Journal of Universal Computer Science, vol. 22, no. 8 (2016), 1072-1096
submitted: 27/4/15, accepted: 29/7/16, appeared: 1/8/16 © *J.UCS*

CMSN: An Efficient and Effective Agent Lookup for Mobile Agent Middleware[1]

Hiroaki Fukuda
(Shibaura Institute of Technology
3-7-5 Toyosu Koto Tokyo Japan
hiroaki@shibaura-it.ac.jp)

Paul Leger
(Universidad Católica del Norte, Escuela de Ciencias Empresariales
Coquimbo, Chile
pleger@ucn.cl)

Keita Namiki
(JTEC CORPORATION
1-10-7 Kyobashi Cyuo Tokyo Japan
k-namiki@j-tec-cor.net)

Abstract: A Wireless Sensor Network (WSN) is typically deployed in a location in which no electrical source is provided, meaning that sufficient battery life is crucial. Applications for WSNs require implementations of complex operations such as network administration. To simplify the development of these applications, several mobile agent middleware solutions have been proposed. Applications for these middleware frameworks are executed by communication among agents; therefore, a common operation is to look up agents. Because existing proposals do not have much technical support for an efficient approach to look up agents, every lookup consumes a significant amount of battery power and time. In addition, current approaches can fail in their lookup operations if the target agent moves during a lookup operation. This paper proposes Chord for Mobile agent on Sensor Network (CMSN), an efficient and effective lookup for mobile agent middleware. CMSN is inspired by Chord for Sensor Networks (CSN), which introduces hierarchical ring structures and a distributed hash table algorithm to improve lookup performance. Unfortunately, CSN cannot be applied to mobile agent middleware solutions because CSN always requires a base station and assumes no agent migration between nodes. Unlikely CSN, CMSN is designed for an environment where agents can freely move between nodes without dependency on a special node such as a base station. In addition, CMSN leverages a feature where the location of a node is stationary in order to improve lookup performance with simplified algorithms. We evaluate and compare CMSN in terms of performance, effective lookups, and battery consumption.

Key Words: Wireless sensor networks, mobile agent middleware, agent lookup, Agilla, CSN, routing protocol
Category: H.4.3, D.1.8, D.4

[1] An earlier version of this article has been presented on IEEE International Conference on Embedded Software and Systems 2015 (ICESS 2015).

1 Introduction

Wireless Sensor Networks (WSNs) [Yick et al.(2008)] consist of a number of sensor nodes[2] used to detect and react to events. WSNs are applied in various domains, such as animal monitoring [Wark et al.(2007)], environmental observations [Chien-Liang et al.(2009)], smart home [Augusto and Nugent(2006)], and inventory tracking [McKelvin et al.(2005)]. The wide heterogeneity of hardware, software, and network resources poses significant coordination problems (*e.g.,* network connections, adaptations to environmental changes) and demands thorough knowledge of technologies [Raychoudfury et al.(2013)]. In addition, the heterogeneity of hardware and software makes it difficult to modularize pieces of code to obtain software engineer benefits such as reusability, maintainability, and flexibility. In order to migrate these problems and help application developers, some middleware solutions have been proposed [Blum et al.(2004), Boulis et al.(2003), Chien-Liang et al.(2009), Hui and Culler(2004)].

The proposed middleware solutions allow developers to build applications over a simplified abstraction layer. Middleware can be classified into three groups: data-oriented middleware, which abstracts a number of nodes into one; event-based middleware, which provides callback methods that are invoked when certain events are dispatched (*e.g.,* intruder detection); and mobile agent middleware, which executes applications that are based on collaborations between agents. Regarding completely distributed operations to avoid a single point of failure, mobile agent middleware solutions are appropriate because each (software) agent behaves autonomously to adapt its operating environment, and can migrate from node to node while maintaining its working state if necessary (*e.g.,* breaking down a node). Among these proposals, mobile agent-based middleware is suitable for complicated applications including fire detection, intrusion tracking, and robot navigation because of its autonomous nature.

Because of the embedded, pervasive nature of WSNs, each node is usually deployed in a location in which no electrical source is provided or there is difficulty in doing so (*e.g.,* forest, mine). Therefore, each node has to use a battery, which must work for a long time (*e.g.,* years) [Madden et al.(2005)]. Considering the battery problem in mobile agent middleware, the agent needs to find the node location where the target agent works in order to communicate and progress the operation in question. This search operation basically requires sending a packet over nodes, which consumes a large amount of battery power because sending a single bit can consume the same energy as executing 1,000 instructions [Levis et al.(2002)]. Therefore, the lookup agent location is a crucial problem in mobile agent middleware.

However, current mobile agent middleware solutions do not provide much

[2] For now, we will use the term "node" to refer to "sensor node".

technical support for the efficient looking up of agents. As a consequence, an agent has to look up the location of its target agent in an ad-hoc manner, basically visiting every node. This lookup approach consumes a significant (and unnecessary) amount of the battery power of a node. In addition, existing lookup approaches for these types of middleware do not always work correctly. For example, if the target agent moves during the lookup operation, this agent cannot be found.

This paper proposes Chord for Mobile agent on Sensor Network (CMSN), an efficient and effective agent lookup for a mobile agent middleware in WSNs. CMSN is based on Chord for Sensor Network (CSN [Ali and Uzmi(2004)]), a distributed hash table (DHT) algorithm, to look up agents. Concretely, the contributions of CMSN are:

1. An improvement in terms of time performance and battery consumption of nodes in a WSN when a lookup operation is executed.
2. The ability to locate an agent even if this agent is moving between nodes.

Although this proposal does not require particular requirements to be implemented in a mobile agent middleware, we have validated CMSN using Agilla [Chien-Liang et al.(2009)], a well-known middleware for mobile agent-based applications for WSNs. This implementation allows us to show improvements in runtime performance, effective lookups, and battery consumption.

This paper is organized as follows. Section 2 introduces agent lookups in mobile agent middleware solutions and highlights their existing lookup problems. Section 3 introduces the key concept on which our proposal is based: DHT algorithms. Section 4 describes our proposal for efficiently looking up agents in WSNs. Through an evaluation, Section 5 shows that our efficient lookup addresses the problems presented in Section 2. Section 6 discusses related work and Section 7 concludes.

2 Lookup of Agents in a WSN

Using Agilla [Chien-Liang et al.(2009)], this section first explains a mobile agent middleware. It then illustrates the need to look up agents and describes problems of existing agent lookup approaches used in these middleware frameworks.

2.1 Agilla: A Mobile Agent Middleware

Agilla is well-known middleware for mobile agents on WSNs [Yick et al.(2008)]. Figure 1 shows Agilla's architecture, which works over an operating system

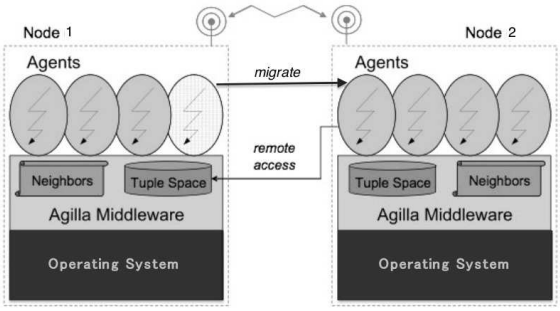


Figure 1: Agilla's architecture (adaptation from [Chien-Liang et al.(2009)]).

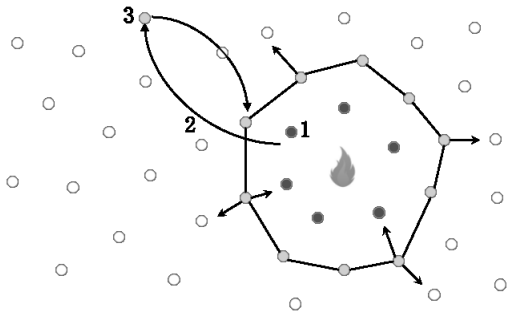


Figure 2: A WSN to detect and react to a fire cited from Agilla's example [Chien-Liang et al.(2009)]. (*1*) Agent A detects a fire. (*2*) this agent notifies to Agent B; (*3*) Agent B travels and clones itself around the fire.

like TinyOS [Hill et al.(2000)]. Every node of Agilla has an interpreter to execute a number of mobile agent programs. In addition, these agents can migrate into the network. Coordination among agents is supported by a *tuple space* [Gelernter(1985)] and a *neighbors* table. A tuple space is a shared memory that is located on each node and can be remotely accessed by agents, and a neighbors table contains the list of nodes where a node can send a packet by single-hop transmission [Kleinrock and Silvester(1978)]. To use the physical resources (*e.g.,* environmental temperature sensors) of a WSN, agents communicate with the node's operating system using an interface.

2.2 Looking Up Agents

This section issultrates the need to look up agents through a fire control application:

Fires are dangerous, and need to be tracked to be controled. A WSN can be used for this purpose [Chien-Liang et al.(2009)]. Figure 2 shows the big picture

of an agent-based application that detects and reacts to a fire. When a fire breaks out, an agent detects this fire (*1*) and must *find* a fire tracker agent to notify about the fire (*2*). This fire tracker agent migrates to the fire and clones itself to form a perimeter (*3*). The perimeter is continuously adjusted based on the fire's evolution. Depending on the physical resources available in the WSN, this WSN might notify firefighters or control the fire by itself.

2.3 Shortcomings of Existing Lookups

Notification to an agent requires that this agent must first be found. Existing mobile agent middleware such as Agilla does not provide much technical support for looking up agents. Basically, it is necessary to visit every node to find the target agent. This solution has three shortcomings:

- ***Runtime performance.*** Visiting every node takes unnecessary time, leading to a delay in reacting to certain events (*e.g.,* a fire).
- ***Migrations during lookups.*** This solution does not always work because the target agent can migrate to another node during the lookup operation. In WSN applications, programmers sometimes write the agent location explicitly when an agent needs to communicate with other agents.
- ***Battery consumption.*** In each visit, a node executes a local lookup operation to find the target agent, which consumes the battery energy. In the worst case, all nodes execute a local lookup. Note that this issue is different from the first one, because an improvement of runtime performance does not necessarily save the battery of each node of a WSN (*e.g.,* consider a node that is used for all nodes to answer node locations).

The last drawback is crucial for a WSN because these networks are commonly deployed in locations in which either electrical source is difficult to provide, or it is not easy to replace the batteries of these nodes. Hence, these batteries have to work for a long time [Madden et al.(2005)]. Next, we describe the requirements for an efficient approach to looking up agents in WSNs.

3 DHT Algorithms to Look Up Agents

CMSN uses Distributed Hash Table (DHT) algorithms [Stoica et al.(2001)]. The section first discusses the need for these algorithms in CMSN, and then explains DHT algorithms and their use in looking up agents.

3.1 Why Use a DHT Algorithm to Look Up Agents?

Centralized or distributed algorithms can be used to look up agents. On the one hand, a centralized approach uses a base station to track and look up agent locations. In the fire tracker in Agilla, Fire Detector the fire detector agent, which detects the fire, convey this to the base station. Fire tracker agents are then injected to a WSN by a base station. On the other hand, a distributed approach uses potentially all the nodes of a WSN instead of one base station. We briefly describe the advantages and drawbacks of both kinds of algorithms:

Centralized.: The clear advantage of this approach is that it requires only *one* base station to look up agents in a WSN. However, focused on the nodes around the base station, resulting in a significant increase in battery consumption of these nodes until their totally energy is spent. As a consequence, these nodes can no longer work and the base station cannot receive any more requests. Therefore, no more lookup operations can be executed.

Distributed.: In this approach, a node partially maintains agent locations and all nodes carry out the lookup operation. Although this approach is more difficult to implement than the centralized one, the distributed approach is more suitable for the following three reasons. First, this approach is fault tolerant, and a change in nodes causes a minimal amount of disruption. Second, it is possible to scale to a large number of nodes. Third, a lookup operation can work for longer than the centralized approach because battery consumption is distributed among all the nodes of a WSN. A longer lifetime for this operation is crucial for scenarios like fire tracking.

3.2 DHT Algorithms

Because DHT algorithms are used to find objects in a distributed manner, it is possible to use these algorithms to find agents in WSNs. DHT algorithms use a *hash table* (*i.e.,* a key-value table). For this proposal, keys are agent identifiers and values are agent locations (*i.e.,* nodes). We next explain two DHT algorithms below.

3.2.1 Chord

Although many DHT algorithms have been proposed [Ratnasamy et al.(2001), Rowstron and Druschel(2001), Zhao et al.(2004)], we explain the Chord algorithm because of its efficient lookup [Stoica et al.(2001), Ali and Uzmi(2004)]. As a DHT algorithm, Chord uses a DHT. When a node receives an agent lookup request, this node first looks it up on its local hash table and the request is forwarded to another node if the lookup operation is not resolved. To perform this operation, Chord uses a ring formation of nodes, enabling a node

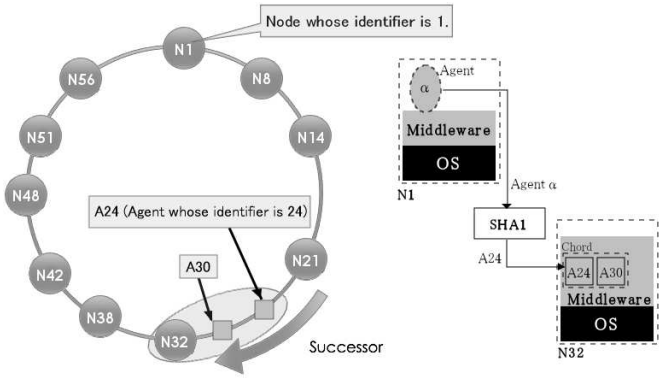


Figure 3: A Chord ring with 10 nodes and 2 agents.

to forward its lookup request. Figure 3 shows a Chord ring example, which uses SHA1 [Eastlake 3rd et al.(2001)], a cryptographic hash function, to assign identifiers to agents and their locations (*i.e.,* nodes). In Figure 3, an agent that is working on node N1 is assigned the identifier as 24 by using the SHA1 hash function. A node points to its *successor*, whose identifier is greater. For example, N21 points to successor N32. Using these identifiers, the distance between a node and an agent can be defined by clockwise subtractions of their identifiers. For example, the distance between A24 and N21 is farther than the distance between A24 and N32. Although an agent stays in a certain node, its location can be managed by another node. In Chord, the location of an agent is managed by the node whose distance from it is the shortest. For example, Figure 3 shows that A24 is working on N1, but its location is managed by N32, meaning that the local hash table of N32 has the entry "A32 $\rightarrow$ N1".

To look up an agent location, imagine two identifiers, k and r, where k is a node identifier (*i.e.,* Nk) and r is an agent identifier (*i.e.,* Ar). Suppose that Nk receives a lookup request for Ar. If r is greater than k and less than the identifier of the successor of Nk, this successor can resolve the request about Ar. Otherwise, Nk forwards this request to its successors until the location of Ar is resolved. We illustrate this procedure using Figure 3. In this figure, suppose that N14 receives a lookup request for A30. N14 forwards this request to its successor N21 because 30 is greater than 21. Then, N21 also forwards this request to its successor N32, which can resolve the location of A30 because 30 is greater than node 21 and less than N32. Regarding the time complexity of this lookup, Chord can look up a node in $O(logN)$, where N is the number of nodes in a ring. This efficiency is possible because each node uses a *finger table* to keep a path to other nodes, where the distance between node k and its successor is 2^{k-1} nodes (for more about Chord's complexity, see [Stoica et al.(2001)]).

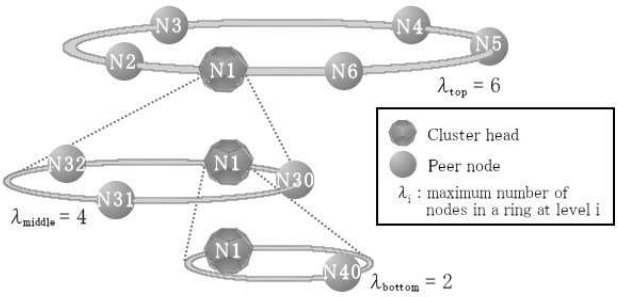


Figure 4: Example of CSN's rings (a hierarchy of rings of Chord).

Drawbacks to apply to WSNs. In Chord, a ring is created independent of the *physical distance* among nodes, defined by a certain hash function. Therefor, Chord cannot directly be applied to WSNs because a packet loss usually happens when the physical distance among nodes gets longer.

3.2.2 CSN

CSN, a variant of Chord addresses the physical distance issues that are drawbacks for Chord in relation to WSNs. CSN considers the physical distance among nodes to select a successor. CSN introduces a hierarchical clustering approach, where each cluster of nodes follows the ring formation of Chord. Each node keeps a finger table of $O(logN)$ size about other nodes in its ring, and a node resolves its lookup requests by sending $O(logN)$ messages to other nodes. In addition, as shown in Figure 4, a super node called a *cluster head* is selected in each ring and this node joins the upper ring as a member node. For example, N1 is a cluster head at the top level as well as the middle and bottom levels. Figure 4 shows that the number of nodes in a lower ring is less than the nodes in its upper ring. As a result, CSN guarantees that the lookup efficiency is $O(MlogN)$, where N is the number of nodes that joins the highest level and M is the maximum path length of the energy efficient path between nodes + 1 (see [Ali and Uzmi(2004)] for more details).

Drawbacks to apply to mobile agent middleware. CSN cannot be applied to look up mobile agents in WSNs for two reasons. First, CSN assumes that a base station starts ring creations and lookups. Second, a number of nodes may not be joined to a certain ring because of the physical distance between nodes in the ring creation phase. These reasons means that CSN is not appropriate for the efficient use of mobile agent middleware solutions because an agent may start a lookup at an arbitrary node. Apart from the structure of a ring, a lookup in CSN

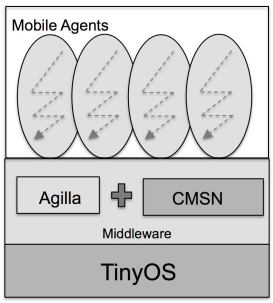


Figure 5: An overview of the architecture of a node in CMSN.

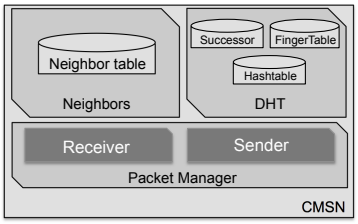


Figure 6: CMSN's architecture.

may not always work because the algorithm does not concern agent migrations, leading to an unsuccessful lookup even though the target agent is on a WSN.

4 CMSN: An Efficient Lookup of Agents

The core of CMSN uses CSN. We extend the architecture of every node of a WSN to implement CMSN (see Figure 5). This implementation is on top of TinyOS [Hill et al.(2000)], an operating system for WSNs. To implement CMSN, we use the nesC language [Gay et al.(2003)], a C extension for event-driven programming. Figure 6 shows the three new components of a node:

1. ***Packet Manager.*** This includes *Receiver* and *Sender* components. Based on the received packets, the Receiver component decides whether or not a node allocates an agent process. The Sender component adjusts the timing to send packets in order to prevent interference.

2. ***Neighbors.*** This collects packets and gathers the strength of each radio wave to make a neighbor table. To determine the physical distance, we use the strength of a radio wave when a packet is received[3].

3. ***DHT.*** This component allows nodes to support DHT algorithms, meaning that it controls the information on rings: successors, predecessors, and finger tables.

4.1 Creation of Rings by an Example

CMSN uses hierarchical cluster structures of CSN's rings. As shown in Figure 4, each cluster contains a *cluster head* that joins at least two rings (*e.g.,* top

[3] The use of the strength of a radio wave to measure physical distance is commonly accepted [Awad et al.(2007)].

and middle levels). In addition, a node has to communicate with another node that is next to it in a single hop to save its battery consumption. Suppose that two cluster heads cannot communicate with each other by a single hop manner because of the physical distance between them (as in CSN), leading to multi-hop communications. Therefore, the selection of the cluster head in each ring based on the physical distance is crucial. Because of this restriction, we manually choose *cluster heads* at each level. If we apply this approach to the Internet-based peer-to-peer (P2P) network, it causes a problem when a *cluster head* disappears (*i.e.,* shutting down a computer), resulting in nodes in the same ring that cannot communicate with other levels. However, this approach is feasible because the number of nodes is fixed and each node is not commonly added or removed in WSNs[4].

As an example, Figures 7(a, b, and c) illustrates the main stages of the ring creations. Figure 7a shows the initial state of a grid of 6 x 6 nodes. As shown at each level of Figure 7a, we specify each cluster head manually. Figure 7b-(t) shows a top-level ring that consists of six nodes. As shown in Figure 7b-(m), these six nodes are also cluster heads at the middle level. As a consequence, six rings at the middle-level are created (Figure 7c-(m)). As well as the top level, the nodes that belong to the middle level rings are also cluster heads at the bottom level. Therefore, as shown in Figure 7c-(b), 18 rings are created at the bottom level.

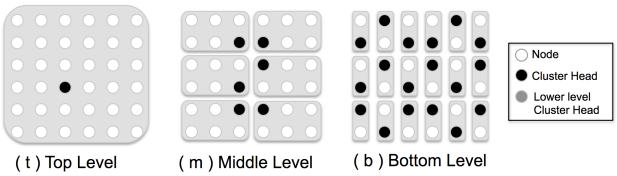
Next, by using Figure 8, we explain how to create a ring in Figure 8, which corresponds to the top-level ring from Figure 7a-(t) to Figure 7b-(t), and the numbers in Figure 8 correspond to the following numbered items:

1. ***Broadcast a Creation message.*** Node N1, which is a cluster head, begins sending a message to create a ring, named a Creation message. Note that, in WSNs, every message is sent in a broadcast manner. When a node receives this kind of message, it has to choose whether the message will be accepted or not.

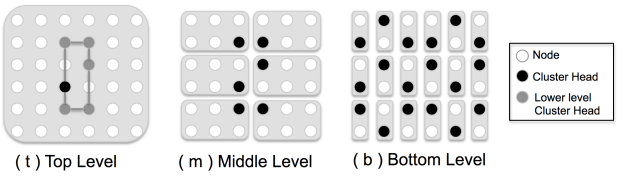
2. ***Reply ACKs against the Creation message.*** Every node that receives and accepts the Creation message becomes a cluster head at the middle level. In this example, nodes N2, N3, N5, N6, and N7 will send an ACK to N1.

3. **Create neighbor relations.** After node N1 receives ACKs from other nodes, this chooses a successor out of the nodes that reply ACKs. In the current implementation, we use Received Signal Stremgth Indicator (*RSSI*), which means the strength of the radio wave for this choice. As a consequence,

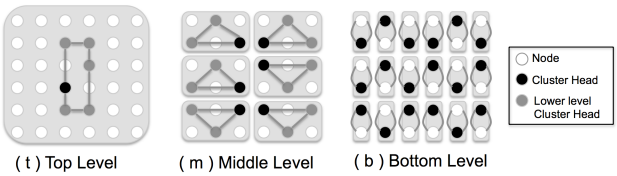
[4] If the battery of a node is totally consumed, the topology changes. Although this change is now ignored, it is possible to apply other approaches like a leader election algorithm to address this issue.



(a) Initial state of nodes at each level.



(b) After creating a ring at the top level.



(c) After creating all rings of each level.

Figure 7: Different stages of the creation of three levels of CMSN's rings.

node N2, which is the closest node to N1, is chosen. Then, N1 sends a message to N2 to inform of this choice. After receiving this message, node N2 registers node N1 as its predecessor, creating a link between No1 and N2.

4. ***Repeat and ignore the Creation message.*** A node that is chosen as a successor starts sending the Creation message. Therefore, the link between nodes N2 and N3 is created following the steps 1–3. Note that, in Figure 8-(ii), when node N3 sends a Creation message, node N1 is chosen based on its *RSSI* but it is undesirable because a ring among node N1, N2, and N3 will be created, meaning that the ring creation at the top level finishes. To prevent this scenario, a node that starts creating a ring (*e.g.,* node N1) ignores any Creation messages. At the same time, a node that has a predecessor and a successor (*e.g.,* node N2) does not reply either. As a consequence, all links between two nodes except the final link (between N7 and N1) are created.

5. ***Send a Complete message.*** The link between node N6 and N7 is created by step 4. In Figure 8-(ii), when node N7 sends a Creation message, no nodes reply. If node N7 cannot receive any ACK from others within a predefined period of time, node N7 will send a message to complete the creation of a ring, named the Complete message. A node that starts creating a ring (*e.g.,* node

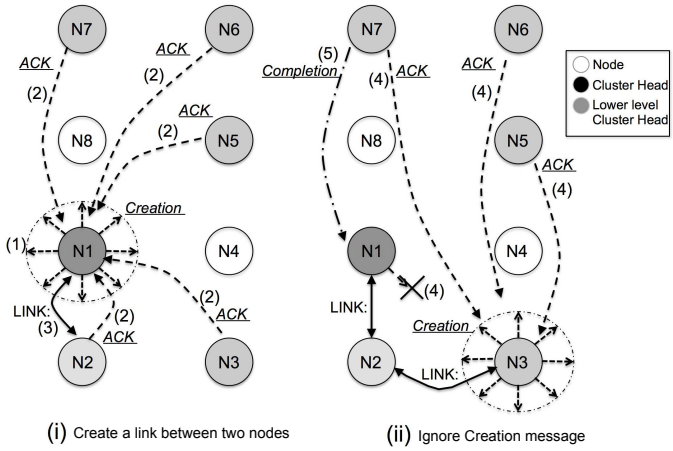


Figure 8: Illustration of steps for a ring creation based on the physical distance among nodes.

N1) replies to this Complete message. As a consequence, the final link between nodes N1 and N7 is created[5].

After creating the ring at the upper level, CMSN repeats the previous process to create rings at the lower levels. Thus, CMSN ends up creating all the rings at every level.

4.2 Assigning Identifiers

DHT algorithms must effectively manage a hash table in a distributed manner. Therefore, assigning an identifier to each node and agent is crucial. In Chord, it is not necessary to know how many nodes in a ring beforehand because Chord uses only one ring, so all nodes belong to this ring. Instead, similar to CSN, CMSN needs to define the maximum number of nodes per ring beforehand. We make use of this information to assign well organized identifiers. We first explain an algorithm for assigning identifiers, which is illustrated with a concrete example thereafter.

Defining node identifiers.: Although Chord and CSN use the same hash function (*e.g., SHA1*) to assign an identifier to each node and agent, CMSN uses a hash function only for agent identifiers. Node identifiers are assigned by the following three equations:

$$scope(l) = \begin{cases} \frac{Max(\text{hash})}{N(l)} & (l = 0) \\ \frac{scope(l-1)}{N(l)} & (otherwise) \end{cases} \quad (1)$$

[5] Similar to CSN, the number of nodes that compose each level ring is predefined.

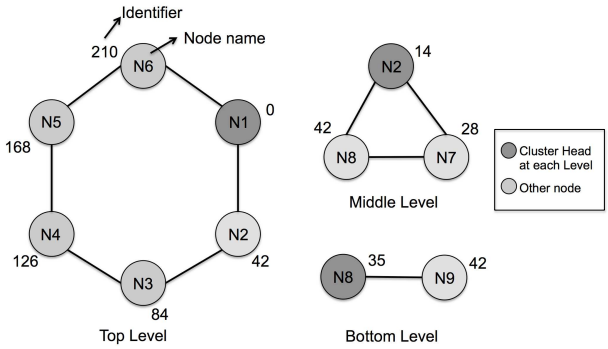


Figure 9: Identifier assignments at three levels.

$$node_id_l(i) = node_id_l(i-1) + scope(l) \tag{2}$$

$$node_id_l(0) = \begin{cases} 0 & (l=0) \\ scope(l) + OF_{l-1} & (otherwise) \end{cases} \tag{3}$$

In these equations, l is the ring level (*e.g.,* the top level). We assume that $l = 0$ corresponds to the top level where the identifier assignment begins. $Max(hash)$ is the maximum number of the hash function that is used for agent identifiers. For example, if we use the *SHA1* hash function, $Max(hash)$ must be $2^{160}-1$. $N(l)$ is the number of nodes per ring, which is manually defined as explained above. By using these variables, $scope(l)$ is calculated as equation (1). $Scope(l)$ represents a unit of boundary that each node needs to manage in a ring. Because the node assignment process starts from a cluster head per ring where a node identifier is defined in an inductive manner as shown in equation (2), and $node_id_l(0)$ in equation (3) always refers to a cluster head at each level. In equation (3), OF_{l-1} represents the identifier of a node that is a predecessor of the node at the upper level. Next, we illustrate this algorithm.

Example of assignment of identifiers to nine nodes.: Figure 9 illustrates an assigning identifier example with a hash function of 8 bits. The figure shows three levels of rings: top, middle, and bottom. Each level has a cluster head: node N1 for the top level, and node N2 for the middle and bottom levels. According to the equation (1), $N(0)$ is 6 and $Max(0)$ is 2^8-1;, $scope(0)$ is 42. From equations (2) and (3), the identifiers of N1 and its successor (*i.e.,* N2) at the top level are 0 and 42, respectively. At the middle level, $N(1)$ is 3 manually assigned, and $scope(1)$ is calculated as 14 (42/3). Note that although the identifier of N2 at the top level is 42, its identifier at the middle level is calculated again by equation (3). From the other case in the equation (3), OF is 0 because the predecessor of N2 at the top level is N1, the identifier of which is 0. As a result, the identifier

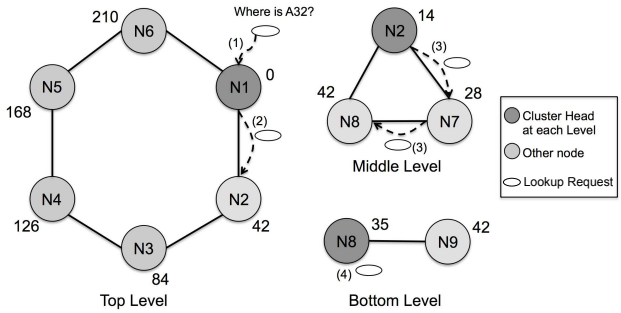


Figure 10: Illustration of lookup behavior steps in CMSN.

of N2 at the middle level is calculated as 14 (14 + 0) Finally, the identifiers of N7 and N8 at the middle level are calculated as 28 and 42 by equation (2). By repeating this algorithm, unique and well organized identifiers are assigned to every node on the WSNs.

4.3 Lookup Behavior

Lookup operations of CMSN are based on Chord, which mainly executes either a lookup operation in the local hash table of a node if the target identifier (*e.g.,* agent ID) is less than the node identifier, or otherwise forwards the request to its successor. Similarly to CSN, a cluster head has at least two successors; therefore, a cluster head needs to choose to which successor it must forward the received request when forwarding. The concrete situation of this decision is described at step 3 in the following example.

Figure 10 illustrates the process of a lookup operation. This example starts when a node N1 receives a lookup request for an agent whose identifier is 32. The numbers in this figure correspond to the following numbered items:

1. ***Receive a lookup request for an agent.*** In this example, node N1 receives a lookup request for an agent whose identifier is 32 (A32).

2. ***Forward the request to the same level ring.*** As described in the assigning identifier process (section 4.2), N1 knows that its rings at the middle and bottom levels manage identifiers from 211 to 255, meaning that A32 is not managed by N1 and its lower rings. Therefore, N1 forwards the request to its successor (*i.e.,* N2) at the top level.

3. ***Forward the request to the lower level ring.*** N2 compares its identifier at the top level (*i.e.,* 42) with 32, meaning that the location of the agent must be managed by N2 or its bottom rings. Because the identifier of N2 at

the middle level is 14, which is less than 32, N2 forwards the request to the successor at the middle level (*i.e.,* N10). For the same reason, N10 forwards the request to N11.

4. ***Answer the location of an agent or NotFound.*** The identifier of N11 at the middle level is 42, therefore N11 and its bottom ring must manage the location of A32. The identifier of N11 at the bottom level is 35, therefore N11 is the candidate that must manage the identifier of the request. Finally, N11 replies with the location of A32 (*i.e.,* the node identifier) if the local hash table contains the location, otherwise N8 returns a NotFound message.

4.4 Agent Migration

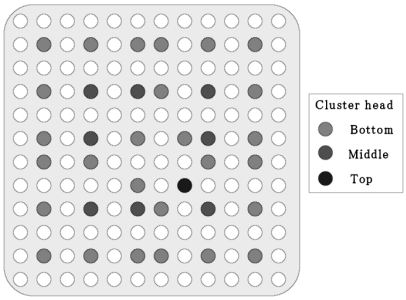
As described in section 2.3, if an agent migrates to another node during a lookup operation, the lookup will fail. To prevent this situation, we apply an approach similar to Mobile Internet Protocol version 6 (MIPv6) [Johnson et al.(2011)]. As an agent finds a location of another agent (A1) using a lookup operation, A1 can also look up the node that manages its own location in order to block a reply to an answer against a lookup request. To prevent lookup failures during agent migration, an agent sends an UnderMigration message to a node that manages its location. For example, suppose an agent (A2), which stays at node (N10), migrates to a node (N20), whose location is managed by a node (N15). In this case, A1 sends an UnderMigration message to N15 before its migration and announces its new location (*i.e.,* N20) to N15 after arriving at N20.

4.5 Summary of CMSN Advantages

To carry out an efficient and effective an agent lookup, CMSN uses novel strategies for (*1*) ring creation; (*2*) identifier assignment; (*3*) lookup behavior; and (*4*) agent migration. The implementation of the first two strategies is implemented by simple algorithms that leverage two reasonable assumptions in certain WSNs: their node number is fixed and these nodes do not move. To implement the strategies *3* and *4*, CMSN does consider that the migration of an agent during a lookup process. Therefore, CMSN can overcome the drawbacks of other solutions (*e.g.,* CSN): an agent cannot start lookups at an arbitrary node without a base station, and a lookup could fail if a target agent moves on a WSN.

5 Evaluation

This section shows that CMSN addresses the problems presented in section 2.3: runtime performance, migrations during lookup operations, and battery consumption. To show to what extent our proposal addresses previous problems,



Level	Number of cluster	Maximum number of nodes per ring
Top	1	9
Middle	9	4
Bottom	36	2

Figure 11: Configuration of rings used for the experiment.

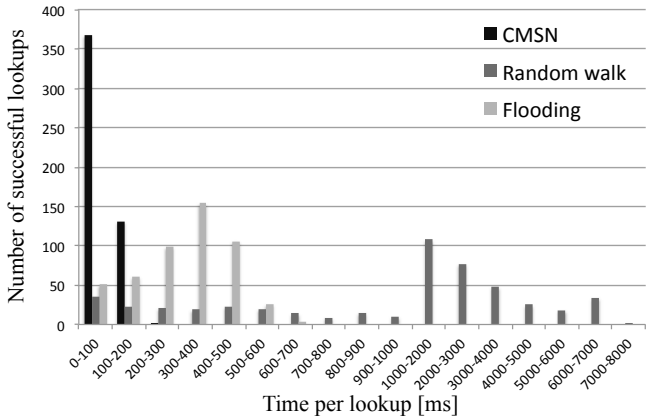


Figure 12: Histogram that shows the number of successful lookups and with their average time.

we compare CMSN to two algorithms: Random Walk [Gkantsidis et al.(2004)], a search algorithm for P2P network, and Flooding [Heinzelman et al.(1999)], a routing algorithm used to send requests between nodes.

Experiment setup. To realize the experiment with the aforementioned approaches, we use PowerTOSSIMZ [Perla et al.(2008)], an extension of the emulator TOSSIM [Levis et al.(2003)] for TinyOS. This emulator works for TinyOS 2.1.1 and is deployed on an Intel Core i5 (2.4 GHz) with 8GB of RAM running Ubuntu 12.04 (x32). Figure 11 shows the simulation configuration. The grid is

Approach	DHT creation [ms]	Per lookup [ms]	STDEV [ms]
CMSN	25,676	71	41
Random Walk	0	2,046	1,862
Flooding	0	307	144

Table 1: Average time of successful lookups in CMSN, Random Walk, and Flooding.

Approach	Lookups	Failed lookups	Failed lookup rate (%)
CMSN	500	0	0.0
Random Walk	500	31	6.2
Flooding	500	5	1.0

Table 2: Comparison of failed lookup rates of CMSN, Random Walk, and Flooding.

composed of a 12×12 nodes. CMSN rings are split into three levels. The top level contains 1 cluster and 9 nodes, the middle level contains 9 clusters and 4 nodes, and finally the bottom level contains 36 clusters and 2 nodes. Using this configuration, a set of agents are deployed and these agents randomly migrate between nodes . Then, a lookup operation starts when a node receives an agent lookup request. Using Random Walk, Flooding, and CMSN, we carried out 500 lookup operations to get an average result of these approaches.

To address runtime performance and battery consumption concerns, we evaluated these aspects in two stages: creation of rings (*i.e.,* DHT structures) and per lookup operation. To prevent infinite lookup operations in Random Walk and Flooding, we introduced time to live (TTL), as TCP/IP does in order to represent a *failed lookup*. This lookup operation takes a greater period of time compared to a threshold.

5.1 Runtime Performance

Figure 12 shows the results of the experiment with each approach regarding runtime performance. In this figure, the X-axis indicates different time ranges (*e.g.,* 0–100 ms) and the Y-axis corresponds to the number of successful lookups for a certain time range. CMSN is able to resolve more than half of the lookups in the time range 0-100 (ms). Although Flooding takes less time for lookups than Random Walk, Flooding's time ranges is 200–500 ms.

Regarding the average time for lookups, Table 1 shows the evaluation of each approach and the time for a ring creation in CMSN. The average time to lookup an agent in CMSN is about 1/30 of Random Walk and 1/4 of Flooding. As CMSN

Battery capacity: 21,600 Joules.		
Approach	DHT creation	Per lookup
CMSN	1.556 (0.0072%)	0.0049 (0.226×10^{-4}%)
Random Walk	0 (0%)	0.1170 (5.415×10^{-4}%)
Flooding	0 (0%)	0.0432 (1.998×10^{-4}%)

Table 3: Comparison of the battery consumption between CMSN, Random Walk, and Flooding.

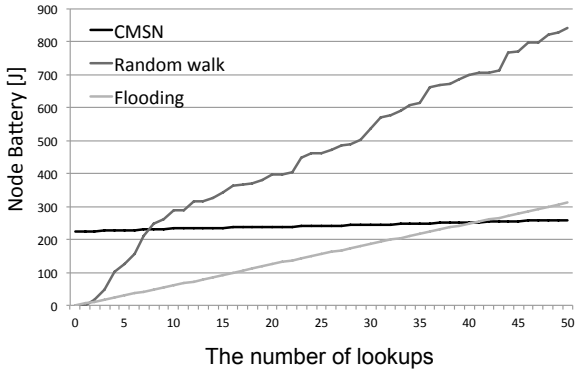


Figure 13: Comparison of CMSN to Random Walk and Flooding.

extends from CSN, ring creations requires time. In this experiment, CMSN spent about 26 (sec) which is quite slow compared to a lookup operation. However, the ring creation only occurs once and is amortized when lookup increases (*e.g.,* for years).

5.2 Migrations during Lookups

This section compares the failed lookups of each approach. As described in Section 2.3, lookup operations of some proposals fail because of agent migrations. For example, in a pure Random Walk approach, a lookup request is forwarded until the target agent is found, meaning that this operation never stops if the agent cannot be found.

To compare failed lookups, we introduced TTL. We assign a certain value as TTL to a lookup request then forward it to another node after decrementing this value. If this value becomes 0, the request is no longer forwarded, implying that this lookup fails. For this experiment, we assigned 1,000 to TTL because this number is big enough for the experiment grid of nodes (*i.e.,* 144 nodes). Table 2 shows a comparison of failed lookups. The failed lookup rate of CMSN is 0%,

while those of Random Walk and Flooding were 6.2% and 1% respectively. This result means that lookup requests of Random Walk and Flooding may return "agent not found" when an agent is at a WSN.

5.3 Battery Consumption

Table 3 shows the comparison of battery consumption between Random Walk, Flooding, and CMSN. Although our proposal consumes battery power to create DHT structures, it only consumes 0.0072% of the battery of each node to create it. In addition, CMSN consumes 0.226×10^{-4}% of the battery power per lookup operation. In contrast, Random Walk and Flooding consume 5.5415×10^{-4}% and 1.998×10^{-4}% of each node respectively.

As an additional evaluation, Figure 13 shows the relationship between battery consumption and the number of lookup operations in Random Walk, Flooding and CMSN. As CMSN requires additional battery power to create structures, Random Walk and Flooding consume less battery power when the number of lookup operation is less than 7 times and 41 times respectively. However, when the number of lookup operation increases to 7 or 41, CMSN consumes less battery power.

Based on previous results, CMSN consumes a significant amount of battery power at the beginning, but this consumption decreases when the number of lookup requests increases to more than 41. In the mobile agent frameworks of WSNs, fewer than 41 lookup operations is unusual because they are designed to work from one to two years [Madden et al.(2005)].

6 Related Work

In distributed algorithms, locating agents requires a lookup request to be sent to a node, which replies with the agent location if this node knows, otherwise this node forwards the request to another node. This behavior is similar to routing algorithms on WSNs, which decide where a node should forward a request. Therefore, a comparison of CMSN with routing algorithms is worthwhile. We review and compare those routing algorithms categorized by network structures: flat, hierarchical, and DHT [Al-karaki and Kamal(2004)]. In addition we review a domain name server (DNS) [IETF(1987)] based lookup approach that, to the best of our knowledge, it is the only proposal for lookups that can handle agent migration for a mobile agent-based middleware.

6.1 Flat Routing

In flat routing approaches, all nodes typically play the same role and collaborate to perform tasks. Although a number of protocols have been proposed so far in this category, we review two pioneering approaches.

SPIN [Heinzelman et al.(1999)] disseminates all the information at each node to every node in the network. SPIN makes use of the property that nodes in close proximity have similar information, and hence it is only necessary to distribute information to far nodes. To carry out this behavior, SPIN introduces a meta-data to completely describe their collected data, and performs three-stage negotiations before data transmissions: *ADV*, *REQ* and *DATA*. *ADV* is used to announce new data, *REQ* to request data, and *DATA* is the actual message. The algorithm starts when a node obtains new data that must be shared. The node first broadcasts an *ADV* message containing the metadata. If a neighbor is interested in the data, this node sends a *REQ* message for *DATA* and this is sent to it. Otherwise, *REQ* and *DATA* are not sent, resulting in the elimination of transmission, improving energy use. Similar to SPIN, a CMSN node only forwards a query to its successor node in the ring, and not all the nodes. However, SPIN was developed to disseminate information to all nodes on a WSN, implying that performance for an operation like lookup is affected.

Directed diffusion [Intanagonwiwat et al.(2003)] proposes a new approach that is based on data aggregation. The main idea of this approach is to combine the data coming from different sources en route by eliminating redundancy, minimizing the number of transmissions and resulting in nodes energy savings. Directed diffusion introduces a *gradient* concept: strength between two nodes. A sink node, which is similar to a base station, generates and propagates a query to a WSN. Intermediate nodes locally propagate the query towards a target node that senses corresponding data. The data is sent back to the sink node by using the same path from which it was sent. Because of the local knowledge of nodes, the same data or query may be transmitted to a node from multiple paths. Every time a node transmits a data or query, this node updates its gradients based on attributes like the radio wave strength. Using this gradient, every node is able to choose which node receives the query or data, meaning that direct diffusion minimizes redundant paths. However, directed diffusion is not optimum because the reinforced path can vary due to the gradient calculation that only uses the local knowledge of a node. CMSN does not suffer from such redundancy, because one path is always chosen when a node transmits data due to the ring structure; thus, CMSN saves node batteries.

6.2 Hierarchical Cluster-Based Routing

Hierarchical cluster-based routing is well-known for its techniques for scalable and efficient communications. Therefore, the idea behind of hierarchical routing is utilized to perform energy-efficient routing on WSNs. The main idea of cluster-based routing is that higher energy nodes are used to process and transmit data to a base station, while lower energy nodes sense in the proximity of the target, resulting in an improved use of node energy. However,

these solutions have the problem of the base station (section 3.1). For example, **LEACH** [Heinzelman et al.(2002)] introduces distributed cluster formation, which randomly selects a small set of nodes as the *cluster head* and rotates this role to distribute the energy consumption among WSN nodes. A cluster head compresses data arriving from its cluster and transmits the aggregated data to a base station. Some LEACH extensions like PEGASIS [Lindsey et al.(2002)], and CogLEACH[Eletreby et al.(2014)] optimize energy use through the selection of the closest neighbors and the minimization of data between a cluster head and the nodes of this cluster. These protocols are more appropriate when there is a need for constant monitoring by minimizing battery consumption in a WSN. However, the effectiveness of each lookup was out of focus because they assume that every cluster head can transmit a packet to a base station directly (*i.e.,* a single hop). This assumption is not feasible for the realm of WSN because a node has a limited transmission length. CMSN considers multi-hop transmissions to be adapted to a real WSN.

6.3 DHT-based routing

DHT-based routing, such as CSN, is an approach that has been proposed recently, where nodes must necessarily cooperate to maintain and administer references to data [Ali and Uzmi(2004), Caesar et al.(2006), Awad et al.(2011a)]. In addition, each node stores a partial view of the whole distributed system that effectively distributes the routing information. To achieve the effective routing, each algorithm based on this approach maintains routing information that reduces hop counts to the destination of a node. We now review how some DHT algorithms reduce forwarding requests to carry out an efficient routing.

VRR [Caesar et al.(2006)] is a DHT-based network routing algorithm. VRR is implemented directly on the top of a link layer, which is a media access control (MAC) layer in the protocol TCP/IP. In VRR, nodes assign themselves random and location-independent identifiers in a ring structure. Random node identifiers lead to long physical routing paths, resulting in additional energy consumption. Unlike VRR, which only contains one ring, CMSN introduces hierarchical rings and assigns according to a strategy that takes into account the physical distance among nodes, meaning that a node can forward data to its neighbor (*i.e.,* successor) with a lower cost.

MVR [Gao and Li(2009)] makes use of multilevel virtual rings, similar to hierarchical rings, to reduce hop counts from a source node to a destination node. MVR proposes an autonomous multilevel ring creation algorithm; however, it does not always succeed because nodes will move, meaning that the backbone selection message in MVR cannot reach to the neighbor nodes. In addition, MVR utilizes levels that contain a set of nodes with a unique identifier per level.

As MVR is an *address-centric* routing algorithm, an agent location cannot be mapped on the same ring structure. Although MVR uses hierarchical rings like CMSN, MVR cannot be applied to lookup in a mobile agent middleware because the latter they requires *data-centric* approaches, since an agent can move to different nodes. The identifier strategy of CMSN can map identifiers of nodes and agents on the same ring while the use of hierarchical rings reduces the hop counts of a lookup request.

VCP and VCP-m [Awad et al.(2011a)][Anwit et al.(2014)] introduces the *virtual cord protocol*, which efficiently combines data lookup with routing techniques. VCP accomplishes this by placing all nodes on a virtual cord (*i.e.,* a value) that is also used to associate with the data, the VCP-m then modifies the joining nodes process. As in Chord, a hash function is used to create values in a predefined range, and each node maintains a part of the entire range. In addition, locally available neighborhood information is exploited for greedy routing that may reduce hop counts to a destination. Similar to the original Chord, VCP and VCP-m use a ring formation of nodes, and each node has a greater valued successor node of a greater value. Each node basically forwards a request to its successor. Furthermore, a node is able to compare the value of a request with the node values that are assigned to the physically close nodes. A *greedy* node forwards a request to a closer node when its value is greater than the value request. The greedy routing relies on the probability that a node with a closer value to the destination exists around when a node forwards a lookup request. Hence, VCP and VCP-m cannot guarantee the lookup time. Instead, CMSN adapts a Chord based routing that can provide strong guarantees of lookup time, because our proposal is not based on probabilities.

6.4 Discovery Service for Mobile Agent Middleware

MLDS [Bhattacharya et al.(2008)] uses a directory service for supporting of agent migrations in WSNs through a tree-based structure similar to DNS. A node that has children nodes is called a *Clusterhead* in MLDS. A node where agents stay at keeps sending a location update message to its *Clusterhead* at a certain time interval. This periodic message includes agent IDs on a child node. This message is propagated to the root node at the end. As a consequence, an agent can lookup the location of another agent at an arbitrary node and also the location of an agent is updated when it migrates to another node. However, because of the time interval, a *Clusterhead* may have stale information. In addition, this strategy will waste battery power when an agent stays at a node beyond a period of time. CMSN does not need periodic messages to keep agent locations because of the agent migration strategy, meaning that unnecessary battery power is not consumed. In addition, a node can always keep the current location of each agent, resulting in a node always returning a correct answer for a lookup.

7 Conclusion

Because WSNs promise useful solutions in the real world, this is a motivation to address issues like environmental changes, network connections, and distributed operations. In addition, these applications must execute operations efficiently because the nodes of a WSN commonly use batteries to work. To address these issues in an efficient manner and simplify the development of WSN applications, different middleware solutions have been proposed. One such solution is based on agent interactions. For this kind of middleware, an agent must know the exact location of its target agent before interacting. However, existing middleware solutions, including Agilla, do not provide much technical support for efficiently looking up agents. An inefficient implementation of an agent lookup consumes unnecessary time and battery, which is crucial for a WSN. In addition, agent lookup operations of mobile agent middleware cannot always return the correct agent location when migration is supported. This paper proposes CMSN, an efficient and effective agent lookup, which is based on the CSN algorithm. To show the effectiveness and efficiency of our proposal, we implemented CMSN as an Agilla extension on TinyOS.

References

[Al-karaki and Kamal(2004)] Al-karaki, J. N., Kamal, A. E.: "Routing techniques in wireless sensor networks: A survey"; IEEE Wireless Communications; 11 (2004), 6–28.

[Ali and Uzmi(2004)] Ali, M., Uzmi, Z. A.: "CSN: A network protocol for serving dynamic queries in large-scale wireless sensor networks"; Proceeding of the Second Annual Conference on Communication Networks and Services Research; 165–174; 2004.

[Augusto and Nugent(2006)] Augusto, J. C., Nugent, C. D.: "The role of artificial intelligence"; J. G. Carbonell, J. Siekmann, eds., Designing Smart Homes; Lecture Notes in Computer Science; 2006.

[Awad et al.(2007)] Awad, A., Frunzke, T., Dressler, F.: "Adaptive distance estimation and localization in wsn using rssi measures"; Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on; 471–478; 2007.

[Awad et al.(2011a)] Awad, A., German, R., Dressler, F.: "Exploiting virtual coordinates for improved routing performance in sensor networks"; Mobile Computing, IEEE Transactions on; 10 (2011a), 9, 1214–1226.

[Anwit et al.(2014)] R. Anwit and P. Kumar and M. P. Singh: "Virtual Coordinates Routing Using VCP-M in Wireless Sensor Network"; In Proceedings of Computational Intelligence and Communication Networks (CICN), IEEE, 402-407, 2014.

[Bhattacharya et al.(2008)] Bhattacharya, S., Fok, C., Lu, C., Roman, G.: "MLDS: A flexible location directory service for tiered sensor networks"; Computer Communications; 31 (2008), 6, 1160–1172.

[Blum et al.(2004)] Blum, T. A. B., Cao, Q., Chen, Y., Evans, D., George, J., George, S., Gu, L., He, T., Krishnamurthy, S., Luo, L., Son, H., Stankovic, J., Stoleru, R., Wood, A.: "EnviroTrack: Towards an environmental computing paradigm for distributed sensor networks"; Proceedings of the 24th International Conference on Distributed Computing System; 582–589; 2004.

[Boulis et al.(2003)] Boulis, A., Han, C.-C., Srivastava, M. B.: "Design and implementation of a framework for efficient and programmable sensor networks"; Proceedings of the 1st International Conference on Mobile Systems, Applications and Services; MobiSys '03; 187–200; San Francisco, California, 2003.

[Butun et al.(2014)] Butun, I., Morgera, S. D., Sankar, R.: "A survey of intrusion detection systems in wireless sensor networks"; IEEE Communications Surveys and Tutorials; 16 (2014), 1, 266–282.

[Caesar et al.(2006)] Caesar, M., Castro, M., Nightingale, E. B., O'Shea, G., Rowstron, A.: "Virtual ring routing: Network routing inspired by dhts"; ACM SIGCOMM Computer Communication Review; 36 (2006), 4, 351–362.

[Chien-Liang et al.(2009)] Chien-Liang, F., Roman, G.-C., Lu, C.: "Agilla: A mobile agent midleware for self-adaptive wireless sensor networks"; ACM Transactions on Autonomous and Adaptive System; 4 (2009), 3, 1–26.

[Eastlake 3rd et al.(2001)] Eastlake 3rd, D., Jones, P., US, S. H. A.: "SHA1"; (2001).

[FLIR(2014)] FLIR: "FLIR: Systems for airport security"; (2014).

[Gao and Li(2009)] Gao, L., Li, M.: "Multi-level virtual ring: a foundation network architecture to support peer-to-peer application in wireless sensor network"; Telecommunication Networks and Applications Conference (ATNAC), 2009 Australasian; 1–6; 2009.

[Gay et al.(2003)] Gay, D., Levis, P., Behren, R., Welsh, M., Brewer, E., Culler, D.: "The nesc language: A holistic approach to networked embedded systems"; ACM SIGPLAN Notices; 38 (2003), 5, 1–11.

[Levis et al.(2002)] Levis, P., E., Culler, D.: "Mate: A Tiny Virtual Machine for Sensor Networks"; Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS X) '02; 85–95; 2002.

[Gelernter(1985)] Gelernter, D.: "Generative communication in Linda"; ACM Transactions on Programming Languages and Systems; 7 (1985), 1, 80–112.

[Gkantsidis et al.(2004)] Gkantsidis, C., Mihail, M., Saberi, A.: "Random walks in peer-to-peer networks"; INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies; volume 1; Hong Kong, China, 2004.

[Heinzelman et al.(2002)] Heinzelman, W. B., Chandrakasan, A. P., Balakrishnan, H.: "An application-specific protocol architecture for wireless microsensor networks"; IEEE Transactions on Wireless Communications; 1 (2002), 4, 660–670.

[Heinzelman et al.(1999)] Heinzelman, W. R., Kulik, J., Balakrishnan, H.: "Adaptive protocols for information dissemination in wireless sensor networks"; Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking; MobiCom '99; 174–185; ACM, Seattle, Washington, USA, 1999.

[Hill et al.(2000)] Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, K., David andr Pister: "System architecture directions for networked sensors"; ACM SIGARCH Computer Architecture News; 28 (2000), 5, 93–104.

[Hui and Culler(2004)] Hui, J. W., Culler, D.: "The dynamic behavior of a data dissemination protocol for network programming at scale"; Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems; SenSys '04; 81–94; Baltimore, MD, USA, 2004.

[IETF(1987)] IETF: "DOMAIN NAMES - implementation and specification"; (1987).

[Intanagonwiwat et al.(2003)] Intanagonwiwat, C., Govindan, R., Estrin, D., Heideman n, J., Silva, F.: "Directed diffusion for wireless sensor networking"; IEEE/ACM Transactions on Networking (TON); 11 (2003), 1, 2–16.

[Johnson et al.(2011)] Johnson, D., Perkins, C., Arkko, J.: "Mobile Internet Protocol Version 6 (RFC 6275)"; (2011).

[Kleinrock and Silvester(1978)] Kleinrock, L., Silvester, J.: "Optimum transmission radio for packet radio networks or why six is a magic number"; Proceedings of the IEEE National Telecommunications Conference; Birimingham, Alabama, 1978.

[Levis et al.(2003)] Levis, P., Lee, N., Welsh, M., Culler, D.: "TOSSIM: Accurate and scalable simulation of entire TinyOS applications"; Proceedings of the 1st International Conference on Embedded Networked Sensor Systems; 126–137; 2003.

[Lindsey et al.(2002)] Lindsey, S., Raghavendra, C., Sivalingam, K. M.: "Data gathering algorithms in sensor networks using energy metrics"; IEEE Transactions on Parallel and Distributed Systems (TPDS); 13 (2002), 9, 924–935.

[Madden et al.(2005)] Madden, S. R., Franklin, M. J., Hellerstein, J. M., Hong, W.: "TinyDB: An acquisitional query processing system for sensor networks"; ACM Transactions on Database Systems; 30 (2005), 1, 122–173.

[Manjeshwar and Agrawal(2001)] Manjeshwar, A., Agrawal, D. P.: "TEEN: Arouting protocol for enhanced efficiency in wireless sensor networks"; Proceedings of the 15th International Parallel & Distributed Processing Symposium; IPDPS '01; IEEE Computer Society, Washington, DC, USA, 2001.

[McKelvin et al.(2005)] McKelvin, M. L., Jr., Williams, M. L., Berry, N. M.: "Integrated radio frequency identification and wireless sensor network architecture for automated inventory management and tracking applications"; Proceedings of the 2005 Conference on Diversity in Computing; TAPIA '05; 44–47; ACM, Albuquerque, New Mexico, USA, 2005.

[Perla et al.(2008)] Perla, E., Catháin, A., Carbajo, R. S., Huggard, M., Goldrick, C. M.: "PowerTOSSIM z: realistic energy modelling for wireless sensor network environments"; Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and networks; 35–42; 2008.

[Ratnasamy et al.(2001)] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: "A scalable content-addressable network"; ACM SIGCOMM Computer Communication Review; 31 (2001), 4, 161–172.

[Rowstron and Druschel(2001)] Rowstron, A. I. T., Druschel, P.: "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems"; Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg; Middleware '01; 329–350; Springer-Verlag, London, UK, UK, 2001.

[Stoica et al.(2001)] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakrishnan, H.: "Chord: A scalable peer-to-peer lookup service for internet applications"; ACM SIGCOMM Computer Communication Review; 31 (2001), 4, 149–160.

[Wark et al.(2007)] Wark, T., Crossman, C., Hu, W., Guo, Y., Valencia, P., Sikka, P., Corke, P., Lee, C., Henshall, J., Prayaga, K., O'Grady, J., Reed, M., Fisher, A.: "The design and evaluation of a mobile sensor/actuator network for autonomous animal control"; Proceedings of the 6th International Conference on Information Processing in Sensor Networks (IPSN '07); 206–215; ACM, Cambridge, Massachusetts, USA, 2007.

[Yick et al.(2008)] Yick, J., Mukherjee, B., Ghosal, D.: "Wireless sensor network survey"; Computer Networks; 52 (2008), 12, 2292–2330.

[Zhao et al.(2004)] Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D., Kubiatowicz, J. D.: "Tapestry: A resilient global-scale overlay for service deployment"; IEEE Journal on Selected Areas in Communications; 22 (2004), 41–53.

[Eletreby et al.(2014)] R. M. Eletreby and H. M. Elsayed and M. M. Khairy: "CogLEACH: A spectrum aware clustering protocol for cognitive radio sensor networks"; Proceedings of 9th International Conference on Cognitive Radio Oriented Wireless Networks and Communications (CROWNCOM); 179-184; IEEE, 2014.
the 6th International Conference on Information Processing in Sensor Networks (IPSN '07); 206–215; ACM, Cambridge, Massachusetts, USA, 2007.

[Raychoudfury et al.(2013)] Vasker, R., Jiannong, C., Mohan, Z., Daqiand: "Middleware for Pervasive Computing: A Survey"; ACM Pervasive Mob. Comput. 24 (2013), 177–200.