An Efficient Agent Lookup Approach in Middlewares for Mobile Agents

Hiroaki Fukuda
Shibaura Institute of Technology, 3-7-5 Koto, Toyosu, Tokyo, 135-8548, Japan

Keywords: Wireless Sensor Network, Mobile Agent, Distributed Hash Table, Location Management.

Abstract: A Wireless Sensor Network (WSN) is typically deployed on the place in which no electric source is provided, meaning that the battery consumption concern is crucial. Due to their deeply-embedded pervasive nature, applications running on WSNs need to adapt to the changes in physical environment or user preferences. In addition, developers of these applications must pay attention to a set of additional concerns such as limited hardware resources and the management of a set sensor nodes. To appropriately address these concerns, middlewares based on mobile agent based like Agilla have been proposed. Applications for these middlewares are executed by the communications among agents, thereby a common task is to look up the agents. In this paper, we propose an efficient lookup approach for mobile agent based middlewares in WSNs. We evaluate the advantages of our proposal through the comparison with traditional lookup approaches.

1 INTRODUCTION

Wireless Sensor Networks (WSNs) (Yick et al., 2008) consist in a number of sensor nodes and have been used to detect events and/or collect data in various domains such as environment observation and inventory tracking. Due to their embedded pervasive nature, each node is usually deployed on the place in which no electric source is provided and/or picking them up is not an easy task (*e.g.* inside of a forest). Therefore, each node is usually working by dry cell batteries yet these nodes require to keep working for a long time (*e.g.* a few years (Madden et al., 2005)).

Developing an application for WSNs is difficult because developers need to pay attention to a network environment dynamically changing, the management of a set of sensor nodes, low-level instructions related to physical hardwares. Several middlewares for WSNs have been proposed to make a development easier (Chien-Liang et al., 2009; Madden et al., 2005; Blum et al., 2004). It is possible to classify these middlewares into three types: (1) data oriented approaches that abstract a number of sensor nodes as one abstraction: WSN, (2) event-based approaches that provide callback methods, that are invoked when certain events are dispatched, and (3) mobile agent approaches, where their applications are based on the collaboration of mobile agents. In this paper, we focus on the mobile agent based approach because this approach is adequate when multiple applications are running on a WSN (Chien-Liang et al., 2009). Agilla (Chien-Liang et al., 2009), a middleware for mobile agents for WSNs, allows an application to restrict itself to only reside on relevant nodes. As WSN environments change, this application can be self-adapted through the migration of its agents to the required locations. Moreover, an application in Agilla is executed by interactions among agents, meaning that an agent needs to know the exact location of other agents. Although Agilla allows agents to interact with other agents, this middleware (like similar proposals) does not provide much technical support to efficiently lookup the agent locations. As a result, an agent has to look up the location of its target agent by ad-hoc manner, resulting in the consumption of a significant and unnecessary amount of the battery of sensor nodes.

In this paper, we propose an efficient lookup approach for mobile agent based middlewares in WSNs. Our proposal borrows ideas from CSN (Chord for WSNs) (Ali and Uzmi, 2004). As a consequence of our proposal, it is possible to save battery consumption of every node in a WSN. We implement our proposal on TinyOS, which is one of the platform for WSNs. Finally, we show the advantages of our approach via a simulations that compare battery consumption to traditional lookup approaches.

Paper Roadmap. Section 2 briefly introduces Agilla and points out the battery consumption problem. Section 3 presents key concepts on which our proposal is based, and Section 4 describes our proposal. Section

Fukuda H..
An Efficient Agent Lookup Approach in Middlewares for Mobile Agents.
DOI: 10.5220/0005554100810087
In *Proceedings of the 12th International Conference on Wireless Information Networks and Systems* (WINSYS-2015), pages 81-87
ISBN: 978-989-758-119-9

5 shows that our proposal addresses the problems and Section 6 concludes.

2 AGILLA: A MIDDLEWARE FOR WSNS

This section briefly describes Agilla and identifies the battery consumption problem.

2.1 Overview of Agilla

Agilla (Chien-Liang et al., 2009) is a well-known middleware for mobile agents on Wireless Sensors Networks (WSNs) (Yick et al., 2008). Each node of Agilla has an interpreter to execute mobile agent programs. Developers can write a program using ISA (Instruction Set Architecture) prepared by Agilla. In addition, this middleware can also execute a number of agent programs on a node. By using these features, a program can restrict to reside only on relevant nodes and reactions to changes of the environment.

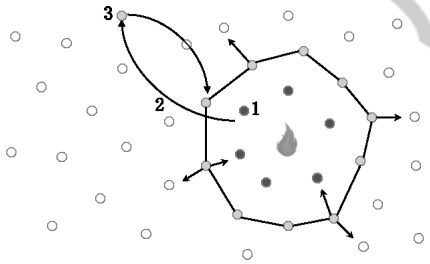


Figure 1: An overview of the fire detection and tracking application.

2.2 Battery Consumption Problem

Using an example about the fire control, we introduce the battery consumption problem. Figure 1 shows an overview of the fire detection and tracking application in Agilla. When a fire breaks out, agents detect this fire (1) and send a message to a tracker agent (2). The tracker agent migrates to the fire and clone itself to form a perimeter. The perimeter is continuously adjusted based on the evolution of the fire. To send a message to a tracker agent, an agent must know the exact location of this agent tracker. Since Agilla (like similar proposals) does not provide methods to lookup the location of a agent on the fly, the location of a tracker agent is directly written in agents, meaning that a tracker agent should keep staying on a determined node. Otherwise, an agent needs to lookup the location of a tracker agent every time, resulting in consuming the battery of nodes very soon.

3 DHT TO LOOK UP AGENTS

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

3.1 Why a Distributed Hash Table Algorithm to Look Up Agents?

Centralized or distributed algorithms can be used to look up agents. A centralized approach uses a base station to keep and look up agent locations. In Fire tracker in Agilla, Fire Detector agent, which detects the fire, notify it to the base station. Fire Tracker agents are then injected to WSN by base station. With this scenario, traffic gets focused on the nodes around the base station, resulting in a significant increase of 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 more requests. Therefore, no more lookup operations can be executed, requiring a distributed algorithm.

3.2 Distributed Hash Table 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. We next explain two DHT algorithms.

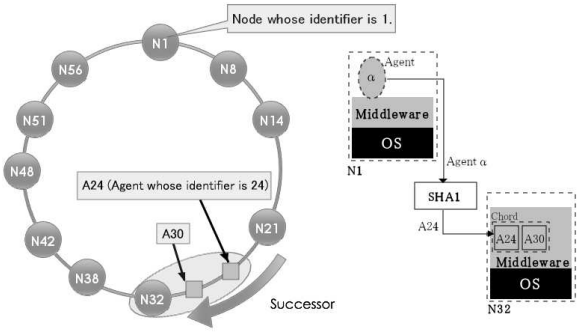


Figure 2: A Chord ring with 10 nodes and 2 agents.

Chord. In Chord (Stoica et al., 2001), when a node receives an agent lookup request, this node first looks up on its local hash table and the request is forwarded to another node while the lookup operation is not resolved. To perform this operation, Chord uses a ring formation of nodes, enabling a node to forward its lookup request. Figure 2 shows a Chord ring example, which uses SHA1, to assign identifiers to agents and their locations (*i.e.*, nodes). In Figure 2, an agent

that is working on node N1 is assigned its identifier as 24 by using SHA1 hash function. A node points to its *successor*, whose identifier is greater. For example, N21 points to successor N32. Using these identifiers, a 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. In Chord, the location of an agent is managed by the node whose distance between them is the shortest. For example, Figure 2 shows that A24 is working on N1, but its location is managed by N32, meaning that the local hash table of N32 has an entry "A32 $\rightarrow$ N1".

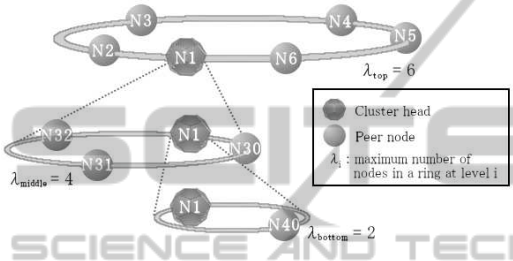


Figure 3: Example of CSN's rings (a hierarchy of rings of Chord).

CSN. In WSNs, a packet loss usually happens when the *physical* distance between two nodes is getting longer. As the Chord algorithm does not consider this kind of distance when a ring is created, CSN, a variant of Chord (Chord for Sensor Nodes), addresses the physical distance issue because CSN considers the distance to select a successor. The algorithm introduces a hierarchical clustering approach, where each cluster of nodes follows the ring formation of Chord. Using the hierarchical structure shown in Figure 3, CSN guarantees its efficiency (see (Ali and Uzmi, 2004) more details). However, CSN cannot be applied to lookups of mobile agents in WSNs because of several reasons. CSN assumes that a base station starts ring creations and lookups. This assumption is not adequate for mobile agent middlewares because an agent requires starting a lookup at an arbitrary node.

4 AN EFFICIENT LOOKUP OF AGENTS

The core of our proposal consists of two stages: Creation of Rings and Lookup Behavior. This section explain them in detail.

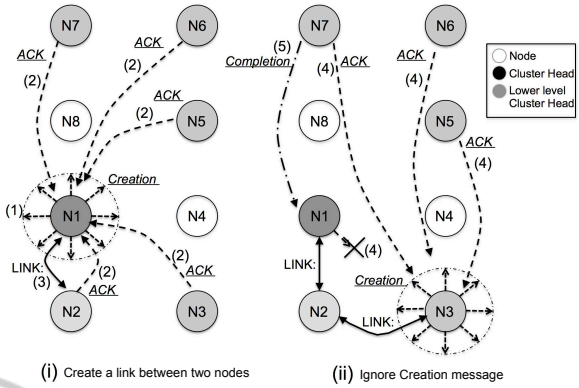


Figure 4: Illustration of steps for a ring creation based on the physical distance among nodes.

4.1 Creation of Rings by an Example

Our proposal uses hierarchical cluster structures of CSN's rings. As shown in Figure 3, each cluster contains a *cluster head* which joins at least two rings (*e.g.*, top and middle level). In addition, a node has to communicate with another node which is next to it by single hop manner to save its battery consumption. 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. This approach implicitly assume that any nodes are added and removed at runtime. Different from the Internet based P2P network, this assumption is now feasible because the number of nodes is fixed and each node is not commonly added or removed in WSNs[1] used in Agilla.

Next, by using Figure 4, we explain how to create a ring, which corresponds to the top level ring and the numbers in Figure 4 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 Creation message. Note that, in WSNs, every message is sent as 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.

[1]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.

3. **Create Neighbor Relations.** After node N1 receives ACKs from other nodes, this chooses a successor out of nodes that reply ACKs. In current implementation, we use *RSSI* that means the strength of the radio wave for this choice. As a consequence, node N2, which is the closest node from N1, is chosen. Then, N1 sends a message to N2 to inform this choice. After receiving this message, node N2 registers node N1 as its predecessor; creating a link between N1 and N2.
4. ***Repeat and Ignore Creation Message.*** A node that is chosen as a successor starts sending the Creation message. Therefore, the link between node N2 and N3 is created following the steps 1 to 3. Note that, in Figure 4-(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 which starts creating a ring (*e.g.*, node N1) ignores any Creation messages. At the same time, a node which 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 are created by step 4. In Figure 4-(ii), when node N7 sends a Creation message, no node replies. If node N7 cannot receive any ACK from others under a predefined period of time, node N7 will send a message to complete the creation of a ring, named Complete message. A node that starts creating a ring (*e.g.*, node N1) replies against this Complete message. As a consequence, the final link between node N1 and N7 is created[2].

After creating the ring of the upper level, our proposal repeats the previous process to create rings of lower levels, then ends up creating all rings at every level.

4.2 Assigning Identifiers

DHT algorithms must effectively manage a hash table in a distributed manner. Thereby, assigning an identifier to each node and agent is crucial. In Chord, it is not necessary how many nodes in a ring beforehand because Chord uses only one ring, so all nodes belong to this ring. Instead, similar to CSN, our proposal 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 later.

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

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

$$node\_id_l(i) = node\_id_l(i-1) + scope(l) \quad (2)$$

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

In these equations, l is the ring level (*e.g.*, 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 that 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 by 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 in the upper level.

Figure 5 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 level. 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 identifier 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 to 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 *otherwise* 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 of N2 at the middle level is calculated to 14 (14 + 0) Finally, the identifiers of N7 and N8 at the middle level are calculated 28 and 42 by equation (2). By repeating this

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

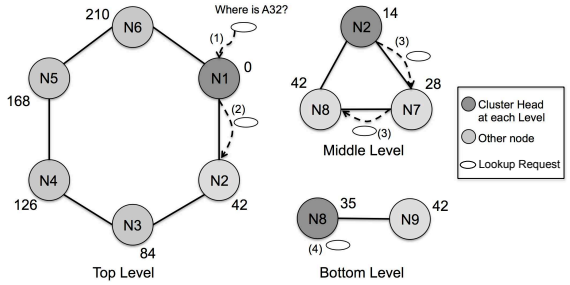


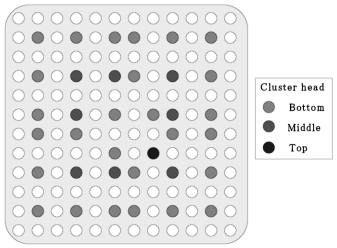
Figure 5: Illustration of assignment identifiers and lookup behavior steps in our proposal.

algorithm, unique and well organized identifiers are assigned to every node on WSNs.

4.3 Lookup Behavior

The lookup behavior in our proposal is based on Chord. Similarly to CSN, a cluster head has at least two successors; thereby, 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. The lookup example shown in Figure 5 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 from 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 of the middle and bottom level manage identifiers from 211 to 255, meaning that A32 is not managed by N1 and its lower rings. Thereby, 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.*, N7). For the same reason, N7 forwards the request to N8.
4. ***Answer the location of an agent or NotFound.*** The identifier of N8 at the middle level is 42, therefore N8 and its bottom ring must manage the location of A32. The identifier of N8 at the bottom level is 35, therefore N8 is the candidate that must manage the identifier of the request. Finally, N7 replies the location of A32 (*i.e.*, node identifier) if the local hash table contains the location, otherwise N8 returns NotFound message.



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

Figure 6: Configuration of rings used for the experiment.

5 EVALUATION

This section shows that our proposal addresses the battery consumption problem. To show to what extent our proposal addresses the problem, we compare our proposal 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.

Table 1: Comparison of the battery consumption between our proposal, Random Walk, and Flooding.

Battery capacity: 21,600 Joules.		
Approach	DHT creation	Per lookup
CMSN	1.556	0.0049
RW	0	0.1170
Flooding	0	0.0432

Experiment Setup. For these approaches, we use PowerTOSSIMZ (Perla et al., 2008) for TinyOS. This emulator works for TinyOS 2.1.1 and is deployed on Intel Core i5 (2.4 GHz) with 8GB of RAM running Ubuntu 12.04 (x32). Figure 6 shows the simulation configuration. The grid is composed of a 12×12 nodes. In this experiment, 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 our proposal, we carried out 500 lookup operations to get an average result of these ap-

proaches. For only our proposal, we evaluated in two stages: *a)* creation of rings (*i.e.,* DHT structures) and *b)* per lookup operation. To prevent infinite lookup operations in Random Walk and Flooding, we introduced time to live (TTL) as IP does in order to represent a *failed lookup*: a lookup operation that takes a greater period of time than a threshold.

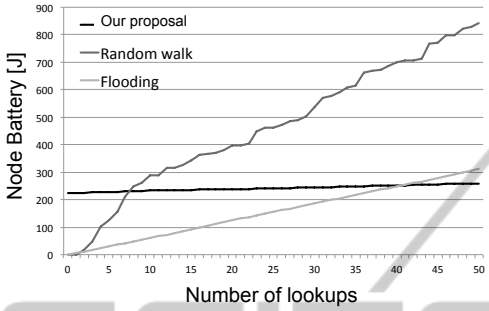


Figure 7: Comparison of our proposal to Random Walk and Flooding.

5.1 Result and Discussion

Table 1 shows the comparison of battery consumption between Random Walk, Flooding and our proposal. Although our proposal consumes to create DHT structures, this only consumes 1.556J (0.0072%) of the battery of each node to create them. In addition, our proposal consumes 0.0049J (0.226×10^{-4}%) of the battery per lookup operation. In contrast, Random Walk and Flooding consume 0.1170J (5.5415×10^{-4}%) and 0.0432J (1.998×10^{-4}%) of each node respectively. Figure 7 also shows the relationship between battery consumption and the number of lookup operations in Random Walk, Flooding and our proposal. As our proposal requires additional battery to create structures, Random Walk and Flooding consume less battery 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, our proposal consumes less battery.

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

6 CONCLUSIONS

As WSNs are exposed to a real world, many of their applications are required to adapt to the environment changes. To address these problems, different middlewares for mobile agents have been proposed, where an application is composed of a set of agents and is executed by the interactions of these agents. For this approach, an agent needs to know the exact location of its target agent beforehand. However, existing proposals, including Agilla, do not support an efficient lookup mechanism to lookup agents. In this paper, we propose an approach that borrows ideas from the CSN algorithm to efficiently lookup agents in WSNs within a specific period time. This efficient lookup allows WSN nodes to save battery consumption. We implement our proposal on the TinyOS environment and verified its advantages via a comparison with traditional lookup methods.

REFERENCES

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

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., and Wood, A. (2004). EnviroTrack: Towards an environmental computing paradigm for distributed sensor networks. In *Proceedings of the 24th International Conference on Distributed Computing System*, pages 582–589.

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

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

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

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

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

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable

peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160.

Yick, J., Mukherjee, B., and Ghosal, D. (2008). Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330.