View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Crossref

Hindawi
Mobile Information Systems
Volume 2017, Article ID 4789814, 14 pages
https://doi.org/10.1155/2017/4789814

Research Article

A Distributed Relation Detection Approach in the Internet of Things

Weiping Zhu,[1] Hongliang Lu,[2,3] Xiaohui Cui,[1] and Jiannong Cao[3]

[1]*International School of Software, Wuhan University, Wuhan, China*
[2]*Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha, China*
[3]*Department of Computing, Hong Kong Polytechnic University, Kowloon, Hong Kong*

Correspondence should be addressed to Weiping Zhu; cswpzhu@gmail.com

Received 22 May 2017; Revised 6 August 2017; Accepted 16 August 2017; Published 28 September 2017

Academic Editor: Floriano Scioscia

Copyright © 2017 Weiping Zhu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In the Internet of Things, it is important to detect the various relations among objects for mining useful knowledge. Existing works on relation detection are based on centralized processing, which is not suitable for the Internet of Things owing to the unavailability of a server, one-point failure, computation bottleneck, and moving of objects. In this paper, we propose a distributed approach to detect relations among objects. We first build a system model for this problem that supports generic forms of relations and both physical time and logical time. Based on this, we design the Distributed Relation Detection Approach (DRDA), which utilizes a distributed spanning tree to detect relations using in-network processing. DRDA can coordinate the distributed tree-building process of objects and automatically change the depth of the routing tree to a proper value. Optimization among multiple relation detection tasks is also considered. Extensive simulations were performed and the results show that the proposed approach outperforms existing approaches in terms of the energy consumption.

1. Introduction

With the progress of wireless communication, mobile computing, and smart sensing and controlling, it is now possible for objects in our daily life to exchange information and form a network; this is called the Internet of Things (IOT) [1]. The concept of IOT is widely used in many fields such as intelligent transportation, smart home/office, and mobile medical healthcare [2, 3].

In the IOT, there are numerous kinds of relations among objects. The examples include that a key resides near a television and that two objects are used by users consecutively. More complex relations may involve multiple objects distributed in different physical places. These relations can be used to search required objects for users or recommend objects of interest to users [4–6].

In order to detect these relations, we need to gather the data from relative objects together. Existing works that utilize relations in the IOT [4–7] assume that a central server exists and all the data are collected to it for processing. This may not be feasible since (1) in many scenarios there is no such central server; (2) if an object is predefined as the central server, it could suffer from one-point failure, computation bottleneck, and difficulties when moving away from other objects; (3) transmitting raw data to a central server for processing could incur much energy consumption. Therefore, it is highly demanded that the system can adaptively determine a routing structure (usually a tree) along which data is gathered, relation is detected using in-network processing, and final result is stored. This processing should be done in a distributed manner; that is, no global coordinator exists in the system. Moreover, if multiple relations need to be detected, the processing of them should be shared for saving energy.

In this paper, we propose a distributed approach to gather data needed for a relation and detect the relation. We first build a system model for this problem following the custom of distributed computing. Then, we propose a three-layer solution framework for solving the problem. Based

on this, an algorithm called Distributed Relation Detection Approach (DRDA) is presented. The algorithm is to build a depth autooptimized breadth-first spanning tree and detect the relation based on it. The optimization among different relation detection processes is also considered. Extensive simulations are performed to validate the effectiveness of the proposed approach. The results demonstrate that the proposed approach is effective and outperforms existing approaches. In summary, this paper makes the following contributions.

(i) A system model is built for distributed relation detection in the IOT.

(ii) A distributed approach is proposed to build a routing tree for detecting relations. Multiple objects may trigger the routing tree building concurrently but a single tree is formed. This tree can automatically optimize its depth to a proper value. The optimization among different relation detection processes is also considered.

(iii) Extensive simulations are conducted to validate the effectiveness of the proposed approach. The evaluation results show that the proposed approach is effective and outperforms the existing approaches in terms of the energy consumption.

The remainder of this paper is organized as follows: Section 2 describes the system models used in this paper and formulates the problem. Then, the Distributed Relation Detection Approach is illustrated in Section 3. The simulation results of validating the proposed approach are reported in Section 4. Section 5 reviews the related works and finally Section 6 concludes the paper.

This paper is based on our conference paper [8]. In this version, we extend the approach to automatically change the depth of the routing tree based on breadth-first traversal and optimize the process among multiple relation detection tasks to further reduce the energy consumption. Additional discussion regarding relation detection under different time modalities is also included in this paper.

2. System Model and the Problem

This section describes the system model used in this paper. We do not assume a central server existing in the system.

In the IOT, there are multiple objects distributed in different physical places. We assume that a subset of the objects have data related to a specific relation detection. These data need to be gathered to be analyzed. Usually we build a routing tree for the IOT to facilitate the gathering. The routing tree includes all the objects that are directly involved in the relation, and some other objects that help to forward the data. All these objects know the definition of the relation and can trigger the routing tree building. However, a single tree should be formed eventually. We assume that each relation is defined in a restricted area and hence the message transmitted in the tree building process is also restricted. After the routing tree is built, the relation is detected using in-network processing based on the routing tree. There are multiple relations needed to be detected in the IOT, and hence multiple routing trees are needed. In many cases, the routing trees for different relations can be reused if possible. There are various metrics to measure how effective the routing tree is, including energy efficiency, latency, overhead, and adaptiveness. In this paper, we aim to minimize the energy consumption considering that objects in the IOT usually have limited energy capability.

The relations among these objects are based on their attributes such as ID, place, and sensing data. An attribute can be denoted by its name, its value, and the time interval when the value holds, in the form of *(attributeName*, *attributeValue*, *startTime*, *endTime)*. We call such an attribute an *interval-based attribute*. An interval-based attribute of an object is based on two *events*, each of which describes a change of the attribute. An event is denoted by *(attributeName*, *attributeValue*, *timestamp)* where *timestamp* describes when the attribute changes into the value of *attributeValue*. Because the value of an attribute changes with time, we have a sequence of events ordered by their timestamps. Except for the first event, each occurrence of an event completes the time interval of the old attribute value and starts a time interval of the new attribute value. For example, when a smart phone moves, it may have the events *(location*, *office A*, *8:00 AM)* and *(location*, *office B*, *8:30 AM)*. The latter completes the interval-based attribute *(location*, *office A*, *8:00 AM*, *8:30 AM)*, and generates a new interval-based attribute that starts from *8:30 AM* and has not yet completed. An example of the process is shown in Figure 1.

A relation is a function defined on the attributes of objects. They are typically described by predicates. A simple predicate can be directly determined by the attributes of an object. For example, $\text{location}_i = A$ is such a predicate denoting that the location of object i is at A. More complex predicates can be classified into *conjunctive predicates* and *relational predicates* [9–11]. Conjunctive predicates are in the form of $P_1 \wedge P_2 \wedge \cdots$, where P_i $(i = 1, 2, \ldots)$ is a simple predicate of object i. Relational predicates support arbitrary complex operations among simple predicates. For example, a relational predicate can be used to represent the case $P_1 + P_2 = 200$ using the plus operation. This paper is based on relational predicates, which means that a wider range of scenarios can be supported [12].

As a distributed system, IOT can be investigated based on *physical time* or *logical time*. When considering physical time, we assume that each object has a physical clock and the clock is well synchronized with those of other objects. The sequence of event occurrences in the system can be determined by measuring the timestamps of physical clocks. For logical time, an object does not have a physical clock, and the sequence of event occurrences is determined by the exchange of messages among objects. If there is a message sent from object P_1 to object P_2, it is certain that the message sending event of P_1 occurs before the message receiving event of P_2. Applying the transitive rule, the sequences of more events can be determined.

For physical time, the most common practice is to infer the relations under the *instantaneously* modality [13]. It denotes that the conditions of a relation are satisfied at a time point of physical time. This modality is difficult to determine

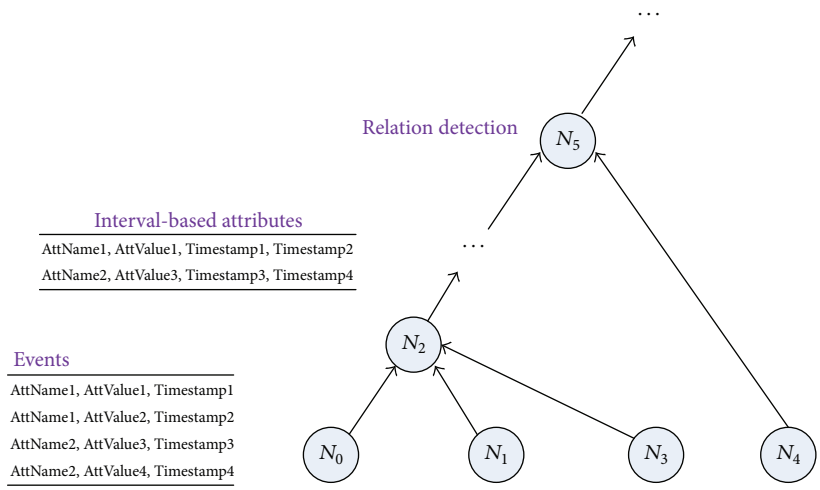


Figure 1: Example of relation detection process.

under logical time. Due to the loose restriction of logical time, corresponding sequences of events may have multiple possibilities in reality [9, 11]. A conservative practice is to employ *definitely* modality [14], which denotes that the conditions of a relation are satisfied in any of the possibilities. We use the pairwise overlapping of time intervals to implement the check under *definitely* modality. Given a list of objects involved in a relation, this modality determines a set of time intervals, one time interval corresponding to one object, such that the time intervals are pairwise overlapped and make the relation true. Two time intervals I_1 and I_2 are said to be overlapped [15] if they satisfy

$$I_1.\text{startTime} < I_2.\text{endTime} \wedge I_2.\text{startTime} < I_1.\text{endTime}. \quad (1)$$

It can be inferred that if such a set of time intervals are found, in reality there exists a time point that the specified relation is true. This check also applies to the *instantaneously* modality if physical time is used.

Our problem can be described as follows: given the IOT including multiple objects and a collection of relation detection tasks, the problem is to build proper routing trees for these tasks in a distributed manner and determine the processing in the trees such that the overall energy consumption for detecting these relations is minimized.

We believe that the problem investigated in this paper applies to many IOT applications. For example, in a smart office, smart chairs can sense the activities of people who sat on them, and their data are collected to infer whether a meeting starts. Concurrently, multiple RFID readers can be used to sense the people nearby and infer the distribution of people in the office. And the working computers can be monitored to compute the workload of electrical circuits. These relations can be detected based on the routing trees built among the smart devices. Subsequently, further actions can be performed, for example, turning on a projector, adjusting an air conditioner, and turning off a computer.

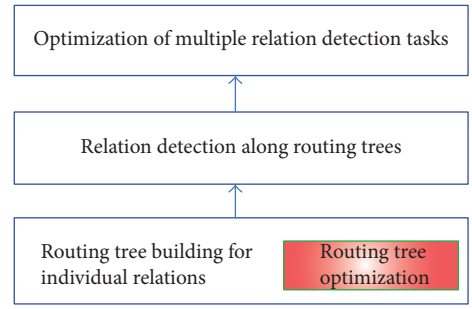


Figure 2: Solution framework of distributed relation detection.

3. Proposed Solution

In this section, we propose a solution for detecting relations in the IOT. We first briefly discuss the basic idea of the approach and then illustrate the details. Finally, we further discuss the proposed approach.

3.1. Framework Architecture. We propose a three-layer framework for solving the distributed relation detection problem discussed in this paper. As shown in Figure 2, the framework includes routing tree building for individual relations (the first layer), relation detection based on routing trees (the second layer), and optimization of multiple relation detection tasks (the third layer).

In the first layer, we build routing trees among the objects. The trees are based on individual relations; hence, one routing tree corresponds to one relation. Multiple objects may start the routing tree building process concurrently and proper coordination among these objects ensures a single routing tree is achieved. The routing trees are optimized in this layer considering two factors: the relation to be detected and the topology of the IOT. In the second layer, the events generated by the objects are forwarded along the routing tree, and the relation is checked during the forwarding. In the third layer, the routing trees are further optimized considering multiple relation detection tasks. Rather than building each

routing tree independently, it is possible to reuse some parts of the routing trees and corresponding processing to achieve minimal overall energy consumption.

3.2. Design Rationale. In existing relation detection approaches, every object sends its events to a server. The server generates interval-based attributes, extracts a set of interval-based attributes whose time intervals are pairwisely overlapped, and determines whether they satisfy the conditions of the relation. In our problem, such a server is not available. Although a fixed object can be selected as the server to solve this problem, but this could bring in more problems including one-point failure, computation bottleneck, and difficulties when moving away from other objects. Instead, we build a routing tree dynamically based on characteristics of objects on site. The relation can be detected when collecting related data along the tree by using in-network processing, and the result is stored in the root of the tree. There are two issues that needed to be solved in this approach. One is that an object needs to be selected as the root of the routing tree. The other is that proper coordination is required when multiple objects trigger the building of the routing tree in a distributed manner.

For the first issue, we need to guarantee the uniqueness and the optimization of the root. The simplest algorithm is to select a particular object as the root. However, this may lead to a routing tree with a large depth, along which a lot of data are required to transmit. We should adapt the root of the tree during the tree building to keep a proper depth. For the second issue, when the tree-building processes started by different objects conflict with each other, a proper method is required to terminate some of the buildings and allow others to continue.

When the detection of individual relations is solved, we consider the scenario where multiple relations need to be detected. For each relation, a routing tree can be built using the approach described above. The optimization among multiple relation detection tasks can be performed by reusing parts of their routing trees and corresponding processing. An example can be shown in the Figure 3, relation Φ_1 includes objects N_0, N_1, and N_2; relation Φ_2 includes the objects from N_0 to N_4; hence the routing tree built for Φ_1 can be reused for Φ_2. Reuse of a routing tree can save the detection cost of a relation; however, it may also introduce additional transmitting data and distance. We will consider this in the detailed algorithm design.

3.3. Data Structure. Each object keeps several variables for the execution of our algorithm. The variable *events* stores the generated events. Q_i ($i = 1, \ldots, n$) is a queue that stores interval-based attributes gathered from child object i ($i = 1, \ldots, n$) in the routing tree. *parent* and *children* record the parent object and children objects, respectively. *root* records the root of the routing tree. *depth*[*rID*] denotes the depth of the current object with regard to relation *rID*. To facilitate the process of tree-building, *neighbors* is used to record all the objects that an object can communicate with, *toSend* is used to record all the objects in *neighbors* that requires to be visited, and *visited* is used to record all the objects in *neighbors* that have been visited. *RS* records the IDs of relations in which the current object has been involved. The notations used in this paper are summarized in Notations.

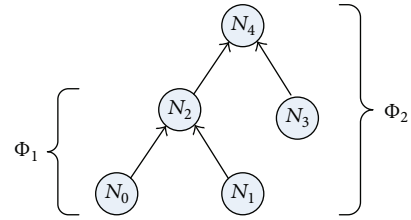


Figure 3: Routing trees for detecting multiple relations.

3.4. Detailed Algorithms. We propose Distributed Relation Detection Approach (DRDA) to detect relations in the IOT. This approach includes three steps. The first step is building a routing tree for each relation detection task. The second step is gathering information from the objects and checking the relations. The third step is optimizing the process for multiple relation detection tasks. The details are given in Algorithms 1, 2, and 3. Notably, DRDA extends our previous approach proposed in [8] by using breadth-first traversal strategy and enabling automatic change of the depth of the routing tree (in Algorithm 1), supporting more time modalities (in Algorithm 2), and optimizing the tree building process for multiple relation detection tasks (in Algorithm 3).

Algorithm 1 is responsible for building a routing tree for each relation in a distributed manner. This algorithm is based on the distributed breadth-first traversal approach [16]. In that work, the system starts from node A to traverse a graph following breadth-first manner and simultaneously builds a spanning tree rooted at A. Our problem has different requirements compared with that work. First, the root is not determined in advance; rather, it is determined in the tree-building process. Second, the objects involved in a relation may start routing tree building concurrently, but a single routing tree should be formed eventually. Compared with our previous work [8], we change the tree-building strategy from depth-first to breadth-first, because we observed that the breadth-first tree has a less depth and more compact structure comparing with the depth-first tree.

As shown in function *buildRoutingTree*, each object initiates the tree-building process by regarding itself to be the root of the tree (i.e., *parent=ID*) and then sending a tree-building message "go" to all its neighbors. The ID of the relation to be detected is added to *RS*, and *toSend* is initialized to its neighbors. Then, the objects respond to different messages received, as shown in function *receiveMsg*. There are five types of messages: "go," "continue," "stop," "no," and "reverse."

The processing of "go" messages is depicted in lines 1–24. First, current object is verified whether it has already received a "go" message from another initiator regarding the current relation (line 2), where the received message is recorded in *RS*. If it is true, there exist trees initialed by two objects, and they require to be merged. We merge the tree with a smaller depth to the tree with a greater depth because later reverse operations can be minimized. A "reverse" message is sent to

```
    Function: buildRoutingTree(String rID)
(1) parent = ID, RS = RS ∪ {rID}, toSend = neighbors
(2) send Msg("go", ID, rID, 0) to each object in toSend
    Function: receiveMsg(Msg msg, Node from)
(1) if msg.type="go" then
(2)     if msg.rID ∈ RS and msg.root ≠ root then
(3)         if msg.depth + depth[msg.rID] > threshold then
(4)             send Msg("reverse",ID,msg.rID,0) to parent and from
(5)         else if msg.depth > depth[msg.rID] then
(6)             send Msg("reverse", msg.root, msg.rID, msg.depth+1) to parent
(7)         else
(8)             send Msg("reverse", root, msg.rID, depth[msg.rID]) to from
(9)         end
(10)    end
(11)    if parent=∅ then
(12)        parent = from, root = msg.root, toSend = neighbors \ {from}
(13)        RS = RS ∪ {msg.rID}, depth[msg.rID]=msg.depth+1,
(14)        if (toSend ≠ ∅) then
(15)            send Msg("continue", root, msg.rID, ⊥) to from
(16)        else
(17)            send Msg("stop", root, msg.rID, ⊥) to from
(18)        end
(19)    else if parent=from then
(20)        send Msg("go", root, msg.rID, msg.depth+1) to each object in toSend
(21)        visited=∅, root = msg.root, depth[msg.rID]=msg.depth+1
(22)    else
(23)        send Msg("no", root, msg.rID, ⊥) to from
(24)    end
(25) else if msg.type="continue", "stop" or "no" then
(26)    visited = visited ∪ {from}
(27)    if msg.type="continue" or "stop" then
(28)        children = children ∪ {from}
(29)    end
(30)    if msg.type="stop" or "no" then
(31)        toSend = toSend \ {from}
(32)    end
(33)    if toSend=∅ then
(34)        if parent=ID then
(35)            trigger the relation detection process
(36)        else
(37)            send Msg("stop", root, msg.rID, ⊥) to parent
(38)        end
(39)    else if visited=toSend then
(40)        if parent=ID then
(41)            send Msg("go", ID, msg.rID, 0) to each object in toSend
(42)            visited=∅
(43)        else
(44)            send Msg("continue", root, msg.rID, ⊥) to from
(45)        end
(46)    end
(47) else if msg.type="reverse" then
(48)    send Msg("revserse", msg.root, msg.rID, msg.depth+1) to parent
(49)    parent = from, root=msg.root, depth[msg.rID] = msg.depth+1
(50)    modify children in parent and current node
(51) end
```

Algorithm 1: Distributed routing tree building algorithm.

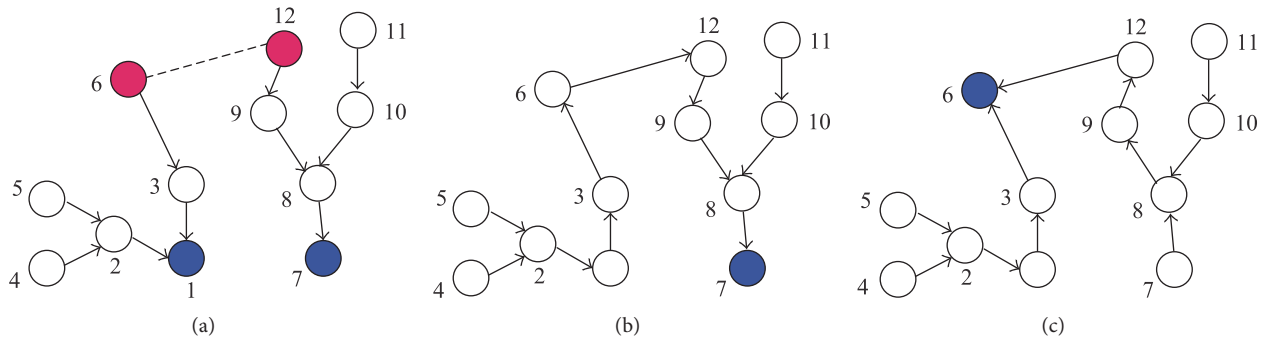


FIGURE 4: Automatic depth change in a routing tree (a) object 6 receives "go" message first from object 3 and then object 12; (b) the tree rooted at object 1 merges into the tree rooted at object 7; (c) object 6 is elected as the root of the result tree and the objects in previous two trees reverse their attachment.

the tree whose depth is smaller in order to reverse its routing direction to the current object (lines 5–9).

Following this execution, the depth of routing tree may be quite large. An example of this can be seen in Figure 4. As shown in Figure 4(a), there are two routing trees initiated by object 1 and object 7. After object 6 receives "go" message from both object 3 and object 12, it requests the tree rooted at object 1 to merge into the tree rooted at object 7, because the depth of the latter is 4 and the depth of the former is 2. Object 6 sends a "reverse" message to its parent, object 3, and then to object 1, to implement the mergence. The merged routing tree is shown in Figure 4(b). The problem is that the depth of the final tree is the sum of the previous two trees, which will further increase when more routing trees merge into it. A large tree depth causes significant energy consumption and long latency. To solve this problem, we design an automatic depth change strategy. The sum of the depths of two routing trees is compared with a threshold. If it is greater than the threshold, the current object is elected as the root of the final tree and "reverse" messages are sent to the two trees (e.g., both object 3 and object 12). The result of this process can be seen in Figure 4(c). Lines 3-4 of function *receiveMsg* show the implementation of the automatic depth change strategy.

If the traversal does not conflict with that of other initiators, the processing is similar to the traditional breadth-first graph traversal. The algorithm first checks whether the current object has been visited (line 11). If the object has not been visited previously ($parent = \varnothing$), the object sets its parent to the sending object *from*, records the root object, and initializes *toSend* to the objects required to forward tree-building messages (line 12). The object also updates *RS* to include the processing relation and updates its depth in the tree (line 13). Next, it checks whether there are neighboring objects that have not yet been visited (line 14). If this is true, the object returns a "continue" message to its parent to denote that this is a child node and some of its neighbors are required to be visited (line 15). Otherwise, the object returns a "stop" message to its parent to denote that this is a child node and all its neighbors have been visited (line 17). If the object has been already visited previously, there are two cases to process the received message. If the message is sent from the parent, it is a message required to be forwarded to the objects having not been visited; thus a "go" message is sent to the objects in *toSend* (lines 20-21). Otherwise, a "no" message is returned to the object *from* (line 23).

The processing of "continue," "stop," and "no" messages is shown in lines 25–46. First, the message is added to *visited* (line 26). If the type of the message is "continue" or "stop," it is known that the sending object has not been visited previously and hence should be added to *children* (lines 27–29). If the type of the message is "stop" or "no," it is known that all the neighbors of the sending object have been visited or the sending object is not a child object; then the sending object can be removed from *toSend* (lines 30–32). Subsequently, the object is checked to determine whether all the neighbors have been visited. If it is not true and all the feedback messages are received (line 39), a "continue" message is sent back to its parent (line 44), and eventually the root will initialize another "go" message to continue the breadth-first traverse (lines 41-42). If all the neighbors have been visited, a "stop" message can be sent back to its parent (line 37) and eventually the root knows that the routing tree is built and then triggers the processing of relation detection (line 35).

Finally, "reverse" messages are processed in lines 47–51. The parent is changed to *from*, the recorded root is updated by the message, and the depth of the object in the tree is changed to *msg.depth*+1 (line 49). *children* also requires modification (line 50) and its trivial operations are omitted here. The "reverse" message is further sent to its previous parent to conduct the same operations (line 48).

Algorithm 2 is responsible for relation detection based on the routing tree. Notably, this algorithm holds in the *definitely* modality if logical time is used and the *instantaneously* modality if physical time is used. We assume that each object generates the required time intervals and sends these to the parent for relation detection. Function *sendAttributeInterval* is invoked when a local event occurs (i.e., the value of an attribute changes). The purpose is to generate time intervals for further processing. It first checks whether the changed attribute belongs to the relation (line 1) and then further checks whether it is the first time to generate this time interval (line 2). If this is the case, a time interval is composed that includes the value of the attribute, the start time, and the end time. The start time is set to current timestamp t, and the end

```
    Function: sendAttributeInterval(String attributeID, Data value, Relation r, Timestamp t)
(1) if attributeID ∈ r.attributes then
(2)     if events.get(attributeID) ≠ ∅ then
(3)         Interval interval = events.get(attributeID)
(4)         interval.endTime = t
(5)         send Info(r.ID, attributeID, interval)
(6)     end
(7)     Interval interval = (value, t, ⊥)
(8)     events.get(attributeID).add(interval)
(9) end
    Function: receiveInterval(Interval I_m, NodeID m, Relation r)
(1) Q_m.enqueue(I_m)
(2) if Q_m.size=1 then
(3)     updatedQueue.add(m)
(4) end
(5) while updatedQueues ≠ ∅ do
(6)     foreach i ∈ updatedQueues do
(7)         if Q_i ≠ ∅ then
(8)             X = Q_i.head
(9)             for j = 1 to n do
(10)                Y = Q_j.head
(11)                if X.startTime ≥ Y.endTime then
(12)                    newUpdatedQueues.add(j)
(13)                end
(14)                if Y.startTime ≥ X.endTime then
(15)                    newUpdatedQueues.add(i)
(16)                end
(17)            end
(18)        end
(19)    endfch
(20)    foreach i ∈ newUpdatedQueues do
(21)        Q_i.dequeue();
(22)    endfch
(23)    updatedQueues = newUpdatedQueues
(24) end
(25) if Q_i (i = 1, ..., n) ≠ ∅ then
(26)    if r exists in Q_i (i = 1, ..., n) then
(27)        if parent ! = ID then
(28)            send all Q (i = 1, ..., n) to parent
(29)        else
(30)            relation is detected and recorded
(31)        end
(32)    end
(33) end
```

Algorithm 2: Relation detection algorithm.

time is set to $\perp$ denoting that it is not determined (line 7). This time interval is added to *events* (line 8). Subsequently, when another local event occurs, function *sendAttributeInterval* is invoked again with the new value. The time interval previously stored in *events* is completed by setting its end time to the timestamp t (lines 3-4). Then, the time interval is sent to its parent in the routing tree (line 5).

The relation detection process is illustrated in function *receiveInterval*. This function is based on [15]. It first determines a collection of pairwise overlapped time intervals, one

```
Function: reuseRoutingTree(Relation r, Set rSet)
(1) sharedTree = ∅
(2) foreach i ∈ rSet do
(3)     find the max subtree s in i's routing tree that matched r
(4)     if ‖s‖ > sharedTree then
(5)         sharedTree = s
(6)     end
(7) endfch
(8) get the node x that is the root of sharedTree
(9) find the node y in r that connects to sharedTree
(10) calculate the energy consumption E1 when transmitting the result of sharedTree from x to y
(11) calculate the energy consumption E2 when using an independent routing tree
(12) if E1 < E2 then
(13)     reuse sharedTree for r
(14) else
(15)     construct an independent routing tree for r
(16) end
```

Algorithm 3: Routing tree reuse algorithm.

time interval corresponding to one object (lines 1–24). The details are as follows. When a time interval I_m is received, it is placed into the corresponding queue and recorded in *updatedQueue* (lines 1–4). Then, the algorithm attempts to make all the heads of Q_i ($i = 1, \ldots, n$) pairwise overlapped. The condition for pairwise overlapping is presented in Section 2. This is checked in lines 11–16. For a pair of time intervals that do not satisfy the condition, the time interval with the smaller start time is deleted (placed into *newUpdatedQueues*) and replaced by the following time interval. Finally, the pairwise overlapped time intervals are obtained and checked with the relation (lines 25-26). If the relation exists based on the current information, it is further sent to its parent object (line 28) and finally the decision is made at the root of the routing tree (line 30).

Algorithm 3 is responsible for reusing routing trees among different relation detection tasks. In this paper, we aim to minimize the energy consumption by reusing the routing trees. Suppose that we want to build a routing tree for relation r. The variable *sharedTree* is used to record the tree that can possibly be shared with previous built routing trees. In lines 2–7, the algorithm checks each relation and determines the maximal subtree s that is common between it and r. When s is determined, its cardinality is compared with previously determined shared tree *sharedTree* and the one with the greater value is retained. In lines 8–16, the algorithm compares the costs when using and not using the shared tree. If *sharedTree* is used, the result in this subtree can be generated only once, and hence no additional cost is required. However, it requires additional energy E_1 to transmit the result from the root of *sharedTree* to the node linking to *sharedTree* in r. If *sharedTree* is not used, it requires to build a new routing tree with the energy consumption of E_2. If E_1 is less than E_2, we reuse *sharedTree*; otherwise an independent routing tree is built.

3.5. Discussion. In this section, we further discuss issues that require additional considerations in the algorithms.

The first issue regards the detection of multiple relations. In practice, an object can be involved in multiple relation detection tasks. These tasks may require different sensing data of the object. In order to distinguish these tasks, we should use different variables (such as *parent*, *children*, *visited*, and Q_i) for each task.

The second issue is the detection under logical time. Compared with physical time, logical time is more complex owing to uncertainty. We use Figure 5 to illustrate the difference between physical time and logical time. The sequence of event e_1^1 of object P_1 and event e_2^1 of object P_2 can be determined by their timestamps directly under physical time. Conversely, the temporal relations under logical time can only be determined by message passing. If there is a message from e_1^1 to e_2^2, it is certain that e_1^1 occurs before e_2^2. However, the relation between e_1^1 of and e_2^1 cannot be determined. It means two possibilities exist in reality, e_1^1 before e_2^1 or e_2^1 before e_1^1. Relation detection under logical time is usually based on two modalities, *possibly* or *definitely*. *possibly* modality denotes that a relation holds in one of the possibilities, and *definitely* modality denotes that a relation holds in each of the possibilities. *possibly* or *definitely* can be further extended to *occurrence probability* [17], which measures the probability that a relation holds in reality. In this paper, relation detection is based on the *definitely* modality. The relation detection under *possibly* modality or *occurrence probability* is left as future work.

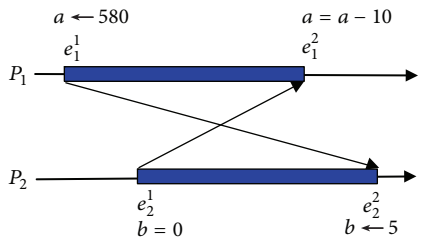


FIGURE 5: Temporal relations under logical clock.

The last issue is the moving of objects in the IOT. When some objects move away, the routing trees including them need to be revised. There are two kinds of approaches for the rebuilding of routing trees. One is to detect the moving of objects and revise the routing tree in real time. The other is to periodically perform the detection and revision. The data being transmitted may be lost due to the moving of objects, and hence some message acknowledgement and retransmission strategy need to be added to the proposed approach.

4. Performance Evaluation

Simulations were performed to validate the effectiveness of the proposed approach. We compared the energy consumption of DRDA with that of the centralized approach proposed by Raychoudhury et al. in [4] (denoted as the centralized approach) and the approach proposed in our previous work [8] (denoted as DRDA-D). Energy consumption is measured by the number of packages transmitted during the relation detection.

We conducted the simulations in a region of 40 m × 40 m where 200 objects were randomly scattered. Two objects can communicate with each other if a link between them exists. The links between objects follow the commonly used Waxman model [18] with an additional constraint of communication range. Given two objects u and v, a link between them exists according to the probability, $P\{(u, v)\}$:

$$P\{(u,v)\} = \begin{cases} \beta \times \exp \dfrac{-d(u,v)}{L \times \alpha} & (d(u,v) \le r) \\ 0 & (d(u,v) > r), \end{cases} \quad (2)$$

where β denotes the density of links, α denotes the ratio of short links to long links, L denotes the maximum distance between two objects, $d(u, v)$ denotes the distance between objects u and v, and r denotes the communication range of the objects. The values of α and β are between 0 and 1. There are several relations (specified by predicates) required to be detected, and each predicate has the length of l. In the simulations, we let $\alpha = 0.2$, $\beta = 0.8$, $r = 8$, and $l = 10$, unless otherwise specified. We assume that the routing table of each object has been established and the amount of data to be transmitted does not include those used for routing discovery. We executed each simulation 100 times and then calculated the average value.

4.1. Impact of the Length of Predicates. The length of a predicate is correlated to the number of objects required for the relation detection. We varied the length of predicates from 5 to 19 and check its effect on the number of packages transmitted in the relation detection. The result is shown in Figure 6.

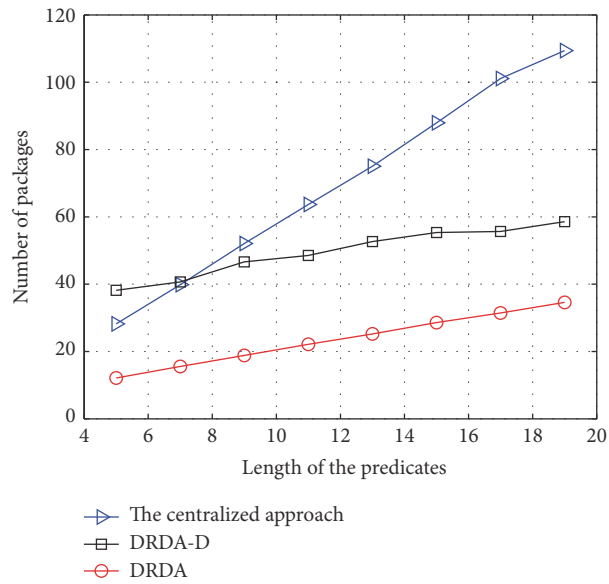


FIGURE 6: Number of packages for relation detection with different lengths of predicates.

It can be observed that the number of packages required for all the three approaches tends to increase when the length of predicates increases. This is because an increment of the length of predicates indicates an increment of the number of objects related to the predicate. A relation detecting task must obtain all the related information of all such objects. Hence, more packages are required to be transmitted. The number of packages transmitted in the centralized approach increases approximately linearly with the length of predicates. The number of packages required for DRDA-D is more than that required for the centralized approach when the length of predicates is less than 7 and less than that required for the centralized approach when the length of predicates is greater than 7. This illustrates that the in-network processing is more evident when there are a sufficient number of objects in the routing tree. DRDA further reduces the number of data transmission compared with DRDA-D and achieves the best performance in all lengths of predicates. This can be explained by the fact that the breadth-first traversal and the adaptiveness of the depth of the routing tree result in more compact routing trees and more in-network processing. When the length of predicates is 19, the number of packages of DRDA is 31.6% that of the centralized approach and 59.0% that of DRDA-D.

4.2. Impact of the Density of Links. The communication network among objects is controlled by three parameters, the density of links, the ratio of short links to long links, and the communication range. We first varied the density of links from 0.3 to 0.9 and compared the performance of the three approaches in relation detection. The result is shown in Figure 7.

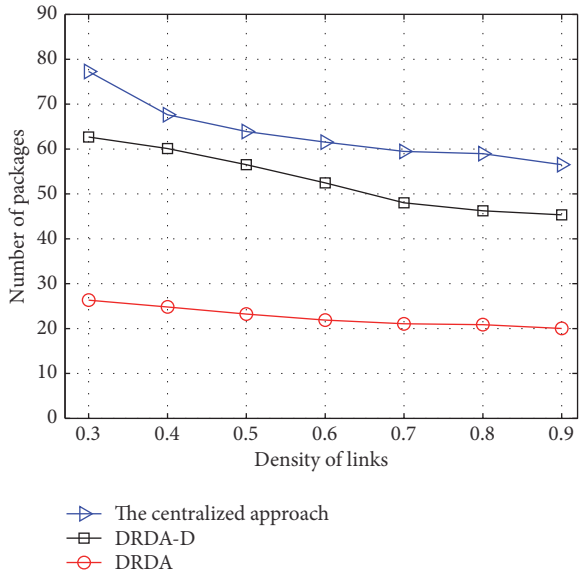


Figure 7: Number of packages for relation detection with different densities of links.

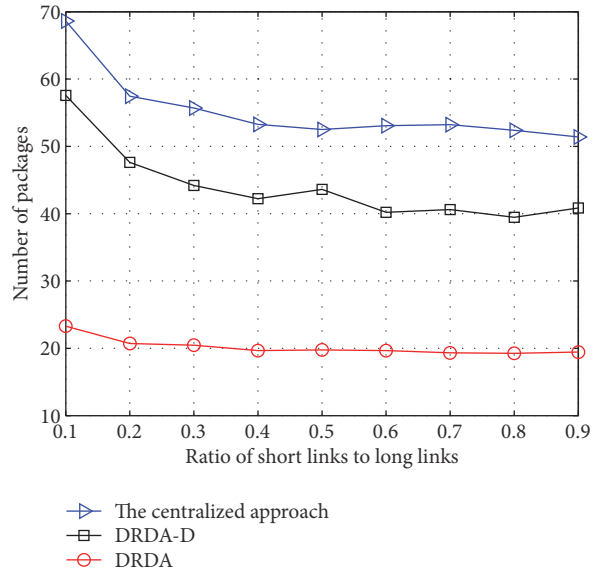


Figure 8: Number of packages for relation detection with different ratios of short links to long links.

When the density of links increases, the number of packages transmitted decreases. For the centralized approach, this is because more links provide more paths from an object to the root of the routing tree, and hence a shorter path can be used for data transmission. For DRDA-D and DRDA, more links not only reduce the hops between objects but also provide more opportunities for in-network processing. In all cases, DRDA outperforms the other two approaches in terms of the number of packages transmitted. DRDA is less sensitive to the change of the density of links, comparing with the other two approaches.

4.3. Impact of the Ratio of Short Links to Long Links. The ratio of short links to long links is another important parameter (denoted as α) influencing the communication network. We varied α from 0.1 to 0.9 and compared the performances of the three approaches in relation detection. The result is shown in Figure 8.

When α increases, the number of packages transmitted decreases. This is because the increment of α results in more long links, which reduces the number of transmissions between two nodes. The reduction is more evident when α is less than 0.3. However, when α is larger than 0.3, its impact on the number of packages transmitted is negligible. This is because the links are also controlled by the communication range. Although α increases, the length of links cannot be increased due to such constraint. According to the figure, the number of packages transmitted of the centralized approach is much more than that of the DRDA-D and DRDA. DRDA is least sensitive to the change of α among the three approaches.

4.4. Impact of the Communication Range. The communication range of an object is important for the wireless communications. One of the necessary conditions for two objects to communicate with each other is that the distance between them is less than the communication range. We varied the communication range from 10 m to 50 m and compared the performances of the three approaches in relation detection. The result is shown in Figure 9.

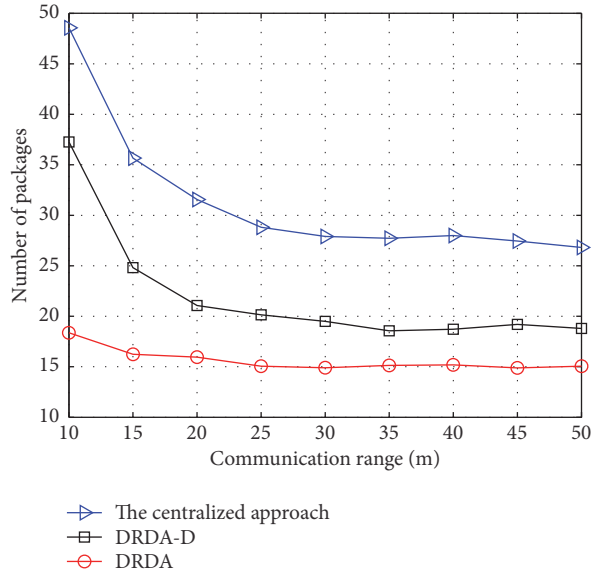


Figure 9: Number of packages for relation detection with different communication ranges.

It can be observed that when the communication range increases, the number of packages transmitted decreases. This is nature because less forwarding through intermediate objects is required. The reduction on the number of packages is significant when the communication range is between 10 m and 25 m and marginal when the communication range is more than 25 m. According to the figure, the number of packages transmitted of the centralized approach is more than that of DRDA-D and DRDA. DRDA requires a small

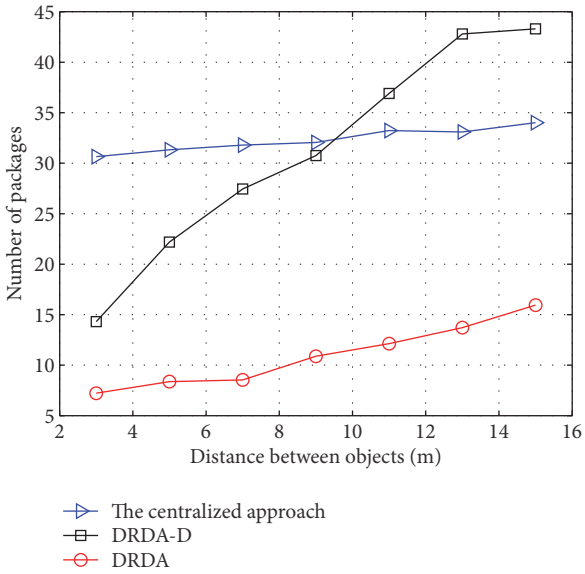


Figure 10: Number of packages for relation detection with different distances between objects.

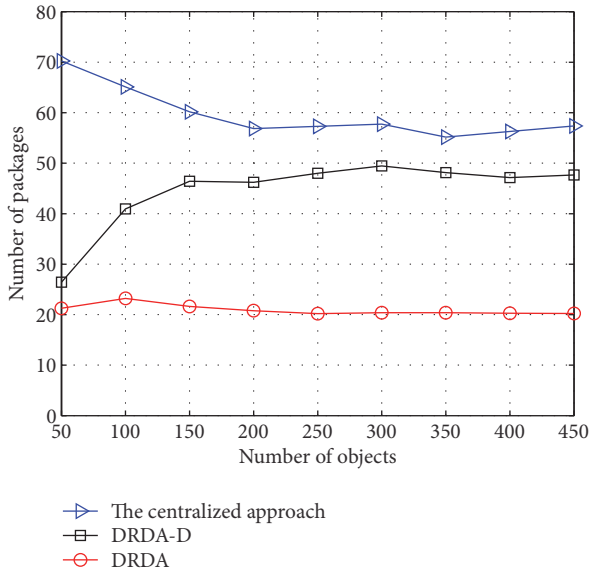


Figure 11: Number of packages for relation detection with different sizes of networks.

number of packages to be transmitted even though the communication range is small, and the performance is quite stable in different communication ranges.

4.5. Impact of the Distance between Objects. Multiple objects are usually involved in the relation detection, and the distance between objects affects the structure of the routing tree. We varied the distance between objects from 3 m to 15 m and checked its effect on the number of packages transmitted in relation detection. During this process, we set the length of predicates to 6. The result is shown in Figure 10.

According to the figure, the number of packages transmitted of the centralized approach is rarely affected by the distance between objects. This is because the centralized approach directly connects the objects to the central server, and the distance between objects is not a concern. The numbers of packages transmitted of DRDA-D and DRDA increase as the increment of the distance between objects. This can be explained by the fact that the objects related to the predicate transmit their data to an intermediate object and the transmission distance increases when the distance between objects increases. DRDA-D outperforms the centralized approach when the distance between objects is less than 9 m and underperforms the centralized approach when it is greater than 9 m. DRDA demonstrates superior performance in all cases compared with the centralized approach and DRDA-D.

4.6. Impact of the Number of Objects. We compared the performances of the three approaches in a network with different sizes. By varying the number of objects in the network from 50 to 450, we checked the number of packages transmitted for relation detection. The result is shown in Figure 11.

As the number of objects increases, the number of packages transmitted of the centralized approach gradually decreases from approximately 70 to 56 and maintains this level. This is because if there are only a small number of objects, the shortest paths are difficult to obtain, whereas if the number of objects is sufficiently large, the added objects do not affect the shortest paths. By contrast, as the number of objects increases, the number of packages transmitted of DRDA-D gradually increases from approximately 27 to 49 and maintains this level. This is because DRDA-D performs depth-first traversal and hence a routing tree with a large depth is built when the number of objects is limited. DRDA has the best performance, approximately 22 in different sizes of networks. It verifies that the breadth-first traversal used in DRDA is effective in constructing a compact routing tree.

4.7. Impact of the Reuse of Routing Trees. We further verify the effectiveness of the reuse of routing trees. According to the design of DRDA, when there are multiple predicates (i.e., relations) required to be detected, some parts of their routing trees can be shared. We consider a scenario of multiple predicates each of which involves ten objects. Nine of them are at the lower-left region, and the other one is at the upper-right region. Five of the objects are commonly required for all the predicates. We vary the number of predicates to check the performance of DRDA reusing routing trees and DRDA building the trees independently. The result is shown in Figure 12.

It can be observed that DRDA reusing routing trees requires fewer packages to be transmitted. This is because the transmission in the reused routing tree can be saved. The number of packages saved increases as the number of predicates increases. When there are 19 predicates, approximately 32.7% of packages can be saved, compared with building the routing trees independently.

4.8. Execution Time of DRDA. Although the major concern of this paper is the energy consumption, we keep the

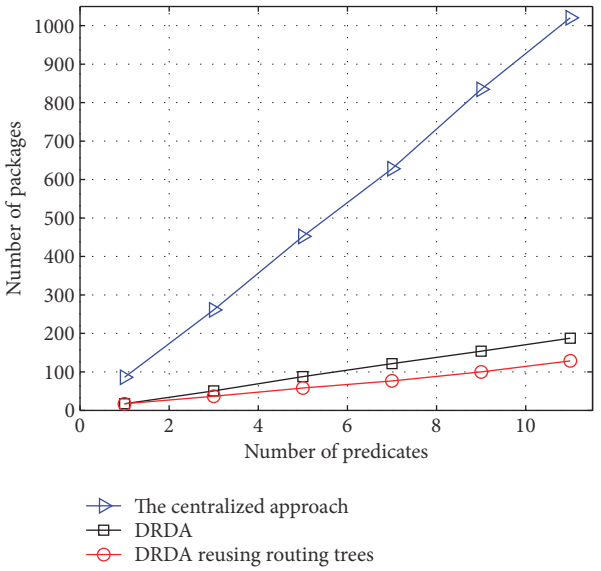


Figure 12: Number of packages for relation detection with different number of relations.

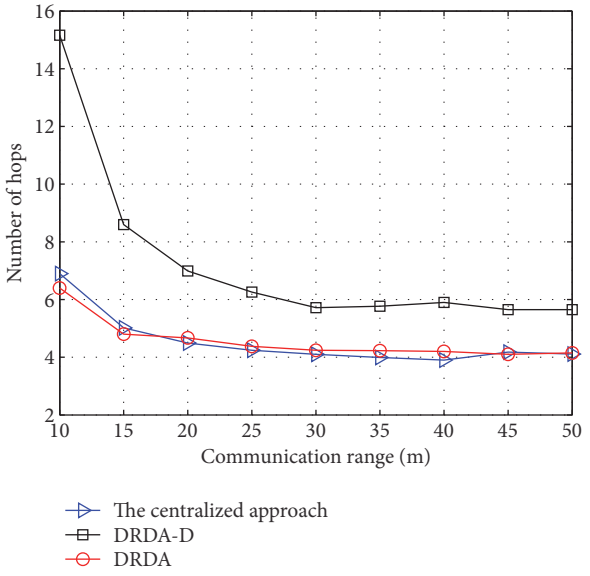


Figure 13: Number of hops for relation detection with different communication ranges.

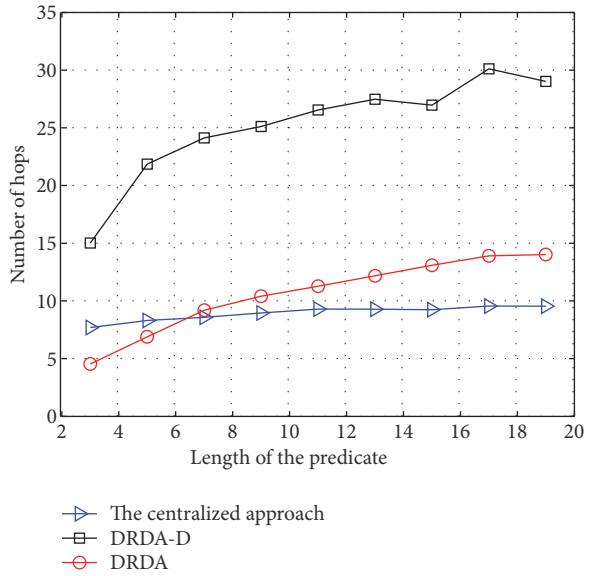


Figure 14: Number of hops for relation detection with different lengths of predicates.

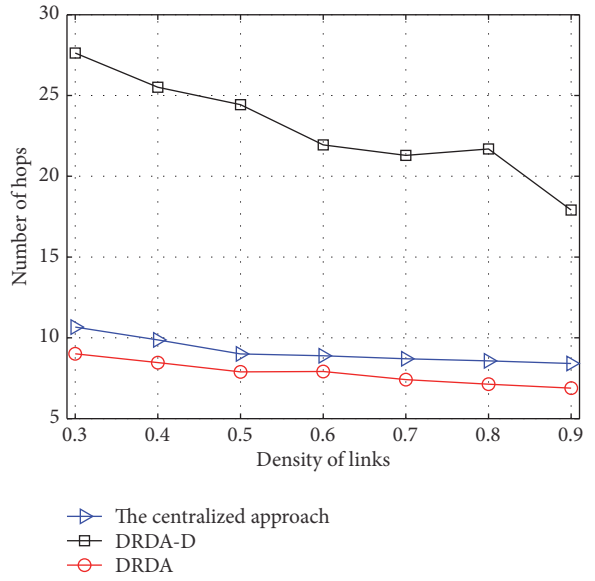


Figure 15: Number of hops for relation detection with different densities of links.

execution time as short as possible. We varied the communication range, the length of predicates, and the density of links and checked the execution time of the three approaches in relation detection. The other parameters have similar effects on the execution time and hence their results are omitted. The execution time is measured by the maximum number of hops required for an object to transmit its data to the root of the routing tree. The results are shown in Figures 13–15.

It can be observed that DRDA and the centralized approach have similar execution time in different settings of parameters. DRDA demonstrates less execution time when the communicating range is less than 20 m, the length of predicates is less than 7, and the density of links is greater than 0.3. In all settings, the execution time of DRDA is less than that of DRDA-D.

5. Related Works

In the literature, there are some works related to relation detection in the IOT. In wireless sensor networks and pervasive computing, extracting the relations among objects is important. In [19], Khelil et al. collect the data (e.g., temperature) from sensor nodes and compose a spatial and temporal map based on the data for further analysis. Similarly,

SENSID [20] is a middleware for wireless sensor networks to detect spatial and temporal event patterns specified by the user. In pervasive computing, Huang et al. check inconsistent context information based on concurrent event detection [7]. It assumes asynchronous communications, and the solution is based on vector clock and time interval analysis. Applications such as Snoogle [21], Microsearch [22], and FiMS [23] are designed to search physical objects based on their relations. Raychoudhury et al. further build a context map based on the relations in the IOT for navigation among objects [4]. Mietz et al. extend the search of physical objects by providing semantic Web-based [24] data storage format and the query language. Perera et al. [25] propose a model for sensor search based on users priorities and characteristics of sensors (e.g., reliability, accuracy, and battery life). Chen et al. consider the users' dynamic and changing context for object search [5]. Yao et al. [6] recommend things to users based on relations across heterogeneous entities of IOT including user-thing relations, user-user relations, and thing-thing relations. All these works assume that a central server exists and all data are collected in it for processing. This is not suitable for the IOT that includes a large number of distributed objects and supports the relation detection requirements that may change frequently.

There are also works for the detection of different kinds of predicates. Predicates are used to specify the relations of interest, based on the attributes of objects. Zhu et al. compose a research framework for predicate detection [17]. In that framework, predicate detection is categorized according to three metrics: predicate type, detection cardinality, and detection modality. Predicate type refers to the forms of predicates, including simple conjunctive, simple relational, simple sequence, and interval-constraint sequence. Simple conjunctive denotes a conjunctive expression of states of processes, and simple relational can specify arbitrary operations between states of processes [9, 11]. Simple sequence and interval-constraint sequence further specify the temporal sequence of states [26]. More detailed classifications may also consider the stability of predicates. A stable predicate is a predicate that remains true once it is true; an unstable predicate is a predicate that is true intermittently [27]. Detection cardinality includes single detection and repeated detection [28]. Detection modality includes physical time and logical time, denoting that a predicate is satisfied in terms of physical time and logical time, respectively. There are different detection approaches for different types of predicates. For the detection of simple relational and stable predicates based on physical time, we can get the snapshot of system [27] and analyze the results. For the detection of unstable predicates based on logical time, we need to consider multiple possible observations existing in the system [12]. *Definitely* and *possibly* modalities are introduced for such a detection [14]. Zhu et al. extend these to a more generic occurrence probability that can measure the probability that a predicate holds in different observations [17]. Kshemkalyani et al. specify the fine-grained relations of time intervals under logical time [13, 29]. The works above are useful for our research, especially for specifying relations. However, they cannot be directly used for solving our problem. In this paper, we utilize the ideas of unstable and simple relational predicate detection and adapt them to the relation detection in the IOT.

6. Conclusion

In this paper, we studied the relation detection problem in the IOT. Existing works on relation detection mainly focus on centralized processing, which suffers from problems including unavailability of a server, one-point failure, computation bottleneck, and moving away of objects. Realizing the drawbacks of the prior works, we proposed DRDA to achieve distributed relation detection in the IOT. DRDA supports generic forms of relations and both physical time and logical time modalities. For each task, a spanning tree is built in a distributed manner. Proper coordination among objects and automatic depth change of the tree are implemented. The optimization among multiple relation detection tasks is also considered. We performed extensive simulations to validate the effectiveness of the proposed approach. The results show that the proposed approach can reduce the amount of data to be transmitted.

Notations

ID:	ID of an object
rID:	ID of a relation
RS:	Relation IDs in which the current object has been involved
depth[*rID*]:	Depth of the current object in the routing tree regarding relation *rID*
parent:	ID of the parent of the current object in the routing tree
children:	IDs of the children of the current object in the routing tree
root:	ID of the root of the routing tree
neighbors:	IDs of the neighboring objects
toSend:	IDs of the objects to be visited in the process of the routing tree building
visited:	IDs of the objects already considered in the process of the routing tree building
Msg(Type, rootID, relationID, depth):	Message used in the routing tree building, containing message type, ID of the initiator, ID of relation, and depth from the initiator
r, $r.id$, $r.attributes$:	Relation, its ID, and its attributes
interval(AttributeValue, StartTime, EndTime):	Time interval with attribute value, start time, and end time
events:	Events record for different attributes
Info(RelationID, AttributeID, Interval):	Message used in relation detection, including ID of relation, ID of attribute, and time interval
Q_i ($i = 1, \ldots, n$):	Queue storing time intervals from child object i

updatedQueue, *newUpdatedQueue*:	Queues storing time intervals in relation detection.

Conflicts of Interest

The authors declare no conflicts of interest.

Acknowledgments

This research is supported in part by Outstanding Young Academic Talents Start-Up Funds of Wuhan University (no. 216-410100004), the Fundamental Research Funds for the Central Universities of China (no. 2042015kf0042), National Natural Science Foundation of China (no. 61502351), and Natural Science Foundation of Hubei, China (no. 2015CFB340).

References

[1] N. Gershenfeld, R. Krikorian, and D. Cohen, "The internet of things," *Scientific American*, vol. 291, no. 4, pp. 76–81, 2004.

[2] L. Atzori, A. Iera, and G. Morabito, "The internet of things: a survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[3] D. Miorandi, S. Sicari, F. de Pellegrini, and I. Chlamtac, "Internet of things: vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[4] V. Raychoudhury, J. Cao, W. Zhu, and A. Kshemkalyani, "Context map for navigating the physical world," in *Proceedings of the Euromicro International Conference on Parallel, Distributed & Network-based Processing (PDP)*, pp. 146–153, 2012.

[5] Y. Chen, J. Zhou, and M. Guo, "A context-aware search system for Internet of Things based on hierarchical context model," *Telecommunication Systems*, vol. 62, no. 1, pp. 77–91, 2016.

[6] L. Yao, Q. Z. Sheng, A. H. H. Ngu, and X. Li, "Things of interest recommendation by leveraging heterogeneous relations in the internet of things," *ACM Transactions on Internet Technology*, vol. 16, no. 2, article no. 9, 2016.

[7] Y. Huang, X. Ma, J. Cao, X. Tao, and J. Lu, "Concurrent event detection for asynchronous consistency checking of pervasive context," in *Proceedings of the 7th Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2009*, Galveston, TX, USA, March 2009.

[8] W. Zhu, H. Lu, and X. Cui, "Distributed relation discovery in internet of things," in *Proceedings of the 2014 International Conference on Cloud Computing and Big Data, CCBD 2014*, pp. 39–46, Wuhan, China, November 2014.

[9] V. K. Garg, "Detection of Weak Unstable Predicates in Distributed Programs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 3, pp. 299–307, 1994.

[10] M. Hurfin, M. Mizuno, M. Raynal, and M. Singhal, "Efficient distributed detection of conjunctions of local predicates," *IEEE Transactions on Software Engineering*, vol. 24, no. 8, pp. 664–677, 1998.

[11] V. K. Garg and B. Waldecker, "Detection of strong unstable predicates in distributed programs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 12, pp. 1323–1333, 1996.

[12] M. Raynal, *Distributed algorithms for message-passing systems*, Springer, Berlin, Germany, 2013.

[13] A. D. Kshemkalyani and J. Cao, "Predicate detection in asynchronous pervasive environments," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 62, no. 9, pp. 1823–1836, 2013.

[14] R. Cooper and K. Marzullo, "Consistent detection of global predicates," in *Proceedings of the ACM/OCR Workshop on Parallel and Distributed Debugging*, pp. 163–173, 1991.

[15] A. D. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*, Chapter 11, Cambridge University Press, Cambridge, UK, 2008.

[16] Y. Z. Zhu and T. Y. Cheung, "A new distributed breadth-first-search algorithm," *Information Processing Letters*, vol. 25, no. 5, pp. 329–333, 1987.

[17] W. Zhu, J. Cao, and M. Raynal, "Predicate detection in asynchronous distributed systems: a probabilistic approach," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 65, no. 1, pp. 173–186, 2016.

[18] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, 1988.

[19] A. Khelil, F. K. Shaikh, B. Ayari, and N. Suri, "MWM: A map-based world model for wireless sensor networks," in *Proceedings of the 2nd International ICST Conference on Autonomic Computing and Communication Systems, AUTONOMICS 2008*, Turin, Italy, September 2008.

[20] M. Kranz, *SENSID: A Sensor Network Situation Detector [Honours thesis]*, School of Computer Science and Software Engineering, The University of Western Australia, 2005.

[21] H. D. Wang, C. C. Tan, and Q. Li, "Snoogle: a search engine for pervasive environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 8, pp. 1188–1202, 2010.

[22] C. C. Tan, B. Sheng, H. Wang, and Q. Li, "Microsearch: When search engines meet small devices," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5013, pp. 93–110, 2008.

[23] J. Nickels, P. Knierim, B. Könings et al., "Find my stuff: Supporting physical objects search with relative positioning," in *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp 2013*, pp. 325–334, Zurich, Switzerland, September 2013.

[24] R. Mietz, S. Groppe, K. Römer, and D. Pfisterer, "Semantic models for scalable search in the internet of things," *Journal of Sensor and Actuator Networks*, vol. 2, no. 2, pp. 172–195, 2013.

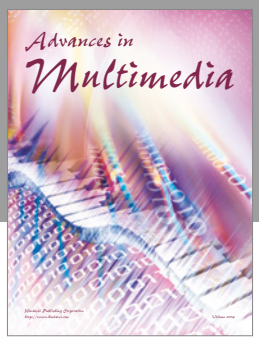
[25] C. Perera, A. Zaslavsky, P. Christen, M. Compton, and D. Georgakopoulos, "Context-aware sensor search, selection and ranking model for internet of things middleware," in *Proceedings of the 14th International Conference on Mobile Data Management, MDM 2013*, pp. 314–322, Milan, Italy, June 2013.

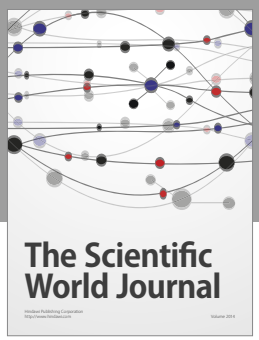
[26] O. Babaoglu and M. Raynal, "Specification and Verification of Dynamic Properties in Distributed Computations," *Journal of Parallel and Distributed Computing*, vol. 28, no. 2, pp. 173–185, 1995.

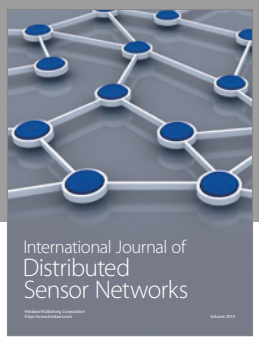
[27] K. M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 3, no. 1, pp. 63–75, 1985.

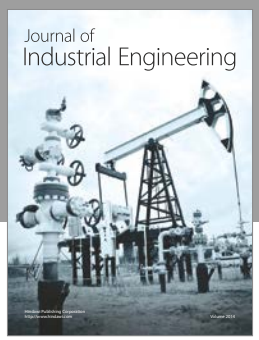
[28] A. D. Kshemkalyani, "Repeated detection of conjunctive predicates in distributed executions," *Information Processing Letters*, vol. 111, no. 9, pp. 447–452, 2011.

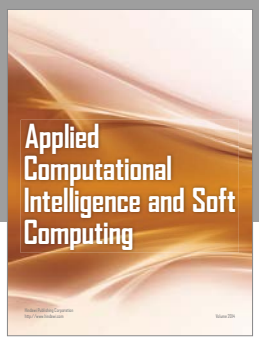
[29] A. D. Kshemkalyani, "Temporal interactions of intervals in distributed systems," *Journal of Computer and System Sciences*, vol. 52, no. 2, pp. 287–298, 1996.

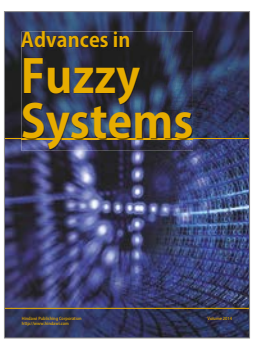


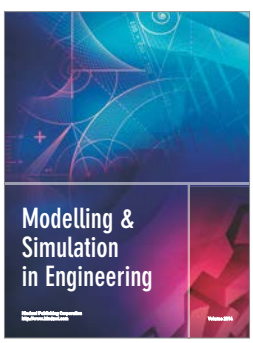


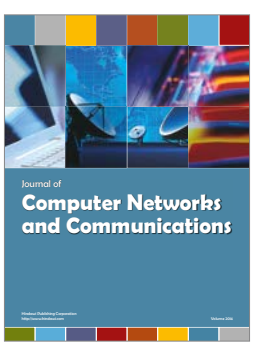
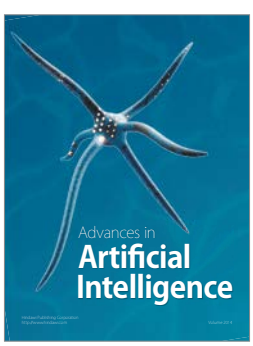
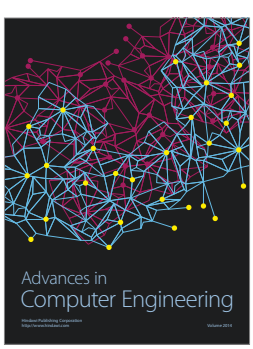
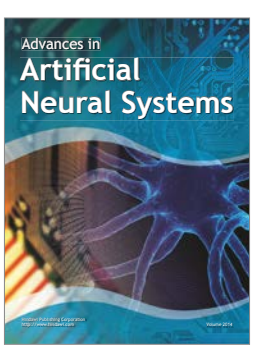
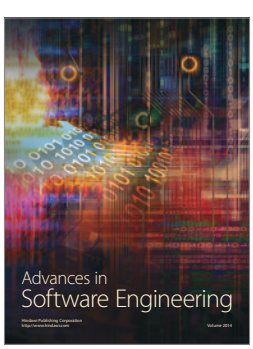
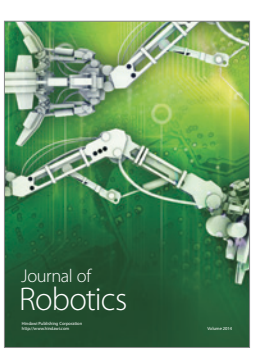
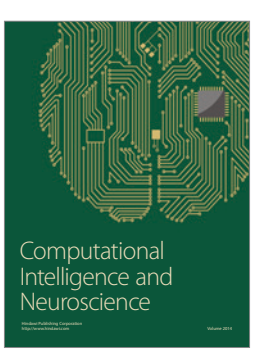












Hindawi
Submit your manuscripts at
https://www.hindawi.com