intermediate devices could be smart TVs, network routers that have capabilities of computing, storing and transmitting data. We consider the intermediate devices as gateways. The remainder things only play the role of perceiving surroundings and sending data to gateways and they are categorized into sensors.
- In building spaces such as room, lobby, corridor, and floor, all devices operate at the same time in different smart contexts without human interactions. For example, on the one hand, when room door detects an open operation, this means someone comes in the room. Then lights and air-condition are turned on automatically. In this way, the lights and air-condition in this room are cooperated by the smart room context. On the other hand, some devices in the room above are also managed by another other smart contexts. Concretely, all room and corridor air conditions in the same floor are linked with a centralized air-condition management component. These air condition devices thus will be controlled to ensure that floor temperature does not differential more than two degrees among diverse spaces.

From the scenario described above, it can be seen that there are many smart contexts, which can intersect and operate simultaneously in a large IoT environment. The intersections aspect could be taken shape because the complexity of physical network connections as well as gateway and sensor capabilities. If we would like to manage things in the building, our control data flows can be sent to a devices through many ways. As a result, we need to have an suitable architecture to organize and mechanisms to discover things in the environment. Until now, there are not existing IoT solutions that enable to cooperate completely intersecting contexts including thing identification and discovery. In this way, the work presented in this document deals with the following questions:

- At the logical management layer, how can we manage things in heterogeneous environment with intersection of many IoT contexts?
- How does a thing join in and out the environment?
- How to determine effectively location to control a thing in the environment?

3 Related Work

There are a lot of existing works related to this study. In this section, we classify roughly these works into several groups according to their approaches as follows:

Thing identification. Identification is crucial for IoT to name and thus enables to manage things. IoT identification methods could be divided into two types. At the low level, the first type uses physical tags to identify devices. Typical example is RIFD technology [4]. The RFID tag represents a simple chip or label attached to things. RFID reader thus transmits a query signal to the tag and receives reflected signal from the tag to identify the things. At higher level, the second type addresses things by ID or IP. The difference among two methods is that tag-based identification often is not globally unique. In contrast, addressing may help uniquely identify objects. This work described in this paper employs the second type to identify things via IDs.

Multiple Chord rings. In [5], N. Antonopoulos proposed an novel approach that uses multiple Chord rings in looking up multi-dimensional data. In this way, data is distributed and labeled on many rings. Each of them corresponds with a keyword (i.e. a dimension). There is a central Chord ring, which integrates these secondary rings and thus provides search mechanism for entire system. Structurally, with a central ring and many secondary rings, this architecture includes two levels of organization with different roles.

Multiple Chord rings in IoT. Using DHT seems to be an useful solution for object discovery in IoT. Effort presented in [6] brought forward a model, in which many unstructured peer-to-peer networks are linked together through super nodes. These nodes thus are located in a Chord ring. In other words, the super nodes are elements of Chord's ring and the unstructured peer-to-peer networks will be linked and their peers could be discovered based on the DHT identifications. However, the multiple peer Chord ring architecture is not mentioned in the work. P. Liu in [7] proposed another architecture with Chord rings, which correspond with two levels to identify objects using many different RFID standards. Each node in the high level Chord ring is a specific standard and it is linked to the low level ring, where nodes with the same standard are connected together. In [8], the authors also exploited two-level architecture with a DGT (Discovery Geographic Table) and a DLS (Discovery Location Service) to connect IoT networks that can be deployed to very large scale. The common feature of those works above is that they built hierarchical P2P architectures using one or more rings. In those architectures, each ring node plays the especial role as a transit station for discovery operations. Nonetheless, there are still not solutions dealing with using an peer model including multiple rings to identify and discover IoT objects.

Chord DHT optimization. Several existing works focused on exploiting Chord DHT or creating Chord variants to optimize in a certain application domain. The work described in [9] figured out the lack of balancing load in Chord's discovery mechanism. In this way, the author proposed a token to estimate and select next node in figure table (FT) instead of selecting the nearest key node as in basic Chord. Other works such as [10] and [11] towards using Chord DHT to optimize lookup process, especially in network routing problem.

As compared with the works described above, our contributions include:

- Proposing a novel logical architecture to manage the huge number of IoT objects at the same time by integrating multiple peer Chord rings. These rings can share one or many common nodes. In the architecture, there is no central or high-level ring like the existing studies and therefore our system achieves scalability without depending on any main ring.
- Proposing hash methods to identify uniquely rings, gateways and sensors for the architecture.

- Proposing mechanisms to discover any objects in the multiple Chord rings.
- Providing mechanisms for objects join in and out in the proposed architecture.

With those characteristics, our approach provides a solution for the issues presented in the section sect:scenario.

4 Logical Management Architecture Designs

In this section, multiple peer Chord rings architecture designs and discovery mechanism are described concretely.

4.1 Chord Algorithm

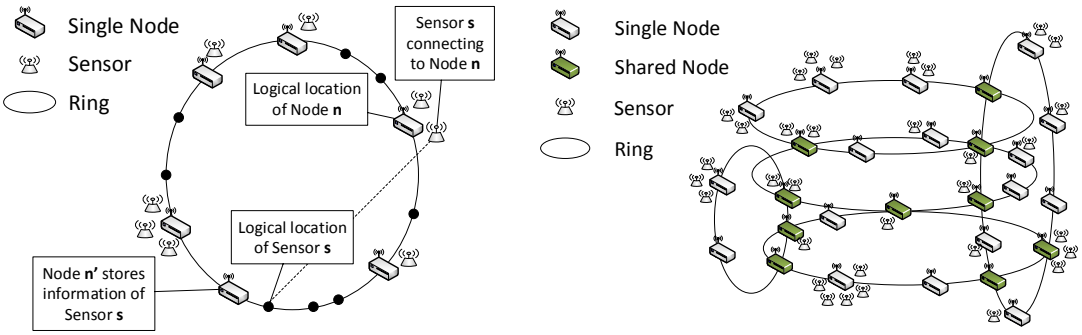
Chord[2] is widely known as one of popular DHT algorithms to reduce significally the number of query and update messages in peer-to-peer (P2P) network. By using consistent hashing[12], Chord organizes nodes into a ring. Thus, a ring can cover the huge number of nodes (up to 2^{160} with Secure Hash Algorithms, for example SHA-1[13] provides 160-bit keys). Each node maintains routing information of other $O(\log N)$ nodes. Hence, $O(\log N)$ query messages are required for each lookup request and $O(\log^2 N)$ update messages are transmitted for each node's joining or leave request. Due to its advantage features such as load balance, decentralization, scalability, availability and flexible naming, Chord is applied to design our logical management architecture in order to meet requirements given in section 2.

4.2 Multiple Peer Chord Rings Architecture

The multiple peer Chord rings architecture is come into being by three components, namely ring, node, and sensor, which are defined as follows:

- Ring - symbolizes a single logical management network. Ring uses Chord as a base method to organize and discover things.
- Node - is an object located on the ring as a peer of the ring. Node corresponds with a gateway device in fact. We divide nodes into two types:
  - Single node - is a node that is joining in only one ring.
  - Shared node - is a node that is simultaneously joining in at least two rings.
- Sensor - is an object representing sensors in practice. In our multi-ring architecture, sensors cannot join directly in rings like nodes (i.e gateways). The reason is that sensors manifest characteristics of real device, which are limited by capabilities of communication, computing and storage. Therefore, sensors have to connect to a certain gateway and depend on the capabilities of the gateway. However, in order to manage easily sensors in our proposed architecture, we store connection information between sensor and gateway in dispersion way on nodes of ring.

As presented, our multi-ring Chord architecture includes many peer rings that intersect each other via one or more shared nodes. Particularly, on each ring, things (i.e. nodes and sensors) organization and discovery operations are performed independently based on traditional Chord. Figure 1a illustrates a specific example for a single Chord ring, where nodes and sensors are



(a) Things organization on a single ring (b) Things organization on multi-ring architecture

Figure 1: Multiple Peer Chord Rings Architecture

mapped to logical positions on the virtual ring circle. A concrete example for our architecture with multiple peer rings is shown by Figure 1b. To ensure discovery processes on different rings, while Successor, Predecessor, Finger Table (FT) data are maintained for a single ring conforming to Chord, we propose a new data called Ring Table (RT). Thus the table provides routing data and based on that, discovery queries can be routed to send messages from a ring to others. RT is designed as follows: it is created in the same manner of FT and distributed on every ring node. RT consists of a row number corresponding with the number of distinct successor nodes that appear in FT. RT is illustrated by a specific example in Figure 2. Each row of RT contains 2 data fields:

- NodeID indicates a distinct successor node that appears in FT.
- RingIDs lists ring IDs that the node is joining in at the time.

The reason for creating a new RT instead of adding a field ringIDs to FT is to avoid checking and updating repeatedly RingIDs of the same successor node on many rows of FT. Meanwhile, for each node, only one row is required with RT. Otherwise, because a node can join simultaneously in multiple rings, for each of those rings, the node requires an individual ring data including information of ringID, Successor, Predecessor, FT and RT. Based on the data, thing organization and discovery on each ring are maintained stably and independently.

4.3 Thing Discovery Mechanisms

In order to realize discovery process, our system has to provide the following mechanisms: identification, key insert/remove, key lookup and node join/leave. Firstly, the identification offers a naming method for rings and objects (i.e. nodes and sensors) on our management layer. While key insert/remove brings forward mechanisms to store and delete object discovery information distributed on ring nodes, the key lookup allows finding objects among multiple peer rings. Finally, node join/leave ensures that the node joining/leave processes are performed precisely in different use cases. In our design, meanwhile nodes are peers on Chord rings, sensors are not. Then sensors are managed via nodes, and they only take key places on the ring. Consequently, sensor joining/leave (i.e. sensor connection/disconnection to/from gateway) already are done via key insert/remove mechanisms above.

4.3.1 Identification

On each ring in our architecture, nodes and sensors also are assigned a 160-bit identifier using consistent hashing method with SHA-1 hash function. Thus, each node involved ring has a key-value pair calculated by formula: $nodekey = hash(NodeID)$. On the other hand, each sensor connected to node also has a key-value using formula: $sensorkey = hash(SensorID)$ (both NodeID and SensorID are identifiers in physical layer of devices). These keys are logical identifier of devices on our management layer and they are also used to organize and discover them.

Due to multiple peer rings architecture, rings has to be distinguished with each other like node and sensor identifications. Hence, an identifier called RingID is used to identify rings. According to Chord, a ring is abstractly initialized by a first node, as a result, the node also can be employed to generate a RingID. Thus RingIDs are created by hashing a value including ID of the first node and a timestamp. In this way, the hashing function for rings is as follows: $RingID = hash(first_NodeID|timestamp)$. The timestamp is used to avoid collision in the case of rings are generated by a same first node. With the proposed method, RingIDs become unique and enable to identify efficiently rings in our architecture.

4.3.2 Key Insert and Remove

When connecting to IoT environment managed by our architecture, both nodes and sensors also are provided keys through the mechanism presented in subsection 4.3.1. After that their discovery information are stored in dispersion way on their keys's successor(s). Specifically, for node joining case, when a node participates in a ring, its key-value pair is stored in its key's successor on this ring based on traditional Chord. If the node participates in many rings at the same time (a shared node), its key-value pair then is stored on its key's successors on all those rings. Therefore, each ring has a successor of the node key. Similarly, for sensor joining case, when a sensor connects to a single node, key-value pair containing sensor connection information to the node is stored in the key's successor on that ring. If a sensor connects to a shared node, its key-value is stored simultaneously on the key's successors on rings, which the shared node is joining in at the same time.

On the other hand, when a node leaves or a sensor disconnects from our architecture, their key(s) is removed from the key's successor(s) in the similar way of key insert operation described above.

Based on Chord, there is a high probability that both cases of node joining/leave and sensor connection/disconnection to/from single node also need $O(\log N)$ messages to be completed. Meanwhile, in case of a sensor connects to a shared node, $O(M \log N)$ messages are required, in which M is the number of rings that the node is joining in, and N is number of nodes on each rings.

4.3.3 Key Lookup

Key lookup operations could be categorized into two types:

- Inner-ring lookup finds successor of the key on the current ring as well as checks the existence of the key on its successor.

- External-ring lookup finds shared nodes in order to jump to another rings in case of the key is not found on the current ring via inner-ring lookup process.

Two key lookup types above are used one after another in the manner of exhausting all rings until finding our the desired key.

Algorithm 1: External-ring lookup algorithm

```
1  Function n.externalringLookup(passed_rings_list, curent_ringID)
2      shared_nodes ← [ ];
3      ring_data ← n.get_Ring_Data(curent_ringID);
4      ring_table ← ring_data.ring_table;
       // Step1:  Find shared nodes in route_table of current ring data
5      foreach row in ring_table do
6          if row.ringIDs has ringIDs not in passed_rings_list then
7              Append row.nodeID and new found ringIDs into shared_nodes;
8      if shared_nodes!= [ ] then
           // if new ringID found, return
9          response.cache_node ← ring_data.successor;
10         response.shared_nodes ← shared_nodes;
11         return response;
12     else
           // if new ringID not found, conduct Step 2
           // Step 2:  Foward to its successor on current ring
13         succ ← ring_data.successor;
14         response ← succ.externalringLookup(passed_rings_list, curent_ringID);
15         return response;
```

Because each ring in our architecture uses Chord, hence, the inner-ring lookup operation also conforms to Chord mechanisms and depends on FT to send lookup query messages. External-ring lookup operation uses RT to find out shared nodes that link the current ring with others. Algorithm 1 describes the external-ring lookup operation, which could be summarized in 2 steps as follows:

1. Looking up a shared node in RT.
2. In the case of no shared node is found, external ring lookup request is sent to the current node's successor to perform again from the above step.

These two steps above are repeated until a shared node is found out or all nodes on the current ring are passed. In Algorithm 1, to avoid repeating inner-ring lookup operations on processed rings before, two parameters (in lookup request/response) *passed_rings_list* and *cache_node* are designed to mark all passed rings and the successor of current node (which found out a certain shared node) respectively. On the one hand, once a ring ID is found in RT, it will be compared with IDs in *passed_rings_list* to ensure the ring has been not passed yet. On the other hand, *cache_node* is used to eliminate repeats of the shared nodes lookup process with passed nodes on the previous ring after unsuccessful shared node find and to continue to discover the desired key on another new ring.

For concrete example, with 2^5 key-space hashing function, a key lookup operation is illustrated by Figure 2, where node $N0$ (which has keyID = 0) receives a key lookup request from user to find information of sensor S with keyID $k = 24$. Since sensor S connects to single node $N15$ on ring $R3$, the key's value is only stored at $N30$ despite $N31$ on ring $R1$ and $N28$ on ring $R2$ are also successors of key k. This operation is performed as follows.

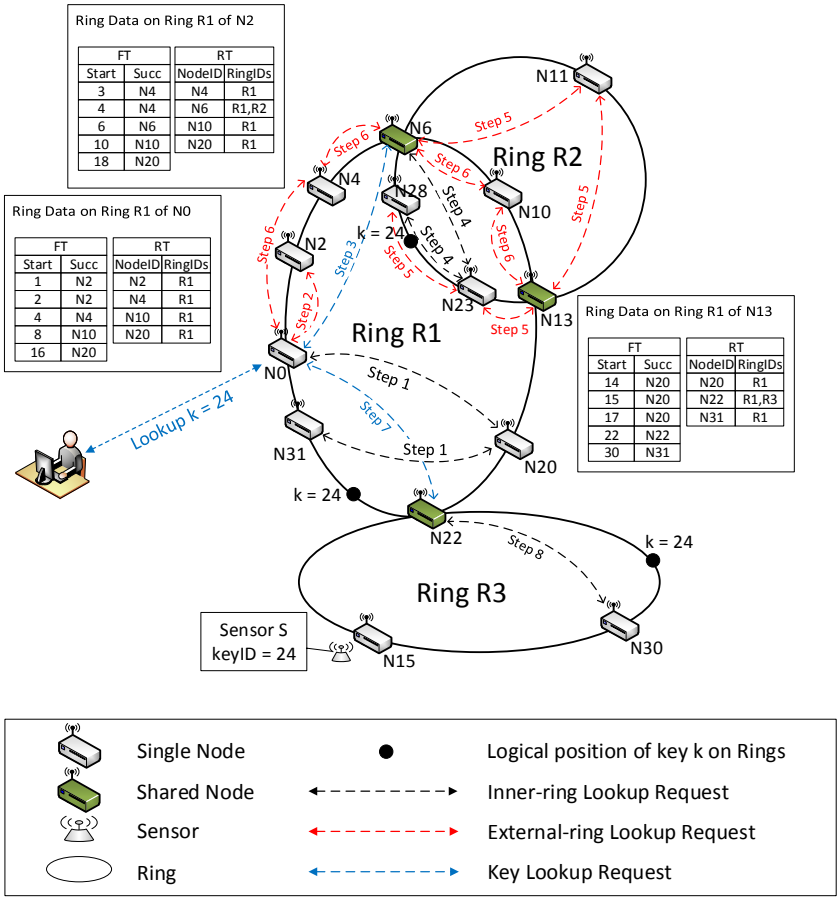


Figure 2: Example of a specific key lookup operation

At Step 1, inner-ring lookup messages are transmitted among $N0$, $N20$ and $N31$. Although $N31$ is a successor of $k = 24$, the key's value is not managed by $N31$. Hence, $N0$ calls the external-ring lookup operation. Because RT of $N0$ does not consist of any shared node information, $N0$ sends an external-ring lookup message to its successor - $N2$ depicted by Step 2 line. At $N2$, shared node $N6$ linking to ring $R2$ is found, then a response message containing address of shared node $N6$ and *Cache_Node* $N4$ (successor of current node $N2$) are returned $N0$. After that $N0$ forwards a key lookup request to $N6$ as Step 3 line. From this time, inner-ring lookup operation and external-ring lookup are performed as Step 4 and Step 5 on ring $R2$. After all nodes on ring $R2$ are queried and cannot find out the desired key as well as any new rings, a key-not-found response is returned to $N0$ from $N6$, then $N0$ sends a request message to last *Cache_Node* $N4$ at Step 2 to find new external rings from ring $R1$. In this way, shared node $N22$ is found out by $N13$ and value of keyID $k = 24$ is found at node $N30$ on ring $R3$ according to traditional Chord.

Theoretically, a key lookup operation on a N-node ring requires $O(\log N)$ query messages with high probability. When a key lookup request is sent to a node, the best case is that the key is managed by another node on the same ring with that node. Then it requires only $O(\log N)$

query messages of inner-ring lookup. Otherwise, extra $O(N)$ messages of external-ring lookup are forwarded to route the key lookup request to another ring. Hence, assuming that there are M rings to be passed during the operation, it requires $O(M \log N)$ messages of inner-ring and $O((M-1)N)$ massages of external-ring lookup.

4.3.4 Node Join and Leave

As mentioned in subsection 4.3.2, when a node joins in a ring, its key-value pair is inserted into the ring. Though, because the node plays a peer role in the ring (as defined in 4.2), ring data of its neighbor nodes are needed to be updated. In our architecture, we assume that gateways participate in our system successively. For concrete instance, gateway G cannot perform simultaneously connection processes with two other gateways $G1$ and $G2$. Instead, this gateway has to connect to $G1$ firstly. After the connection process is done, then gateway G connects to $G2$ as usual. Basically, node join operation is based on Chord. However, two features are designed in order to suit the multiple peer rings architecture, including:

- In case of a new node joins in a ring and becomes a single node, ring data (as presented in subsection 4.2, the data include information of ringID, Successor, Predecessor, FT and RT) of the new node will be initialized and its neighbor nodes (also on that ring) will automatically update their ring data containing information of the new node.
- In case of an existing node (already is a single or shared node in system) joins in another ring to become a shared node. The node initializes a new ring data (ringID, Successor, Predecessor, FT and RT) corresponding with the new joined ring. Then its neighbor nodes on the new ring update automatically their ring data. One the other hand, its neighbor nodes on last rings also update their RT to indicate routes to go to the new ring.

Node leave process from ring(s) also is an update information operation like joining operation. Therefore leave action is realized in the similar way of node joining.

Theoretically, the number of updated nodes on a ring is $O(\log N)$ and number of update messages is $O(\log^2 N)$, with N is number of nodes on the ring. In this way, generally, when occurring any node changes (joining or leave), the number of updated nodes on multi-ring architecture is $O(M \log N)$ and number of messages to complete this update is $O(M \log^2 N)$ with M is number of rings which the node is joining in.

5 Experiments

In order to evaluate the proposed architecture operations, we carried out four tests, including:

- Key lookup and node joining on single ring to analyze operation costs and prove the similarity among our ring design and traditional Chord.
- Node joining operation on multi-ring architecture to prove its operability.
- Key lookup on multi-ring architecture to evaluate key cost finding in different cases.
- Fault existence on multi-ring architecture to test key lookup operation in case of occurring faulty nodes.

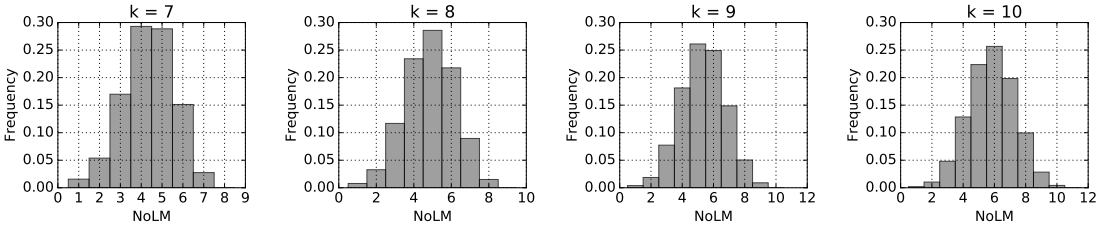


Figure 3: Probability distribution of NoLM with different k from 7 to 10

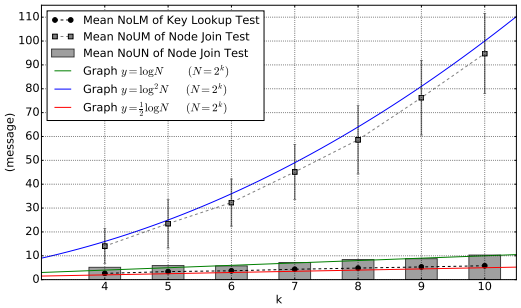


Figure 4: Correlation among mean NoLM, NoUN and NoUM on a single ring

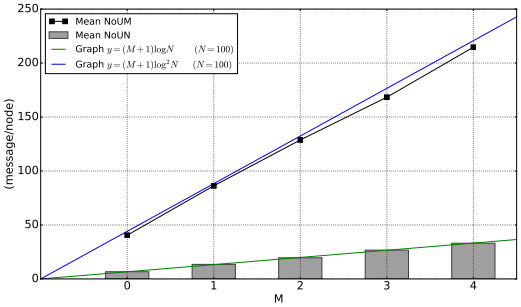


Figure 5: Mean NoUN and NoUM of node join operation on multiple rings

5.1 Experimental Setup

Due to the difficulties to have a real testbed with full hardware and software for a large IoT environment, we build a simulation tool, where ring, node, sensor are considered as programming objects. In this approach, a strong configuration and control interface was developed using Python language for all our tests as well as evaluations. For each of test, we propose different topologies that have diverse characteristics. The different number of nodes and sensors also are created for these topologies in order to gain generality for the tests. Otherwise, although in a strong distributed IoT environment, physical communication cost among things plays the important role in performance assessment for entire system, but with the goal of experimenting the overlay management layer based on multiple peer Chord ring architecture, in this work, we focus on measuring the proposal effects by using the number of sent/received message as the main metrics. We also calculate the mean number of updated nodes to show and prove the operation of node joining mechanism in our architecture.

5.2 Key Lookup and Node Joining on Single Ring

In first test, we simulate $N = 2^k$ nodes for a single ring. The number of sensors connecting to each node is disposed randomly in range from 5 to 10. We change k from 4 to 10 and carry out tests in succession. On the one hand, for key lookup operation, we order each node to lookup all sensors on the ring and measure the number of lookup messages (NoLM). On the other hand, for node joining operation, a new nodes is created and joined in the ring several times. We average the number of updated node (NoUN) and update messages (NoUM) in the system.

Figure 3 shows achieved results from measurement of the NoLM with different k from 7 to 10. It can make an important observation here: NoLM fluctuation is about $\frac{1}{2}\log N$. This phenomenon is identical and explained in [2]. Figure 4 illustrates the node joining test

outcomes, where the mean NoUN increases approximately $k = \log N$ and mean NoUM also augments exponentially base two. Through gained results, it can remark that costs for key lookup is lower than node join operation. In summary, the first experiment demonstrates our single ring with nodes and sensors object operates similarly with traditional Chord although these objects are disposed suitably on the ring.

5.3 Node Joining on Multi-ring Architecture

The goal of this test is to evaluate node joining operation on multi-ring architecture. We use five separate rings called $R1$, $R2$, $R3$, $R4$, and $R5$. Each ring contains 100 nodes. A new node X is created to join in these rings. Firstly, the new node X joins in ring $R1$, and becomes a single node on ring $R1$. Next, we continue to order X to join in ring $R2$. At the time, X becomes a shared node among $R1$ and $R2$. Node joining operation is repeated sequentially to connect X with $R3$, $R4$ and $R5$. At the time, node X is a shared node of all five rings. We carry out the test several times to obtain NoUN and NoUM for joining process. The gained results are shown by Figure 5.

It is clearly to see that NoUN and NoUM increase linearly as compared with the number of rings, which node is participating in. For example, node X is participating in ring $R1$, $R2$, and $R3$. If X continues to join in ring $R4$, all four rings are required to update with about $\log N$ updated nodes on each ring. A value observation could be made from this test: NoUN and NoUM are approximate $(M+1)\log N$ and $(M+1)\log^2 N$ respectively with M is ring number, which the joining node is attending before participating in another ring. The achieved outcomes of this test are conformable with the node joining operation design described in the previous section.

5.4 Key Lookup on Multi-ring Architecture

NoLM of key lookup operation on multi-ring architecture is measured in this test. NoLM covers the number of inner-ring lookup messages (NoIRLM) and number of external-ring lookup messages (NoERLM). We evaluate two instances, namely: looking up existing keys and non-existing keys.

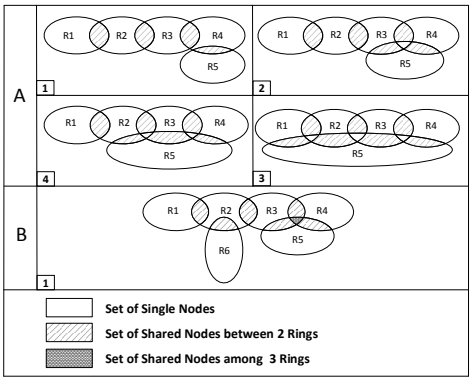


Figure 6: Different topologies

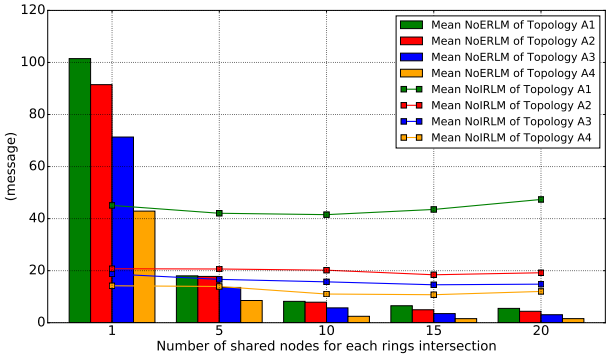


Figure 7: Mean NoIRLM and NoERLM of Topologies A1,A2,A3,A4

5.4.1 Existing Keys Lookup

For first instance, we use four topologies, including $A1$, $A2$, $A3$ and $A4$ as showed in Figure 6. 100 nodes is set for each ring. The number of sensors are disposed in range from 5 to 10 per node. We consider five cases with different shared node number for each topology. The number of shared node is 1, 5, 10, 15 and 20. From each node on ring $R1$, lookup requests are sent to find all sensors stored on ring $R5$. Obtained results are presented by Figure 7. There are some remarks from this test as follows.

First, we examine performance of each topology. It is easily to see that the difference of NoIRLM is insignificant despite the number of shared nodes is diverse in test cases. This proves that the cost of inner-ring lookup operation does not be affected by the number of shared nodes. On the other hand, NoERLM is reduced significantly in inverse proportion to the shared nodes number. This could be explained as follows: in the case of the number of shared nodes increases, it creates more opportunities to find out quickly route to go to other rings. It also means that the performance is better in this case. Second, both NoIRLM and NoERLM decrease when ring $R5$ is linked to other closer rings to $R1$. The reason is that the forwarding message path from ring $R1$ to ring $R5$ is shorter. Third, NoERLM is considerably lower than total node number, even it also is smaller than NoIRLM while the number of shared nodes is greater than or equal to five. This indicates that RT works effectively to route messages among rings.

5.4.2 Non-existing Keys Lookup

In the second instance, we create topology $B1$ in Figure 6 and look up several keys that are not stored by any nodes. The goal is to assess the worst cost of lookup operation. We set up six independent rings (from $R1$ to $R6$). The node number N on separate ring is configured as introduced in Table 2. We carry out tests with two separate use cases corresponding to the number of shared nodes, which are shown in Table 1. Also in this table, linking schemata among these rings are described. The intersection is created by the shared nodes. At this time, the node number on each ring is N' written in Table 2. It could be recognized that N' is greater or equal to N because shared nodes are counted repeatedly.

Linked Rings	Number of shared Nodes	
	Use Case 1	Use Case 2
R2 → R1	5	15
R3 → R2	6	16
R6 → R2	10	20
R4 → R3	4	9
R5 → R3	6	11
R5 → R4 → R3	3	8
R4 → R4	5	10

Table 1: Rings' links description

Ring	N	Use Case 1			Use Case 2		
		N'	$NoIRLM$	$NoERLM$	N'	$NoIRLM$	$NoERLM$
1	100	105	3.61	105	115	3.97	115
2	110	126	4.23	126	146	4.59	146
3	120	133	4.28	133	148	4.51	148
4	130	138	4.42	138	148	4.61	148
5	140	140	4.39	140	140	4.42	140
6	150	150	4.63	150	150	4.59	150
Total	750	792	25.56	792	847	26.69	847

Table 2: Mean NoIRLM and NoERLM

As mentioned in section 4.3.3, generally, the number of required messages for a lookup operation covers $O(M \log N)$ of inner-ring and $O((M-1)N)$ of external-ring lookup operations in case of existing key, with N is number of node on each ring and M is number of passed rings. However, in this test, the desired key does not exist. Hence, all nodes in the topology are passed by lookup queries. Concretely, $NoERLM$ on each ring is equal to N' as shown in Table 2. On the other hand, $NoIRLM$ on each ring is approximate $\frac{1}{2} \log N'$. This phenomenon manifests that inner-ring lookup is only performed once per ring. In summary, the result of this experiment demonstrates *rings_passed_list* and *cache_node* have effect in avoiding repeat of inner-ring lookup on passed rings and external-ring lookup on passed nodes on each rings as our expected design.

5.5 Fault Existence on Multi-ring Architecture

This test is carried out to evaluate the capability of lookup operation in case of faulty nodes existence. We use topology $A1$ and $A4$ illustrated in Figure 6 with 100 nodes disposed for each ring, in which there are 10 shared nodes among each ring pair. In entire system, each node is linked randomly from 5 to 10 sensors. We experiment the number of faulty nodes in range from 1 to 5 with two cases: faulty nodes are single nodes and faulty nodes are shared nodes. In this way, each node on ring $R1$ order to look up all sensors on ring $R5$. Assuming that the lookup process is considered to be failed if there are any messages sent to faulty nodes, the average success rates are measured and shown by Figure8.

With topology $A1$, there is only one way to find out keys stored on ring $R5$ (via ring $R2$, $R3$, $R4$ and $R5$ in succession and their shared nodes). Therefore, faulty shared nodes cause the lower average success rates as compared with faulty single nodes case. Meanwhile, with topology $A4$, there are many ways to reach ring $R5$. This makes a higher probability of success rate than topology $A1$. In addition, the effect of faulty shared nodes number is insignificant in this topology. In summary, with the existence of faults, our architecture still work. However, due its features, diverse ring integration topologies will affect to the ability of successful object lookup.

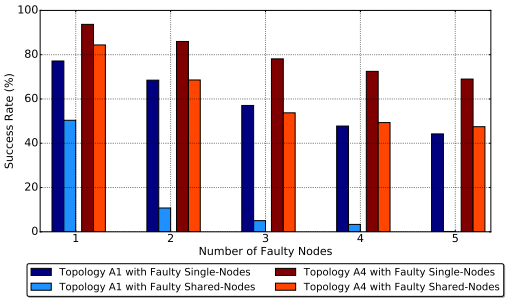


Figure 8: Success Rate of Key Lookup Operation in Fault Existence Experiment

6 Conclusion and Future Works

This works presented in the document concentrate on proposing a management architecture for things in large-scale IoT environment. The architecture thus is an overlay network providing mechanisms to organize, control interaction and discover objects represented physical things. In our proposal, multiple peer Chord rings is employed to suit the fact that it may have different smart contexts intersecting each other in a large space. In this direction, we define a way to dispose and manage things on those logical rings using node and sensor objects corresponding with gateways and sensors. Together with building rings, operation mechanisms for the multi-ring architecture also are proposed, consisting of thing identification based on consistent hashing, key insert/remove, key lookup for nodes and sensors (which are categorized into two operation types: inner-ring and external-ring lookup), and node joining/leave. The architecture is tested by simulating a large-scale environment with a large number of nodes and sensors to evaluate performance and operation costs of the entire system with various topologies and properties. The achieved results prove that our proposal has operability and feasibility to apply to practice.

Although this study has brought forward a novel logical architecture in managing things, however there are still scopes of this work that can be continued to improve in the near future, namely enabling fault-tolerance and stabilization functionalities, finding out the shortest path on multiple rings for thing discovery problem associating with cost evaluation instead of our exhaustive method. Besides, the energy cost for thing control also affects system performance, especially in IoT environment. Hence, an energy monitoring model for the architecture will be designed in our plan.

Acknowledgment

This research is supported by the Vietnamese MOETs project "Research and development of software framework to integrate IoT gateways for fog computing deployed on multi-cloud environment, the Slovak national VEGA 2/0167/16 project Methods and algorithms for the semantic processing of Big Data in distributed com- puting environment, and the EU Research and Innovation programme Horizon 2020 H2020-654142 EGI-Engage project Engaging the Research Community towards an Open Science Commons.

References

[1] T. D. Cao, H. H. Hoang, H. X. Huynh, B. M. Nguyen, T. V. Pham, Q. Tran-Minh, V. T. Tran, and H. L. Truong. Iot services for solving critical problems in vietnam: A research landscape and directions. *IEEE Internet Computing*, 20(5):76–81, Sept 2016.

[2] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

[3] Binh Nguyen, Viet Tran, and Ladislav Hluchy. Abstraction layer for development and deployment of cloud services. *Computer Science*, 13(3):79, 2012.

[4] R. Want. An introduction to rfid technology. *IEEE Pervasive Computing*, 5(1):25–33, Jan 2006.

[5] Nick Antonopoulos, James Salter, and Roger M. A. Peel. A multi-ring method for efficient multi-dimensional data lookup in p2p networks. In Hamid R. Arabnia and Rose Joshua, editors, *FCS*, pages 10–16. CSREA Press, 2005.

[6] Lai Wenyan and Liao Qing. A kind of p2p-based method of dynamic discovery of resources in iot. Nov 2012.

[7] P. Liu, N. Kong, Y. Tian, X. Lee, and B. Yan. A dht-based discovery service for rfid network. In *2014 IEEE International Conference on Internet of Things (iThings), and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom)*, pages 344–347, Sept 2014.

[8] S. Cirani, L. Davoli, G. Ferrari, R. Lone, P. Medagliani, M. Picone, and L. Veltri. A scalable and self-configuring architecture for service discovery in the internet of things. *IEEE Internet of Things Journal*, 1(5):508–521, Oct 2014.

[9] Kun Huang and Dafang Zhang. Dht-based lightweight broadcast algorithms in large-scale computing infrastructures. *Future Gener. Comput. Syst.*, 26(3):291–303, March 2010.

[10] Zhiyong Xu, Rui Min, and Yiming Hu. Hieras: a dht based hierarchical p2p routing algorithm. In *2003 International Conference on Parallel Processing, 2003. Proceedings.*, pages 187–194, Oct 2003.

[11] M. Cai, M. Frank, J. Chen, and P. Szekely. Maan: a multi-attribute addressable network for grid information services. In *Proceedings. First Latin American Web Congress*, pages 184–191, Nov 2003.

[12] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 654–663, New York, NY, USA, 1997. ACM.

[13] D. Eastlake, 3rd and P. Jones. Us secure hash algorithm 1 (sha1), 2001.