PROCEEDINGS for the

IEEE International Conference on Pervasive Services 2005

(ICPS '05)

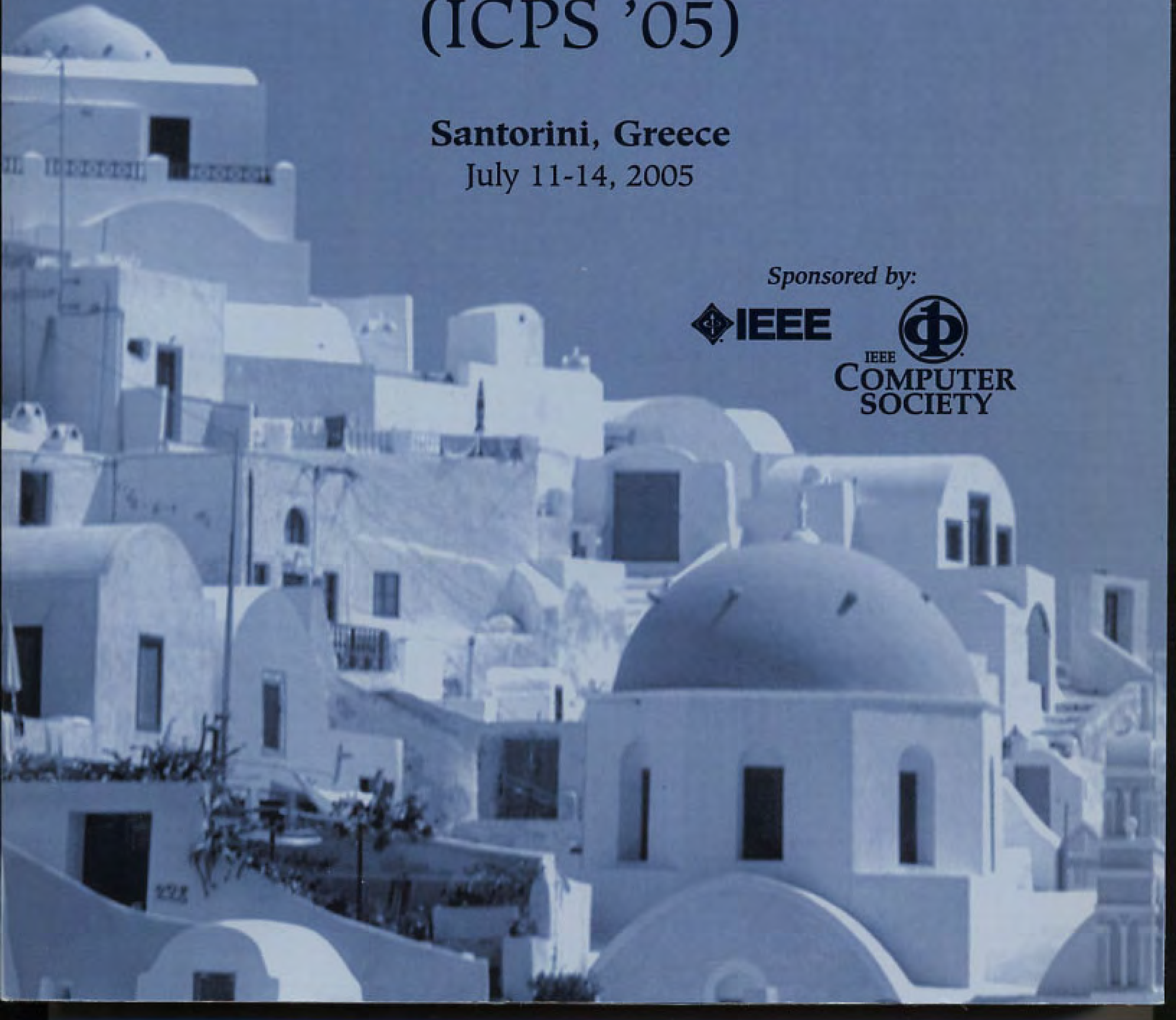
Santorini, Greece

July 11-14, 2005

Sponsored by:

IEEE

IEEE COMPUTER SOCIETY



IEEE Catalog Number:	05EX1040
ISBN:	0-7803-9032-6
Library of Congress:	2005922346

Proc. of IEEE Int'l Conference on Pervasive Services 2005 (ICPS'05), pp.198-207 (July 2005).

A Rule-Based Discovery Mechanism of Network Topology among Ubiquitous Chips

Yasue Kishino†, Tsutomu Terada†, Masahiko Tsukamoto‡, Tomoki Yoshihisa†, Keisuke Hayakawa*, Atsushi Kashitani⋆, and Shojiro Nishio†

† *Graduate School of Information Science and Technology, Osaka University*
‡ *Faculty of Engineering, Kobe University*
∗ *3rd Systems Operations Unit, NEC Electronics*
⋆ *Internet Systems Research Laboratories, NEC Corp.*

Abstract

In this paper, we propose a new network topology discovery mechanism among ubiquitous chips, which are rule-based, event-driven input/output (I/O) control devices to compose ubiquitous computing environments. Since they achieve flexibility by describing behavior in a set of rules, we employ a rule-based approach to discover network topology. In ubiquitous computing environments, we use various communication methods and applications at the same time. Therefore, our flexible discovery mechanism works well in ubiquitous computing environments. Moreover, we verified our algorithm by implementing it on a topology discovery simulator and actual prototype devices of ubiquitous chips.

1 Introduction

Recent evolutions in the miniaturization of computers and such component devices as microchips, sensors, and wireless modules have contributed to the realization of ubiquitous computing environments [4, 9, 11]. We previously proposed a new ubiquitous computing environment using rule-based input/output (I/O) control devices [10] called *ubiquitous chip* (Figure 1). We expect that in the future, ubiquitous chips will be embedded into such any artifacts as furniture, appliances, walls, and floors because they can provide various services to support human daily life.

The behaviors of a ubiquitous chip are described by a set of event-driven rules. Although the ubiquitous chip has low processing power and small memory, it can dynamically change its behavior by modifying stored rules. Moreover, we can also change devices connected to a ubiquitous chip at any time. Such software/hardware modifications can be performed while applications are running. Using ubiquitous chips, we can compose flexible applications for ubiquitous computing environments. Since a ubiquitous chip can achieve complex behaviors by cooperating with other ubiquitous chips, we can freely move an artifact and add/remove ubiquitous chips and devices to/from the location.

To realize various flexible applications, the system needs to know the network topology among ubiquitous chips. Therefore, we propose a network-topology discovery mechanism for ubiquitous chips that employs a rule-based approach to discover network topology, and we achieve flexible discovery methods that work well in ubiquitous computing environments.

The reminder of this paper is organized as follows. Section 2 outlines the ubiquitous chip, and Section 3 describes our mechanism that discovers network topology based on a rule-based approach. Sections 4 and 5 explain the realization and verification of our algorithm using a topology discovery simulator and actual prototype ubiquitous chip devices. Section 6 discusses our methods, and Section 7 sets forth the conclusion and planned future work.

2 Ubiquitous Chip

2.1 Overview

Figure 1 shows a prototype of the ubiquitous chip. Along its sides it has five digital input ports, one analog input port, twelve digital output ports, and two serial communication ports. As shown in Figure 2, we can attach several input/output devices to these ports.

The behaviors of ubiquitous chips are described by ECA rules, which have been used for describing behaviors in event-driven databases. An ECA rules consists of Event, Condition, and Action. Event is an occurring event, Condition is a condition for executing actions, and Action is an operation to be carried

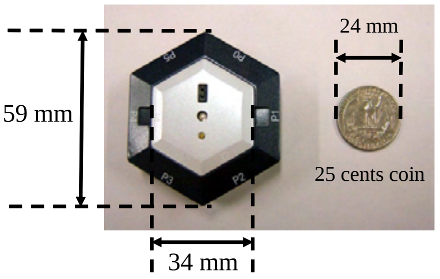


Figure 1. Ubiquitous Chip

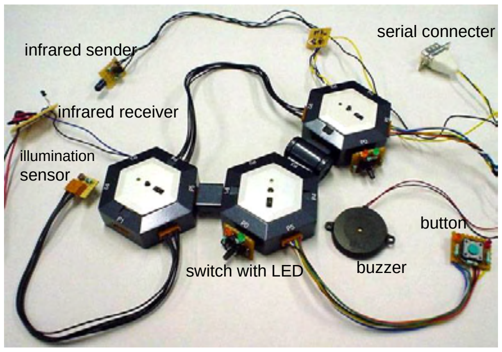


Figure 2. Attachments for Ubiquitous Chip

Table 1. Events

Name	Contents
RECEIVE_MESSAGE	receiving eight types of messages via a serial port
RECEIVE_DATA	1 byte data reception via a serial port
TIMER	Firing a timer
NONE	Evaluating conditions at all times

Table 2. Actions

Name	Contents
OUTPUT	On/Off control of output ports
OUTPUT_STATE	On/Off control of state variables
TIMER_SET	Setting a new timer
SEND_MESSAGE	Sending a message
SEND_DATA	Sending a 1 byte data
SEND_COMMAND	Sending a command to control another ubiquitous chip
HW_CONTROL	Hardware control

out. Since a ubiquitous chip has little processing power and small memory, we simplified the language specifications of ECA rules while maintaining the ability to fulfill various requirements in ubiquitous computing environments. Tables 1 and 2 show the lists of events and actions that can be done on ubiquitous chips. We can describe the expiration of a timer and the reception of a message as an event, the situation of input-ports and internal variables as a condition, and output ports control and sending a message as an action.

2.2 Communication Function

The communication function is one of the most important features of ubiquitous chips. A ubiquitous chip can communicate with other chips via its serial communication ports. These ports transmit information by means of serial cables or wireless communication modules. We provide various wireless communication units for the ubiquitous chip, such as an Infrared (IR) unit, a Radio Frequency (RF) unit, and a bluetooth unit. All of these units work with low battery consumption, and they have different communication areas from several centimeters to dozens of meters. These characteristics contribute to flexible ubiquitous computing environments with ubiquitous chips. Figure 3 shows a prototype unit of an RF communication module attached to the ubiquitous chip to make wireless serial ports.

Action for Communication

The ubiquitous chip handles three types of packets: *message*, *data*, and *command*. As shown in Table 2, the SEND_MESSAGE action sends a message that has a specific ID (0–7) via a serial communication port. The SEND_DATA action sends 1 byte data specified in the rules or that is input from the analog port. We designed the SEND_MESSAGE action for controlling other ubiquitous chips and the SEND_DATA action for collecting data from connected sensors and sending it to another ubiquitous chip or a server. We can manage ECA rules stored in remote ubiquitous chips by sending several commands with the SEND_COMMAND action. Table 3 shows a list of commands. By sending these commands, we can add, delete, enable, and disable specific ECA rules. Using these commands, the ubiquitous chip can control surrounding ubiquitous chips.

Multihop Function

The ubiquitous chip has two communication modes: *single-hop* and *multihop*. In the single-hop mode, the ubiquitous chip processes a packet as soon as it received the packet. On the other hand, in the multihop mode, a packet is sent to its destination according to specified communication pathways. This method enables a ubiquitous chip to send a message to another ubiquitous chip located outside of its direct communication area.

Table 3. Commands for the SEND_COMMAND

Name	Contents
ADD_ECA	Adding a new ECA rule
DELETE_ECA	Deleting a specific ECA rule(s)
ENABLE_ECA	Enabling a specific ECA rule(s)
DISABLE_ECA	Disabling a specific ECA rule(s)
REQUEST_ECA	Requesting a specific ECA rule

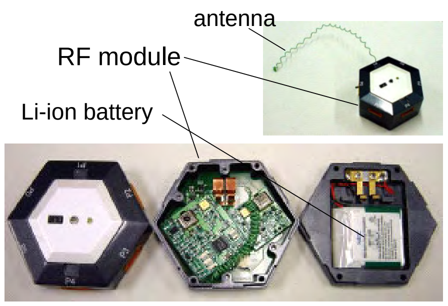


Figure 3. Wireless communication module

When a ubiquitous chip sends a packet via multihop communication, it adds a multihop header to the packet that consists of a list of ubiquitous chip IDs, including destination information, communication pathway, length of the ID list, and the current pointer on the list. Figure 4 shows the format of the multihop command. Each ubiquitous chip has 1 byte ID. We can specify a particular ID (broadcast ID), which means the ID of all ubiquitous chips. When a ubiquitous chip receives a multihop packet, it follows these steps:

Step 1: The ubiquitous chip checks the ID of the current pointer. If it matches the ID of the ubiquitous chip or the broadcast ID and the ID of the ubiquitous chip is not contained in the list, it goes to the next step. Otherwise, the ubiquitous chip ignores the packet.

Step 2: If the current pointer and the length of the list are not same (the ubiquitous chip is not at the end of the pathway), the ubiquitous chip increments the current pointer and sends the packet to ubiquitous chips in the next step. Otherwise, it goes to the next step. When the current pointer points the broadcast ID, the ubiquitous chip inserts its own ID into the list instead of the broadcast ID to avoid packet circulation.

Step 3: If the last ID of the ubiquitous chip matches its own ID or the broadcast ID, the ubiquitous chip processes the contents of the packet.

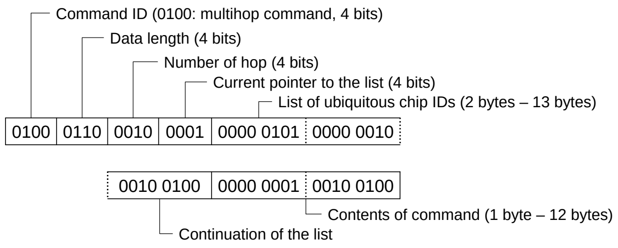

Command ID (0100: multihop command, 4 bits)
Data length (4 bits)
Number of hop (4 bits)
Current pointer to the list (4 bits)
List of ubiquitous chip IDs (2 bytes – 13 bytes)
0100
0110
0010
0001
0000 0101
0000 0010
0010 0100
0000 0001
0010 0100
Contents of command (1 byte – 12 bytes)
Continuation of the list

Figure 4. Format of multihop commands

3 Rule-Based Topology Discovery

3.1 Scenario

In our assumption, dozens of ubiquitous chips are embedded all over the place to which various sensors and output devices are connected (illustrated in Figure 5). These ubiquitous chips collect data from sensors, send messages, and control attached devices according to the received data via wireless multihop communication. A server, which is considered as a laptop or a PDA, manages network topology when necessary. We consider the following scenarios.

Building automation system:

Using these devices, we assume a building into which many kinds of sensors and ubiquitous chips have been embedded to develop a building automation system. In the surrounding area of important sensors, the system should discover all communication pathways among ubiquitous chips (mesh topology) because stable communication capacities are required. On the other hand, in an environment of unimportant sensors, it is not required. Moreover, to conserve the batteries of ubiquitous chips, the system is also required to discover minimum pathways to such sensors to reduce the number of messages. For example, in a building where entrance and exit management is important, door switches are important and the system should discover mesh topology in the surrounding areas of such sensors. In a building using a temperature management system, temperature sensors are important.

Networking based on reliability:

We consider an environment where the network reliability varies from place to place. In such environment, the system needs to repeat sending messages to correct errors at places of low network connectivity. On the other hand, at places of stable network, such repetition is unnecessary. Therefore, the system needs to change its method

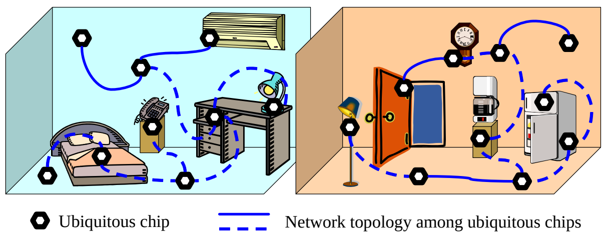


Figure 5. Image of our assumed environment

of discovering network topology according to network reliability. In particular, we have to consider this problem in a system in an environment that uses both wired and wireless networks.

In this way, applications in ubiquitous computing environments need a flexible network-topology discovery mechanism. Moreover such flexible mechanism cannot be realized by using conventional protocols in ad hoc network areas.

3.2 Approaches

A flexible discovery mechanism should discover only a necessary part of the network topology because the server has to minimize the number of messages, and the required communication structure varies depending on surrounding situations and application requirements.

Moreover, such a flexible mechanism should have the capability to change itself dynamically to handle various application requirements and communication methods. This feature also enables the system to adopt new discovery methods in the future. Additionally, we suppose that embedded devices such as the ubiquitous chip have little memory for storing routing tables.

For constructing such a flexible mechanism, we propose a rule-based topology-discovery mechanism. The introduction of a rule-based mechanism leads to the following advantages:

- The mechanism can dynamically change its own behavior because behavior is represented as a set of rules and the ubiquitous chip can add, delete, enable, and disable these rules.
- When a new method is proposed, we can easily implement it by modifying rules in embedded ubiquitous chips without any firmware update.

4 Topology Discovery for Ubiquitous Chips

4.1 Environmental Assumptions

We realized our proposed mechanism on ubiquitous chips and consider that this mechanism will be used in the following situations:

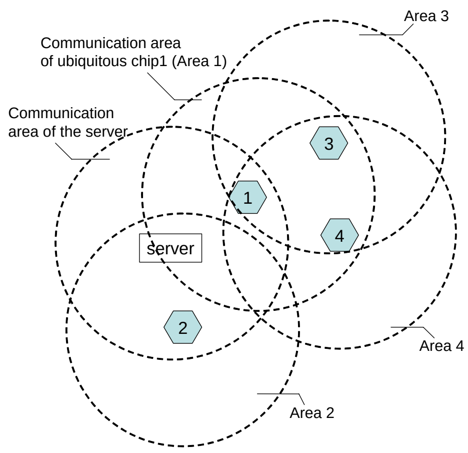
- There are dozens of ubiquitous chips in a room and some applications converge. A user brings his PDA to the room, modifies the configuration of the ubiquitous chips, and customizes applications by using his PDA.
- When a user enters a room with his PDA and some ubiquitous chips, he incorporates these ubiquitous chips into previous applications to support him by buzzers and vibration motors. His PDA discovers the network topology among ubiquitous chips, including additional ones, and modifies the stored rules on the previous ubiquitous chips to collaborate with the new ones.
- When a new piece of furniture, which includes an embedded ubiquitous chip, is added to the room, someone modifies the rules on the ubiquitous chips in the room.
- In cases where network topology changes frequently, we place a server in the room to manage topology and maintain applications.

Since the ubiquitous chip already has a rule-based engine, we realize the rule-based topology-discovery method using ECA rules and the communication functions of the ubiquitous chip. We assume that the network topology changes infrequently. Therefore, in actual applications, the server shifts two modes: *application* and *topology maintenance*. The server shifts to the topology maintenance mode at regular intervals to check the topology modification. This shift is also realized by the ENABLE_ECA and the DISABLE_ECA commands.

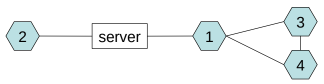
The ubiquitous chip and ECA rules have some limitations. The maximum length of the packet is 16 bytes, including the multihop header and information for multihop communication in Action of the ADD_ECA command. For example, when we add a 4 byte rule to another ubiquitous chip via multihop communication, the maximum number of hops is seven. We cannot add a rule including action using long multihop via long hop communication.

The IDs of ubiquitous chips are 1 byte. In this paper, we assume that the IDs of ubiquitous chips in a place are unique.

Figure 6 shows an example of topology discovery. When ubiquitous chips are allocated as illustrated in Figure 6 (a), the server manages ubiquitous chip IDs as a mesh topology (Figure 6 (b)). Each node of the mesh has information of its ID, the ID of its parent, the IDs of its children, and the other IDs that can communicate with the ubiquitous chip (Figure 6 (c)).



(a): Allocation of ubiquitous chips in the real world



(b): Mesh topology of ubiquitous chip

(c): Content of each node

ID	parent	children	other
server	-	1, 2	-
1	server	3, 4	-
2	server	-	-
3	1	-	4
4	1	-	3

Figure 6. An example of the topology discovery

4.2 Fundamental Methods

We realized two methods to analyze communication topology among ubiquitous chips: *direct* and *indirect*. The direct method can reduce the number of packets, and the indirect method can discover ubiquitous chips in longer distance. Since both protocols are realized by using ECA rules, they can be easily implemented on ubiquitous chips. On the other hand, our method requires a server at the discovery of network topology. The server modifies the ECA rules in ubiquitous chips at the location, sends inquiry messages to them, receives reply messages from them, and analyses the network topology among them.

In the followings, we explain the two methods in detail that can discover a mesh topology among ubiquitous chips. After that, we explain a method that maintains topology.

4.2.1 Direct Method

Table 4 shows the ECA rules for the direct method. The procedure for discovering a network topology is as follows:

Table 4. ECA rules for direct method

WAIT_UCID
E: C: Flag on A: Setting a timer for a proportional time to its ID
REPLY_MESSAGE
E: Reception of REQUEST_UCID message C: A: Flag on
E: Expiration of the timer C: A: Sending the REPLY_UCID message
REQUEST_UCID: Requesting the ubiquitous chip ID REPLY_UCID: Sending the ID to the server

Step 1: All ubiquitous chips have WAIT_UCID rule beforehand.

Step 2: The server sends the ADD_ECA command to store the REPLY_MESSAGE rules to the ubiquitous chips that can directly communicate with the server.

Step 3: It sends the REQUEST_UCID message to them.

Step 4: It receives the REPLY_UCID messages that are replies to the message from them.

Step 5: It records IDs in replies and adds them to the mesh topology information.

After these operations, the server begins to discover ubiquitous chips that can communicate with the server via two-hop communication. The server can discover ubiquitous chips in the second hop, as in steps 2–5.

Step 6: The server selects the ubiquitous chip as the *current* ubiquitous chip from those handled in the previous hop, which a minimum of IDs.

Step 7: The server sends the ADD_ECA command to store the REPLY_MESSAGE rules in the ubiquitous chips within a hop from the current ubiquitous chip. At this point, the server sends a multihop command with the multihop header, which includes the ID of the current ubiquitous chip as the first hop and the broadcast ID as the second hop.

Step 8: It sends the REQUEST_UCID message to these ubiquitous chips.

Step 9: It receives the REPLY_UCID messages from them.

Step 10: It adds these IDs to the mesh topology. If an ID has already been in the topology, the server records the ID to the direct communication list for the current ubiquitous chip. Otherwise, the server records the ID as a child of the current ubiquitous chip.

Step 11: It deletes the REPLY_MESSAGE rules stored in Step 7 to prevent the sending of superfluous REPLY_UCID messages.

Table 5. ECA rules for indirect method

SEND_MESSAGES
E: C: Flag on A: Waiting for a proportional time to its ID and sending the REPLY_UCID_M messages
REPLY_MESSAGES
E: Reception of the REQUEST_UCID_M message C: A: Flag on
RELAY_MESSAGE
E: Reception of REPLY_UCID_M message C: A: Sending the REPLY_UCID_M message
REQUEST_UCID_M: Requesting the ubiquitous chip ID REPLY_UCID_M: Sending the ID to the server by a set of messages

Step 12: If there is another ubiquitous chip that has not been selected as the current ubiquitous chip, it is selected as the new current ubiquitous chip and the procedure returns to Step 7.

Step 13: The server selects new current ubiquitous chip, which has the minimum ID among the children of the previous hop. In the same way, the server repeats this operation over the third hop.

Step 14: If the number of hops is over the limit or all ubiquitous chips are discovered, the server finishes the procedure.

4.2.2 Indirect Method

Table 5 shows the ECA rules for the indirect method in which ubiquitous chips send their IDs to the server as a set of four messages, and the server discovers their IDs by reconstructing these messages. For example, if a ubiquitous chip, whose ID is 57 H (01010111), wants to send its ID to the server, the ID is divided into four messages whose IDs are 01, 01, 01, and 11. When the server receives these four messages, it reconstructs the original ID from them. By this means, the server can discover a ubiquitous chip ID instead of using the header of the message in the direct method. Therefore, this method works well where there are ubiquitous chips farther from the server compared with the direct method.

The discovery procedure of network topology by the indirect method is similar to the direct method. The differences are that all ubiquitous chips have a SEND_MESSAGES rule beforehand, and the server uses the REPLY_MESSAGES rule instead of the REPLY_MESSAGE rule in the direct method and the use of the RELAY_MESSAGE rule. The RELAY_MESSAGE rule is added to a ubiquitous chip when it becomes the current ubiquitous chip. The current ubiquitous chip receives the REPLY_UCID_M messages that represent the IDs of the new ubiquitous chips by the SEND_MESSAGES rule, and it relays them to the server or its parent by the RELAY_MESSAGE rule. The RELAY_MESSAGE rule is disabled to avoid duplicate message receptions when the current ubiquitous chip changes. It is enabled again when the first child of the current ubiquitous chip becomes the current one.

Table 6. ECA rules for discovery of a new ubiquitous chip

ADVERTISE_MESSAGE (for a new ubiquitous chip)
E: C: Flag off A: Setting a timer to repeat sending the NEW_UC message, and Flag on
E: Expiration of the timer C: A: Sending the NEW_UC message
STOP_ADVERTISE (for the new ubiquitous chip)
E: Reception of the STOP_NEW_UC message C: A: Killing the timer
NOTIFY_DISCOVER (for the others)
E: Reception of the NEW_UC message C: A: Setting timers 1 and 2
E: Expiration of timer 1 C: A: Sending the STOP_NEW_UC message
E: Expiration of timer 2 C: A: Turning on the flag to cancel the WAIT_UCID and SEND_MESSAGES rules
NEW_UC: Advertise from a new ubiquitous chip STOP_NEWUC: Stop the NEW_UC message

4.2.3 Comparisons between the two Methods

Comparing our two methods, the indirect method needs more messages than the direct method. Although a ubiquitous chip sends only one message to notify its ID to the server in the direct method, the indirect method consumes four messages for the same notification.

On the other hand, since the indirect method relays messages to the server without using long multihop path information, it is more flexible and scalable than the direct method. In the indirect method, a server can discover network topology in maximum seven hops, while three hops in the direct method. Moreover, we can easily enhance the indirect method to handle more than eight hops by adding several ECA rules.

Table 7. ECA rules for discovery of disappearing ubiquitous

CHECK_NEXT (for the parent ubiquitous chip)
E: C: Flag 1 is off A: Setting timer 3 to repeat sending the CHECK_ALIVE message, and flag 1 is on and flag 2 off
E: Expiration of timer 3 C: A: Sending the CHECK_ALIVE message to the child ubiquitous chip, and flag 2 is on
E: Reception of the REPLY_ALIVE message C: A: Flag 2 is off
REPLY_CHECK (for the child ubiquitous chip)
E: Reception of the CHECK_ALIVE message C: A: Sending the REPLY_ALIVE message
NOTIFY_DISAPPEAR (for the parent ubiquitous chip)
E: Expiration of timer 3 C: Flag 2 is on A: Turning on the flag to cancel the WAIT_UCID and the SEND_MESSAGES rules on
CHECK_ALIVE: Requesting a message to the child ubiquitous chip REPLY_ALIVE: Reply to CHECK_ALIVE message that means the ubiquitous chip is alive

4.3 Maintenance

We also realize a function to handle network topology changes that consists of two discoveries: new ubiquitous chips and disappearing.

Table 6 shows a set of ECA rules that discover new ubiquitous chips. In the indirect method, after the server discovers the network topology, it modifies the opponent of the SEND_MESSAGES rule to the parent. A new ubiquitous chip repeats sending the NEW_UC messages to surrounding ubiquitous chips by the ADVERTISE_MESSAGE rule. When a ubiquitous chip receives the message, it sends a STOP_NEW_UC message to the new ubiquitous chip and sends its ID to the server (by the NOTIFY_DISCOVER rule), and the new ubiquitous chip stops sending a message (by the STOP_ADVERTISE rule). When the server receives the ID, it selects the ubiquitous chip as the current ubiquitous chip and inquires about new ubiquitous chip IDs in the same way already described. In the direct method, the server may receive IDs from multiple ubiquitous chips that discovered the same new ubiquitous chip. In such situations, the server selects one of them to be the current one and disables the RELAY_MESSAGE rule from the other ones to avoid maintenance malfunctions.

Table 7 shows a rule set to detect disappearing ubiquitous chips. Using these rules, the server can check connections between two ubiquitous chips. Note that, this time the function of these rules is limited to check only one connection for a ubiquitous chip.

Between a pair of ubiquitous chips, the one nearer the server becomes the *parent*, who repeats sending the CHECK_ALIVE message to the other ubiquitous chip called the *child* by the CHECK_NEXT rule. When the child receives the message, it replies with the REPLY_ALIVE message (by the REPLY_CHECK rule). If the parent cannot receive this reply message, it sends its ID to the server (by the NOTIFY_DISAPPEAR rule). When the server receives the ID, it selects the parent as the current one and discovers the topology of its children.

Table 8. ECA rules for dynamic method

(a): Modified ECA rules based on the indirect method (case 2)

SEND_MESSAGES (replaced from indirect method)
E: C: Flag on A: Waiting a proportional time to its ID and sending REPLY_UCID_M messages several times one by one

(b): Modified ECA rules based on the indirect method (case 3)

SEND_MESSAGES (replaced from indirect method)
E: C: Flag on A: Waiting a proportional time to its ID and repeating send REPLY_UCID_M messages one by one
E: Reception of STOP_REPEAT message C: A: Changing flag to send next REPLY_UCID_M message
RELAY_MESSAGES (added to the current ubiquitous chip)
E: Reception of REPLY_UCID_M message C: A: Sending STOP_REPEAT message

(c): Added ECA rules to new ubiquitous chips to detect network reliability

CHECK_NETWORK
E: Reception of the CHECK_NETWORK message C: A: Sending the REPLY_CHECK message several times
CHECK_NETWORK: Requesting a message to new ubiquitous chips REPLY_CHECK: Reply to CHECK_NETWORK, which means network reliability

4.4 Dynamic Method

This method is suitable for environments where network reliability varies from place to place. In places with many errors and lost messages, message repetition is effective to discover correct topology. On the other hand, such repetition is unnecessary at places of stable networks. In a rule-based approach, we can easily solve this problem by changing rules according to network reliability.

Here, we describe a method for solving this problem based on the indirect method. In this method, we use several kinds of rule sets according to network reliability:

Case 1 (High reliability): The server adds the normal rule set of the indirect method.

Case 2 (Slightly low reliability): The server adds another rule set, shown in Table 8 (a), in which the ubiquitous chip repeats sending the REPLY_UCID_M message one by one. The number of repetitions depends on the degree of reliability.

Case 3 (Low reliability): The server adds the other rule set, shown in Table 8 (b), in which the ubiquitous chip repeats sending the REPLY_UCID_M message until it receives a reply message from the current ubiquitous chip. The server also adds a rule to the current ubiquitous chip to send the message when it receives the REPLY_UCID_M message. These operations are identical to the ADVERTISE_MESSAGE and STOP_ADVERTISE rules in Table 6.

Network reliability is detected by using the rules shown in Table 8 (c). The CHECK_NETWORK rule is added to all ubiquitous chips beforehand. When the new current ubiquitous chip is selected, the server sends one CHECK_NETWORK message to the new ubiquitous chips that can directly communicate with the current ubiquitous chips. When the new ubiquitous chip receives the CHECK_NETWORK message, it sends a REPLY_CHECK message. The server counts the number of received REPLY_CHECK messages and repeats this operation several times. After that, the server knows the reliability from the number of received REPLY_CHECK messages.

By dynamically selecting these rules, the server can realize an efficient topology discovery method. Moreover, the system can also use the reliability information in application rules.

All of these proposed methods are realized by ECA rules, and modification of firmware is unnecessary. Therefore, we can also easily realize new topology discovery methods by replacing them.

5 Implementation

5.1 Simulator

We implemented a simulator to verify the proposed methods. Figure 7 shows a screen shot of the simulator. We can place ubiquitous chips on the left side of the simulator, and the results are shown on the right side.

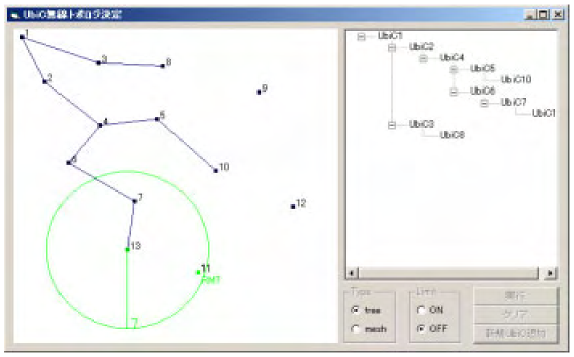


Figure 7. Simulator of topology discovery.

Table 9. Rule set for direct method

WAIT_UCID	
E:	
C: S0=1	
A: S0=0, T(ID × 100 ms)	
REPLY_MESSAGE (2 rules)	
E: RM(7)	E: Timer
C: S0=0	C:
A: S0=1	A: SM_M(6)

T: Setting a timer for a proportional time to its ID
RM: RECEIVE_MESSAGE event
SM_M: SEND_MESSAGE (multihop mode) action
Message 7: REQUEST_UCID message
Message 6: REPLY_MESSAGE message
S0: Flag for timer

In the simulator, the wireless communication area of the ubiquitous chip is set to a fixed range. Tables 9 and 10 show rule sets for the direct and indirect methods respectively.

The direct method cannot discover ubiquitous chips placed over five hops. The indirect method can discover ubiquitous chips placed within eight hops. However, the direct method operates a fourth as fast as the indirect method, and the number of messages is 35% to 50% as many as in the indirect method.

5.2 Verification on Prototype Devices

We also verified the direct method using actual prototype devices of ubiquitous chips. Figure 8 shows two examples of topology structures, and snapshots of verification. In this verification, we made the timer interval in the WAIT_UCID rule twice as long as that in Table 9 to prevent wireless communication reflections on the RF modules.

In Figure 8 (a), two ubiquitous chips (IDs: 6 and 8) within a hop can directly communicate with the server. Although ubiquitous chips 6 and 8 received Message 7 from the server at the same time, they replied one by one according to the WAIT_UCID rule, verifying that the server discovered the topology of these ubiquitous

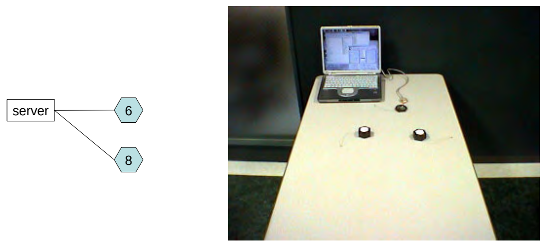
Table 10. Rule set for indirect method

REPLY_MESSAGES	RELAY_MESSAGES (4 rules)			
E: RM(5) C: S1=0 A: S1=1	E: RM(0) C: A: SM_M(0)	E: RM(1) C: A: SM_M(1)	E: RM(2) C: A: SM_M(2)	E: RM(3) C: A: SM_M(3)

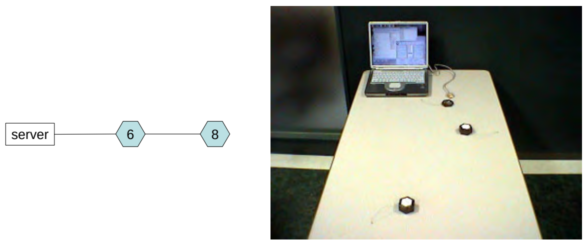
SEND_MESSAGES (5 rules)		
E: C: S1=1 A: S1=0, T(ID × 500 ms)	E: Timer C: S3=1, S4=1 A: S3=0, S4=1, SM(D)	E: Timer C: S3=0, S4=1 A: S3=1, S4=1, SM(C), T(100 ms)
E: Timer C: S3=1, S4=0 A: S3=0, S4=0, SM(B), T(100 ms)	E: Timer C: S3=0, S4=0 A: S3=1, S4=0, SM(A), T(100 ms)	

T: Setting a timer
RM: RECEIVE_MESSAGE event
SM: SEND_MESSAGE (single-hop mode) action
SM_M: SEND_MESSAGE (multihop mode) action
S1: Flag for the timer

Message 0 – 3: REPLY_UCID_M messages
Message A – D: Messages that represent ubiquitous chips ID (actually, these are Messages 0 – 3)
Message 5: REQUEST_UCID_M message
S3, S4: Flag for sending messages in order.



(a): Topology 1



(b): Topology 2

Figure 8. Topologies for the verification and snapshots

chips.

In Figure 8 (b), there is one ubiquitous chip (ID: 6) in the first hop and another ubiquitous chip (ID: 8) in the second hop. Ubiquitous chip 8 sends/receives messages and commands them to/from the server through ubiquitous chip 6. After discovering ubiquitous chip 6, the server selected it as the current one and was able to discover ubiquitous chip 8.

6 Considerations

6.1 Advantages of rule-based approach

We can also consider the hybrid method in which the server uses the direct method in shorter distances than the limit of this method and the indirect method in longer distances. In addition, servers sometimes might discover not mesh topology but tree topology because of messages reduction and saving batteries.

These customizations and adaptations are required to construct flexible applications in ubiquitous computing environments. In our approach, such customizations are easily realized by changing ECA rules, unlike conventional methods.

On the other hand, topology discovery methods realized in the firmware can be implemented in an optimum way. Therefore, our approach leads to some delay and overhead.

6.2 Related Works

MICA is a platform for wireless sensor networks that can automatically discover network topology [6]. We can collect sensor information from MICA devices. Moreover, many kinds of multihop networking protocols are implemented on MICA devices [5]. However, MICA cannot dynamically change the discovery method for network topology and it cannot change its behavior and attached devices during applications are active. This is the different point from ubiquitous chip, which is controlled by ECA rules.

Smart-Its is a tiny computer system to be embedded into everyday objects [1]. In Smart-Its, no flexible network topology discovery mechanism is proposed. We believe that we can implement our topology discovery mechanism on Smart-Its devices.

In recent years, several topology discovery methods have been proposed in the research area of ad hoc networks, which are networks constructed temporarily only by mobile hosts. Such topology discovery methods are divided into two types: proactive routing and reactive routing. In proactive routings such as DSDV [7] and OLSR [2], mobile hosts regularly exchange control messages and maintain routing information. In reactive routing, such as AODV [8] and DSR [3], mobile hosts have local routing tables and inquiry about routing paths when sending message. Since these routing protocols require enough processing power to calculate routing information and sufficient memory to maintain routing tables, it is difficult to apply these conventional protocols to ubiquitous chips. Moreover, since these methods are static, the system cannot dynamically change their topology discovery methods.

7 Conclusion

In this paper, we proposed a rule-based topology discovery mechanism for ubiquitous chips. In this mechanism and using our method, we can change discovery method flexibility by modifying stored ECA rules and develop flexible applications for ubiquitous computing environments. We also verified our algorithm by implementing it on a topology discovery simulator and actual prototype devices of ubiquitous chips.

In the future, we plan to develop a framework to modify application behaviors according to network topology. We also plan to develop topology detection and routing methods considering the electricity consumption of wireless communication and actuators.

Acknowledgments

This research was supported in part by “The 21st Century Center of Excellence Program” of the Japanese Ministry of Education, Culture, Sports, Science and Technology.

References

[1] M. Beigl and H. Gellersen: Smart-Its: An Embedded Platform for Smart Objects, Smart Objects Conference (sOc) 2003, 2003.

[2] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot: Optimized Link State Routing Protocol for Ad Hoc Networks, in Proc. of IEEE INMIC 2001, 2001.

[3] D. B. Johnson and D. A. Maltz: Dynamic Source Routing in Ad Hoc Wireless Networks, in Proc. of Mobile Computing 1996, 1996.

[4] J. Kahn, R. Katz, and K. Pister: Mobile Networking for Smart Dust, In Proc. of ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom99), pp. 271–278, 1999.

[5] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler: The Emergence of Networking Abstractions and Techniques in TinyOS, in Proc. of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004), 2004.

[6] MICA, http://www.xbow.com/products/Wireless_Sensor_Networks.htm.

[7] C. E. Perkins and P. Bhagwat: Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers, in Proc. of SIGCOMM 1994, 1994.

[8] C. E. Perkins and E. M. Royer: Ad-hoc On-Demand Distance Vector Routing, in Proc. of WMCSA1999, 1999.

[9] K. Sakamura: TRON: Total Architecture, in Proc. of Architecture Workshop in Japan'84, pp. 41–50, 1984.

[10] T. Terada, M. Tsukamoto, K. Hayakawa, T. Yoshihisa, Y. Kishino, A. Kashitani, and S. Nishio: Ubiquitous Chip: a Rule-based I/O Control Device for Ubiquitous Computing, in Proc. of Pervasive2004, pp. 238–253, 2004.

[11] M. Weiser: The Computer for the Twenty-first Century, Scientific American, Vol. 265, No. 3, pp. 94–104, 1991.